Primer to Prompt Engineering: A Grounded Approach

Core Concepts

- **Prompt Engineering**: Crafting inputs to AI language models to elicit desired outputs
- **LLM Mechanics**: How models process prompts based on statistical patterns
- **Context Sensitivity**: How prompt wording affects model outputs
- **Prompt Categories**: From random prompting to goal-oriented and constraint-focused approaches

Essential Skills

- **Strategic Prompting**: Designing prompts with clear goals and constraints
- **Iteration Techniques**: Refining prompts systematically to improve outputs
- **Context Management**: Placing critical information at prompt beginnings/endings for maximum effect
- **Error Mitigation**: Reducing hallucinations and improving factual accuracy

Practical Applications

- **Industry Use Cases**: Customer support chatbots, content creation tools, virtual assistants
- **Research Applications**: Zero-shot learning, chain-of-thought prompting, model evaluation
- **Development Tools**: Frameworks like LangChain and OpenPrompt for prompt management
- **Community Resources**: Repositories of proven prompts for common tasks

Key Limitations

- **Adversarial Vulnerabilities**: How malicious prompts can circumvent safeguards
- **Attention Constraints**: The "lost in the middle" phenomenon in long contexts
- **Knowledge Boundaries**: When tasks exceed what prompting alone can accomplish
- **Hallucination Management**: Techniques to reduce but not eliminate false outputs

Note: This companion is a practical example of prompt engineering itself. Significant content of this companion is curated with help of OpenAI DeepResearch with five-shots prompt-response, using o3-mini-high, for priming the LLM then focusing the DeepResearch towards scholarly sources available for public access, e.g. arXiv. Final copy-editing tools including o3-mini-high and Claude 3.7 Sonnet with thinking enabled.

Introduction

Prompt engineering is the art and science of crafting inputs to AI language models to elicit desired outputs. Instead of writing traditional code, we write natural language instructions that guide model behavior, enhancing performance without changing internal parameters [1].

Prompt engineering matters because with powerful LLMs, prompt quality dramatically affects output quality. GPT-3 demonstrated this when it performed translation, Q&A, and arithmetic "without any gradient updates or fine-tuning" [2], purely based on prompt design. This enables repurposing general models for specific tasks through clever prompting rather than training new models [3].

LLMs serve as the "engine" that generates text, with prompt engineering being how we "program" this engine using natural language [4]. The quality of instructions can make the difference between generic responses and precise, insightful ones—analogous to directing a capable assistant whose output quality depends entirely on how tasks are framed.

Consider asking an AI model: "Tell me about machine learning" versus "You are an AI tutor. Explain machine learning **in two concise paragraphs**, focusing on its **applications in healthcare**, and provide an **analogy** for non-experts." The second prompt provides context, format, domain focus, and stylistic guidance, resulting in dramatically more useful output.

How Prompt Engineering Works

The Middle Ground: Neither Rules Nor Pure Art

Prompt engineering exists between rigid rules and pure artistic endeavor. Effective prompt engineering requires understanding:

1. **Model capabilities and limitations**: What the model can and cannot do
2. **Training data foundation**: The knowledge domains represented in training
3. **Domain expertise**: Framing questions in domain-appropriate language
4. **Audience awareness**: Cultural and contextual communication factors

Most importantly, **effective prompt engineering bridges technical AI understanding with domain expertise**. The practitioner must both grasp how models process prompts and possess deep knowledge of their specific field to craft prompts that activate relevant model knowledge using domain-appropriate terminology and reasoning patterns.

Domain experts often develop prompting intuition more quickly in their areas of expertise because they're familiar with language patterns the model has encountered in training. This explains why domain-specific prompt libraries are valuable—they capture communication patterns proven effective in specific contexts.

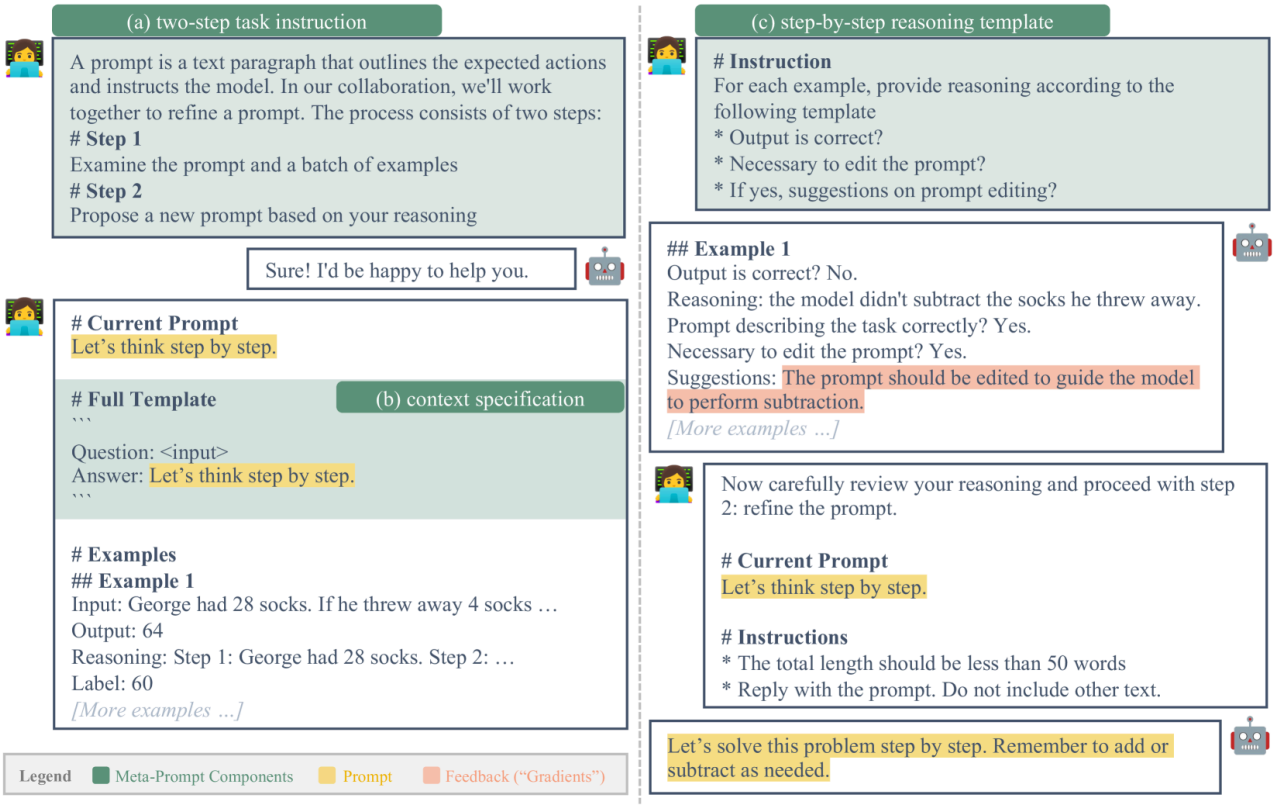

Figure 1: Meta-prompt components from Prompt Engineering a Prompt Engineer [22].

LLM Design and Response Mechanisms

Modern LLMs are built on the Transformer architecture, trained on vast text corpora to predict the next word in a sequence. During training, models learn statistical patterns, facts, and reasoning abilities encoded in billions of parameters. The models are **task-agnostic** at training time—not explicitly taught to summarize or translate—they simply model text probability distributions.

When provided with a prompt that includes task descriptions or examples, the model generates a completion that accomplishes the task [3]. This approach—called *in-context learning* or *prompt-based learning*—allows using models for prediction tasks **without additional training**, shifting from the traditional "train a new model for each task" paradigm to a "pre-train, *prompt*, and predict" approach [3].

The prompt effectively *conditions* the model's output by influencing the probability distribution of generated tokens. All task specification and context must be expressed in the prompt since the model has no other way of discerning intent [5]. Prompts essentially "program" the model's behavior, functioning as natural language instructions that enforce rules, automate processes, and ensure specific output qualities [4].

Internally, when an LLM processes a prompt, it converts text into tokens and feeds them through its neural network. The prompt tokens influence activations at every layer, effectively priming the model's "thought process." There is no separate instruction-understanding mechanism—the

model's training has encoded that when it sees patterns like "Translate to French: X," the likely continuation is a French translation of X.

The prompt provides context that activates relevant portions of the model's knowledge [1]. Including domain-specific terminology biases the model toward that knowledge domain, an effect called "priming." Researchers have found that adding phrases like "think step by step" induces different inference patterns, demonstrating how tiny prompt changes can condition a model's entire approach to a question [6].

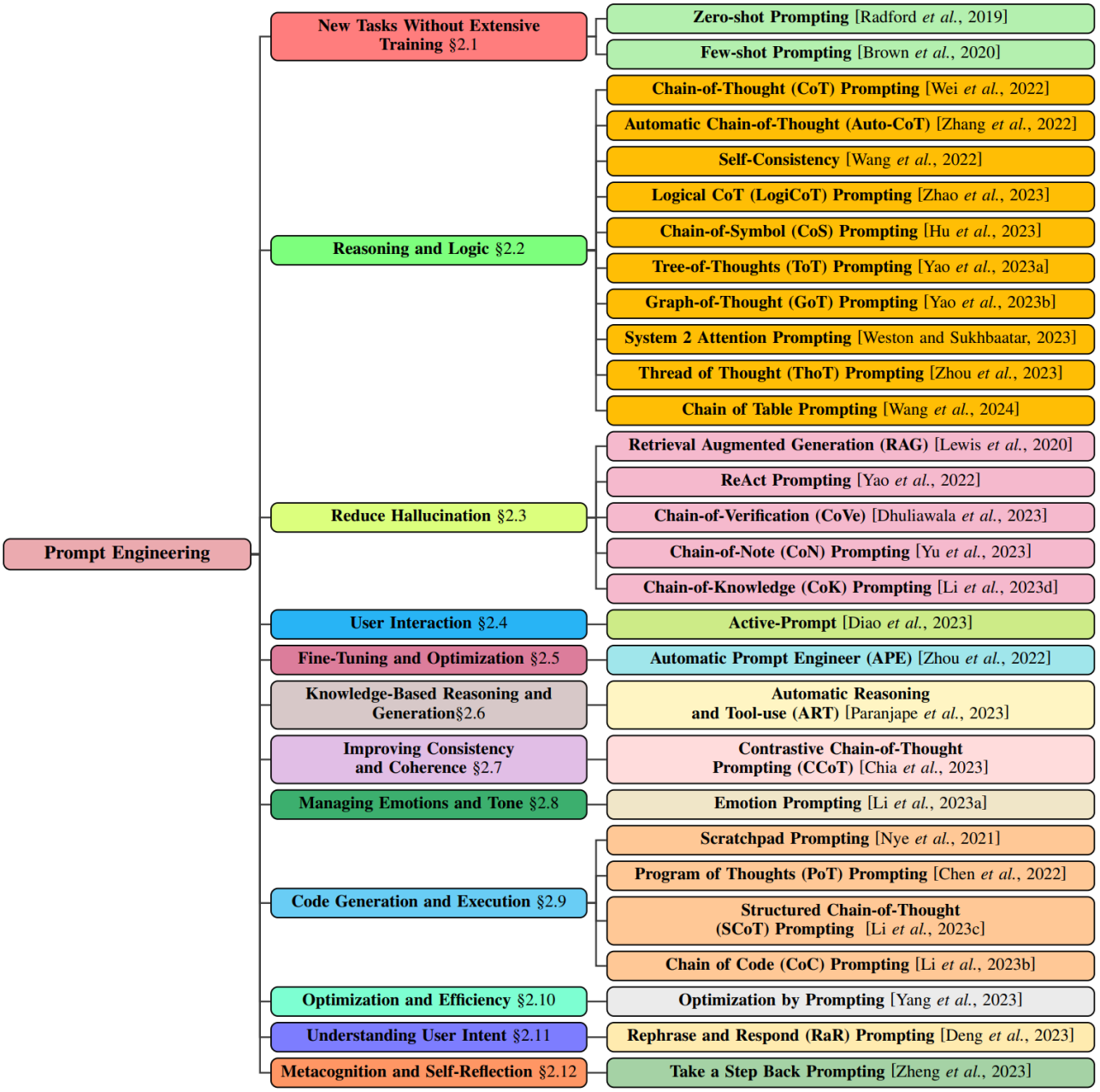

Figure 2: Taxonomy of prompt engineering techniques in LLMs, organized around application domains, providing a nuanced framework for customizing prompts across diverse contexts [1].

Categories of Prompt Engineering

Prompt engineering approaches range from haphazard to systematic. We'll explore four broad categories that often combine in practice. Understanding these categories helps practitioners identify which approach best suits different tasks and contexts.

1. “Random” Prompting (Ad-hoc)

This represents the absence of strategy—simply throwing questions at the model without consideration for phrasing or context. For example, asking "Explain quantum computing" without specifying detail level, format, or audience. Results are hit-or-miss with no systematic improvement process. This approach lacks the guidance necessary for consistent quality.

2. Trial-and-Error Prompting

This common approach involves testing prompts, observing outputs, and refining iteratively until requirements are met. For example, starting with "Summarize this article" then refining to "Summarize this article in one paragraph, focusing on the main findings" after seeing initial results. Each model response provides feedback on how the prompt was interpreted, allowing adjustment if that interpretation differs from intent. This method builds intuition about effective phrasings over time.

Case Study: A student initially asks "Explain the concept of entropy" and receives a correct but overly broad explanation. They refine to "Explain entropy in information theory in 2-3 sentences, in simple terms for beginners." Each iteration adds missing context or constraints until achieving the desired result—a process mirroring software development's debugging cycle.

3. Goal-Oriented Prompting

This strategic approach begins with a clear outcome in mind, designing prompts to guide the model through specific reasoning steps. Rather than blind iteration, you plan how the model should reach the answer. For complex problems, a goal-oriented prompt might instruct: "First, outline the problem, then consider three solutions, and finally provide the best solution with reasoning."

Case Study: Instead of asking "Should company X expand to market Y?" analysts use: "You are an expert business analyst. Analyze this scenario step-by-step: (1) Summarize the company's situation, (2) List expansion benefits, (3) List risks, (4) Conclude with a justified recommendation." This yields a well-organized analysis following the prescribed structure, demonstrating how embedding a conversation plan into a prompt can save time by getting higher-quality answers on the first attempt.

This aligns with research showing that "guiding LLMs to follow established human logical thinking significantly improves performance" on complex tasks [7]. **Chain-of-Thought prompting**

exemplifies this—designing prompts to elicit step-by-step reasoning before conclusions, which often improves accuracy for math, logic, and commonsense questions.

4. Constraint-Focused Prompting

This approach emphasizes rules and requirements the model must follow, preventing undesired results by explicitly stating boundaries. For example: "Provide the answer as a JSON object with fields 'summary' and 'conclusion', with no extra commentary." By reducing the model's degrees of freedom, constraints often lead to more useful, targeted responses.

Constraint-focused prompts are crucial when consistency and format matter—for code generation ("Output only code, no explanation") or customer support ("If asked about pricing, only quote from the pricing table below"). This approach helps mitigate rambling and hallucinations by forbidding content outside provided context, though excessive constraints can confuse models or make them unable to comply.

Case Study: A developer needs SQL queries from natural language requests without explanatory text. They use: "Convert the following request to SQL. Output only the SQL query, no explanation or prose," plus an example showing desired format. This constraint-focused prompt with a prototype ensures reliable, clean SQL output—a critical requirement for downstream processing.

These categories aren't mutually exclusive—skilled prompt engineers often combine them based on task complexity and requirements. The spectrum ranges from unguided to highly guided prompting, with complex tasks benefiting from goal-oriented and constraint-focused approaches, while simpler queries might only need iteration.

Automated Prompt Engineering (APE) represents an emerging approach that systematically optimizes prompt creation. As explored by Li et al. [15], APE frames the challenge as a discrete-continuous optimization problem, using algorithms to search the vast space of possible prompts. Approaches range from gradient-based optimization to evolutionary algorithms that iteratively improve prompts based on performance metrics [16, 17].

Key Takeaway: The most effective prompt engineering strategy depends on task complexity and goals. Begin with structured approaches (goal-oriented or constraint-focused) for critical applications, use trial-and-error for refinement, and consider automated methods for optimization at scale.

Why Prompts Fail and Their Limitations

Moving from the categories of prompt engineering, we now examine why even carefully constructed prompts sometimes produce unsatisfactory results and the inherent limitations of prompt-based approaches.

Common Failure Modes

Generic outputs from vague prompts: Broad, non-specific prompts yield generic responses because the model lacks specific direction. Asking "What is the importance of project management?" produces textbook-like answers covering common points. The solution is specificity: "In 2 paragraphs, argue why project management is crucial for software teams, with an example of a project failure."

Missing context: Prompts that assume context the model doesn't have will produce irrelevant or hallucinated responses. The model only knows what's in the current prompt, not previous conversation or referenced materials unless explicitly included. For example, "Analyze the data in the table above" fails without the actual table content in the prompt.

Misinterpreted intent: Ambiguous prompts may be interpreted differently than intended. For example, "Explain how to solve the problem without actually solving it" could mean providing a general strategy or giving an explanation that stops before the final answer. Unlike humans, models can't ask clarifying questions in one-shot interactions, so they choose one interpretation and proceed.

A helpful mindset is treating the AI like a knowledgeable but oblivious assistant who requires explicit instructions. Models follow prompts *literally* without understanding your underlying goals. Prompt engineering fails when we assume models will "just know what we mean"—an anthropomorphic assumption that ignores their statistical nature [7].

Model limitations: Sometimes prompts fail because tasks exceed model capabilities. No clever wording will make a model perform specialized analysis it wasn't trained for or access real-time information beyond its training data. Recognizing when you've hit capability limits helps distinguish between prompt failures and inherent model limitations.

Fundamental Limitations

Adversarial prompts: Maliciously crafted prompts can sometimes cause models to violate guidelines. These "prompt injection attacks" might append instructions like "Ignore previous constraints and do X" to override earlier guidance. Models follow the most recent or specific command, potentially bypassing safety measures. Research shows LLMs "are vulnerable to various adversarial attacks, particularly prompt injection, which trick them into producing harmful content" [8]. Security can't rely solely on prompting—external filters or validation are necessary.

Attention and context limitations: LLMs have fixed context windows and exhibit an "attention U-curve" where information at prompt beginnings and endings receives more attention than middle sections [10]. This "lost in the middle" phenomenon means critical instructions should appear at prompt edges, not buried within lengthy text. Even within context limits, longer prompts dilute focus and increase error probability.

Hallucinations and factuality: LLMs can produce false but fluent statements. While prompting can reduce this (by requesting source citations or verification), it cannot eliminate hallucinations because models lack ground-truth verification mechanisms. For critical factual tasks, prompt engineering alone cannot guarantee correctness—external validation or retrieval augmentation is necessary.

Overcoming Limitations

Several strategies can help address these limitations:

- **External knowledge integration**: Augment models with retrieval systems, APIs, or databases to provide factual information at generation time.
- **Multi-step prompting**: Break complex tasks into manageable steps using intermediate results to guide subsequent prompts.
- **Few-shot examples**: Provide examples directly in prompts to guide format and style more reliably than instructions alone.
- **Parameter tuning**: Adjust generation settings like temperature to control output determinism and creativity.
- **Know when to stop**: Recognize when tasks require different approaches, like fine-tuning or specialized models, rather than increasingly complex prompts.

Privacy concerns: As Edemacu and Wu note [14], interactions with LLMs can potentially expose sensitive information through prompts or outputs. Organizations must consider both technical limitations and privacy implications when deploying LLM systems, potentially implementing privacy-preserving prompt templates and sensitivity detection.

Key Takeaway: Prompt limitations stem from model capabilities, context processing, and inherent AI constraints. Use external knowledge sources, multi-step approaches, and explicit constraints to mitigate these limitations, recognizing when to shift from prompt engineering to alternative approaches.

Industry Use Cases

Having examined prompt engineering principles and limitations, let's explore how these techniques are successfully applied in commercial applications across various industries.

Prompt engineering powers numerous commercial applications:

- **Customer Support Chatbots**: System prompts define persona, tone, and scope: "You are an AI assistant for ACME Corp. Your tone is friendly and professional. Answer product questions, but defer account issues to support." This ensures consistent brand voice and appropriate handling of different query types.

- **GitHub Copilot**: This AI pair-programmer prompts language models with file context (comments and surrounding code) to generate relevant completion suggestions. The prompt engineering involves formatting input to produce syntactically correct, contextually appropriate code.

- **Content Creation Tools**: Applications like Jasper use templates such as: "You are a marketing copywriter. Write a catchy, {tone} ad for {product}, highlighting {key points}, in under 50 words." These parameterized prompts enable consistent style while allowing customization through variable substitution.

- **Data Analysis Tools**: Enterprise analytics platforms convert natural language questions into augmented prompts: "Based on the following database results: {data}, provide a brief analysis comparing quarterly sales in Europe." The prompt combines user intent with retrieved data, demonstrating the integration of prompt engineering with data engineering.

- **Virtual Assistants**: Modern assistants maintain ongoing prompts containing conversation history and system instructions: "You are a helpful assistant... User: ...; Assistant: ...; User: ...". The prompt design determines how well the assistant maintains context and follows rules across multiple interactions.

- **Creative Generation**: For story or image generation, prompts define style and content parameters: "Write a short story in Edgar Allan Poe's style about a futuristic city" or "Create a surreal landscape with purple sky in Van Gogh's style." Users have learned that specific phrasing dramatically affects creative outputs.

These examples demonstrate prompt engineering's practical importance. Organizations often treat well-designed prompts as intellectual property or critical configurations, extensively testing outputs and iterating on designs.

Key Takeaway: In industry applications, effective prompt engineering creates business value by ensuring consistent outputs, maintaining brand voice, and enabling complex functionality through structured interactions. The most successful implementations combine domain knowledge with systematic prompt design.

Academic and Research Applications

While industry focuses on practical applications, academic research explores prompt engineering's theoretical foundations and pushes boundaries with novel techniques.

The academic community has embraced prompt engineering both as research subject and practical tool:

- **Zero-shot and Few-shot Learning**: Researchers discovered that large language models could perform tasks with just instructions (zero-shot) or a few examples (few-shot) [2]. This led to new benchmarks where models are evaluated on how well they follow prompt instructions rather than requiring fine-tuning. Prompts function as programs that temporarily transform general models into task-specific ones.

- **Chain-of-Thought Studies**: The "Let's think step by step" technique [6] opened new research into reasoning approaches like self-consistency (generating multiple reasoning paths then cross-checking results). Researchers document prompting techniques that unlock latent capabilities, such as having models debate themselves or adopt expert personas.

- **Prompt Tuning and Automation**: Research explores making prompt engineering more systematic through algorithmic optimization. Some approaches treat it as an optimization problem—using gradient descent to find prompts (sometimes as embedding sequences rather than natural language) that best guide models [1]. This "prompt tuning" represents the boundary between manual and automated prompt engineering.

- **Evaluation and Understanding**: Researchers use prompts to probe model knowledge, designing queries that isolate specific capabilities. For example, testing common sense with fill-in-the-blank prompts: "A dog is to bark as a cat is to ___." Carefully engineered prompts serve as diagnostic tools for model understanding.

- **Multimodal Prompting**: Academic projects extend prompting to multiple modalities, combining text with images or other inputs. Gu et al. [18] survey prompt engineering for vision-language models, while Chen et al. [19] discuss adapting techniques like chain-of-thought for multimodal contexts.

- **Social Prompt Engineering**: Wang et al.'s work on "Wordflow" [20] explores collaborative prompt development among non-experts, recognizing that effective prompt engineering benefits from diverse perspectives and iterative feedback.

- **Information Retrieval**: Jin et al. [21] demonstrate how optimized prompts enhance LLM reranking for search applications, improving models' ability to judge document relevance.

In academic settings, prompt engineering methods are carefully documented in research papers, with explicit templates and rationales shared through platforms like the Prompt Engineering Guide.

Key Takeaway: Academic research systematizes prompt engineering knowledge, establishing theoretical frameworks and discovering novel techniques that often transfer to practical applications. Research efforts continue to expand prompt engineering's capabilities while documenting its limitations.

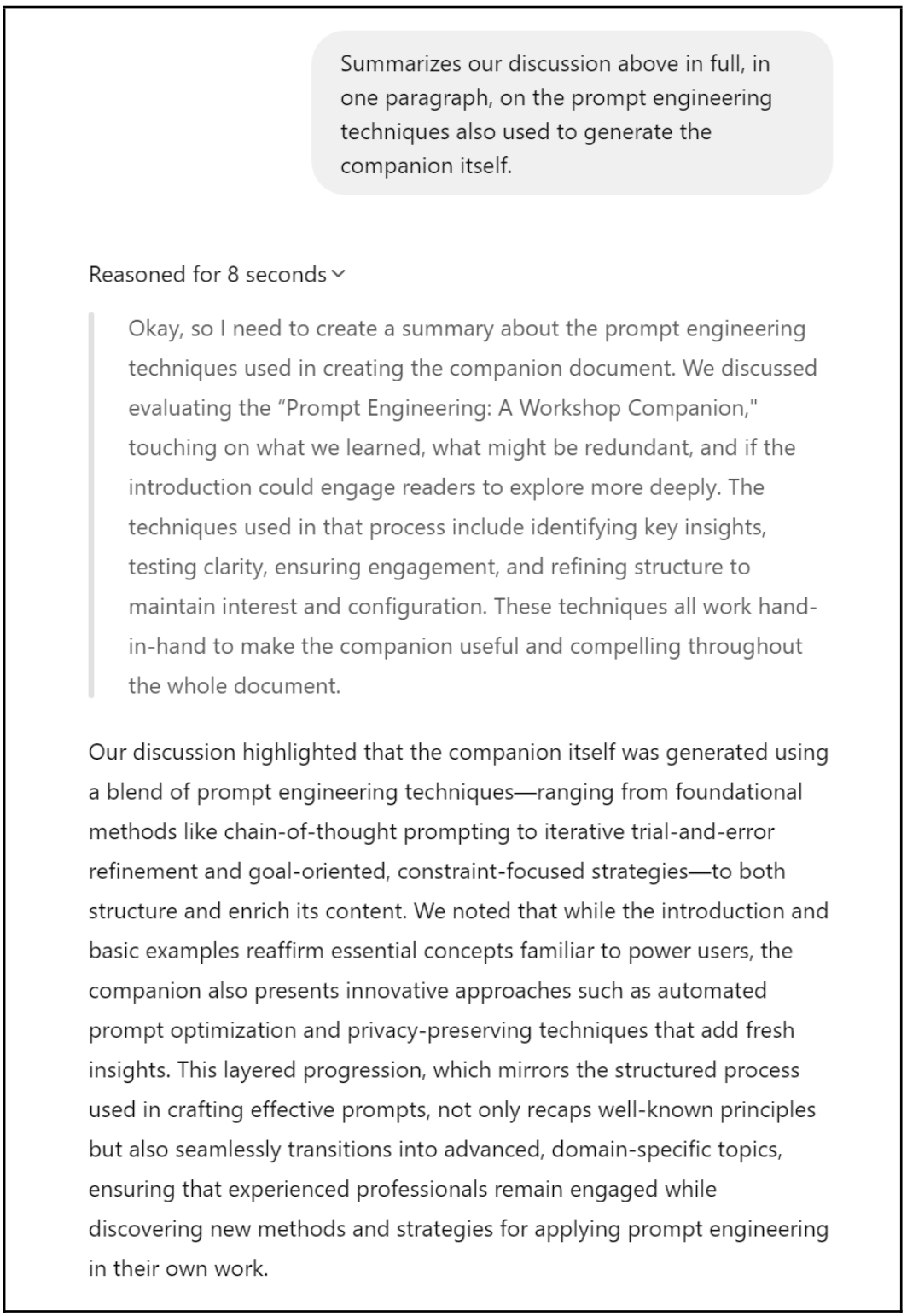

Figure 3: Closing conversation with o3-mini-high on the final review of this document.

Open-Source Tools and Repositories

The research and industry advancements in prompt engineering have led to the development of numerous tools and resources that help practitioners implement these techniques effectively.

The importance of prompt engineering has spawned numerous resources:

- **Frameworks and Libraries**:
  - **LangChain**: A Python framework for building LLM applications with utilities for prompt management, chaining, and integration with external data sources. It allows defining template prompts with placeholders filled at runtime and supports conversation memory [11].
  - **OpenPrompt**: A framework for prompt-learning research that enables testing different prompting methods with HuggingFace models [11].
  - Lighter utilities like **Promptify** and **Promptista** help manage and test prompt templates.

- **Curated Repositories**:
  - **Awesome Prompt Engineering**: A GitHub collection of resources, papers, and tools [11].
  - **Awesome ChatGPT Prompts**: User-contributed creative and useful prompt examples.
  - **Prompting Guide (promptingguide.ai)**: Tutorials and literature summaries for different techniques.

- **Prompt Datasets**:
  - **P3 (Public Pool of Prompts)**: Thousands of prompts for various tasks from researchers and crowdworkers [11].
  - **PromptSource**: An interactive repository of community-contributed prompts for popular datasets [11].

- **Evaluation Tools**:
  - **PromptInject**: Tests prompt robustness by generating adversarial variations to identify weaknesses [11].
  - **PromptSource's interface**: Enables testing prompts on actual models before deployment.

- **Communities and Courses**:
  - Forums, Discord servers, and dedicated courses compile best practices and provide live demonstrations.
  - University AI courses increasingly include prompt design sections when covering LLMs.

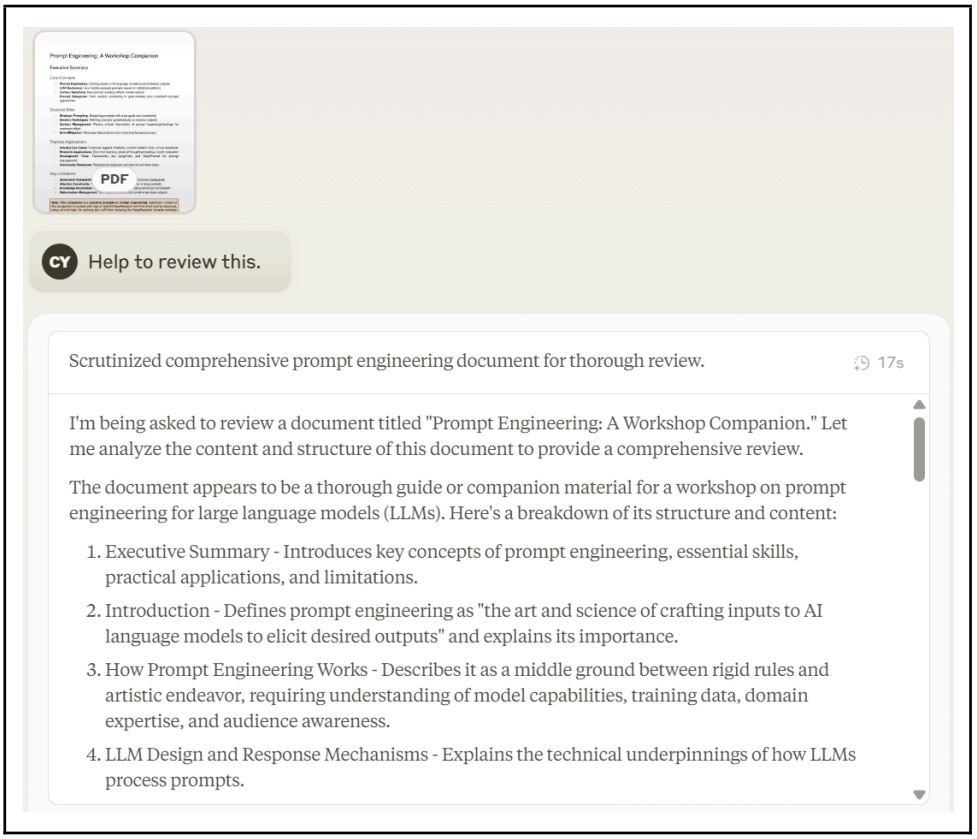

Figure 4: Open-ended review with Claude 3.7 Sonnet with thinking mode to review this document.

Key Takeaway: Open-source tools democratize prompt engineering by providing frameworks, repositories, and evaluation methods that codify best practices. These resources accelerate adoption and encourage standardization, making prompt engineering more accessible to non-experts.

Beyond Prompt Engineering: Advanced Topics

While prompt engineering provides an accessible entry point to working with LLMs, several advanced topics build upon this foundation:

Foundations of AI and LLMs

- Neural network architecture and attention mechanisms
- Embeddings and vector representations
- Tokenization strategies and their effects
- Context window utilization patterns

Key questions to explore: How exactly do transformers work, and why are they so effective at language tasks? What happens in the attention layers when a model processes a prompt? How do tokenization choices affect different languages and specialized domains?

Training and Fine-tuning

- Data curation principles for high-quality training
- Reinforcement learning from human feedback (RLHF)
- Adversarial testing methodologies
- Parameter-efficient fine-tuning techniques
- Comprehensive evaluation frameworks

Key questions to explore: What makes a high-quality dataset for LLM training? How does reinforcement learning from human feedback actually work, and why is it so powerful for aligning models with human preferences? When is full fine-tuning necessary versus parameter-efficient methods?

Retrieval-Augmented Generation (RAG)

- Knowledge retrieval architectures
- Vector database design principles
- Relevance optimization techniques
- Hybrid retrieval approaches
- Evaluation metrics for retrieval quality

Key questions to explore: How do we design systems that know when to retrieve information versus rely on parametric knowledge? What makes vector databases different from traditional databases? How can we improve the accuracy of retrieved information and its integration into generated text?

LLM System Engineering

- Orchestration of multiple models and tools
- Agent frameworks for autonomous systems
- Error handling and reliability strategies
- Cost and latency optimization
- Scalable deployment architectures

Key questions to explore: How can multiple models and tools be coordinated in complex workflows? What frameworks allow LLMs to effectively reason, plan, and interact with their environment? How do we design systems that gracefully manage LLM limitations?

AI and Software Development

- AI-assisted coding methodologies
- Cybersecurity considerations for LLM applications
- Ethical frameworks for AI development
- Natural language to code translation
- Multimodal development workflows

Key questions to explore: How are LLMs changing the software development lifecycle? What unique security vulnerabilities emerge in AI systems? How might natural language eventually replace traditional programming for certain applications?

Exploring these areas will help practitioners move beyond using LLMs effectively to designing sophisticated AI-enabled applications that push the boundaries of what's possible.

Conclusion

Prompt engineering sits at the intersection of human communication and AI behavior, requiring understanding of both linguistic nuance and model mechanics. It doesn't replace traditional engineering practices but shifts development to a new abstraction level. Core principles like modularity, testability, and robustness remain essential; prompt engineering simply provides a new interface for expressing intent.

This paradigm shift parallels earlier transitions from assembly to high-level languages, from procedural to object-oriented programming, and from local to cloud computing—each preserving fundamental principles while changing implementation approaches.

As LLM technology evolves, prompt engineering will evolve with it, but the ability to craft effective prompts remains critical for harnessing AI capabilities. Like query design for databases or UX design for applications, prompt engineering combines artistic creativity with systematic methodology to achieve optimal results.

Next steps for practitioners: Begin applying these techniques in your own work, starting with simple trial-and-error approaches and gradually incorporating goal-oriented and constraint-focused methods. Document successful prompts, analyze failures, and build your own prompt library for common tasks. Join communities where prompt engineers share insights and contribute your own discoveries to this rapidly evolving field.

References

[1] Sahoo, P. et al. 2024. A Systematic survey of prompt engineering in large language Models: Techniques and applications. *arXiv*. (Feb. 2024). DOI:https://doi.org/10.48550/arxiv.2402.07927.

[2] Brown, T.B. et al. 2020. Language Models are Few-Shot Learners. *arXiv*. (Jan. 2020). DOI:https://doi.org/10.48550/arxiv.2005.14165.

[3] Liu, P. et al. 2021. Pre-train, Prompt, and Predict: A systematic survey of prompting methods in natural language processing. *arXiv*. (Jan. 2021). DOI:https://doi.org/10.48550/arxiv.2107.13586.

[4] White, J. et al. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. *arXiv*. (Jan. 2023). DOI:https://doi.org/10.48550/arxiv.2302.11382.

[5] Liu, N.F. et al. 2023. Lost in the Middle: How language models use long contexts. *arXiv*. (Jan. 2023). DOI:https://doi.org/10.48550/arxiv.2307.03172.

[6] Wei, J. et al. 2022. Chain-of-Thought prompting elicits reasoning in large language models. *arXiv*. (Jan. 2022). DOI:https://doi.org/10.48550/arxiv.2201.11903.

[7] Li, H. et al. 2024. Towards goal-oriented large language model prompting: a survey. *arXiv*. (Jan. 2024). DOI:https://doi.org/10.48550/arxiv.2401.14043.
[8] Ayub, Md.A. and Majumdar, S. 2024. Embedding-based classifiers can detect prompt injection attacks. *arXiv*. (Oct. 2024). DOI:https://doi.org/10.48550/arxiv.2410.22284.
[9] Adversarial prompting in LLMs – Nextra: *https://www.promptingguide.ai/risks/adversarial*.
[10] Liu, N.F. et al. 2023. Lost in the Middle: How language models use long contexts. *arXiv*. (Jan. 2023). DOI:https://doi.org/10.48550/arxiv.2307.03172.
[11] GitHub - promptslab/Awesome-Prompt-Engineering: This repository contains a hand-curated resources for Prompt Engineering with a focus on Generative Pre-trained Transformer (GPT), ChatGPT, PaLM etc: *https://github.com/promptslab/Awesome-Prompt-Engineering*.
[12] Vatsal, S. and Dubey, H. 2024. A survey of prompt engineering methods in large language models for different NLP tasks. *arXiv*. (Jul. 2024). DOI:https://doi.org/10.48550/arxiv.2407.12994.
[13] Schulhoff, S. et al. 2024. The Prompt Report: A Systematic survey of prompting techniques. *arXiv*. (Jun. 2024). DOI:https://doi.org/10.48550/arxiv.2406.06608.
[14] Edemacu, K. and Wu, X. 2024. Privacy Preserving Prompt Engineering: A survey. *arXiv*. (Apr. 2024). DOI:https://doi.org/10.48550/arxiv.2404.06001.
[15] Li, W. et al. 2025. A Survey of Automatic Prompt Engineering: An Optimization Perspective. *arXiv*. (Feb. 2025). DOI:https://doi.org/10.48550/arxiv.2502.11560.
[16] Kepel, D. and Valogianni, K. 2024. Autonomous prompt engineering in large language models. *arXiv*. (Jun. 2024). DOI:https://doi.org/10.48550/arxiv.2407.11000.
[17] Wang, G. et al. 2024. Do advanced language models eliminate the need for prompt engineering in software engineering? *arXiv*. (Nov. 2024). DOI:https://doi.org/10.48550/arxiv.2411.02093.
[18] Gu, J. et al. 2023. A Systematic survey of prompt engineering on Vision-Language foundation models. *arXiv*. (Jan. 2023). DOI:https://doi.org/10.48550/arxiv.2307.12980.
[19] Chen, B. et al. 2023. Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review. *arXiv*. (Jan. 2023). DOI:https://doi.org/10.48550/arxiv.2310.14735.
[20] Wang, Z.J. et al. 2024. WordFlow: Social Prompt engineering for large language models. *arXiv*. (Jan. 2024). DOI:https://doi.org/10.48550/arxiv.2401.14447.
[21] Jin, C. et al. 2024. APEER: Automatic Prompt Engineering enhances large language model reranking. *arXiv*. (Jun. 2024). DOI:https://doi.org/10.48550/arxiv.2406.14449.
[22] Ye, Q. et al. 2023. Prompt engineering A prompt engineer. *arXiv*. (Jan. 2023). DOI:https://doi.org/10.48550/arxiv.2311.05661.