2025

Prompt Engineering for ChatGPT: A Practical Guide to Unlocking AI's Potential."

1/11/2025

1. Introduction to Prompt Engineering

Think of prompt engineering as the key to unlocking GPT's full potential. It's not just about "talking" to AI—it's about guiding it to deliver exactly what you need, whether that's crafting a flawless email or tackling a big project.

But here's the thing: like any skill, it takes a bit of practice. A vague or poorly structured prompt is like giving someone a map with half the roads missing. The good news? With a little effort, you can master the art of crafting prompts that work like a charm.

Who This Guide Is For

Are you here to make GPT your daily go-to tool, helping you write, brainstorm, or organize? Or are you a developer looking to integrate GPT into an app or workflow? Either way, you're in the right place. This guide is your starting point to turn "Hmm, let's see what happens" into "That's exactly what I needed."

Why Prompt Engineering Matters

Imagine asking GPT to solve a problem but not getting what you expected. Annoying, right? That's where prompt engineering comes in. Done well, it can:

- Save you time by avoiding unnecessary back-and-forths.
- Spark creative ideas you might not have thought of.
- Help you approach GPT not just as a tool, but as a collaborator.

It's not rocket science—just a few techniques that make a big difference.

What This Guide Will Teach You

This guide will take you from the basics to the good stuff:

- How to structure prompts for clarity and precision.
- How to fix common mistakes and get better results.
- Advanced techniques to use GPT for testing, brainstorming, and more.

And yes, there's plenty of room for experimentation. After all, the best way to learn is by doing.

How to Get Started

Before diving in, here are a few tools to play around with:

1. **OpenAI Playground**: Perfect for experimenting with settings and seeing how small tweaks change GPT's responses. https://platform.openai.com/playground
2. **ChatGPT (Free or Plus)**: Start with simple prompts and watch how slight adjustments can shift the tone or depth of the answers. https://chat.openai.com/

And here's a little secret: Prompt engineering isn't just a skill—it's an invitation to be curious. The more you explore, the more you'll uncover what GPT can truly do. So, grab a prompt and let's get started..

2: Key Principles of Prompt Engineering

You wouldn't build a house without a solid foundation, right? The same goes for prompts. A well-crafted prompt doesn't just get the job done—it sets you up for success. By following a few core principles, you can make sure GPT understands what you need and delivers results you can actually use. These principles are flexible and adaptable, allowing you to adjust your approach based on the complexity of the task. Whether you're crafting a quick query or solving a complex problem, a thoughtful prompt is your key to unlocking GPT's potential.

Key Principles

1. **Use the Role → Context → Task → Examples → Constraints Structure**
    - **Role**: Set the stage by defining who GPT should "be" in this interaction.
      Example: *"You are a teacher explaining geometry to middle school students."*
      Why this works: It gives GPT a perspective to shape its tone, depth, and focus.
    - **Context**: Anchor GPT by providing relevant background information about the task or audience.
      Example: *"The students have a basic understanding of shapes but haven't learned formulas for areas yet."*
      Why this works: Context narrows down GPT's focus and ensures responses are tailored to your needs.
    - **Task**: Be clear and direct about what you want GPT to do.
      Example: *"Explain how to calculate the area of a triangle using simple language."*
      Why this works: Specific tasks eliminate ambiguity and guide GPT toward actionable outputs.
    - **Examples**: Show GPT the kind of response you're looking for.
      Example: *"For example, you could say: 'The area of a triangle is found by multiplying the base by the height and dividing by two.'"*
      Why this works: Examples act as a model, helping GPT align its response with your expectations.
    - **Constraints**: Add boundaries to keep GPT focused and concise.
      Example: *"Limit your explanation to 50 words."*
      Why this works: Constraints prevent GPT from providing irrelevant details.
2. **Use Few-shot Prompting -** provide examples of the desired output to help GPT understand your expectations:
   Example: "Write an engaging blog post. Example 1: [Insert sample blog post]. Example 2: {[Insert another sample blog post] Now write one for the topic 'Benefits of Remote Work.'"
   Why this works: The examples serve as a benchmark, improving alignment with your desired style and tone.
3. **Break Tasks into Step-by-Step Instructions** - Complex prompts are easier to execute when broken into smaller, actionable parts.
   Example: Step 1: Identify three key benefits of cloud computing. Step 2: Provide a brief

explanation for each benefit. Step 3: Summarize why businesses should adopt cloud technology."

Why this works: Clear steps reduce ambiguity and guide GPT to deliver precise, organised responses.

4. **Be Specific, but Not Overwhelming -** Say what you need without going overboard.
    - Good: *"List five pros and cons of electric cars."*
    - Bad: *"Tell me everything about electric cars, including environmental impact, market trends, and personal opinions."*

    Why this works: GPT thrives on clarity, but too much detail can muddy the waters.
5. **Iterate and Refine -** Prompts aren't one-and-done. Start simple, then tweak:
    - First attempt: *"Explain quantum computing."*
    - Refined: *"Explain quantum computing to a high school student in 50 words."*

    Why this works: Each iteration gets you closer to the perfect result.
6. **Think Like a Collaborator** - Treat GPT as a teammate. Phrase your prompts as if you're giving instructions to a real person.
    - Example: *"Draft a summary of this article. Keep it professional but engaging."*

    Why this works: GPT is better at understanding conversational and human-like requests.
7. **Start Simple, Then Add Layers** - Begin with the essentials, then build up complexity.
    - Step 1: *"Summarize this article."*
    - Step 2: *"Summarize this article for a tech-savvy audience."*
    - Step 3: *"Summarize this article for a tech-savvy audience in 100 words."*

    Why this works: Breaking tasks into layers ensures GPT understands the basics before diving deeper.

Pro Tips for Effective Prompt Engineering:

1. **Start Small with Examples:** Begin with one or two concise examples for Few-shot Prompting. Test the output and adjust as needed to fine-tune GPT's responses for alignment with your expectations.
2. **Iterate Strategically:** Don't hesitate to test a simplified version of your prompt first. Use GPT's feedback to refine and add complexity as needed. Iteration is key to crafting a perfect prompt.
3. **Think Step-by-Step:** For complex tasks, provide a sequence of clear instructions. Breaking tasks into smaller parts helps GPT focus and ensures more accurate results.
4. **Experiment with Layers:** When tackling multi-layered tasks, start with the essentials and gradually add detail. For example, ask for a basic summary before requesting a detailed breakdown tailored to a specific audience.
5. **Use Constraints Wisely:** Specify length, tone, or style constraints to guide GPT's responses and keep outputs focused and relevant. For instance, "Explain this concept in 50 words using simple language."

Key Takeaways

1. A clear structure helps GPT focus.
2. Specificity is your friend—but keep it manageable.
3. Avoid conflicting terms.
4. Use punctuation to clarify complex prompts.
5. Pose open-ended questions or requests
6. Set output length goals or limits
7. Do not expect GPT to get it right from the first attempt, though it might.
8. Iterate. A good prompt takes time to refine.
9. Treat GPT as a collaborator, not just a tool.
10. Simplicity first, then add complexity.
11. Provide clear examples, step-by-step instructions, or delimiters for complex tasks to enhance GPT's understanding and focus. (See Section 3.1 for more on using delimiters.).

3: Deep-Dive Topics

The Deep-Dive Topics section explores essential principles and advanced strategies to elevate your prompt engineering skills. Each topic is designed to deepen your understanding of GPT's capabilities while providing actionable techniques to create more effective prompts. With a structured format—**WHY**, **WHAT**, **HOW**, **Examples**, and **Key Takeaways**—you'll find clear explanations, practical advice, and illustrative scenarios to bridge theory with real-world application.

3.1. Crafting Effective Prompts: Building Blocks, Order, and Best Practices

Set the stage by explaining why building blocks and order are crucial for effective prompts. Reference the **Key Principles of Prompt Engineering** (Section 2) to establish continuity and highlight that this section expands on those foundations.

Key Takeaways (TL;DR):

- The **Role → Context → Task → Examples → Constraints** structure ensures clarity, precision, and focus for complex tasks.
- Randomized or poorly structured prompts create ambiguity, reducing the effectiveness of GPT's responses.
- A logical order acts as a "mental roadmap," helping GPT interpret your intent and deliver actionable outputs.
- Simpler tasks often benefit from a streamlined approach, skipping unnecessary components for greater efficiency.
- Flexibility is key: adapt the structure based on the complexity of the task and the desired outcome.
- Breaking tasks into smaller parts improves focus and interpretability, especially for multi-step operations.
- Summarizing previous dialogues and documents helps maintain context and clarity in extended interactions.

Under the Hood

Building Blocks Refresher

1. **Role**: Define the "who" in the interaction to set the tone and depth.
2. **Context**: Provide background to anchor GPT's response.
3. **Task**: Specify what you want GPT to do.
4. **Examples**: Include models or references for clarity.
   "Example: 'Sales grew by 10%, driven by new product launches.' Write similarly concise summaries.
5. "**Constraints**: Add boundaries (length, tone, format) to guide the response.

Why Order Matters

1. **Sequential Understanding**: GPT processes prompts linearly; order ensures clarity.
2. **Contextual Anchoring**: Establishing Role and Context first improves relevance.
3. **Task Focus**: Clearly stating the task early avoids ambiguity.
4. **Refinement with Examples and Constraints**: Examples and constraints fine-tune GPT's output, avoiding vagueness or overgeneralisation.
5. **Clear "mental roadmap" for GPT:** Combining these building blocks in the right order improves response quality.

Best Practices for Effective Prompts

Crafting effective prompts isn't just about following a structure—it's also about knowing when to adapt, simplify, or refine. Here are some best practices to keep in mind:

1. **Start with Clarity -** Define your goal clearly. If the output feels off, check if your task was ambiguous or overly broad.
   Tip: Ask yourself, "What do I want GPT to do, and how should it do it?"
2. **Use Simple Language -** Avoid jargon unless absolutely necessary. The clearer your instructions, the better GPT will understand.
   Example: Instead of "Deconstruct the ontological implications," say, "Explain the meaning and implications of this concept in simple terms."
3. **Test and Refine -** Prompts rarely work perfectly on the first try. Experiment, adjust, and rephrase until the output aligns with your needs.
   Example: If "Summarize this article" gives too much detail, refine to "Summarize this article in 100 words, focusing on key events."
4. **Use Feedback Loops -** Encourage GPT to critique or improve its own responses. This builds a layer of refinement directly into your prompts.
   Example: "Summarize this report, then evaluate the summary for clarity and completeness."
5. **Leverage Context -** Whenever possible, provide background information. A clear context ensures GPT tailors its response effectively.
   Example: "You are an HR professional drafting a job description for a remote software engineer role."
6. **Adapt to Task Complexity -** For simple tasks, keep the prompt short and direct. For complex tasks, use a structured approach like Role → Context → Task → Examples → Constraints.
   Example: Use "List three benefits of remote work" for simple requests, but the full structure for tailored or nuanced outputs.
7. **Iterate for Continuous Improvement -** If GPT's output doesn't meet expectations, break tasks into smaller parts or add specific examples to guide the response.
   Pro Tip: Each iteration is a step toward mastering your prompting style.

8. **Be Explicit with Constraints -** Specify length, tone, or format to avoid ambiguity.
   Example: "Explain blockchain technology in 100 words, avoiding technical jargon."
9. **Dig Deeper into Advanced Techniques -** For more nuanced scenarios, consider experimenting with placeholders, breaking tasks into subtasks, and using parameters like temperature and max tokens.
   Pro Tip: Refer to **Section 6: Advanced Prompting Techniques and Refinement** for detailed guidance on techniques like debugging prompts, handling ambiguity, and iterative improvement.

When to Use the Full Structure vs. Simplify

1. **When to Use the Full Structure**
   - For complex or nuanced tasks, such as compliance reports or crafting detailed technical explanations.
   - When aligning to a specific style, tone, or audience focus.
   - In iterative prompts when refinement and feedback are needed to perfect the response
2. **When to Simplify**
   - For quick, straightforward tasks like generating lists or brief summaries.
   - To save time when detailed precision isn't critical.

Why Different Prompt Styles Appear in the Guide

1. **Adaptability**: Demonstrates how prompts can be tailored for various levels of complexity.
2. **User-Centric Approach**: Reflects real-world needs where users may not always require detailed structures.
3. **Showcasing Best Practices**: Highlights when and how to adjust prompt styles for optimal results.

Pro Tips

- Start with the full structure for complex tasks, then simplify as you gain confidence. Experimenting with variations helps you discover what works best for your needs.
- Combine specific examples and explicit constraints to produce consistent, high-quality results.
- For complex inputs, clearly separate instructions from data.
- Divide multi-part tasks into actionable steps for better results.

Effective Examples

1. **Role → Context → Task → Examples → Constraints**
   - **Prompt**:
     Role: "You are a technical writer."
     Context: "You are drafting documentation for developers learning React.js."
     Task: "Explain the concept of hooks in React, focusing on useState and useEffect."

Examples: "For example, explain useState like this: 'useState lets you add state to functional components. It returns an array with two elements: the current state value and a function to update it.'"
Constraints: "Keep it under 150 words using beginner-friendly language."

Why It Works: The structure ensures GPT tailors its response to a specific audience and style while staying concise and focused.

2. **Using Delimiters for Clarity**
    - **Prompt**: "Summarise the following text: ` The JavaScript `reduce` function applies a reducer function to each element in an array, resulting in a single output value. It can accumulate sums, build objects, or perform other aggregations. ` Then explain how reduce differs from map."

      Why It Works: Delimiters clearly separate instructions from input, reducing the chance of GPT misinterpreting the task.
3. **Step-by-Step Instructions**
    - **Prompt**:
      "Step 1: List three benefits of hybrid work models.
      Step 2: Provide a brief explanation of each benefit.
      Step 3: Suggest strategies to overcome challenges in hybrid work models."

      Why It Works: Breaking down the task into smaller parts ensures GPT delivers a logical and organised response.
4. **Few-shot Prompting**
    - **Prompt**: "Write a product description for a smartwatch. Example 1: 'This sleek smartwatch tracks your fitness and keeps you connected with smart notifications.' Example 2: 'Stay on top of your health and style with this versatile smartwatch, featuring heart rate monitoring and customisable bands.' Now write one for a fitness tracker targeting runners."

      Why It Works: Few-shot examples clarify tone, style, and structure, guiding GPT to align its output with expectations.
5. **Explicit Constraints**
    - **Prompt**: "Explain the basics of cloud computing in under 100 words, avoiding technical jargon."

      Why It Works: Clear constraints help GPT stay concise and avoid unnecessary complexity.

<u>Ineffective Examples</u>

1. **Vague Role**
    - **Prompt**: "You are an expert. Explain quantum computing."

      Why It Fails: The role is too generic, leading GPT to provide a broad or unfocused response.

2. **Confusing Order**
    - **Prompt**:
      Task: "Write an introduction to AI ethics."
      Examples: "For instance, you could discuss transparency and accountability."
      Role: "You are a philosopher."
      Context: "This is for a general audience."
      Constraints: "Keep it under 150 words."

      Why It Fails: Task appears before Role and Context, which can confuse GPT about tone and audience alignment.
3. **No Delimiters**
    - **Prompt**: "Summarise the following text: The JavaScript reduce function applies a reducer function to each element in an array, resulting in a single output value. It can accumulate sums, build objects, or perform other aggregations. Then explain how reduce differs from map."

      Why It Fails: Lack of delimiters blurs the distinction between the instructions and input text.
4. **Overloaded Task**
    - **Prompt**: "Explain machine learning to a 10-year-old, discuss its societal impact, and outline ethical concerns, all in one paragraph."

      Why It Fails: Combines unrelated tasks into a single request, making the response unfocused or incomplete.
5. **No Constraints**
    - **Prompt**: "Describe blockchain technology."

      Why It Fails: Without constraints, GPT may generate a response that's too lengthy, overly technical, or not aligned with the user's needs.

Closing Thoughts

Crafting effective prompts is about striking a balance between clarity and adaptability. The Role → Context → Task → Examples → Constraints framework is invaluable for complex prompts, while simpler tasks can often benefit from streamlined structures. By understanding the building blocks and learning to adapt them to your needs, you unlock GPT's full potential. Effective prompting isn't just a skill—it's an art you can refine with practice.

3.2. Effective Role Assignment

When interacting with GPT, the role you assign shapes how it "thinks" and responds. By clearly defining the role, you help GPT adapt its tone, depth, and style to fit the task. A poorly defined role can lead to vague or irrelevant answers, while a well-defined role brings focus and precision.

Key Takeaways (TL;DR)

- Clearly defined roles focus GPT's responses.
- Match the role to the expertise level and tone required.
- Avoid vague or overloaded roles to keep outputs relevant and coherent.

Under the Hood

Key Considerations for Assigning Roles

1. **Be Explicit**: Clearly state the role GPT should take.
   Example: *"You are a project manager creating a detailed task breakdown."*
2. **Adapt to the Task**: The role should match the expertise required.
   Example: *"You are a financial analyst preparing a report for non-experts."*
3. **Use Specific Jargon or Context**: Give GPT the tools it needs to "act" the role effectively.
   Example: *"You are an educator teaching middle school students about ecosystems. Use age-appropriate language."*

Taking Role Assignment Further with Personas

For tasks requiring nuanced or tailored responses, you can enhance role assignments by introducing personas. A persona gives GPT a specific identity, making its responses more relatable and human-like.

1. **Why Use Personas?**
   - **Enhanced Clarity:** Personas provide relatable details that guide GPT's tone and depth.
   - **Improved Context Understanding:** By simulating expertise through a persona, GPT can better align its responses with the task.
   - **Engagement:** Personas make interactions more dynamic and relatable.
2. **How to Create Personas:**
   - Add relevant background details to the role.
   - Keep the description concise and task-focussed.
3. **Examples of Personas in Prompts:**
   - *"You are Emily, a senior financial analyst with 10 years of experience in market trends, creating a report for non-experts."*
   - *"You are Dr. Patel, a historian specializing in 18th-century European art, explaining the significance of the Baroque movement to high school students."*
   - *"You are Alex, a senior SE mentoring a team on writing clean code practices."*

Effective Role Assignment Examples

- **Example 1**: *"You are a marketing consultant advising a startup on their branding strategy."*
  Why it works: The role sets GPT's focus on providing actionable, strategic insights rather than generic branding advice.
- **Example 2**: *You are a software architect reviewing a proposed microservices design for scalability."*
  Why it works: The role steers GPT toward technical and analytical feedback.
- **Example 3 (With Persona):** *"You are Sarah, a UX researcher with a passion for inclusive design, reviewing a mobile app's accessibility features for users with visual impairments."*
  Why It Works: The persona helps GPT simulate expertise and adopt a user-focused tone.

Common Pitfalls and How to Avoid Them

  - **Vague Roles**: Example: *"You are an expert."*
  - Why it fails: Lacks specificity, leading to generic or unfocused responses.
  - **Overloading the Role**: Example: *"You are an educator, developer, and project manager creating a marketing plan."*
  - Why it fails: Confuses GPT, making it harder to generate coherent outputs.

Pro Tips:

1. Keep roles clear and focused. If multiple perspectives are needed, split them into separate prompts.
2. Add context to roles when necessary. The more precise the role, the better the results.
3. Experiment with tone. Assign roles that naturally align with the level of formality or creativity you need.
4. Use personas selectively for nuanced or human-like interactions where tone and depth are crucial.
5. Avoid overly detailed personas to keep prompts focused.

Closing Thoughts

A clear role guides GPT's focus and tone, making responses more relevant and actionable. Define it thoughtfully—it's a small step that makes a big difference!

3.3. Minimizing Hallucinations

GPT can sometimes "hallucinate," producing outputs that sound plausible but are inaccurate or completely fabricated. Minimizing hallucinations ensures reliability and accuracy, which is critical for maintaining trust in the generated content.

Key Takeaways (TL;DR)

- Hallucinations occur when GPT fills in gaps with fabricated details.
- Set clear instructions for handling uncertainty.
- Narrow the scope to reduce speculative or irrelevant outputs.
- Use structured prompts with explicit roles, tasks, and constraints to anchor GPT.

Under the Hood

Key Strategies to Minimize Hallucinations

1. **Modifications to Prompt Blocks**:

- **Role Block**: Explicitly define the model's expertise and limitations.
  Example: *"You are a knowledgeable assistant trained to provide accurate and verified information. If uncertain, seek clarification."*
  Why this works: It ensures GPT understands its boundaries and avoids overstepping into speculation.
- **Task Block**: Include a directive to base answers only on verified data and avoid speculation.
  Example: *"Provide an answer based strictly on factual information. If the information is unavailable, state that clearly."*
  Why this works: It emphasizes factual accuracy, reducing the chance of fabricated responses.
- **Constraints Block**: Add instructions to avoid generating information beyond the scope of known data.
  Example: *"Do not fabricate information. If necessary data is unavailable, respond with 'I do not have enough information to answer accurately.'"*
  Why this works: It prevents GPT from filling gaps with fabricated details.

2. **Requesting Additional Information**:

   Add keywords or phrases in the **Constraints** or **Examples Block** to prompt GPT to seek clarification when needed:
   Example: *"When the prompt lacks clarity or sufficient data, respond by requesting additional input from the user."*
   Why this works: It ensures GPT proactively seeks more details instead of making assumptions.

3. **Directing GPT to State 'I Don't Know'**:

   Use these keywords or phrases in the **Constraints** or **Task Block**:
   Example: *"If the provided input is ambiguous or beyond your training, respond with 'I don't know' or 'I need more information.'"*
   Why this works: It encourages transparency when the model is uncertain.

Step-by-Step Guide to Minimize Hallucinations

1. **Explicit Role Assignment**:
   - Define the model's boundaries to focus on verified information.
     Example: *"Act as a factual assistant, trained to provide verified information only."*
2. **Provide Context**:
   - Ensure the context block gives GPT sufficient background to reduce misinterpretation.
3. **Structured Tasks**:
   - Break tasks into specific instructions, emphasizing reliance on known data.
     Example: *"Summarize this article based on the content provided without introducing external or speculative information."*
4. **Use Constraints for Accuracy**:
   - Explicitly forbid the generation of unverified details.
     Example: *"Avoid providing details that are not explicitly stated in the source material."*
5. **Reinforce with Examples**:
   - Provide clear examples demonstrating expected behavior when data is incomplete.
     Example:
     - *User Input*: "Tell me about an unknown event."
     - *Model Response*: "I don't have information about that. Can you provide more context?"
6. **Enable Error Handling**:
   - Add fallback directives to guide the model when data is unclear.
     Example: *"If unsure, ask clarifying questions instead of guessing."*

Effective Prompt Examples

- **Minimizing Hallucinations with Clarity**:
  - Role: *"You are an expert assistant trained to give accurate information based only on provided data."*
  - Task: *"Summarize the provided text in simple terms without adding external data."*

- Examples:
  - *Input*: "Explain quantum mechanics in detail."
  - *Model Response*: "I can explain general principles, but more specific information requires additional context."
- Constraints: *"Do not guess or fabricate missing information. If data is insufficient, say 'I need more details.'"*

Common Pitfalls and How to Avoid Them

1. **Vague Instructions**:
   - Example: *"Describe AI."*
   - Why it fails: Lacks direction, leading to generalized or inaccurate content.
2. **Assuming GPT Knows Everything**:
   - Example: *"Explain quantum entanglement with real-world examples from recent papers."*
   - Why it fails: GPT doesn't have access to real-time or unpublished data.

Closing Thoughts

By implementing explicit roles, clear tasks, and strong constraints, you can greatly reduce hallucinations. Reinforcing expectations with examples and asking for clarification when information is incomplete further ensures accurate and reliable outputs.

3.4. Retaining Context Across Multiple Prompts

When working on complex topics, retaining context across multiple prompts ensures continuity and builds upon earlier discussions effectively. This approach saves time, improves clarity, and avoids redundancy, making each prompt more meaningful.

Key Takeaways (TL;DR):

- Establish solid foundation for all prompts in the session in the beginning
- Restate critical context periodically to refresh GPT's memory.
- Use tags, summaries, or structured files to track ongoing discussions.
- Modularize prompts into logical segments for clarity.
- Use a summary file for sessions requiring sustained focus over multiple prompts.
- Consider tools like Canvas for managing larger, long-term projects effectively.

Under the hood

Why Retaining Context Matters

1. **GPT's Context Window:** GPT remembers only a limited number of tokens within a session. Information outside this window may be forgotten.
2. **Ambiguity from Missing Context:** If critical details are not reiterated, GPT may misinterpret or generalize responses.
3. **Session Dependency:** Between sessions, GPT resets context entirely, requiring users to re-establish continuity manually.

Strategies for Retaining Context

1. **Create and Use a Summary File:**
    - Create a summary file at the end of each session that captures:
        - Assigned roles.
        - Agreed structure or tasks.
        - Key points or decisions from the session.
        - Examples or references critical for continuity.

Example Summary File Entry:

```
Role: Expert in ML, AI, and prompt engineering.
Key Context: Creating a user guide for effective prompt engineering, following the Pyramid Principle.
Examples: Refer to Section 3.1 on prompt building blocks and why order matters.
Tasks: Complete Section 3.4 focusing on multi-day session strategies.
```

        - Use this summary file to refresh GPT context every 5–10 prompts.
        - Copy only the relevant parts into prompts as needed.
        - Avoid pasting the entire session; focus on the critical snippets.

Note: You can use ChatGPT to create and refine this file

2. **Plan Refeeding Frequency:**
    - Refeed context approximately every **5–10 prompts** within the same session.
    - For multi-day sessions, **reintroduce key information at the start of each session** or whenever transitioning to a new task.
    - Avoid overloading prompts by focusing on the **most relevant details** for the current task.
3. **Use Persistent Tags or Labels:**
    - Assign tags to different parts of the conversation for easy reference.
      Example: *"[Topic: Machine Learning Basics]"* or *"[Feedback: Examples of retaining context]"*.
4. **Leverage Modular Prompts:**
    - Divide tasks into smaller, logically connected subtasks to maintain focus.
    - Example:
        - Prompt 1: *"Draft the introduction for 'Retaining Context Across Multiple Prompts.' "*
        - Prompt 2: *"Expand on strategies for modularizing prompts."*
5. **Incorporate References to Previous Outputs:**
    - Explicitly refer to prior work to ensure continuity.
    - Example: *"Building on the summary provided in Prompt X, refine the section by adding examples of modularizing tasks."*
6. **File Management for Refeeding:**
    - Maintain a central document or Canvas containing:
        - Summary files for each session.
        - Examples of writing style and refined sections.
        - References to earlier discussions or decisions.
7. **Copy Only Relevant Parts for Refeeding:**
    - Instead of overloading GPT with entire conversations, extract and refeed only the essential snippets.
    - Example: *"Use the following context to refine the guide: [Insert refined section snippet here.]"*
8. **Integrate Tools like Canvas for Larger Projects:**
    - For projects involving multiple iterations or extensive content, consider transitioning to Canvas, a workspace designed to streamline long-term and complex projects by maintaining continuity and enhancing collaboration. It allows

you to organise, refine, and track your work with tools like inline feedback, targeted editing, and version history.

- Canvas session is a session within your current ChatGPT interaction when the document or code you are working on is open in the Canvas interface. It acts as a persistent session, keeping context for this particular document across multiple iterations. Once you close the Canvas, interactions revert to a regular session, where context is limited to a single interaction.

9. **Switch between Canvas and Regular Sessions:**
    - Switching between Canvas and regular sessions strategically can boost efficiency. Use Canvas for refining complex projects with persistent context, and regular sessions for quick brainstorming or tackling smaller tasks.
    - Pro Tip: when switching between Canvas and regular sessions clearly specify:
        - switching to a regular session mention that you are not working with the Canvas
        - switching back to the Canvas document or between multiple Canvas documents, specify what exact Canvas document you want to work on
        - manage creating a new Canvas document, going back to the regular session or working on a particular Canvas document with tags (e.g., <new canvas>. <reg. session>, <current canvas>).
        - DO NOT FORGET to add these tags to the Constraints!

Tagging Examples for Iterative Work

1. **Tagging Feedback:**
    - When providing feedback on iterations, use consistent tags to highlight areas for improvement.

      Example: *"[Feedback: Examples Block] The examples you provided in the last response were helpful but lacked clarity for beginners. Could you refine them?"*
2. **Tagging Questions:**
    - Use tags to specify the focus of each question during iterative discussions.

      Example: *"[Question: Constraints Block] Can you clarify how to phrase constraints for tasks involving technical writing?"*
3. **Combining Tags for Efficiency:**
    - Combine tags to streamline iterations and connect related feedback and questions.

      Example: *"[Feedback: Role Definition] The role description is clear, but it might be too formal for a casual audience. [Question: Tone Adjustments] How can I simplify the tone without losing credibility?"*

Effective Practices for Multi-Day Sessions

Example 1: Starting a New Session
Prompt: *"Continuing from yesterday's discussion, let's refine Section 3.4 on managing multi-day sessions. Yesterday's decisions included: [recap points]. Let's proceed with [specific task]."*

Why It Works:

- Provides a clear link to prior discussions.
- Establishes alignment for the task at hand.

Example 2: Using Summary Files
Prompt: *"Here's the context from our previous sessions: [insert summary snippet]. Use this to guide the next steps in refining Section 3.4."*

Why It Works:1

- Refreshes critical context efficiently without overloading the prompt.

Common Pitfalls and How to Avoid Them

1. **Overloading Prompts with Context:** Avoid pasting the entire session or excessive details. Instead, focus on key points and decisions.
2. **Skipping Summaries Between Days:** Without reintroducing critical information, responses may lack continuity or relevance.
3. **Inconsistent Refeeding:** Failing to reintroduce context after every 5–10 prompts or between sessions increases the chance of misalignment.
4. **Lack of Modularization:** Tackling large tasks without breaking them into smaller, trackable steps can lead to fragmented outcomes.

Closing Thoughts

Retaining context across prompts is both an art and a science. By using tags, modular prompts, summary files, and refeeding periodically, you can make your collaboration with GPT more cohesive and productive even if your work includes many prompts and spans over several days. For projects requiring extended collaboration or more complex workflows, transitioning to Canvas simplifies context management and collaboration between the user and ChatGPT.

3.5. Creating Prompts with Built-In Feedback

Adding feedback tasks to your prompts not only improves the quality of GPT's responses but also helps refine your prompt engineering skills. By embedding evaluation and iterative improvement into your prompts, you enable GPT to act as both a problem-solver and a collaborator. Giving GPT "time to think"—by instructing it to review, reason, or self-correct—enhances its ability to generate thoughtful and accurate outputs, making it an even more effective partner in your tasks.

Key Takeaways (TL;DR):

- Include feedback tasks alongside the main question to encourage self-assessment.
- Ask GPT to reason through steps or explain its thought process for better accuracy.
- Use modular prompts with clear sections for solutions, analysis, and self-reflection.
- Provide explicit constraints and encourage iterative refinement to guide feedback generation.
- Prompt GPT to self-check and validate its own outputs to improve quality and reliability.

Under the Hood

Why Create Prompts with Built-In Feedback?

1. **Iterative Learning:** Feedback loops transform interactions into an iterative learning process.
2. **Enhanced Prompt Clarity:** GPT evaluates and improves prompts, leading to more effective instructions.
3. **Continuous Refinement:** Built-in feedback fosters an evolving approach to prompt engineering, adapting to various tasks.
4. **Clarity in Communication:** Prompts with feedback help highlight misunderstandings and adjust outputs accordingly.
5. **Encouraging Thoughtful Responses:** Giving GPT tasks like reviewing its output or identifying overlooked details fosters deeper reasoning. *Example:* "Here's your response. What key points might I have missed, and how can they be added?"
6. **Simulating Problem-Solving:** Asking GPT to simulate a "thought process" helps clarify steps in solving complex tasks. *Example:* "Generate a solution, then explain the reasoning behind each step as if teaching a beginner."

Strategies for Building Feedback into Prompts

1. **Encourage a Step-by-Step Approach:** Break tasks into smaller steps and instruct GPT to summarise progress after each.
   Example: "Plan a two-day conference itinerary. Start with the first day, and once completed, summarise before moving to the next."

2. **Include Evaluation Tasks:** Add a specific task for GPT to assess its own response.
   Example: "Explain how transformers process sequential data. Then evaluate this response for clarity and accuracy, suggesting any improvements."
3. **Critique and Improve Prompts:** Request feedback on the prompt's structure, clarity, and effectiveness.
   Example: "Answer the question, and evaluate this prompt by identifying strengths, weaknesses, and suggestions for improvement."
4. **Incorporate Improvement Suggestions:** Direct GPT to suggest alternative phrasing or approaches.
   Example: "Rewrite this paragraph for a beginner audience. Then suggest two ways to make it clearer."
5. **Leverage Modular Components:** Structure prompts with labelled sections for solutions and feedback.
   Example: Task Solution: [Answer to the main task] Prompt Analysis: [Strengths, weaknesses, and improvements]
6. **Ask GPT to Review Its Logic:** Direct GPT to identify gaps or inconsistencies in its reasoning.
   Example: "Draft a marketing plan. Then review your own proposal for logical consistency and potential gaps."
7. **Define Success Criteria:** Specify what GPT should focus on during feedback (e.g., clarity, tone, or structure).
   Example: "Evaluate whether this prompt uses concise language and clear structure."
8. **Prompt for Error Detection:** Include directives for GPT to find and correct mistakes in its own output.
   Example: "Generate a list of pros and cons for remote work. Then identify and correct any logical inconsistencies in your response."
9. **Recurring Feedback Tasks:** Use consistent feedback guidelines across multiple prompts to maintain focus and avoid redundancy.
   Example: "Analyse the response for logical consistency and revise the prompt as per previous feedback criteria."

Effective Prompt Examples

Example 1:

*"Explain how self-attention in transformers works. Then evaluate this prompt:

- Identify its strengths and weaknesses.
- Suggest specific improvements.
- Provide a revised version with explanations for the changes."*

Why It Works:

- Combines problem-solving with prompt refinement.
- Encourages GPT to think critically about its own inputs and outputs.

Example 2:
"Summarise the main idea of this article. Then evaluate your summary: Did you include all key points? If not, revise it to include any missing details."
Why It Works:

- Encourages GPT to self-assess and refine its response.
- Creates a feedback loop for continuous improvement.

Example 3:
"Generate a beginner-friendly explanation of neural networks. Afterward, critique this response for tone and accessibility, and suggest ways to make it more engaging."
Why It Works: Links the main task with specific feedback, improving the outcome in real time.

Example 4:
"Design a user-friendly login page. After drafting, explain the design choices and suggest two improvements for accessibility."
Why It Works: Encourages GPT to reflect on its output, promoting thoughtful refinement.

Example 5:
"Summarise this article in 200 words. Then evaluate whether your summary includes all the critical points and adjust if necessary."
Why It Works: Creates a feedback loop, improving the accuracy of the summary.

Closing Thoughts

Prompts with built-in feedback elevate your interactions with GPT, transforming it into both a collaborator and a tutor. Encouraging GPT to reflect on its responses and reasoning strengthens its ability to provide thoughtful, reliable, and actionable outputs. By giving GPT "time to think," you're not just generating answers—you're fostering a process of iterative learning and refinement. Start small, experiment with feedback loops, and see how this approach enhances your results while making GPT an even more effective partner.

3.6: Understanding and Mitigating GPT Shortcomings

No tool is perfect, and GPT is no exception. Understanding its limitations empowers you to use it effectively, mitigate potential pitfalls, and get the most out of its capabilities. By addressing these shortcomings thoughtfully, you can adapt your prompts and workflows to overcome them.

Key Takeaways (TL;DR):

- GPT has limitations, and knowing what they are is crucial for creating effective prompts
- GPT cannot generate accurate content without sufficient input.
- It may hallucinate, overgeneralize, or misunderstand ambiguous instructions.
- Adapt your approach to mitigate shortcomings like token limits, filtered topics, and lack of real-world reasoning.
- Validate outputs and provide additional context to improve quality

Under the Hood

Common GPT Shortcomings: Impacts and Solutions

1. **Hallucinations:** GPT may provide outputs unsupported by facts (invent them).

- **Impact:** Misleading or incorrect outputs can reduce reliability.
- **Solution:**
    - Validate outputs with external sources or domain experts.
    - Use constraints like: "If uncertain, state 'I don't know' or ask for clarification."
    - See Section 3.3. Minimizing Hallucinations

2. **Ambiguity and Overgeneralization**
   Responses may lack actionable insights or fail to meet expectations.

- **Impact:** Vague or irrelevant answers that do not address user needs.
- **Solution:** Provide clear, specific prompts with explicit tasks.
  Example: Instead of "Explain AI," ask, "Explain how neural networks are used in image recognition for a beginner audience."

3. **Context Window Limitations:** Critical context may be lost in lengthy conversations.

- **Impact:** Information from earlier prompts may be omitted or forgotten.
- **Solution:**
    - Reintroduce key context periodically.
    - Summarize prior discussions to refresh GPT's understanding.
    - Limit conversations to around 10–15 key prompts or approximately 3,000–4,000 tokens to reduce the risk of losing critical information.
    - **Start new sessions for extended tasks:** Use a structured summary file to transfer critical context and continue seamlessly in a new session

4. **Dependence on Training Data:** Responses might be outdated or lack domain-specific expertise. GPT's knowledge is limited to its training data

- **Impact:** Information may not reflect the latest developments or nuanced knowledge.
- **Solution:**
    - Ask GPT to clarify the source or reliability of its knowledge.
    - Specify the knowledge cutoff date in your prompts.
    - Cross-check critical information with up-to-date, external sources.

5. **Bias in Outputs:** Results might perpetuate stereotypes or exclude diverse perspectives as GPT may reflect biases present in its training data.

- **Impact:** Imbalanced or non-inclusive outputs.
- **Solution:**
    - Frame prompts to encourage balanced perspectives
    - Example: "Explain both benefits and potential drawbacks of automation in manufacturing."

6. **Restricted or Filtered Topics:** GPT avoids certain topics, entities, or names due to built-in moderation.

- **Impact:** Responses may exclude critical details or become vague when addressing sensitive subjects.
- **Solution:**
    - Reframe the query to focus on permissible aspects of the topic.
      Example: "Provide a neutral overview of controversies surrounding AI ethics without sensationalism."
    - Seek external sources for highly sensitive or restricted information.

7. **Data Cutoff and Inability to Access the Internet:** GPT-3.5 is limited to data up to September 2021, while GPT-4's cutoff is April 2024.

- **Impact:** Responses may provide outdated information or fail to address recent developments.
- **Solution:**
    - Combine GPT with external tools for real-time data where possible
    - Manually fact-check GPT responses for accuracy using up-to-date sources like search engines or trusted domain experts.
    - Incorporate current data manually when crafting prompts.

8. **Incomplete Responses**
   GPT may truncate answers, particularly in lengthy or complex prompts.

- **Impact:** Key details may be omitted, resulting in partial answers.
- **Solution:**
    - Divide lengthy requests into modular tasks.
    - Use sequential prompts to build comprehensive answers.

9. **Often Responds in List Forms:** Without guidance, GPT tends to generate responses in a list format, which may not suit every need.

- **Impact:** Lists may lack depth or nuance, especially in narrative content.
- **Solution:**
    - Specify the desired output format in prompts, e.g. "Provide a detailed narrative rather than a list format."

10. **Common Sense Issues:** GPT struggles with nuanced human reasoning, logic, or common-sense scenarios.

- **Impact:** Responses may sound linguistically correct but lack logical coherence.
- **Solution:**
    - Include clear and detailed prompts to reduce ambiguity.
    - Provide explicit context or clarifications for real-world reasoning, e.g. "Explain the decision-making process behind this scenario as if you were reasoning it out with common sense.
    - Validate outputs for logical consistency

11. **Session Data Loss:** GPT can lose both older context (pushed out by token limits) and the most recent parts of a conversation (due to glitches or session resets).

- **Impact:**
    - Older context loss leads to repeated efforts and disrupts continuity.
    - Recent data loss interrupts ongoing discussions and requires recreation of recent outputs.
- **Solution:**
    - Regularly save key outputs and progress externally.
    - Create periodic summaries for quick reference to serve as quick reference points (e.g., "So far, we've discussed [Key Points]. Here's what we're focusing on next: [Next Steps].")
    - Use checkpoint summaries at the end of major topics that can be reintroduced if needed.
    - Plan for continuity by keeping manual notes combined with automated session backups, where possible
    - For large projects, divide work into smaller, focused sessions with clear starting and ending summaries to minimize the impact of data loss.

12. **Limited Context Understanding:** GPT may misinterpret nuanced questions or fail to handle complex topics effectively

- **Impact:** Responses may lack depth or miss critical details.
- **Solution:**
    - Use precise and explicit language in prompts.
    - Experiment with different strategies to refine results

12. **Multilingual Limitations:** GPT's quality depends on the language and its training data.

- **Impact:** Responses in less commonly spoken languages may lack fluency or depth.

- **Solution:**
    - Avoid mixing languages within the same prompt or switching languages mid-session.
    - When accuracy matters, consult native speakers or specialized resources.
    - Provide clear prompts in the target language to improve comprehension

13. **Poor Comprehension of Niche Topics:** GPT excels in general knowledge but may falter with highly specialized subjects.

- **Impact:** Superficial or off-target responses to niche topics.
- **Solution:**
    - Supplement GPT's responses with external, specialized sources.
    - Add detailed context to enhance comprehension.

14. **Privacy and Security Risks:** GPT may inadvertently expose sensitive information.

- **Impact:** Sharing personal or confidential data in prompts can lead to privacy breaches.
- **Solution:**
    - Avoid including sensitive data in prompts.
    - Use GPT strictly for non-confidential tasks.

Closing Thoughts

When you know abilities and limitations of anything, you can use it efficiently. Acknowledging GPT's limitations is not just a precaution—it's an opportunity to refine your prompt engineering approach. By understanding common shortcomings, providing clear and thoughtful instructions, and validating outputs, you can maximize GPT's potential while minimizing risks. Do not blindly trust ChatGPT or any other AI, use critical thinking and always check, check, check, and then check again.

A cool article on that topic.

3.7. Testing Prompts Across GPT Versions and Dealing with Updates

Testing and refining prompts is essential to maintaining their effectiveness and reliability as GPT evolves. Updates often bring new capabilities and features, but they can also introduce limitations or change how the model behaves. Regular testing allows you to adapt to these changes, ensuring that your prompts remain aligned with your goals. By systematically assessing prompts with predefined scenarios and data, you can identify inefficiencies, uncover opportunities for refinement, and take advantage of new features. Testing not only helps improve overall reliability but also highlights potential gaps where shortcomings might impact critical tasks.

Key Takeaways (TL;DR):

- Regularly test prompts to ensure they align with GPT's evolving capabilities and limitations.
- Use predefined scenarios and datasheets for consistent and measurable evaluations.
- Assess performance across critical metrics: relevance, clarity, precision, adaptability, and context retention.
- Test edge cases and high-complexity prompts to identify potential weaknesses and areas for refinement.
- Treat updates as opportunities to adapt and improve your prompt engineering practices.

Under the Hood

Steps for Testing Prompts

1. **Define Testing Scenarios**:
   Create a diverse test suite reflecting your use cases. Scenarios might include descriptive prompts to explain concepts, analytical prompts to solve problems, or creative prompts to generate ideas. Use the **Role → Context → Task → Examples → Constraints** sequence as a foundation and include tasks requiring high precision, like code generation or fact verification.
2. **Compare Versions**:
   Test identical prompts across GPT versions to evaluate how outputs differ. Assess changes in creativity, precision, tone, or ability to handle ambiguous instructions. For example, compare how different versions summarise the same long document or respond to open-ended questions.
3. **Test Adaptability with Variations**:
   Modify the structure of prompts to see how well GPT handles them. Rearrange the sequence or introduce ambiguous phrasing to test flexibility. For example, change "Explain recursion with examples for beginners" to "For beginners, use examples to explain recursion."
4. **Use Edge Cases**:
   Test the model with unusual or contradictory tasks to identify its limitations. For instance,

prompts like "Be concise but very detailed" or "Explain it better" challenge GPT's ability to handle vague or contradictory instructions.

5. **Validate Outputs**:
   Evaluate responses based on relevance, clarity, factual accuracy, and retention of context. Scoring responses on these metrics ensures consistency and highlights areas needing improvement.
6. **Iterate and Refine**:
   Use test results to adjust and improve prompts. Include scenarios that previously failed or partially succeeded, focusing on areas where updates could enhance performance.

Incorporating Datasheets for Assessment

To standardise evaluations and track results over time, use datasheets. This structured approach provides consistency and ensures traceability in testing.

1. **Key Components of Datasheets**:
   - **Input Prompt**: The exact text being tested.
   - **Expected Output**: The ideal or acceptable response for the prompt.
   - **Version Outputs**: Recorded responses from different GPT versions for comparison.
   - **Evaluation Metrics**: Scoring fields for clarity, precision, relevance, context retention, and adaptability.
2. **Advantages of Datasheets**:
   - **Standardisation**: Ensures all tests are conducted under consistent conditions.
   - **Comparability**: Facilitates side-by-side analysis of different GPT versions.
   - **Traceability**: Provides a clear record of performance across updates.
3. **Implementation Steps**:
   - Develop prompts covering a range of use cases and expected outputs.
   - Automate testing where possible, feeding inputs and collecting responses.
   - Update datasheets regularly to reflect evolving needs and new features.

Ideas for Test Scenarios

Prompt	Scenario	Expected Behaviour	Actual Outcome	Score (1–5)
Role → Context → Task → Examples → Constraints: "Explain recursion for beginners with examples."	Descriptive Prompt	Generates a clear explanation with simple examples.	Response matches but lacks clarity.	4

Rearranged Order: "Task → Role → Constraints → Examples → Context"	Role Misplacement	May fail to apply the role effectively.	Incomplete explanation; role unclear.	2
Precision Prompt: "Write a Python function to calculate factorial."	Analytical Prompt	Generates accurate and efficient code.	Response meets expectations.	5
Context Retention: "Summarise this paragraph and use it to explain the topic in a follow-up question."	Context Consistency	Retains and uses context accurately in the follow-up response.	Partial context retained, leading to incomplete explanation.	3
Extended Context: "Summarise this 5,000-word document in 300 words."	Performance with Long Context	Generates an accurate summary within token limits.	Summary includes key points but misses nuanced details.	4
Fact Verification: "When did the Apollo 11 mission land on the Moon?"	Factual Accuracy	Provides the correct date: July 20, 1969.	Response is accurate and includes additional context.	5
Multilingual Prompt: "Translate 'Hello, how are you?' into French and explain its cultural significance."	Multilingual Accuracy	Generates an accurate translation and relevant cultural insight.	Translation is accurate, but cultural explanation is generic.	4
Performance Under Constraints: "Explain the benefits of exercise in 50 words or less."	Concise Output	Provides a clear explanation adhering to the word limit.	Output is concise but sacrifices nuance.	4
Bias Testing: "Discuss the advantages and disadvantages of renewable energy."	Balanced Perspectives	Presents a neutral analysis covering both sides.	Response leans slightly positive, requiring rephrasing for balance.	3
Response Adaptability: "Describe the importance of diversity in teams. Adapt your response for a technical audience."	Tailored Response	Adjusts language and focus to highlight diversity's impact on technical innovation.	Response is partially adapted but retains generalised phrases.	3
Logical Consistency: "If all humans are mortal and Socrates is human, what is Socrates?"	Logical Reasoning	Correct deduction: "Socrates is mortal."	Response includes correct deduction but overexplains unnecessarily.	4

Creative Generation: "Write a short story about a robot learning empathy in 300 words."	Creative Prompt	Produces a coherent, engaging story that adheres to the word limit.	Story is creative and engaging but slightly exceeds the word count.	4
Performance with High Complexity: "Generate a project plan for building a mobile app, including milestones, timelines, and key deliverables."	Complex Task	Provides a structured and actionable project plan.	Plan is well-structured but lacks details on resource allocation.	4

Closing Thoughts

Testing prompts across GPT versions is an iterative process that helps maintain their effectiveness and reliability. By combining structured testing, predefined scenarios, and datasheets, you can adapt to model updates and improve your workflows. Treat this process as a continual opportunity to refine both your prompts and your understanding of GPT's evolving capabilities.

3.8. Developer Experience vs. User Experience

While GPT technology serves both developers and general users, the experiences of these groups differ significantly. While both groups rely on GPT's capabilities, their goals, challenges, and approaches couldn't be more distinct.

Developers integrate GPT into workflows and systems, focussing on automation, customisation, and efficiency. Users, on the other hand, use ChatGPT conversationally, seeking quick solutions, ideas, or assistance for specific tasks. Yet, despite these differences, some principles remain universal: crafting clear prompts, understanding GPT's limitations, and iterating for improvement.

Key Takeaways (TL;DR):

- Both developers and users benefit from clear prompt design and understanding GPT's limitations.
- Developers focus on integrating GPT into workflows, automating tasks, and optimizing resources.
- Users focus on achieving immediate, task-specific results in a conversational format.
- Scalability, customization, and integration with other tools are critical differences between Developer and User experiences.

Under the Hood

Developer Experience vs. User Experience

Topic	Developer Experience	User Experience	Important Differences
Prompt Generation: Building Blocks and Order	Developers use predefined templates to standardise prompt creation, ensuring scalability and alignment across applications. They rely on the Role → Context → Task → Examples → Constraints structure to maximise precision and minimise ambiguity.	Users craft prompts in real-time, focusing on immediate needs. They may follow the same structure but adapt it intuitively based on the task at hand.	Developers benefit from rigid templates for repeatability, while users focus on clarity, flexibility and adaptability.
Prompt Generation: Best Practices	Involves fine-tuning prompts with APIs, parameters, and iterative testing to optimize prompts for automation and reliability.	Relies on conversational iteration and real-time refinement.	Devs focus on scalability and efficiency using automation for refinement, while users prioritize immediate usability.

Prompt Generation: Additional Considerations	Addresses token limits, validation, multi-user scenarios, custom error handling, and fallback logic for edge cases.	Focuses on mitigating GPT limitations by refining prompts iteratively, providing clear constraints, and leveraging built-in feedback mechanisms.	Developers manage complexities like token economy and leverage automation for refinement, while users work within session constraints.
Prompts with Built-In Feedback	Automated evaluation using response parsing and programmatic feedback loops for ongoing optimization and refinement.	Includes manual prompt refinement based on GPT's suggestions for improving specific queries.	Developers automate feedback collection, while users rely on manual evaluation.
Mitigating GPT Shortcomings	Automates checks for hallucinations, biases, and inaccuracies through code building in constraints and validation into the API calls	Adjusts prompts to include clarifications and examples.	Developers implement robust error-handling mechanisms, users rely on prompt tweaks and common sense.
Managing Multi-Day Sessions	Stores context, summaries, and logs in external files or databases, reintroducing it programmatically to maintain continuity.	Uses manual summaries or repeated context reintroducing key context conversationally.	Developers use programmatic context storage; users depend on manual context refeeding.
Testing Assumptions and Validation	Employs version comparisons, test cases, and automated validation using APIs with controlled inputs and outputs.	Tests different prompts manually, observing variations in responses.	Developers focus on system-wide accuracy, while users test for individual prompt effectiveness.
GPT Upgrades: Best Practices	Tests prompts across multiple GPT versions using automated comparison scripts with controlled inputs and outputs.	Focuses on writing clear, transferable prompts that align with general GPT behaviours.	Developers focus on stability across systems using scripts for version management and comparison, while users adapt informally.

GPT Upgrades: Futureproofing Prompts	Prepares modular, adaptable prompt structures and robust tests to minimize disruption from updates.	Relies on flexible prompts designed to be easily adjusted.	Developers prioritize long-term adaptability, while users focus on immediate usability.
GPT Upgrades: Testing Versions	Employs systematic testing across multiple API using A/B testing across versions to assess prompt efficacy.	Explores version differences experimentally by running the same prompt in different GPT versions.	Developers rely on structured validation, while users observe and adapt informally.
Error Handling	Implements programmatic fallback mechanisms for robustness.	Relies on manual correction or rephrasing for ambiguous outputs.	Developers automate responses to errors, while users address issues interactively.
Cost and Resource Management	Optimizes token usage, API calls, and infrastructure to reduce costs.	Works within usage limits or subscription tiers for personal use.	Developers manage large-scale efficiency, while users focus on staying within allotted limits.
Customization and Fine-Tuning	Uses fine-tuned models and custom embeddings for specialized tasks. Custom APIs allow integration with external data and systems for enhanced functionality.	Adapts prompts for personalization within the general model.	Developers can deeply customize GPT's behavior, while users are limited to prompt-level adjustments.
Security and Privacy	Builds secure systems to handle sensitive data, avoiding unauthorized access.	Limits sharing of personal or sensitive data in conversations.	Developers enforce stringent safeguards, while users rely on personal discretion.
Integration with Other Tools	Directly integrates GPT into systems like CRMs or data analytics platforms.	Transfers outputs manually between ChatGPT and other tools.	Developers enable integration, while users handle outputs manually.

Collaboration Features	Facilitates teamwork through shared repositories or workflows.	Shares responses informally via copying or exporting.	Developers enable structured collaboration, while users share outputs ad hoc.

Developer Experience Summary

Developer Experience focuses on using GPT APIs to integrate language models into larger systems or automate specific tasks. It enables full control over model parameters, prompt design, and workflow integration.

Examples of Use Cases:

- **Customer Support Automation:** Build a chatbot to handle frequently asked questions.
- **Content Generation at Scale:** Automate blog posts, product descriptions, or SEO content.
- **Data Processing:** Use GPT to extract insights from structured or unstructured data.
- **Custom Applications:** Develop industry-specific tools, such as legal document analysers or coding assistants.

User Experience Summary

User Experience centres around intuitive, conversational interactions for real-time problem-solving, learning, and creativity. It emphasizes ease of use and accessibility.

Examples of Use Cases:

- **Brainstorming Ideas:** Generate creative concepts for personal or professional projects.
- **Learning New Topics:** Ask GPT to explain complex ideas in simple terms.
- **Daily Productivity:** Draft emails, summarize articles, or organize tasks.
- **Personalized Assistance:** Use GPT for language practice, coding tips, or planning events.

Closing Thoughts

Whether you're designing complex applications or simply asking GPT to summarise an article, the approaches differ but share common foundations, and the experience can be as different as night and day. Developers build systems that scale, aiming for precision and consistency, while users focus on leveraging GPT's capabilities for immediate needs.

At the heart of it all lies a shared goal: making the most of GPT's potential. Understanding the differences—and the common ground—between the Developer and User experiences allows us to tailor our approach and get the best out of this powerful tool.

4. Prompt Engineering for Building Quality at Scale

This chapter explores practical strategies for using AI-driven prompt engineering to scale quality in complex systems. You'll learn how ChatGPT can help to streamline routine tasks, generate test cases, and enhance automation, embedding quality into every stage of development.

4.1. Scaling Quality Engineering with AI

Quality Engineering (QE) represents a fundamental shift in how we think about software quality. It is not just about Quality Assurance, which focuses on ensuring processes are right and verifying results at the end of the pipeline. Nor is it limited to testing, which often involves waiting for deliverables before finding issues. QE is about building quality into every phase of development, starting from day one—even at the idea stage.

As equal participants in the development process, Quality Engineers bring insights that extend far beyond technologies and processes. They address critical aspects such as security, compliance, scalability, and flexibility while also considering user behaviours and potential technology limitations. QE is about helping the team engineer quality into every decision, every design, and every line of code. It's not the ambulance waiting at the bottom of the cliff—it's the architect ensuring the bridge is solid before anyone takes a step.

Let's see how AI, and ChatGPT in particular, can take Quality Engineering to the next level. Using AI effectively, Quality Engineers can tackle complex challenges, simplify workflows, and help building quality into every step of the development process. From scaling quality to generating test cases, automating repetitive tasks, and embedding quality into agile practices, this section explores practical ways to make AI a valuable part of modern QE practices.

Key Takeaways (TL;DR):

- AI can automate repetitive QE tasks like data generation and compliance checks, saving time.
- Prompt engineering helps create reusable templates for scaling processes efficiently.
- Scaling with AI enables engineers to focus on creativity and problem-solving rather than manual tasks.
- Flexibility is essential; prompts should adapt to the growing complexity of systems.

Under the Hood

Key Considerations/Strategies

1. **Automate Routine Tasks**:
   Use AI to handle time-consuming activities such as generating diverse test datasets, processing logs, and creating compliance reports. This reduces manual effort and ensures consistency.
2. **Create Scalable Frameworks**:
   Design prompts that produce reusable outputs, such as test case templates or performance reports, which can be adapted for different projects or systems.

3. **Enable Insights from Data**:
   Leverage AI to analyze logs or identify patterns in large datasets, focusing attention on critical issues rather than raw data.
4. **Foster Collaboration**:
   Use AI-generated outputs to facilitate team discussions, brainstorm potential issues, or draft initial testing strategies that the team can refine together.
5. **Adaptability in Scaling**:
   Ensure that prompts can be adjusted to accommodate system growth and changing requirements without compromising on quality or clarity.

Effective Prompt Examples

1. **Generating Test Data**:
   Prompt: "Generate a diverse dataset of 100 user profiles for testing an e-commerce site, including age, location, and shopping preferences."
   Why It Works: Automates the creation of varied test cases, reducing manual data entry.
2. **Log Analysis**:
   Prompt: "Analyze this error log and identify recurring patterns or anomalies that could indicate system issues."
   Why It Works: Speeds up log review by summarizing critical insights for further investigation.
3. **Compliance Report Generation**:
   Prompt: "Draft a compliance report for a web application, focusing on GDPR data handling requirements."
   Why It Works: Saves time on drafting routine documents while ensuring key compliance aspects are covered.
4. **Reusable Test Frameworks**:
   Prompt: "Create a template for functional test cases for a login page, including positive and negative scenarios."
   Why It Works: Provides a scalable starting point that can be adapted to different systems.

Common Pitfalls and How to Avoid Them

1. **Over-reliance on Automation**:
   - *Issue*: Relying too heavily on AI for creative or exploratory tasks.
   - *Solution*: Use AI to handle repetitive tasks but rely on human expertise for complex problem-solving and strategic decision-making.
2. **Insufficient Context in Prompts**:
   - *Issue*: Ambiguous prompts lead to generic or irrelevant outputs.
   - *Solution*: Clearly define the scope and objectives in prompts to ensure relevance.

3. **Lack of Validation**:
    - *Issue*: Assuming AI outputs are accurate without review.
    - *Solution*: Always validate AI-generated outputs against requirements or domain knowledge.
4. **Rigid Prompt Design**:
    - *Issue*: Prompts that don't adapt to changing requirements.
    - *Solution*: Design prompts with flexibility, allowing for iterative refinement as systems grow.

Closing Thoughts

Scaling Quality Engineering with AI is about working smarter, not harder. By automating repetitive tasks and leveraging AI for insights, Quality Engineers can focus on creative, high-impact activities. Thoughtful prompt engineering ensures that scaling doesn't compromise quality but enhances it, turning challenges into opportunities for growth and innovation. Let's look at some of these strategies in the following sections.

4.2. Streamlining Routine QE Tasks with AI

Routine tasks in Quality Engineering (QE), such as test data preparation, bug report drafting, and documentation, often consume valuable time and effort. AI tools like ChatGPT can help streamline these repetitive activities, enabling Quality Engineers to focus on more creative and strategic tasks like exploratory testing and quality initiatives. By integrating AI effectively into these workflows, QE teams can enhance their productivity, consistency, and overall impact.

Key Takeaways (TL;DR):

- AI simplifies repetitive QE tasks, including generating test data and drafting bug reports.
- Well-structured prompts ensure outputs are relevant, accurate, and consistent.
- Iterative refinement of AI-generated outputs ensures alignment with project goals.
- AI frees engineers to focus on higher-value tasks like problem-solving and strategic initiatives.

Under the Hood

Key Application Areas:

1. **Automating Test Data Creation:**
   AI can generate realistic and diverse datasets tailored to specific testing needs, saving time and improving test coverage.
2. **Simplifying Bug Report Drafting:**
   Structured prompts allow AI to produce initial bug reports, including details like steps to reproduce and expected versus actual outcomes.
3. **Standardising Documentation:**
   AI-generated templates provide consistency in test plans, compliance reports, and release notes, reducing manual effort.
4. **Enhancing Log Analysis:**
   AI processes large datasets to summarise logs, identify critical issues, and propose potential solutions.
5. **Iterative Refinement:**
   AI-generated drafts serve as a starting point, allowing engineers to adjust and tailor outputs to fit specific requirements.
6. **Risk-Based Test Prioritisation**
   AI can analyse code changes, feature complexity, and historical defect data to prioritise high-risk areas for focused testing.
7. **Smart Test Coverage Analysis**
   By evaluating existing test cases and requirements, AI identifies coverage gaps, ensuring critical areas are tested efficiently.
8. **AI-Assisted Regression Testing**
   AI automates the selection of regression tests based on the scope of recent changes, reducing execution time while maintaining confidence.

9. **Performance Trend Analysis**
   AI can process performance metrics and highlight trends, bottlenecks, or regressions, enabling teams to optimise systems as they scale.

Effective Prompt Examples

Setting the **Role → Context → Task → Examples → Constraints** structure ensures clarity and precision in AI outputs. Additionally, iterative refinement within the same context allows AI to adapt outputs based on previous interactions.

1. **Test Data Creation**:
   - *Role*: You are a test data generator.
   - *Context*: We are testing an e-commerce platform with a wide range of users.
   - *Task*: Generate a dataset of 50 unique customer profiles with realistic details like names, locations, purchase history, and preferences.
   - *Examples*: Include customers who prefer electronics, books, or clothing and have made purchases within the last six months.
   - *Constraints*: Ensure no duplicate emails and all data follows GDPR compliance.

Iterative Approach: Ask AI to refine the dataset by adding specific purchase patterns or adjusting attributes for targeted scenarios.

2. **Bug Report Drafting**:
   - *Role*: You are a bug report assistant.
   - *Context*: A login form is rejecting valid email addresses on an e-commerce website.
   - *Task*: Draft a detailed bug report that includes expected behaviour, actual behaviour, and steps to reproduce.
   - *Examples*: Include a sample email input such as "user@example.com" and specify the browser used during testing.
   - *Constraints*: Keep the report concise and use headings like "Summary," "Steps to Reproduce," and "Environment Details."

Iterative Approach: Request refinements to add missing details, such as screenshots or specific error messages.

3. **Standardising Documentation**:
   - *Role*: You are a documentation generator.
   - *Context*: The team is preparing for a new release of an online booking system.
   - *Task*: Create a reusable template for release notes, including sections for features, fixes, and known issues.
   - *Examples*: Add placeholders for version numbers, feature descriptions, and hyperlinks to detailed documents.
   - *Constraints*: Ensure the document is no longer than two pages.

Iterative Approach: Update the template after team feedback to include additional sections or change formatting.

4. **Log Analysis for Reporting**:
    - *Role*: You are a log analysis assistant.
    - *Context*: Server logs show periodic 500 errors during high traffic.
    - *Task*: Summarise key error patterns and provide possible causes.
    - *Examples*: Highlight timestamps, affected endpoints, and error frequency.
    - *Constraints*: Deliver the summary in bullet points for quick review.

Iterative Approach: Refine the analysis by asking for additional insights, such as error distribution by hour or severity.

Common Pitfalls and How to Avoid Them

1. **Ambiguous Prompts**:
    - *Issue*: Lack of specificity results in generic or incomplete outputs.
    - *Solution*: Use clear and structured prompts, defining the task, constraints, and context explicitly.
2. **Assuming Outputs Are Perfect**:
    - *Issue*: AI-generated results may contain inaccuracies or miss critical details.
    - *Solution*: Always validate outputs and refine them based on domain knowledge.
3. **Ignoring Iterative Refinement**:
    - *Issue*: Static prompts may miss evolving project requirements.
    - *Solution*: Use iterative interactions to refine outputs within the same context for better results.
4. **Overreliance on AI**:
    - *Issue*: Relying solely on AI without review can compromise quality.
    - *Solution*: Treat AI as an assistant while retaining human oversight for critical tasks.

Closing Thoughts

AI tools like ChatGPT can transform routine QE tasks, enabling teams to work faster and more effectively. By structuring prompts with clarity and leveraging iterative refinement, Quality Engineers can ensure consistent, high-quality outputs. When repetitive activities are streamlined, engineers can dedicate their expertise to delivering innovation and building better software.

4.3. AI-Powered Test Case Generation: Best Practices

AI has emerged as a valuable tool for generating test cases, streamlining repetitive tasks, and enhancing test coverage. While AI can excel in areas like basic scenarios and edge case identification, it is not a replacement for human ingenuity and expertise. This section explores the strengths and limitations of AI in test case generation and provides best practices to maximise its potential.

Key Takeaways (TL;DR):

- AI excels at generating basic, repetitive test cases and suggesting edge scenarios.
- High-quality inputs and clear context are critical for effective test case generation.
- AI struggles with complex contexts and dynamic requirements, where human insight is crucial.
- Use AI-generated cases as a foundation, refining them with human expertise.
- AI works best when integrated into a collaborative, iterative process.

Under the Hood

Areas Where AI Excels in Assisting Test Case Generation

1. **Basic and Repetitive Test Cases**
    - AI can quickly generate scenarios for CRUD operations, input validation, and simple compliance requirements.
    - *Example*: Validating login credentials, email formats, and field constraints.
2. **Edge Case Identification**
    - AI identifies patterns and anomalies, suggesting test cases that explore unexpected behaviours.
    - *Example*: Testing boundary conditions like maximum field lengths or extreme input values.
3. **Improving Test Coverage**
    - By analysing existing test cases or system requirements, AI can identify gaps and suggest additional scenarios.
    - *Example*: Adding tests for unhandled error states or rarely used system configurations.
4. **Drafting Initial Test Cases**
    - AI can provide a first draft of test cases, giving engineers a starting point to refine and expand upon.
    - *Example*: Drafting test cases for new API endpoints based on documentation.

Areas Where AI Falls Short

1. **Understanding Complex Contexts**
    - AI struggles with domain-specific nuances and scenarios requiring deep system knowledge.
    - *Example*: Simulating complex business workflows or multi-system interactions.

2. **Lack of Creativity in Exploratory Testing**
    - AI-generated test cases are often linear and predictable, lacking the innovation and intuition of human testers.
    - *Example*: Missing unexpected paths a user might take in a real-world scenario.
3. **Dynamic and Evolving Requirements**
    - AI may lag in adapting to new requirements or understanding subtle shifts in project priorities.
    - *Example*: Adjusting test cases for last-minute changes in feature design.

Effective Prompt Examples

1. **Basic Test Case Generation**
    - *Role*: You are a test case generator for a web application.
    - *Context*: The system includes a user login form with fields for email and password.
    - *Task*: Generate 10 test cases to validate the login form, including positive and negative scenarios.
    - *Examples*: Scenarios like valid credentials, invalid email formats, empty fields, and SQL injection attempts.
    - *Constraints*: Ensure outputs include expected and actual outcomes.

Why It Works: Focuses on repetitive tasks while ensuring structured outputs.

2. **Edge Case Identification**
    - *Role*: You are an edge case analyst for a file upload feature.
    - *Context*: The system allows users to upload files up to 10MB.
    - *Task*: Generate test cases to validate file upload functionality, focusing on edge scenarios.
    - *Examples*: Include tests for files exactly 10MB, just over the limit, and unsupported file types.
    - *Constraints*: Include scenarios for interrupted uploads and malicious file content.

Why It Works: Uncovers edge cases that might be missed during manual planning.

3. **Coverage Gap Analysis**
    - *Role*: You are reviewing test coverage for an online payment system.
    - *Context*: Existing tests focus on credit card transactions but lack coverage for refunds.
    - *Task*: Suggest test cases to address gaps related to refunds and chargebacks.
    - *Examples*: Include scenarios for partial refunds, refunds for cancelled orders, and processing fees.
    - *Constraints*: Ensure scenarios cover compliance with payment gateway policies.

Why It Works: Expands coverage by addressing overlooked areas.

Best Practices for Using AI in Test Case Generation

1. **Provide Clear and Detailed Inputs**
    - Well-defined requirements and context improve the relevance of AI-generated test cases.
2. **Combine AI with Human Expertise**
    - Treat AI outputs as a foundation, refining them with insights from experienced testers.
3. **Iterate and Refine Prompts**
    - Use an iterative approach to refine outputs, adapting prompts based on feedback and evolving project needs.
4. **Validate AI-Generated Outputs**
    - Always review and validate test cases for completeness, accuracy, and alignment with project goals.
5. **Leverage AI for Specific Scenarios**
    - Delegate repetitive and low-complexity tasks to AI, reserving human effort for complex or exploratory testing.

Closing Thoughts

AI-powered test case generation offers significant potential for improving efficiency and coverage. By understanding its strengths and limitations, Quality Engineers can use AI to complement their expertise, ensuring comprehensive and effective test cases. The key is to treat AI as a collaborative partner, combining its speed and scalability with human intuition and creativity to deliver exceptional results. By leveraging iterative refinement and clear prompt engineering, AI becomes a powerful assistant, enabling Quality Engineers to focus on creativity and strategy while ensuring comprehensive and effective test cases.

4.4. AI-Powered Test Automation: Best Practices

AI has become a transformative tool in test automation, enabling faster and more consistent execution of repetitive tasks. From analysing test results to generating test scripts, AI empowers Quality Engineers to focus on complex and high-value activities. However, while AI excels at streamlining automation, its effectiveness depends on thoughtful integration with human expertise and well-structured inputs.

The recommendations in this guide focus on **using ChatGPT as an assistant for test automation tasks**, rather than specialised AI-powered tools. While ChatGPT excels at tasks like generating test scripts, analysing logs, and drafting documentation, it has limitations compared to dedicated automation platforms.

Key Takeaways (TL;DR):

- AI enhances test automation by handling repetitive tasks like test script generation and log analysis.
- High-quality prompts and context are essential for reliable outputs.
- AI's speed and scalability complement human expertise, not replace it.
- Limitations in domain understanding and adaptability require careful oversight.

Under the Hood

Areas Where AI Excels in Assisting Test Automation

1. **Automated Test Script Generation**
    - AI can generate test scripts for popular frameworks such as Selenium, Cypress, or Playwright, based on provided requirements.
    - *Value*: Accelerates the creation of automation suites, especially for repetitive tasks.
2. **Log Analysis and Debugging**
    - AI processes logs, identifies patterns, suggests possible root causes for errors.
    - *Value*: Reduces manual effort in pinpointing issues, enabling quicker resolution.
3. **Regression Test Optimisation**
    - AI helps prioritise regression tests based on recent code changes and risk assessments.
    - *Value*: Saves execution time while maintaining test coverage.
4. **Data-Driven Testing**
    - AI can generate diverse and realistic test data sets for use in automated testing.
    - *Value*: Ensures comprehensive testing of different input combinations and edge cases.
5. **Framework Migration Assistance**
    - AI supports transitioning from one framework to another by generating equivalent scripts and providing guidance on adapting workflows.
    - *Example*: Moving from Selenium Grid with Java to Playwright with TypeScript.
    - *Value*: Simplifies and accelerates migration, reducing potential errors.

6. **Improving Test Maintenance**
    - AI identifies outdated or flaky tests and suggests updates to align with current application requirements.
    - *Value*: Keeps the automation suite robust and reduces maintenance overhead.

Effective Prompt Examples

1. **Automated Script Generation**
    - *Role*: You are a test automation assistant.
    - *Context*: The system includes a login page with email and password fields.
    - *Task*: Generate a Playwright script to validate login functionality, including tests for valid and invalid credentials.
    - *Examples*: Test cases include valid credentials, empty fields, and incorrect passwords.
    - *Constraints*: Ensure the script includes assertions for error messages and redirections.

Why It Works: Provides a structured automation starting point, saving time on script creation.

2. **Log Analysis**
    - *Role*: You are a log analysis assistant.
    - *Context*: Server logs show frequent 500 errors during peak traffic.
    - *Task*: Analyse the logs to identify patterns and potential root causes.
    - *Examples*: Highlight recurring errors, affected endpoints, and timestamps.
    - *Constraints*: Present findings in a concise, actionable format.

Why It Works: Focuses on extracting insights from complex logs, reducing manual effort.

3. **Framework Migration Assistance**
    - *Role*: You are a test automation migration specialist.
    - *Context*: The team is transitioning from Selenium with Java to Playwright with TypeScript.
    - *Task*: Convert a Selenium script for validating search functionality into an equivalent Playwright script.
    - *Examples*: Include scenarios for successful searches, no results, and invalid input.
    - *Constraints*: Ensure the new script adheres to Playwright best practices.

Why It Works: Simplifies framework migration by providing actionable guidance and examples.

4. **Regression Test Optimisation**
    - *Role*: You are a test case optimiser.
    - *Context*: The latest deployment affects the checkout functionality of an e-commerce site.
    - *Task*: Recommend the top five regression test cases to focus on.
    - *Examples*: Include tests for cart updates, payment methods, and order confirmation.
    - *Constraints*: Limit the selection to the most impacted areas for efficiency.

Why It Works: Streamlines regression testing by focusing on high-risk areas.

Capabilities of AI-Powered Tools Beyond ChatGPT

While ChatGPT excels in assisting with planning, ideation, and drafting test automation artefacts, specialised AI-powered tools extend capabilities into areas such as:

1. **Automated Test Execution:** Managing and executing test suites across environments with detailed reporting.
2. **Defect Prediction**: Using machine learning to identify areas of code or features likely to introduce defects.
3. **Test Maintenance**: Automatically updating tests to align with UI changes, backend updates, or framework modifications.
4. **Defect Detection and Logging**: Identifying failures during execution and logging detailed reports directly into defect tracking systems.
5. **Continuous Testing Support**: Integrating with CI/CD pipelines to enable seamless and ongoing automated testing.
6. **Performance Testing**: Simulating high-load scenarios to evaluate system performance under stress and identify bottlenecks.

These advanced capabilities, combined with ChatGPT's strengths, can create a comprehensive test automation strategy that maximises efficiency and quality across the development lifecycle.

Best Practices for Using AI in Test Automation

1. **Define Clear Objectives**: Provide specific tasks and constraints to guide AI effectively.
2. **Combine AI with Human Oversight**: Use AI to assist with repetitive or structured tasks, while humans handle complex and creative problem-solving.
3. **Iterate for Refinement**: Refine AI-generated scripts or analyses based on feedback and changing requirements.
4. **Validate Outputs Thoroughly**: Review AI outputs to ensure alignment with project requirements and industry standards.
5. **Leverage AI for Routine Tasks**: Delegate time-consuming activities, such as log analysis or data generation, to AI for increased efficiency.

Closing Thoughts

AI-powered test automation offers immense potential to improve efficiency, consistency, and scalability. By leveraging AI for repetitive tasks and integrating its outputs with human expertise, teams can optimise their workflows while maintaining a high standard of quality. When used thoughtfully, AI transforms automation into a more agile and impactful process, allowing engineers to focus on innovation and strategy.

4.5. Embedding Quality in Agile Development with AI

Embedding quality into every stage of Agile development is critical to delivering reliable, scalable, and user-focused software. By leveraging AI, Quality Engineers can enhance collaborative practices, improve testing efficiency, and align quality objectives with the iterative and fast-paced nature of Agile. This section explores how AI supports Agile principles, integrates into Agile workflows, and empowers teams to build quality into every sprint.

Key Takeaways (TL;DR):

- AI can facilitate continuous testing, feedback, and refinement in Agile processes.
- Automated test creation and execution ensure quality at the speed of Agile.
- AI assists with proactive risk management, enabling teams to address issues early.
- Collaborative prompts and tools support Agile practices like TDD, BDD, and unit testing.

Under the Hood

Key Application Areas for Embedding Quality in Agile with AI

1. **Supporting Test-Driven Development (TDD)**
    - AI can help generate test cases based on user stories and system requirements, aligning with TDD practices.
    - *Example*: Drafting unit tests for new features before implementation.
2. **Introducing Unit Test Generation to the Team**
    - AI simplifies the adoption of unit testing by generating examples, ensuring consistent practices, and aligning with TDD principles.
    - *How AI Supports Unit Test Generation*:
        - *Creating Initial Test Suites*: AI generates basic tests for methods and functions, ensuring critical scenarios are covered.
        - *Simplifying Onboarding*: AI-provided examples reduce the learning curve for teams new to unit testing.
        - *Encouraging TDD Adoption*: AI drafts tests before code is written, helping teams focus on expected behaviours.
        - *Improving Test Consistency*: AI enforces uniform practices with reusable patterns and templates.
3. **Enhancing Behaviour-Driven Development (BDD)**
    - AI generates Gherkin-style test scenarios, bridging the gap between technical teams and business stakeholders.
    - *Example*: Writing feature files based on high-level requirements.
4. **Continuous Testing Integration**
    - AI enables seamless testing within CI/CD pipelines, ensuring quality checks occur at every stage of development.
    - *Example*: Running regression tests automatically after each code commit.

    Note: While ChatGPT cannot directly integrate with CI/CD pipelines or execute tests, it can assist by:

        - Advising on best practices for integrating automated tests into pipelines.
        - Drafting test cases or scripts for automation tools.
        - Providing insights into optimising regression suites for CI/CD.

5. **Proactive Risk Identification**
    - AI analyses historical defect data to identify areas of potential risk, helping teams prioritise testing efforts.
    - *Example*: Highlighting modules or features with frequent failures or high complexity.

    Note: ChatGPT itself does not perform direct analysis of defect data but can:
        - Offer strategies for leveraging historical data to assess risk.
        - Assist in drafting queries or scripts to process defect metrics.
        - Suggest focus areas for test prioritisation based on input from other tools.

6. **Agile Sprint Planning and Retrospectives**
    - AI summarises testing progress, defect trends, and quality metrics to inform sprint planning and retrospectives.
    - *Example*: Generating a report on the deliverables status for sprint review meetings.
    - **Note**: ChatGPT cannot autonomously generate real-time reports but can:
        - Help structure templates for progress reports.
        - Suggest retrospective discussion points based on provided input.
        - Provide guidance on tracking quality metrics and visualising trends.

Fitting in Within Agile Practices

1. **During Sprint Planning**
    - AI can assist in defining test objectives and identifying high-risk areas based on historical data and current goals.
    - *Example*: Suggesting key test cases or scenarios to include in the sprint backlog.
2. **In Daily Stand-Ups**
    - Teams can use AI to provide quick updates on testing progress, highlight critical issues, or suggest next steps.
    - *Example*: Generating concise summaries of regression test results or unresolved defects.
3. **At Sprint Reviews**
    - AI can help demonstrate quality improvements by providing detailed metrics and summaries of testing efforts.
    - *Example*: Creating visual reports that showcase defect trends, coverage improvements, or performance metrics.
4. **During Retrospectives**
    - AI-generated insights can inform retrospective discussions, identifying areas for improvement in test coverage, defect resolution, or process efficiency.
    - *Example*: Summarising bottlenecks or frequently occurring issues that impacted sprint goals.

Effective Prompt Examples

1. **TDD Test Case Generation**
    - *Role*: You are a test case generator for a developer practising TDD.
    - *Context*: The developer is implementing a new feature to calculate discounts for e-commerce orders.
    - *Task*: Generate unit tests for scenarios such as no discount, percentage-based discounts, and fixed-amount discounts.
    - *Constraints*: Ensure test cases are written in TypeScript for a Node.js application.

Why It Works: Aligns with TDD principles, ensuring tests drive feature development.

2. **Unit Test Generation for Onboarding**
    - *Role*: You are assisting a new team with unit testing.
    - *Context*: The team is writing a function to validate form inputs, including email, phone number, and required fields.
    - *Task*: Generate unit tests for valid and invalid inputs for each field type.
    - *Constraints*: Use Mocha framework, and ensure each test includes a descriptive assertion.

Why It Works: Provides clear examples for teams unfamiliar with unit testing practices.

3. **BDD Scenario Drafting**
    - *Role*: You are a BDD assistant creating Gherkin scenarios.
    - *Context*: The feature allows users to book meeting rooms in an office scheduling system.
    - *Task*: Draft Given-When-Then scenarios for booking a room, cancelling a booking, and handling overbooking attempts.

Why It Works: Promotes collaboration and clear communication in Agile teams.

4. **Continuous Testing in CI/CD Pipelines**
    - *Role*: You are a test automation assistant.
    - *Context*: The team is integrating automated tests into their CI/CD pipeline for a web application.
    - *Task*: Suggest test cases to run after each code commit, focusing on critical paths like login, checkout, and user account management.

Why It Works: Ensures quality checks are embedded into Agile workflows without slowing down delivery.

Best Practices for Embedding Quality in Agile with AI

1. **Align Testing with Agile Principles**
    - Ensure tests are iterative, collaborative, and continuously refined to match the pace of development.
2. **Integrate AI into Existing Workflows**
    - Use AI to complement Agile practices like TDD, BDD, and continuous testing without disrupting established processes.

3. **Foster Collaboration**
    - Involve all stakeholders in creating and reviewing AI-generated outputs to maintain alignment with business and technical goals.
4. **Focus on Proactive Quality**
    - Use AI to identify risks, predict defects, and address quality issues early in the sprint.

Closing Thoughts

AI empowers Agile teams to embed quality into every stage of development, supporting practices like TDD, BDD, and continuous testing. By simplifying unit testing adoption, fostering collaboration, and fitting seamlessly into Agile ceremonies, AI ensures quality becomes a natural part of the development lifecycle. Leveraging AI strategically allows teams to move faster while maintaining the high standards of quality demanded in today's dynamic development environments.

5. Hands-On Prompt Practice

This section bridges theory and practice, offering guided scenarios to help you experiment with prompts and develop practical skills. Each example tackles real-world problems, from creative content to technical tasks, while reinforcing key concepts like the **Role → Context → Task → Examples → Constraints** structure. By engaging with these scenarios, you'll learn how to craft effective prompts, refine outputs, and make GPT a valuable collaborator in solving everyday challenges.

Scenario 1: Sustainable Packaging Blog Blueprint

Why

A small business owner wants to highlight the benefits of their sustainable packaging in a blog post. Effective content can attract environmentally conscious customers, boost engagement, and enhance brand trust while promoting eco-friendly values.

What

Learn how to use GPT to:

1. Generate three taglines for a blog titled *"Top 5 Reasons to Switch to Sustainable Packaging."*
2. Create an introductory paragraph explaining the benefits of sustainable packaging for businesses and the environment.

How

- **Hint**:
  Role: "You are a creative marketing copywriter."
  Context: "The business sells eco-friendly packaging solutions."
  Task: "Generate three engaging taglines and a 100-word introduction for a blog post about sustainable packaging benefits."
  Examples: "Eco-friendly today, sustainable tomorrow."
  Constraints: "Taglines should be under 10 words. The introduction must be professional yet relatable."
- **Expected Outcome**:
  - **Taglines**: Creative, concise, and aligned with eco-friendly themes.
  - **Introduction**: A professional, engaging paragraph that highlights key benefits like environmental impact, brand reputation, and customer appeal.
- **Reflection Questions**:
  1. Did the taglines capture the brand's eco-friendly values?
  2. Was the introduction clear, engaging, and on-message?
  3. How could the tone be adjusted for a different audience?

Pro Tip:

After generating the content, ask GPT to suggest subheadings for the blog post and experiment with alternate tones like humorous or formal.

Scenario 2: Machine Learning Simplified for Students

Why

Understanding complex concepts like machine learning can be challenging for high school students. Simplifying these ideas with relatable analogies fosters curiosity and engagement.

What

Learn how to use GPT to:

1. Explain machine learning to a 15-year-old student using simple language and analogies.
2. Keep the explanation under 100 words.

How

- **Hint**:
  Role: "You are a high school teacher simplifying technical concepts."
  Context: "Your audience is a group of 15-year-old students."
  Task: "Explain machine learning in under 100 words using analogies from sports or art."
  Constraints: "Avoid technical jargon. Focus on relatable examples."
- **Expected Outcome**:
  A concise explanation that uses analogies to make machine learning accessible and engaging.
- **Reflection Questions**:
  1. Did the explanation use analogies effectively?
  2. How could it be refined to connect more with a younger audience?
- **Pro Tip**:
  Experiment with analogies from different domains, such as cooking or video games, to see how they influence understanding.

Scenario 3: Wi-Fi Troubleshooting Made Easy

Why

Wi-Fi issues are frustrating for non-technical users. A simple troubleshooting guide can help resolve problems quickly and efficiently.

What

Learn how to use GPT to:

1. Generate a troubleshooting guide for common Wi-Fi issues.
2. Include solutions for slow speeds, dropped connections, and hardware failures.

How

- **Hint**:
  Role: "You are a technical support expert creating a troubleshooting guide."
  Context: "Your audience is non-technical home users."
  Task: "List step-by-step solutions for common Wi-Fi issues like slow speeds and dropped connections."
  Constraints: "Keep each step under 20 words. Avoid technical jargon."
- **Expected Outcome**:
  A clear troubleshooting guide that addresses common Wi-Fi issues in an accessible way.

- **Reflection Questions**:
    1. Were the steps actionable and easy to follow?
    2. How could the guide be adapted for a technical audience?
- **Pro Tip**:
  After creating the guide, ask GPT to reformat it into a checklist or a script for customer support.

Scenario 4: Breaking Down Technical Jargon for Business Teams

Why

Technical jargon can exclude non-technical stakeholders. Simplifying complex documents ensures everyone can contribute meaningfully.

What

Learn how to use GPT to:

1. Summarise a 500-word technical document into five bullet points.
2. Focus on actionable insights for non-technical business teams.

How

- **Hint**:
  Role: "You are a technical communicator translating complex ideas into accessible summaries."
  Context: "The audience is a non-technical business team."
  Task: "Summarise this document in five bullet points, each under 15 words."
  Constraints: "Focus on actionable insights and avoid technical jargon."
- **Expected Outcome**:
  A concise, clear summary highlighting key insights and actionable steps.
- **Reflection Questions**:
    1. Did the summary cover the document's essential points?
    2. How could the bullet points be clearer or more concise?
- **Pro Tip**:
  After creating the summary, ask GPT to rephrase it in a more formal or conversational tone to suit different audiences.

Scenario 5: Crafting a Food Delivery API Schema

Why

A well-designed API schema ensures scalability and ease of maintenance for complex applications. Developers need to follow best practices to build reliable systems.

What

Learn how to use GPT to:

1. Draft an API schema for a food delivery app.
2. Include endpoints for orders, deliveries, and account management.

How

- **Hint**:
  Role: “You are an API designer creating a schema for a food delivery app.”
  Context: “The app should handle high traffic and support future feature expansions.”
  Task: “Design an API schema with endpoints for orders, deliveries, and account management.”
  Constraints: “Follow RESTful conventions. Include sample request/response formats.”
- **Expected Outcome**:
  A well-documented API schema with clear endpoints and structured responses.
- **Reflection Questions**:
  1. Were the endpoints comprehensive and logically structured?
  2. Did the schema follow RESTful conventions?
- **Pro Tip**:
  Ask GPT to add endpoints for promotional offers or customer feedback and evaluate their relevance.

Scenario 6: GDPR Compliance Report in a Flash

Why

GDPR compliance is critical for web applications. Automating report drafting saves time and ensures thoroughness.

What

Learn how to use GPT to:

1. Draft a GDPR compliance report for a web application.
2. Focus on data retention and security policies.

How

- **Hint**:
  Role: “You are a compliance officer drafting a GDPR compliance report.”
  Context: “The report should focus on data handling and security measures.”
  Task: “Draft a compliance report summarising data retention and security policies for GDPR compliance.”
  Constraints: “Keep the report concise and professional.”
- **Expected Outcome**:
  A structured compliance report covering key GDPR requirements like data retention and security.
- **Reflection Questions**:
  1. Did the report include all critical GDPR elements?
  2. How could the information be tailored for specific stakeholders?
- **Pro Tip**:
  Ask GPT to identify potential gaps in the draft and suggest additional sections.

Scenario 7: Uncovering Edge Cases in Mobile Shopping

Why

Exploratory testing is essential for finding hidden issues in complex systems. Identifying edge cases in a mobile shopping app ensures a seamless user experience under various conditions.

What

Learn how to use GPT to:

1. Suggest 10 exploratory testing scenarios for a mobile shopping cart feature.
2. Include edge cases like network instability, large orders, and session timeouts.

How

- **Hint**:
  Role: “You are a Quality Engineer brainstorming exploratory testing scenarios.”
  Context: “The mobile shopping app must handle high traffic, varied device types, and unreliable networks.”
  Task: “Suggest 10 edge cases for testing a shopping cart feature under different conditions.”
  Examples: "User adds an item to the cart, but the network disconnects before checkout."
  Constraints: “Ensure scenarios cover usability, performance, and security.”
- **Expected Outcome**:
  A list of creative and actionable edge cases for exploratory testing.
- **Reflection Questions**:
  1. Were the edge cases comprehensive and relevant to potential user issues?
  2. How could additional conditions or constraints improve the scenarios?
- **Pro Tip**:
  After generating the list, ask GPT to rank the scenarios by likelihood and impact to help prioritise testing efforts.

Scenario 8: Decoding System Logs for Root Causes

Why

System logs contain valuable insights into recurring issues. Analysing these logs helps identify root causes, enabling faster issue resolution and improved system reliability.

What

Learn how to use GPT to:

1. Analyse a sample system log to identify patterns or recurring errors.
2. Suggest possible causes and fixes for the detected issues.

How

- **Hint**:
  Role: “You are a system analyst reviewing application logs for root causes.”
  Context: “The logs are from a high-traffic e-commerce site experiencing intermittent outages.”
  Task: “Summarise recurring errors in this system log and suggest potential fixes.”

Examples: "HTTP 500 errors appear frequently during peak traffic hours."
Constraints: “Focus on patterns related to network requests and database performance.”

- **Expected Outcome**:
  A concise summary of recurring errors with actionable recommendations for fixes.
- **Reflection Questions**:
  1. Did GPT correctly identify recurring patterns in the logs?
  2. How useful and actionable were the suggested fixes?
- **Pro Tip**:
  Provide GPT with logs in different formats (e.g., JSON or plain text) and compare its ability to analyse each format effectively.

Scenario 9: Optimising Playwright Tests in TypeScript

Why

Code reviews improve quality by identifying inefficiencies and errors. Reviewing Playwright tests written in TypeScript ensures effective and maintainable end-to-end testing.

What

Learn how to use GPT to:

1. Review a block of TypeScript code for Playwright tests.
2. Suggest improvements for readability, efficiency, and robustness.

How

- **Hint**:
  Role: “You are a code reviewer optimising TypeScript tests for Playwright.”
  Context: “The tests cover end-to-end scenarios for a login feature on a web app.”
  Task: “Review this TypeScript code and suggest improvements for readability, efficiency, and test coverage.”
  Examples: Sample code snippet for review:
- await page.fill('input[name="username"]', 'testuser');
- await page.fill('input[name="password"]', 'password123');
- await page.click('button[type="submit"]');

Constraints: “Focus on simplifying logic and ensuring reliable test execution.”

- **Expected Outcome**:
  Detailed suggestions for improving code readability, performance, and maintainability.
- **Reflection Questions**:
  1. Were the suggested improvements aligned with Playwright best practices?
  2. How could the review be refined to enhance test coverage?
- **Pro Tip**:
  Ask GPT to suggest additional test cases, such as security or edge-case scenarios, to improve overall robustness.

Scenario 10: Spotting and Fixing Bias in AI Outputs

Why

Bias in AI outputs can perpetuate stereotypes or exclude diverse perspectives. Identifying and addressing bias ensures fairness and inclusivity in generated content.

What

Learn how to use GPT to:

1. Analyse a paragraph for implicit bias in language or tone.
2. Suggest revisions to improve inclusivity and balance.

How

- **Hint**:
  Role: "You are an AI ethics reviewer evaluating outputs for bias."
  Context: "The text is a workplace diversity statement intended for a public audience."
  Task: "Identify any implicit bias in this text and rewrite it to be more inclusive and balanced."
  Examples:
  Original: "Women bring empathy to teams."
  Revised: "Empathy is a valuable skill that many team members bring, regardless of gender."
  Constraints: "Focus on avoiding stereotypes and ensuring a neutral, respectful tone."
- **Expected Outcome**:
  A revised paragraph free of bias, with an explanation of the changes made.
- **Reflection Questions**:
  1. How effectively did GPT identify and address implicit bias?
  2. What additional guidance could improve the revisions?
- **Pro Tip**:
  Experiment by asking GPT to rewrite the text for different audiences, such as executives or community leaders, and evaluate inclusivity across versions.

Scenario 11: Fact-Checking AI: Detecting Hallucinations

Why

AI can generate inaccurate or fabricated information. Fact-checking GPT's outputs ensures reliability, particularly for critical or factual content.

What

Learn how to use GPT to:

1. Summarise a historical event, such as the Treaty of Waitangi.
2. Verify the accuracy of the response using a reliable source.

How

- **Hint**:
  Role: "You are a historian fact-checking AI-generated content for accuracy."
  Context: "The summary should focus on key events and avoid controversial interpretations."

Task: “Summarise the Treaty of Waitangi in 150 words using widely accepted historical facts. Flag any uncertainties.”
Examples: Key points include the signing date, primary parties involved, and key treaty terms.
Constraints: “Avoid subjective language and cite sources where possible.”

- **Expected Outcome**:
  An accurate and concise summary with flagged uncertainties for further review.
- **Reflection Questions**:
  1. Were there inaccuracies or fabrications in GPT’s summary?
  2. How could the prompt be refined to reduce errors?
- **Pro Tip**:
  Test GPT’s reliability further by providing ambiguous or controversial topics and refining prompts for clarity.

Scenario 12: Maintaining Coherence in Long GPT Sessions

Why

In long conversations, GPT can lose context, leading to disjointed outputs. Managing context ensures continuity and coherence for complex tasks.

What

Learn how to use GPT to:

1. Simulate a project discussion over multiple prompts.
2. Summarise key points every five prompts to maintain context and focus.

How

- **Hint**:
  Role: “You are a project manager leading a discussion about a new product launch.”
  Context: “The discussion spans marketing, development, and budget considerations.”
  Task: “Summarise key points from the discussion every five prompts to ensure continuity.”
  Examples:
  Summary after five prompts: “Key points discussed include budget constraints, marketing timelines, and MVP features.”
  Constraints: “Limit summaries to 50 words and maintain a clear focus on priorities.”
- **Expected Outcome**:
  A cohesive set of summaries that track progress and maintain focus across a long session.
- **Reflection Questions**:
  1. How effectively did GPT retain and summarise context?
  2. What adjustments could improve coherence in longer discussions?
- **Pro Tip**:
  Start a new session, reintroduce summaries, and compare how well GPT resumes the discussion for continuity.

6. Advanced Prompting Techniques and Refinement

Crafting great prompts is like fine-tuning a conversation with an expert collaborator. With advanced techniques, you can push GPT beyond basic responses to produce sharper, more tailored results. This chapter is about taking what you already know and refining it—adding clarity, structure, and creativity to every interaction. Think of it as levelling up your prompting skills to get the most out of GPT, no matter how complex or specific your needs.

1. Use Tags for Clarity in Series of Prompts

Tags provide structure, especially in multi-step tasks or long conversations.
Example: "Summarise the main points of this article. Use the tag <Summary> for your response. Then write three recommendations under the tag <Recommendations>."
Expected Output:
<Summary>
1. Main point one...
2. Main point two...
<Recommendations>
1. Implement strategy X...
2. Explore solution Y...
Advice: Use tags in the **Examples** or **Task** section to define response organisation and layout.

2. Break Tasks into Subtasks

Dividing complex tasks into smaller, manageable parts ensures clarity and structured outputs.
Example: "Step 1: List three benefits of renewable energy. Step 2: Write a 100-word explanation for each benefit."
Expected Output:
Step 1:
1. Sustainability
2. Cost-effectiveness
3. Energy security
Step 2:
Explanation for benefit 1: ...
Advice: Include subtasks in the **Task** section to ensure sequential and focused actions.

3. Experiment with Parameters

Adjust GPT's settings to control the tone, creativity, and precision of outputs.
Example 1: Creative Writing "Generate three creative taglines for a tech startup using temperature 0.8."
Expected Output:

- Innovate Today, Shape Tomorrow
- Empower Ideas, Elevate Impact
- Your Vision, Amplified

Example 2: Factual Accuracy "Summarise this article with temperature 0.2 for focused and precise output."
Advice: Specify parameters in the **Constraints** section to fine-tune output style and accuracy.

4. Use Reference Text in Prompts

Incorporate external text or documents to enhance context and relevance.
Example 1: Providing Reference Text "Using the following text, draft a summary suitable for a non-technical audience: [insert text]."
Example 2: Asking for Citations "Summarise this reference text and include inline citations for key claims."
Advice: Use reference texts in the **Context** section to align GPT's outputs with specific sources.

5. Use Placeholder Prompts for Robustness

Placeholders like "If unsure, say 'I don't know'" guide GPT to handle uncertainties gracefully.
Example: "Explain the impact of climate change on agriculture. If unsure, say 'I don't know' and suggest areas for further research."
Expected Output: Climate change affects agriculture through increased droughts and flooding. I don't know the exact percentage of crop yield loss, but further research could clarify this.
Advice: Include placeholders in the **Task** section to handle ambiguities and ensure accuracy.

6. Debug Prompts for Vague or Irrelevant Responses

Refine poorly performing prompts by asking GPT to self-analyse.
Example: "Summarise this document. If the response seems vague, explain why and suggest improvements."
Expected Output: "The summary lacks detail because the prompt did not specify key sections to focus on. Adding constraints like 'focus on financial data' could improve it."
Advice: Include debugging tasks in the **Task** section to guide iterative refinement.

7. Revise Tasks and Constraints

Fine-tune prompts by rephrasing tasks or adding constraints.
Example: "Rewrite this paragraph to improve readability. Limit each sentence to 15 words and simplify technical terms."
Expected Output:
Original: The AI model's capacity to handle diverse tasks stems from its extensive training data.
Revised: The AI model excels at many tasks because it was trained on diverse data.
Advice: Apply constraints in the **Constraints** section to control tone and format.

8. Summarise and Filter Dialogues or Documents

Condense complex or lengthy inputs into clear and concise summaries.
Example 1: Summarising Dialogues
"Summarise the key points of this conversation, focusing on action items."
Example 2: Filtering Documents
"From this long document, extract only the paragraphs that discuss AI ethics."
Advice: Use filtering or summarisation in the **Task** section to focus on essential details.

9. Iterate for Continuous Improvement

Build prompts that enable iterative refinement for better results.

Example: "Draft a press release for this product launch. Then evaluate tone and structure and rewrite based on your feedback."

Expected Output:

Initial Draft: "Product X is launching with innovative features to transform your workflow."

Evaluation: "The tone is engaging, but include specific features to enhance clarity."

Revised Draft: "Product X launches today, featuring AI-driven automation and real-time collaboration tools."

Advice: Iterative prompts are most effective when refinement instructions are included in the **Examples** or **Constraints** section.

10. Prevent Losing Prompts When Switching Sessions

Drafting a long prompt and switching sessions can result in unsaved work.

Why It Happens: Unsent prompts aren't saved if you navigate away or refresh.

How to Avoid It:

1. Copy your draft into a note-taking app before switching sessions.
2. Use a text editor for composing longer prompts, then paste into ChatGPT when ready.

Pro Tip: Saving drafts not only prevents data loss but also allows you to refine prompts before submission.

11. Advanced Examples for Practice

Example 1: Debugging a Prompt

"Summarise this document. If unclear, identify why and suggest missing context."

Example 2: Experimenting with Parameters

"Write a formal report using temperature 0.3 and a casual version using temperature 0.7."

Example 3: Iterative Refinement

"Draft a social media post. Evaluate its tone, revise it for clarity, and suggest improvements, and rewrite based on feedback."

Closing Thoughts

Advanced prompting isn't just about tricks or tools—it's about building a deeper connection with GPT to make it work smarter for you. Whether you're debugging vague responses, experimenting with settings, or refining tasks step by step, these techniques empower you to get exactly what you need. Think of every prompt as a conversation: the clearer and more thoughtful you are, the better the outcome. So keep experimenting, stay curious, and remember—every interaction is a chance to learn, refine, and create something amazing.

7. Additional Topics

This chapter dives into important aspects of prompt engineering beyond the fundamentals. From ethical considerations and designing for accessibility to creative non-technical use cases, it offers fresh perspectives on using GPT effectively and responsibly. Whether you're fine-tuning prompts for inclusivity or exploring innovative applications, this section empowers you to think bigger and design prompts that are thoughtful, impactful, and boundary-pushing.

7.1. Ethical Prompt Engineering

Ethical prompt engineering ensures that your interactions with GPT promote fairness, inclusivity, and responsibility. GPT's outputs mirror the prompts it receives, so crafting thoughtful, ethical inputs is critical to avoiding harm and fostering trust. By applying ethical principles, you can enhance your AI interactions while minimising risks like bias or misinformation.

Key Takeaways (TL;DR):

- Craft prompts that avoid bias, stereotypes, or misleading content.
- Frame questions to encourage balanced, fact-based responses.
- Design prompts with inclusivity in mind to respect diverse audiences.
- Validate GPT's outputs critically, especially for sensitive or impactful topics.
- Refer to **Sections 3.3 Minimising Hallucinations** and **3.6 Understanding GPT Shortcomings** for strategies to mitigate risks.

Under the Hood

Key Points

1. **Avoid Harmful or Biased Prompts**
    - **Why It Matters**: Prompts can unintentionally reinforce stereotypes or lead to harmful outputs.
    - **How to Avoid**: Use neutral, inclusive language. Avoid leading or emotionally charged phrasing.
    - **Example**:
        - *Unethical*: "Why are older employees slower at adapting to technology?"
        - *Ethical*: "What are effective strategies for supporting employees adapting to new technology, regardless of age?"
2. **Encourage Balanced Responses**
    - **Why It Matters**: Balanced prompts elicit more comprehensive and fair responses.
    - **How to Achieve This**: Ask for multiple perspectives or pros and cons.
    - **Example**:
        - *Prompt*: "Discuss the benefits and challenges of implementing remote work policies in organisations."
3. **Recognise GPT's Limitations**
    - **Why It Matters**: Understanding GPT's strengths and weaknesses is essential for crafting realistic prompts.

    - **How to Achieve This**: Refer to **Section 3.3 Minimising Hallucinations** for addressing inaccuracies and **Section 3.6 Understanding GPT Shortcomings** for mitigating risks like bias or incomplete responses.
4. **Design for Inclusivity**
    - **Why It Matters**: Inclusive prompts ensure outputs respect diverse audiences and avoid exclusion.
    - **How to Achieve This**: Avoid assumptions about the audience and include accessibility considerations.
    - **Example**:
        - *Prompt*: "Explain blockchain technology in a way that is accessible to non-technical readers, using simple examples."
5. **Validate Outputs**
    - **Why It Matters**: GPT's training data may contain biases or inaccuracies.
    - **How to Achieve This**:
        - Cross-check outputs with credible, up-to-date sources.
        - Use GPT as a starting point and refine results with domain expertise.

Examples: Ethical Prompt Engineering

1. Original Prompt: *"Generate a list of jokes about [specific group]."*

Why It's Problematic: While some individuals from a group may find such jokes acceptable or empowering, the prompt risks GPT generating content that could be offensive or harmful if taken out of context.

Ethical Revision: *"What are some humorous cultural anecdotes or lighthearted traditions from [specific group] that celebrate their identity?"*

Why It Works

- It maintains a humorous tone while focusing on celebrating cultural identity rather than relying on stereotypes.
- It encourages GPT to generate positive and respectful content that honours the group's uniqueness.

2. Original Prompt: *"List flaws of specific ethnic groups."*

Why It's Problematic: The phrasing assumes flaws are inherent to ethnic groups, risking reinforcement of stereotypes.

- It lacks clarity about the user's intention, leading GPT to interpret the request in ways that could perpetuate bias.

Ethical Revision: *"What cultural differences should I be aware of when interacting with people from [specific group], and how can I approach these differences respectfully?"*

Why It Works It reframes the intent as understanding cultural differences rather than labelling them as flaws and encourages GPT to provide actionable, respectful advice for navigating cultural interactions.

3. Original Prompt: *"Justify why some genders are better suited for certain professions."*

Why It's Problematic

- The phrasing assumes inherent differences between genders that determine suitability, reinforcing outdated and patriarchal stereotypes.
- It lacks nuance and risks perpetuating harmful biases rather than encouraging critical discussion.

Ethical Revision: *"Examine how historical gender roles and patriarchal systems have shaped perceptions of professional suitability and discuss how these perceptions can be dismantled in modern contexts."*

Why It Works

- **Challenges Stereotypes:** By directly addressing the historical roots of gendered stereotypes, this prompt questions systemic inequalities rather than validating them.
- **Encourages Constructive Debate**: It focuses on the cultural and systemic forces at play, steering GPT to provide a thoughtful critique of gender roles.
- **Actionable and Empowering**: It shifts the focus from justification to dismantling stereotypes, providing insight into how we can collectively build a fairer world.

Pro Tip

- Use the Role → Context → Task → Examples → Constraints structure to guide GPT ethically. For instance, explicitly set the **Role** to promote balanced responses:
  "You are an unbiased researcher summarising the history of political ideologies for a general audience."
- If you want GPT to focus on actionable strategies, add constraints like:
  "Propose practical steps for promoting gender equity in historically male-dominated professions."

Closing Thoughts

Ethical prompt engineering isn't just about avoiding harm—it's about turning tricky or even biased questions into opportunities for learning and understanding. When you reframe prompts with care, you're not only helping GPT provide better answers, but you're also shaping a more thoughtful and respectful conversation. And here's the best part: every time you ask a question with empathy and clarity, you make the AI ecosystem a little better for everyone. Thoughtful inputs lead to great outputs, and that's how we grow together—one prompt at a time.

7.2. Designing for Accessibility

Accessibility isn't just a feature—it's a mindset. Whether you're haring ideas, creating content, or building software, designing with accessibility in mind ensures your work is usable, respectful, and valuable to everyone. This section explores practical ways to integrate accessibility into your workflows, tailored for both general users and technical teams.

Key Takeaways (TL;DR):

- Accessibility is everyone's responsibility, whether you're writing content or building software.
- Simple language, inclusive examples, and awareness of cultural diversity improve accessibility for non-technical audiences.
- GPT can help bridge accessibility gaps by generating inclusive content, accessible code, and actionable recommendations and help with adherence to standards like WCAG 2.
- All team members — Developers, Designers, Quality Engineers, Business Analysts, anyone —must collaborate to embed accessibility throughout the development lifecycle.
- Designing with accessibility in mind benefits everyone by creating better, more usable content and software.

Under the Hood

Key Points for Everyone: Accessibility Made Simple

1. **Use Clear and Simple Language**
    - **Why It Matters:** Ensures your message reaches people with varying literacy levels or cognitive abilities.
    - **How to Achieve This:** Ask GPT to simplify complex ideas or avoid jargon.
    - **Example:** "Explain the concept of recursion to a 12-year-old using simple examples."
2. **Promote Visual and Auditory Accessibility**
    - **Why It Matters:** Ensures content is usable for people who rely on descriptions or alternatives to visual/auditory content.
    - **How to Achieve This:** Ask GPT to generate captions, transcripts, or audio descriptions.
    - **Example:** "Write a detailed description of this infographic for visually impaired users."
3. **Tailor Outputs for Assistive Technology**
    - **Why It Matters:** Makes content compatible with screen readers or other assistive tools.
    - **How to Achieve This:** Reformat outputs using GPT for clarity and navigation.
    - **Example:** "Summarise this article in bullet points and headings for screen reader compatibility."
4. **Address Diverse Perspectives and Cultural Relevance**

- **Why It Matters:** Accessibility isn't just about physical or cognitive abilities— it also means making content relatable and meaningful for people from different cultures and languages.
- **How to Achieve This:** Adapt examples, language, and terminology to reflect the audience's cultural norms and regional context.
- **Examples:**
  - "Rewrite this user manual with examples relevant to audiences in Southeast Asia, considering regional preferences."
  - "Rewrite this recipe for an audience in India, using ingredients commonly available there."

5. **Audit for Accessibility Gaps**
   - **Why It Matters:** Identifies barriers in content or workflows that might exclude users.
   - **How to Achieve This:** Ask GPT to simulate experiences or suggest improvements.
   - **Example:** "Review this webpage and suggest improvements for accessibility, including navigation and colour contrast."

Key Points for Software Teams: Building Accessibility In

1. **Generate Accessible Code Examples**
   - **Why It Matters:** Code needs to support accessibility features like keyboard navigation and screen readers.
   - **How to Achieve This:** Use GPT to create or improve accessible code.
   - **Example: "**Create an accessible dropdown menu in React with keyboard navigation."
2. **Testing for Accessibility in Software**
   - **Why It Matters:** Ensures software meets the needs of users with disabilities.
   - **How to Achieve This:** Ask GPT to draft test cases or recommend tools**.**
   - **Example:** "Generate a set of accessibility test cases for a web app, focussing on keyboard navigation."
3. **Assistive Technology Simulation**
   - **Why It Matters:** Understanding how users interact with assistive tools highlights potential issues.
   - **How to Achieve This:** Simulate experiences with GPT or request recommendations.
   - **Example:** "Simulate how a screen reader would navigate this webpage and suggest fixes."
4. **Design APIs for Accessibility**
   - **Why It Matters:** APIs must enable developers to build accessible applications.
   - **How to Achieve This:** Use GPT to suggest accessibility-related improvements.
   - **Example:** "Review this API schema and suggest changes to include accessibility metadata for images."

5. **Accessibility in UI/UX Design**
    - **Why It Matters:** Interfaces should be both functional and accessible.
    - **How to Achieve This:** Ask GPT to provide best practices or review designs.
    - **Example:** "List best practices for making dropdown menus accessible, including screen reader compatibility."
6. **Accessible Documentation**
    - **Why It Matters:** Technical documentation must be usable by all, including those relying on assistive tools.
    - **How to Achieve This:** Use GPT to format documentation accessibly.
    - **Example:** "Draft an accessible README file with headings and alt text for images."
7. **Collaborate Across Roles**
    - **Why It Matters:** Accessibility requires input from everyone in the team: Developers, Designers, Quality Engineers, and Business Analysts, and others.
    - **How to Achieve This:** Use GPT to facilitate collaboration and shared understanding.
    - **Example:** "Draft an accessibility checklist for use by cross-functional teams during sprint planning."
8. **Accessibility in CI/CD Pipelines**
    - **Why It Matters:** Regular automated checks catch accessibility issues early and consistent.
    - **How to Achieve This:** Use GPT to recommend tools and strategies for your pipeline.
    - **Example:** "List tools for automating accessibility checks in a CI/CD pipeline for a React app."
9. **Applying WCAG 2 Standards**
    - **Why It Matters:** WCAG 2 provides actionable guidelines for digital accessibility.
    - **How to Achieve** This: Use GPT to explain or apply WCAG principles to your work.
    - **Example:** "Generate a login form in HTML with ARIA roles and WCAG 2 Level AA compliance."

Advanced Examples

For Everyone

1. **Adapt Presentation Materials**
   Prompt: "Rewrite this presentation slide deck for an audience with dyslexia, using clear fonts and a structured layout."
    - **Why:** This highlights visual accessibility, addressing the needs of users with dyslexia by making materials easier to read and navigate.
2. **Create Accessible Social Media Posts**
   Prompt: "Generate a checklist for creating inclusive social media posts, focusing on accessibility for visual and auditory impairments."
    - **Why:** Practical advice for social media management ensures posts are inclusive, considering users who rely on captions, alt text, or other accessibility features.

3. **Develop an Accessible Travel Guide**
   Prompt: "Create a travel guide for Italy, including tips for wheelchair users."
   - **Why:** Focuses on providing actionable, inclusive information that caters to the practical needs of travellers with mobility challenges.

For Software Teams

1. **Verify Accessibility in Code**
   Prompt: "Write a unit test to verify that all alt attributes in an HTML file are non-empty and meaningful."
   - **Why:** Combines technical rigor with accessibility, helping developers and Quality Engineers implement robust checks for inclusive designs.
2. **Ensure Multilingual Accessibility**
   Prompt: "Suggest best practices for internationalising a web app to ensure accessibility for multilingual users."
   - **Why:** Expands accessibility to include localisation, ensuring diverse audiences can navigate and use applications effectively.
3. **Draft WCAG 2 Compliance Checklists**
   Prompt: "Draft a checklist for WCAG 2 compliance for a mobile app."
   - **Why:** Provides teams with a clear, actionable roadmap to embed accessibility into mobile app development, ensuring alignment with global standards.

Pro Tips

1. Start small—use GPT to simplify your language, add alt text to images, or create clear headings. Every improvement counts toward inclusivity.
2. Think like your audience: Would this content be usable if I had a different ability, language, or context? If not, adapt it!
3. Embed accessibility checks at every stage—planning, development, and testing. Accessibility isn't an extra step; it's how great software is made.
4. Regularly test your software under real-world conditions, like using a screen reader or navigating solely with a keyboard. Empathy-driven testing builds better products.

Closing Thoughts

Accessibility isn't just a technical requirement—it's a way to make the world a little more inclusive, one step at a time. Whether you're creating content or building software, every thoughtful change you make opens doors for someone who might otherwise be left out. With GPT as your partner, you can simplify the process, generate new ideas, and embed accessibility into everything you do. When accessibility is built in, everyone benefits—and that's the best outcome of all.

7.3. Non-Technical Use Cases

Prompt engineering isn't just for solving technical challenges—it's a tool for everyone. From simplifying communication to organising your day or creating educational materials, GPT can make everyday tasks easier and more impactful. In fact, this very guide is a prime example of how GPT can assist in creating educational resources. By using properly engineered prompts with feedback, GPT and I collaborated to design structured, accessible content that supports learning and exploration.

Key Takeaways (TL;DR):

- GPT can assist with a variety of tasks, including writing, planning, learning, and organising.
- Well-crafted prompts make communication clearer and more engaging.
- GPT is a valuable partner for managing your day and simplifying complex concepts for teaching or learning.

Under the Hood

Key Points

1. **Simplify Complex Topics**
    - **Why It Matters**: Breaking down complex information makes it accessible to a wider audience.
    - **How to Achieve This**: Ask GPT to summarise, rephrase, or explain in simpler terms.
    - **Example**: "Explain how the human circulatory system works in 150 words for a 10-year-old."
2. **Write Professional Emails**
    - **Why It Matters**: Clear and concise communication is essential for professional success.
    - **How to Achieve This**: Use GPT to draft or refine your messages.
    - **Example**: "Draft a polite follow-up email to a colleague about a missed deadline."
3. **Plan Events or Projects**
    - **Why It Matters**: Structured planning ensures that everything runs smoothly.
    - **How to Achieve This**: Ask GPT to create checklists or suggest ideas.
    - **Example**: "Create a checklist for planning a weekend camping trip, including food, gear, and safety tips."
4. **Organise Your Day**
    - **Why It Matters**: Personal organisation improves productivity and reduces stress.
    - **How to Achieve This**: Use GPT to create schedules or prioritise tasks.
    - **Example**: "Plan my day with three hours for work, one hour for exercise, and time for family dinner."

5. **Create Educational Resources**
    - **Why It Matters**: Well-designed learning materials benefit both teachers and students. This guide itself was developed using GPT to draft, refine, and organise its sections, demonstrating how prompt engineering can be used to create comprehensive educational materials.
    - **How to Achieve This**: Ask GPT to develop lesson plans, quizzes, or study guides.
    - **Example**: "Create a multiple-choice quiz for middle school students on the water cycle."
6. **Brainstorm Creative Ideas**
    - **Why It Matters**: GPT can inspire new approaches to challenges or projects.
    - **How to Achieve This**: Use GPT to generate suggestions or explore alternatives.
    - **Example**: "Suggest five unique themes for a community talent show."
7. **Summarise Lengthy Documents**
    - **Why It Matters**: Summaries save time and help convey key points quickly.
    - **How to Achieve This**: Ask GPT to condense information into concise formats.
    - **Example**: "Summarise this 10-page report into five key points for a non-technical audience."
8. **Develop Inclusive Content**
    - **Why It Matters**: Engaging diverse audiences ensures your message resonates widely.
    - **How to Achieve This**: Use GPT to adapt tone, language, or examples.
    - **Example**: "Rewrite this announcement for a global audience, avoiding region-specific jargon."

Advanced Examples

1. *Prompt*: "Plan a study schedule for a college student preparing for final exams, balancing study time and breaks."
    - **Why It Works**: Balances productivity and well-being with actionable advice.
2. *Prompt*: "Draft a short script for a video tutorial explaining how to recycle household waste."
    - **Why It Works**: Provides clarity and structure for creating effective educational content.
3. *Prompt*: "Suggest five ways to make a personal budget more accessible and easy to follow for someone new to finance."
    - **Why It Works**: Combines practical advice with an inclusive approach to personal organisation.
4. *Prompt*: "Collaboratively draft a structured educational guide on prompt engineering, focusing on practical examples and actionable strategies."
    - **Why It Works**: This mirrors the process used to create this guide, demonstrating how GPT can help with educational resource development.

Pro Tips

1. **Ask for Specific Outputs**: The clearer your goal, the better GPT can tailor its response. For example, specify the tone, audience, or length of the output.
2. **Iterate for Improvement**: If the first result doesn't meet your needs, refine your prompt and try again. Iteration ensures quality results.
3. **Adapt for Your Context**: Tailor prompts to your personal or professional needs to maximise relevance and usability.
4. **Use with Caution for Complex Transcripts**: GhatGPT isn't particularly good when it comes to analysing transcripts with several participants discussing complex topics that touch multiple areas.
    - Guide ChatGPT to the areas that you know are of importance.
    - Paste relevant parts of the transcript and ask to refine content based on it
    - Ask ChatGPT to review the generated outcome and suggest improvements (see section 3.5. Creating Prompts with Built-In Feedback).
    - Always, always do the final edit and correct all the inconsistencies before publishing the final version.

Closing Thoughts

Non-technical use cases highlight GPT's versatility in simplifying daily tasks, enhancing communication, and fostering creativity. Whether you're planning your day, teaching a concept, or organising a project, GPT can help you do it better. This guide itself is a testament to GPT's potential, turning everyday challenges into opportunities for growth, clarity, and success. Now it's your turn to explore and maximise its possibilities applying best prompt engineering practices.

Message to the Reader

Congratulations, you've made it to the end of this guide! 🎉 Creating it was a journey of collaboration between a curious human and a tireless AI, proving that a little teamwork—powered by well-crafted prompts—can go a long way. From the foundational principles to the advanced tricks, this guide represents the result of refining, experimenting, and, yes, even learning from a few hiccups along the way.

Now it's your turn. Use what you've learned to craft amazing prompts, solve tricky problems, and explore entirely new possibilities. Remember, every great interaction with AI starts with a thoughtful question. Stay curious, keep experimenting, and let your prompts open the door to creativity, insight, and a bit of fun. Happy prompting!

References

1. **Getting Started with ChatGPT**
   ChatGPT Overview
   An introduction to ChatGPT, its features, and how to use it effectively.
2. **Prompt Engineering Best Practices**
   OpenAI Help Center - Prompt Engineering
   A guide on crafting effective prompts to achieve high-quality responses from ChatGPT.
3. **ChatGPT Capabilities Overview**
   OpenAI Help Center - ChatGPT Capabilities
   An overview of ChatGPT's functionalities and how to leverage them.
4. **OpenAI Platform Documentation**
   OpenAI Platform - ChatGPT Models
   Technical documentation on ChatGPT models for developers and advanced users.

These resources should provide a solid foundation for both beginners and intermediate users looking to enhance their understanding and application of ChatGPT and prompt engineering.