**1**

**00:00:00,600 --> 00:00:04,733**

**Welcome to Module 1: Introduction to Prompt Engineering**

for Developers.

**2**

**00:00:04,983 --> 00:00:11,357**

In this module, we explore the foundational concepts

and practical applications of prompt engineering,

**3**

**00:00:11,357 --> 00:00:14,689**

a critical skill for developers working with AI systems.

**4**

**00:00:14,839 --> 00:00:21,271**

Prompt engineering is the process of designing and

**refining inputs to guide AI models, such as GPT-4,**

**5**

**00:00:21,271 --> 00:00:24,916**

to generate accurate, relevant, and optimized outputs.

**6**

**00:00:25,066 --> 00:00:31,393**

This skill is indispensable in software development

as it allows developers to leverage AI models effectively

**7**

**00:00:31,393 --> 00:00:35,004**

for tasks like code generation, debugging, testing,

and more.

**8**

**00:00:35,154 --> 00:00:41,830**

By mastering prompt engineering, developers can improve

the efficiency of their workflows, enhance the quality

**9**

**00:00:41,830 --> 00:00:47,937**

of their outputs, and unlock innovative possibilities

in building AI-driven applications across various

**10**

**00:00:47,937 --> 00:00:48,808**

domains.

**11**

**00:00:49,058 --> 00:00:54,747**

This module introduces you to the core principles

of prompt engineering, starting with understanding

**12**

**00:00:54,747 --> 00:00:57,742**

the structure and components of a well-crafted prompt.

**13**

**00:00:57,892 --> 00:01:04,115**

You will learn how to design inputs that include

clear intent, relevant context, examples, and constraints

**14**

**00:01:04,115 --> 00:01:08,016**

to guide AI responses precisely toward your desired

goals.

**15**

**00:01:08,166 --> 00:01:14,099**

Additionally, this module covers the basics of API

interactions, demonstrating how to integrate AI systems

**16**

**00:01:14,099 --> 00:01:18,382**

into applications using programming languages like

Python and JavaScript.

**17**

**00:01:18,532 --> 00:01:24,465**

You will also explore practical applications, such

as crafting prompts for coding tasks, automating testing,

**18**

**00:01:24,465 --> 00:01:25,824**

and debugging software.

**19**

**00:01:25,974 --> 00:01:32,116**

Case studies and examples will illustrate the best

practices and common pitfalls to avoid, ensuring you

**20**

**00:01:32,116 --> 00:01:35,169**

gain a deep understanding of effective prompt design.

**21**

**00:01:35,419 --> 00:01:42,431**

Through hands-on exercises, you will set up your development

environment, interact with APIs, and iteratively refine

**22**

**00:01:42,431 --> 00:01:44,567**

prompts to optimize AI outputs.

**23**

**00:01:44,717 --> 00:01:50,000**

By the end of this module, you will have a robust

understanding of how to design effective prompts and

**24**

**00:01:50,000 --> 00:01:56,885**

integrate them seamlessly into your development processes,

enabling you to harness the full potential of AI models.

**25**

**00:01:57,035 --> 00:02:02,050**

Let's embark on this exciting journey to master the

art and science of prompt engineering.

**26**

**00:02:03,850 --> 00:02:09,886**

Prompt Engineering is crucial for developers as it

equips developers with the skills to interact effectively

**27**

**00:02:09,886 --> 00:02:12,812**

with AI models, optimizing their responses.

**28**

**00:02:12,962 --> 00:02:19,417**

By understanding prompt structures and API interactions,

developers can enhance the accuracy of AI-generated

**29**

**00:02:19,417 --> 00:02:23,319**

solutions for tasks like code completion, debugging,

and testing.

**30**

**00:02:23,469 --> 00:02:29,273**

This knowledge enables developing more efficient,

AI-driven applications across various industries.

**31**

**00:02:29,423 --> 00:02:36,064**

Furthermore, it fosters innovation by allowing developers

to leverage AI in more practical, impactful ways,

**32**

**00:02:36,064 --> 00:02:41,346**

ultimately improving productivity and addressing complex

software challenges through better prompt design and

**33**

**00:02:41,346 --> 00:02:42,287**

execution.

**34**

**00:02:42,537 --> 00:02:48,412**

This module provides an overview of prompt engineering

and its significance in software development, highlighting

**35**

**00:02:48,412 --> 00:02:51,732**

how it helps developers effectively interact with

AI models.

**36**

**00:02:51,882 --> 00:02:57,176**

It begins with the definition and importance of prompt

engineering, emphasizing its role in improving the

**37**

**00:02:57,176 --> 00:03:02,099**

accuracy of AI responses in various projects, including

code generation and debugging.

**38**

**00:03:02,349 --> 00:03:09,768**

The module delves into the basics of API interaction,

focusing on how to make API requests and handle responses

**39**

**00:03:09,768 --> 00:03:13,727**

in popular programming languages like Python and JavaScript.

**40**

**00:03:13,877 --> 00:03:19,496**

It also guides developers through setting up and

using OpenAI's API via SDKs and libraries.

**41**

**00:03:19,746 --> 00:03:25,737**

Prompt structures are explored in detail, and instructions

on constructing basic prompts tailored to specific

**42**

**00:03:25,737 --> 00:03:29,638**

tasks like code completion, testing, and debugging

are provided.

**43**

**00:03:29,788 --> 00:03:34,200**

Developers learn to craft well-structured prompts

that maximize AI output quality.

**44**

**00:03:34,450 --> 00:03:40,812**

The module includes case studies showcasing real-world

applications of prompt engineering and highlights

**45**

**00:03:40,812 --> 00:03:43,831**

best practices and common pitfalls to avoid.

**46**

**00:03:44,081 --> 00:03:49,828**

Lastly, a hands-on exercise guides learners through

setting up a development environment and making basic

**47**

**00:03:49,828 --> 00:03:55,365**

API calls, providing code examples to reinforce key

concepts and enhance practical skills.

**48**

**00:03:55,615 --> 00:03:58,076**

**1.1 Overview of Prompt Engineering**

**49**

**00:03:58,327 --> 00:04:04,120**

Prompt engineering is crafting effective inputs or

prompts to interact with AI models to obtain accurate

**50**

**00:04:04,120 --> 00:04:05,525**

and relevant outputs.

**51**

**00:04:05,675 --> 00:04:11,712**

Software development involves designing specific,

structured queries or instructions that guide AI models,

**52**

**00:04:11,712 --> 00:04:16,728**

such as GPT, to assist in coding, debugging, testing,

and other development tasks.

**53**

**00:04:16,878 --> 00:04:22,915**

This process allows developers to control the AI's

behavior, ensuring that it generates useful and efficient

**54**

**00:04:22,915 --> 00:04:23,704**

outputs.

**55**

**00:04:23,854 --> 00:04:30,170**

By fine-tuning the prompts, developers can optimize

how AI responds, improving the performance and functionality

**56**

**00:04:30,170 --> 00:04:31,888**

of AI-driven solutions.

**57**

**00:04:32,138 --> 00:04:36,875**

**1.1.1 Definition, Importance of Prompt Engineering**

in Software Development

**58**

**00:04:37,125 --> 00:04:43,499**

At its core, prompt engineering is the practice of

designing input queries for AI systems in a way that

**59**

**00:04:43,499 --> 00:04:46,413**

directs them toward producing the desired outcome.

**60**

**00:04:46,563 --> 00:04:52,705**

**AI models like GPT-4 are trained on vast amounts**

of data, which means their responses can vary based

**61**

**00:04:52,705 --> 00:04:54,179**

on how a prompt is structured.

**62**

**00:04:54,329 --> 00:04:58,788**

An effective prompt will be clear, concise, and designed

with the target outcome in mind.

**63**

**00:04:58,938 --> 00:05:05,184**

For example, when using an AI model to generate code,

a poorly designed prompt may result in irrelevant

**64**

**00:05:05,184 --> 00:05:07,111**

or overly complex code.

**65**

**00:05:07,261 --> 00:05:11,766**

In contrast, a well-crafted prompt will yield accurate

and efficient solutions.

**66**

**00:05:12,016 --> 00:05:16,567**

Prompt engineering can be likened to giving precise

instructions to a skilled assistant.

**67**

**00:05:16,717 --> 00:05:21,129**

The more specific and well-structured the instructions,

the better the results.

**68**

**00:05:21,279 --> 00:05:27,188**

This is particularly important in software development,

where outputs must be predictable, reliable, and optimized

**69**

**00:05:27,188 --> 00:05:28,245**

for performance.

**70**

**00:05:30,045 --> 00:05:31,206**

Elements of Prompt

**71**

**00:05:31,455 --> 00:05:36,993**

Prompts used for instructing or interacting with AI

models typically consist of several key elements to

**72**

**00:05:36,993 --> 00:05:40,976**

communicate the desired input effectively, as shown

in the following figure.

**73**

**00:05:41,126 --> 00:05:42,705**

These elements include:

**74**

**00:05:43,036 --> 00:05:48,713**

* · Context: Providing relevant background information

ensures the AI understands the prompt's purpose.

**75**

**00:05:48,863 --> 00:05:53,554**

This could involve details about the topic or referencing

prior statements or questions.

**76**

**00:05:53,874 --> 00:05:57,101**

* · Intent: Clearly state what you want the AI to do.

**77**

**00:05:57,251 --> 00:06:03,462**

Be specific about the task or type of response, whether

summarizing text, answering a factual question, or

**78**

**00:06:03,462 --> 00:06:04,728**

generating content.

**79**

**00:06:05,071 --> 00:06:11,363**

* · Examples: Including examples helps clarify your

intent and offers reference points for the AI, such

**80**

**00:06:11,363 --> 00:06:14,962**

as sample sentences or similar queries related to

the task.

**81**

**00:06:15,293 --> 00:06:21,017**

* · Format: Specify how you want the response structured,

whether in sentence form, paragraph format, or with

**82**

**00:06:21,017 --> 00:06:23,432**

a particular tone (formal or informal).

**83**

**00:06:23,763 --> 00:06:28,280**

* · Constraints: Set limitations to ensure the response

meets your requirements.

**84**

**00:06:28,430 --> 00:06:33,399**

This could include restricting length, avoiding specific

content, or emphasizing certain points.

**85**

**00:06:33,719 --> 00:06:40,418**

* · Clarifications: For complex prompts, including instructions

for the AI to request clarification can lead to more

**86**

**00:06:40,418 --> 00:06:41,683**

accurate outputs.

**87**

**00:06:42,223 --> 00:06:47,703**

* · User Input: Interactive systems often use prompts

containing user queries to provide context for the

**88**

**00:06:47,703 --> 00:06:48,388**

AI.

**89**

**00:06:48,708 --> 00:06:55,186**

* · Feedback Loop: Include mechanisms to evaluate responses,

allowing improvements over time through model refinement.

**90**

**00:06:55,529 --> 00:07:01,717**

* · Testing and Iteration: Fine-tuning the prompt through

testing and experimentation is essential for optimizing

**91**

**00:07:01,717 --> 00:07:02,913**

AI performance.

**92**

**00:07:05,514 --> 00:07:08,811**

Importance of Prompt Engineering in Software Development

**93**

**00:07:09,061 --> 00:07:14,169**

In software development, prompt engineering plays

a pivotal role for several reasons:

**94**

**00:07:14,489 --> 00:07:16,555**

* · Enhancing Developer Productivity

**95**

**00:07:16,805 --> 00:07:22,947**

Prompt engineering enables developers to leverage

AI models for routine or repetitive tasks like code

**96**

**00:07:22,947 --> 00:07:26,558**

generation, documentation, debugging, and testing.

**97**

**00:07:26,708 --> 00:07:32,141**

By crafting the right prompts, developers can significantly

reduce the time required for these tasks.

**98**

**00:07:32,291 --> 00:07:38,340**

Instead of writing large chunks of boilerplate code,

a developer can instruct the AI to do so using a simple

**99**

**00:07:38,340 --> 00:07:42,415**

prompt, freeing them up to focus on more critical

aspects of the project.

**100**

**00:07:43,065 --> 00:07:43,622**

**2.**

**101**

**00:07:43,772 --> 00:07:45,305**

Improving Code Quality

**102**

**00:07:45,555 --> 00:07:51,685**

AI models can help generate high-quality, clean code

when appropriately given structured prompts.

**103**

**00:07:51,835 --> 00:07:57,953**

They can also assist in providing optimal solutions,

avoiding common errors, and following best practices

**104**

**00:07:57,953 --> 00:07:58,708**

in coding.

**105**

**00:07:58,858 --> 00:08:05,418**

Developers can prompt the AI to suggest improvements

or refactor existing code, ensuring the codebase remains

**106**

**00:08:05,418 --> 00:08:06,939**

efficient and maintainable.

**107**

**00:08:07,189 --> 00:08:12,158**

The AI can analyze the function and provide suggestions

to enhance its efficiency.

**108**

**00:08:12,308 --> 00:08:18,519**

It might identify opportunities to reduce time complexity,

eliminate redundant operations, or suggest alternative

**109**

**00:08:18,519 --> 00:08:19,413**

algorithms.

**110**

**00:08:20,063 --> 00:08:20,713**

**3.**

**111**

**00:08:20,863 --> 00:08:23,557**

Accelerating Debugging and Error Detection

**112**

**00:08:23,807 --> 00:08:27,893**

Debugging is one of the most time-consuming aspects

of software development.

**113**

**00:08:28,043 --> 00:08:34,510**

Prompt engineering can help diagnose issues more

quickly by directing AI models to analyze and debug

**114**

**00:08:34,510 --> 00:08:35,567**

code snippets.

**115**

**00:08:35,717 --> 00:08:41,104**

Instead of manually sifting through hundreds of lines

of code, developers can use prompts to pinpoint the

**116**

**00:08:41,104 --> 00:08:42,218**

issue faster.

**117**

**00:08:42,868 --> 00:08:43,565**

**4.**

**118**

**00:08:43,715 --> 00:08:46,223**

Streamlining Testing and Quality Assurance

**119**

**00:08:46,473 --> 00:08:52,115**

Prompt engineering can also streamline the process

of generating test cases, running simulations, and

**120**

**00:08:52,115 --> 00:08:53,764**

conducting performance tests.

**121**

**00:08:53,914 --> 00:08:59,800**

By crafting detailed prompts, developers can ensure

that AI generates robust test cases covering edge

**122**

**00:08:59,800 --> 00:09:03,712**

conditions, exceptions, and other scenarios that might

be overlooked.

**123**

**00:09:04,362 --> 00:09:06,080**

Required Unittest Cases

**124**

**00:09:06,331 --> 00:09:06,888**

**1.**

**125**

**00:09:07,038 --> 00:09:08,478**

Test successful login.

**126**

**00:09:08,809 --> 00:09:14,533**

* · Test if valid credentials (username and password)

return the message "Login successful."

**127**

**00:09:14,783 --> 00:09:15,340**

**2.**

**128**

**00:09:15,490 --> 00:09:17,626**

Test login with invalid credentials.

**129**

**00:09:17,969 --> 00:09:23,170**

* · Test if an incorrect username or password returns

the message "Invalid credentials."

**130**

**00:09:23,420 --> 00:09:24,070**

**3.**

**131**

**00:09:24,220 --> 00:09:26,960**

Test password change with the correct old password.

**132**

**00:09:27,361 --> 00:09:32,040**

* · Ensure the password can be updated successfully

if the correct old password is provided.

**133**

**00:09:32,290 --> 00:09:32,987**

**4.**

**134**

**00:09:33,137 --> 00:09:35,830**

Test password change with incorrect old password.

**135**

**00:09:37,949 --> 00:09:43,139**

* · Ensure the system rejects the password change if

the provided old password is incorrect.

**136**

**00:09:43,389 --> 00:09:44,132**

**5.**

**137**

**00:09:44,282 --> 00:09:50,923**

Test edge cases (empty or null input).

**138**

**00:09:51,301 --> 00:09:56,514**

* · Test scenarios where the username or password provided

is an empty string or None.

**139**

**00:09:58,314 --> 00:09:59,428**

Explanation:

**140**

**00:09:59,782 --> 00:10:06,969**

* · Setup Method (setUp): Initializes the UserAuth instance

with a mock user dictionary before each test.

**141**

**00:10:07,347 --> 00:10:14,034**

* · Test Methods: Each method covers different scenarios

such as successful login, failed login, successful

**142**

**00:10:14,034 --> 00:10:18,690**

password change, failed password change, and edge

cases like empty input.

**143**

**00:10:19,010 --> 00:10:26,080**

* · Assertions: Use self.assertEqual() to compare the

expected output with the actual result for each test

**144**

**00:10:26,080 --> 00:10:26,695**

case.

**145**

**00:10:26,945 --> 00:10:33,075**

This example tests key functionalities and edge cases

to ensure robustness in the UserAuth class.

**146**

**00:10:33,325 --> 00:10:39,211**

This prompt will prompt the AI to create test cases

that validate the functionality of the user authentication

**147**

**00:10:39,211 --> 00:10:42,567**

class, ensuring that the system behaves as expected.

**148**

**00:10:42,817 --> 00:10:43,560**

**5.**

**149**

**00:10:43,710 --> 00:10:46,682**

Simplifying Documentation and Communication

**150**

**00:10:46,932 --> 00:10:51,297**

Documentation is a crucial but often overlooked part

of software development.

**151**

**00:10:51,447 --> 00:10:57,774**

Using prompt engineering, developers can quickly

generate technical documentation, comments, and explanations

**152**

**00:10:57,774 --> 00:10:59,017**

for complex code.

**153**

**00:10:59,167 --> 00:11:05,019**

This ensures that the project remains well-documented,

easing future maintenance and collaboration.

**154**

**00:11:05,669 --> 00:11:11,230**

Now Let's break down and analyze architecture of a

Prompt Engineering System with connections to various

**155**

**00:11:11,230 --> 00:11:17,232**

components such as a User Interface (UI), Retrieval

Systems, Data API, and Storage in detail.

**156**

**00:11:17,882 --> 00:11:18,439**

**1.**

**157**

**00:11:18,589 --> 00:11:20,215**

User Interface (UI)

**158**

**00:11:20,546 --> 00:11:25,248**

* · Function: The UI is the interaction layer between

the end-user and the system.

**159**

**00:11:25,398 --> 00:11:30,506**

It allows the user to input queries or prompts and

displays the system's output or response.

**160**

**00:11:30,826 --> 00:11:37,246**

* · Connection: The UI is directly linked to the Prompt

Engine, which processes user input and generates responses

**161**

**00:11:37,246 --> 00:11:39,952**

based on the underlying models and data sources.

**162**

**00:11:40,202 --> 00:11:40,759**

**2.**

**163**

**00:11:40,909 --> 00:11:41,931**

Prompt Engine

**164**

**00:11:42,274 --> 00:11:46,360**

* · Core Component: The Prompt Engine is this system's

core processing unit.

**165**

**00:11:46,510 --> 00:11:51,897**

It interprets the input provided through the UI and

generates appropriate responses or instructions.

**166**

**00:11:52,228 --> 00:11:53,262**

· Interaction:

**167**

**00:11:53,512 --> 00:11:59,538**

o User Interface: The Prompt Engine receives inputs

(prompts) from the user through the UI and provides

**168**

**00:11:59,538 --> 00:12:01,917**

outputs (responses) back to the UI.

**169**

**00:12:02,167 --> 00:12:08,889**

o Data API: The Prompt Engine communicates with the

Data API to retrieve or update data required for generating

**170**

**00:12:08,889 --> 00:12:09,783**

responses.

**171**

**00:12:10,033 --> 00:12:15,919**

o Retrieval System: The Prompt Engine can also interact

with Retrieval Systems to fetch relevant information

**172**

**00:12:15,919 --> 00:12:17,742**

that helps construct responses.

**173**

**00:12:17,892 --> 00:12:22,444**

This is particularly useful when prompts require

external knowledge or reference data.

**174**

**00:12:22,694 --> 00:12:23,344**

**3.**

**175**

**00:12:23,494 --> 00:12:24,748**

Retrieval System

**176**

**00:12:25,393 --> 00:12:32,068**

* · Function: The Retrieval Systems serve as data retrieval

mechanisms, responsible for fetching external or internal

**177**

**00:12:32,068 --> 00:12:34,100**

data relevant to the user's query.

**178**

**00:12:34,250 --> 00:12:40,519**

These systems may query databases, search engines,

or external APIs to bring in relevant data.

**179**

**00:12:40,850 --> 00:12:41,884**

· Interaction:

**180**

**00:12:42,134 --> 00:12:47,939**

o The Prompt Engine communicates with one or more

Retrieval Systems to fetch the necessary information

**181**

**00:12:47,939 --> 00:12:49,471**

for constructing the output.

**182**

**00:12:49,721 --> 00:12:56,501**

o There are two separate Retrieval Systems, possibly

indicating redundancy or specialization in retrieving

**183**

**00:12:56,501 --> 00:13:02,632**

different data types (e.g., one for internal data

and the other for external or web data).

**184**

**00:13:02,882 --> 00:13:03,578**

**4.**

**185**

**00:13:03,728 --> 00:13:04,843**

Data API

**186**

**00:13:05,290 --> 00:13:11,687**

* · Data Access Layer: The Data API provides a standardized

interface for the Prompt Engine to interact with the

**187**

**00:13:11,687 --> 00:13:12,477**

Storage.

**188**

**00:13:12,627 --> 00:13:18,362**

It allows for structured access to data, ensuring

the Prompt Engine can query or update the stored data

**189**

**00:13:18,362 --> 00:13:19,175**

efficiently.

**190**

**00:13:19,506 --> 00:13:20,539**

· Interaction:

**191**

**00:13:20,789 --> 00:13:28,080**

o The Prompt Engine relies on the Data API to interact

with Storage, allowing it to access historical data,

**192**

**00:13:28,080 --> 00:13:31,145**

cached responses, or structured datasets.

**193**

**00:13:31,395 --> 00:13:37,676**

o The Data API also communicates with the Retrieval

System, ensuring data consistency and facilitating

**194**

**00:13:37,676 --> 00:13:40,451**

data exchanges between the system components.

**195**

**00:13:40,701 --> 00:13:41,444**

**5.**

**196**

**00:13:41,594 --> 00:13:42,430**

Storage

**197**

**00:13:42,761 --> 00:13:50,285**

* · Purpose: The Storage component is the central repository

that stores all necessary data, including user queries,

**198**

**00:13:50,285 --> 00:13:55,405**

responses, historical data, and possibly the prompt

templates used by the system.

**199**

**00:13:55,736 --> 00:13:56,769**

· Interaction:

**200**

**00:13:57,019 --> 00:14:03,660**

o The Data API facilitates access to Storage, allowing

the system to read from and write to the storage layer.

**201**

**00:14:03,910 --> 00:14:09,297**

o Data that needs to be persisted, such as logs or

past responses, is likely stored here.

**202**

**00:14:09,547 --> 00:14:15,770**

o The Retrieval System may also pull data from Storage

to assist in crafting the response requested by the

**203**

**00:14:15,770 --> 00:14:16,745**

Prompt Engine.

**204**

**00:14:16,995 --> 00:14:18,389**

Overall Flow:

**205**

**00:14:18,639 --> 00:14:19,196**

**1.**

**206**

**00:14:19,346 --> 00:14:23,804**

User Interaction: The user inputs a query or command

through the UI.

**207**

**00:14:24,054 --> 00:14:24,611**

**2.**

**208**

**00:14:24,761 --> 00:14:31,425**

Processing by Prompt Engine: The Prompt Engine receives

the input, interprets it, and determines how to process

**209**

**00:14:31,425 --> 00:14:31,913**

it.

**210**

**00:14:32,163 --> 00:14:32,813**

**3.**

**211**

**00:14:32,963 --> 00:14:34,078**

Data Retrieval:

**212**

**00:14:34,398 --> 00:14:40,516**

* · The Prompt Engine communicates with one or more

Retrieval Systems if it needs external data to complete

**213**

**00:14:40,516 --> 00:14:41,340**

the task.

**214**

**00:14:41,671 --> 00:14:46,188**

* · It can also interact with the Data API to access

or update the central Storage.

**215**

**00:14:46,438 --> 00:14:47,134**

**4.**

**216**

**00:14:47,284 --> 00:14:52,439**

Response Generation: The Prompt Engine generates

an appropriate response based on the data retrieved

**217**

**00:14:52,439 --> 00:14:53,786**

and the internal logic.

**218**

**00:14:54,036 --> 00:14:54,779**

**5.**

**219**

**00:14:54,929 --> 00:15:00,037**

Output: The Prompt Engine returns the response to

the UI, displaying it to the user.

**220**

**00:15:01,838 --> 00:15:07,085**

**1.1.2 Practical Applications of Prompt Engineering**

in Various AI Projects

**221**

**00:15:07,335 --> 00:15:14,324**

Prompt engineering is a crucial aspect of AI development

that involves crafting input prompts to guide AI models

**222**

**00:15:14,324 --> 00:15:16,391**

in generating desired outputs.

**223**

**00:15:16,541 --> 00:15:23,588**

As AI systems like large language models (LLMs) become

more integral to various fields, prompt engineering

**224**

**00:15:23,588 --> 00:15:26,897**

enables developers to harness their potential effectively.

**225**

**00:15:27,047 --> 00:15:32,933**

From generating code to content creation, debugging,

decision-making, and more, well-crafted prompts help

**226**

**00:15:32,933 --> 00:15:37,357**

shape the quality, accuracy, and relevance of AI-generated

responses.

**227**

**00:15:37,607 --> 00:15:44,085**

Let's explore the practical applications of prompt

engineering in various AI projects supported by real-world

**228**

**00:15:44,085 --> 00:15:44,991**

examples.

**229**

**00:15:45,241 --> 00:15:45,798**

**1.**

**230**

**00:15:45,948 --> 00:15:47,852**

Code Generation and Completion

**231**

**00:15:48,102 --> 00:15:53,164**

One of the most common applications of prompt engineering

in AI projects is code generation.

**232**

**00:15:53,314 --> 00:15:57,494**

Developers use AI models to write, complete, and

refactor code snippets.

**233**

**00:15:57,644 --> 00:16:03,693**

With the right prompts, the AI can assist in accelerating

development tasks, reducing errors, and automating

**234**

**00:16:03,693 --> 00:16:05,353**

repetitive coding functions.

**235**

**00:16:05,603 --> 00:16:08,250**

Example: Auto-generating Code Snippets

**236**

**00:16:08,500 --> 00:16:13,713**

Consider a developer working on a Python project that

requires a function to sort a list of dictionaries

**237**

**00:16:13,713 --> 00:16:15,141**

based on a specific key.

**238**

**00:16:15,291 --> 00:16:21,049**

With prompt engineering, the developer can instruct

the AI model with a specific request, such as:

**239**

**00:16:21,299 --> 00:16:26,175**

Prompt: Write a Python function that sorts a list

of dictionaries by a specific key.

**240**

**00:16:26,425 --> 00:16:27,865**

Response from the AI:

**241**

**00:16:28,115 --> 00:16:31,412**

This prompt-driven interaction simplifies coding tasks.

**242**

**00:16:31,562 --> 00:16:37,576**

If the developer needs further customization, they

can tweak the prompt to generate more specific results,

**243**

**00:16:37,576 --> 00:16:41,222**

such as sorting in reverse order or handling missing

keys.

**244**

**00:16:41,472 --> 00:16:47,788**

Impact: By guiding the AI with clear and specific

prompts, developers can generate useful code snippets

**245**

**00:16:47,788 --> 00:16:51,178**

quickly, thus saving time and reducing coding errors.

**246**

**00:16:51,828 --> 00:16:52,385**

**2.**

**247**

**00:16:52,535 --> 00:16:54,114**

Debugging and Code Review

**248**

**00:16:54,364 --> 00:16:59,426**

Another significant application of prompt engineering

is in automated debugging and code review.

**249**

**00:16:59,576 --> 00:17:05,892**

AI models can identify errors, suggest improvements,

and recommend optimizations based on input prompts.

**250**

**00:17:06,042 --> 00:17:12,404**

By framing the issue accurately in a prompt, developers

can utilize AI to streamline the debugging process.

**251**

**00:17:12,654 --> 00:17:14,744**

Example: Finding a Bug in Code

**252**

**00:17:14,994 --> 00:17:19,405**

Imagine a developer facing a bug where a function

isn't returning the expected output.

**253**

**00:17:19,555 --> 00:17:22,063**

The developer can describe the issue in a prompt:

**254**

**00:17:22,313 --> 00:17:27,607**

Prompt: There is a bug in this Python code where the

total sum isn't calculated correctly.

**255**

**00:17:27,757 --> 00:17:29,243**

Can you identify the problem?

**256**

**00:17:29,493 --> 00:17:30,933**

Response from the AI:

**257**

**00:17:31,183 --> 00:17:38,288**

The issue is that the loop is iterating until len(numbers) -

**1, which omits the last element in the list.**

**258**

**00:17:38,438 --> 00:17:42,572**

**You should change range(len(numbers) - 1) to range(len(numbers)).**

**259**

**00:17:42,822 --> 00:17:48,464**

This feedback allows developers to pinpoint issues

quickly, enhance the code, and eliminate bottlenecks

**260**

**00:17:48,464 --> 00:17:49,973**

in the debugging process.

**261**

**00:17:50,223 --> 00:17:56,667**

Impact: Prompt-driven debugging helps to minimize

time spent on identifying issues, ensures more accurate

**262**

**00:17:56,667 --> 00:18:00,440**

code, and enhances productivity in complex development

projects.

**263**

**00:18:01,490 --> 00:18:02,140**

**3.**

**264**

**00:18:02,290 --> 00:18:05,773**

Natural Language Processing (NLP) Applications

**265**

**00:18:06,023 --> 00:18:13,291**

In NLP applications, prompt engineering helps developers

create content generation, summarization, translation,

**266**

**00:18:13,291 --> 00:18:15,172**

and text analysis systems.

**267**

**00:18:15,322 --> 00:18:22,613**

By crafting clear prompts, AI models can be fine-tuned

to generate coherent and contextually accurate responses.

**268**

**00:18:22,863 --> 00:18:24,906**

Example: Text Summarization

**269**

**00:18:25,156 --> 00:18:29,522**

An AI-powered summarization tool can be developed

using prompt engineering.

**270**

**00:18:29,672 --> 00:18:35,849**

If a developer is working on a project that requires

summarizing lengthy legal documents, the AI can be

**271**

**00:18:35,849 --> 00:18:39,238**

guided through a well-constructed prompt to condense

the information.

**272**

**00:18:39,488 --> 00:18:43,250**

Prompt: Summarize the following legal contract in

two sentences.

**273**

**00:18:43,500 --> 00:18:49,769**

Response from the AI: This contract establishes the

terms of a business agreement between Party A and

**274**

**00:18:49,769 --> 00:18:50,605**

Party B.

**275**

**00:18:50,755 --> 00:18:55,539**

It outlines responsibilities, payment terms, and

conditions for termination of the agreement.

**276**

**00:18:55,789 --> 00:19:02,279**

Impact: Prompt engineering in NLP allows developers

to generate summaries, translations, and analyses

**277**

**00:19:02,279 --> 00:19:04,891**

tailored to the context of the input text.

**278**

**00:19:05,041 --> 00:19:09,917**

This is particularly useful in legal tech, customer

service, and content creation.

**279**

**00:19:10,167 --> 00:19:10,864**

**4.**

**280**

**00:19:11,014 --> 00:19:12,685**

AI-Driven Decision-Making

**281**

**00:19:12,935 --> 00:19:18,961**

Prompt engineering can also support decision-making

systems by providing AI models with inputs that simulate

**282**

**00:19:18,961 --> 00:19:22,781**

various scenarios, analyze data, and generate recommendations.

**283**

**00:19:23,031 --> 00:19:25,632**

Example: Recommending Business Strategies

**284**

**00:19:25,881 --> 00:19:30,386**

Imagine an AI system designed to assist businesses

with strategic planning.

**285**

**00:19:30,536 --> 00:19:35,737**

By feeding the AI structured prompts, developers

can enable it to offer insights and recommendations

**286**

**00:19:35,737 --> 00:19:37,223**

based on provided data.

**287**

**00:19:37,473 --> 00:19:43,023**

Prompt: Based on the following quarterly sales data,

recommend two strategies to increase sales in the

**288**

**00:19:43,023 --> 00:19:43,836**

next quarter.

**289**

**00:19:45,636 --> 00:19:46,518**

Bash code

**290**

**00:19:46,768 --> 00:19:49,740**

**Quarter 1: $1,200,000**

**291**

**00:19:49,990 --> 00:19:53,194**

**Quarter 2: $1,100,000**

**292**

**00:19:53,444 --> 00:19:55,999**

**Quarter 3: $900,000**

**293**

**00:19:56,249 --> 00:19:59,267**

**Quarter 4: $850,000**

**294**

**00:19:59,517 --> 00:20:00,957**

Response from the AI:

**295**

**00:20:01,207 --> 00:20:01,764**

**1.**

**296**

**00:20:01,914 --> 00:20:05,676**

Introduce a customer loyalty program to encourage

repeat purchases.

**297**

**00:20:05,926 --> 00:20:06,483**

**2.**

**298**

**00:20:06,633 --> 00:20:11,916**

Increase digital marketing efforts targeting younger

demographics, as their engagement has declined based

**299**

**00:20:11,916 --> 00:20:12,949**

on current trends.

**300**

**00:20:13,199 --> 00:20:19,643**

Impact: In decision-making projects, prompt engineering

allows AI systems to deliver tailored, data-driven

**301**

**00:20:19,643 --> 00:20:24,019**

recommendations that help businesses optimize their

strategies and operations.

**302**

**00:20:24,269 --> 00:20:25,012**

**5.**

**303**

**00:20:25,162 --> 00:20:27,484**

Content Creation and Personalization

**304**

**00:20:27,734 --> 00:20:34,630**

Based on prompt inputs, AI models can generate creative

content like articles, blogs, emails, and social media

**305**

**00:20:34,630 --> 00:20:35,350**

posts.

**306**

**00:20:35,500 --> 00:20:41,723**

Prompt engineering ensures that the generated content

aligns with the desired tone, style, and subject matter.

**307**

**00:20:41,973 --> 00:20:43,784**

Example: Writing a Blog Post

**308**

**00:20:44,035 --> 00:20:49,608**

Developers can craft prompts for a content generation

platform to instruct AI models to write blog posts

**309**

**00:20:49,608 --> 00:20:50,908**

on specific topics.

**310**

**00:20:51,158 --> 00:20:56,916**

**Prompt: Write a 300-word blog post on the importance**

of cloud computing for small businesses.

**311**

**00:20:57,166 --> 00:21:04,051**

Response from the AI: Cloud computing has transformed

how small businesses operate by providing flexible,

**312**

**00:21:04,051 --> 00:21:06,547**

scalable, and cost-effective solutions.

**313**

**00:21:06,697 --> 00:21:12,850**

It enables businesses to store data, run applications,

and manage resources without investing in physical

**314**

**00:21:12,850 --> 00:21:13,524**

hardware.

**315**

**00:21:13,674 --> 00:21:20,199**

Small businesses can improve collaboration, reduce

operational costs, and ensure data security by adopting

**316**

**00:21:20,199 --> 00:21:21,290**

cloud services.

**317**

**00:21:21,440 --> 00:21:27,106**

Furthermore, cloud-based tools allow teams to work

remotely and access information from anywhere.

**318**

**00:21:27,256 --> 00:21:32,922**

As the digital landscape evolves, small businesses

that leverage cloud computing will be better positioned

**319**

**00:21:32,922 --> 00:21:37,147**

to compete with larger enterprises and adapt to changing

market demands.

**320**

**00:21:37,397 --> 00:21:43,806**

Impact: With prompt engineering, content creators

can guide AI to produce high-quality, engaging content

**321**

**00:21:43,806 --> 00:21:47,243**

quickly, saving time and effort in drafting and editing.

**322**

**00:21:47,493 --> 00:21:48,236**

**6.**

**323**

**00:21:48,386 --> 00:21:50,290**

Chatbots and Customer Support

**324**

**00:21:50,540 --> 00:21:56,856**

AI-driven chatbots and virtual assistants rely heavily

on prompt engineering to provide relevant and coherent

**325**

**00:21:56,856 --> 00:21:58,527**

responses to user queries.

**326**

**00:21:58,677 --> 00:22:04,180**

By designing prompts that reflect common user inputs,

developers can improve the chatbot's ability to handle

**327**

**00:22:04,180 --> 00:22:07,222**

various requests, from troubleshooting to product

inquiries.

**328**

**00:22:07,472 --> 00:22:09,422**

Example: Customer Support Bot

**329**

**00:22:09,673 --> 00:22:15,211**

A chatbot designed for an e-commerce website can be

programmed to respond to customer queries using specific

**330**

**00:22:15,211 --> 00:22:15,896**

prompts.

**331**

**00:22:16,146 --> 00:22:21,765**

Prompt: If a user asks about tracking an order, respond

with instructions on how to track their package.

**332**

**00:22:22,015 --> 00:22:28,250**

Response from the AI: To track your order, please

visit our website's 'My Orders' section and click

**333**

**00:22:28,273 --> 00:22:30,328**

'Track Order' next to your purchase.

**334**

**00:22:30,478 --> 00:22:33,961**

You can also enter your tracking number on our shipping

partner's website.

**335**

**00:22:34,211 --> 00:22:40,411**

Impact: Through prompt engineering, chatbots can offer

more accurate, relevant responses, enhancing user

**336**

**00:22:40,411 --> 00:22:44,009**

experience and reducing the workload on human support

agents.

**337**

**00:22:44,259 --> 00:22:45,002**

**7.**

**338**

**00:22:45,152 --> 00:22:46,824**

Testing and Validation

**339**

**00:22:47,074 --> 00:22:53,889**

In software testing, prompt engineering can instruct

AI models to generate test cases, validate functionality,

**340**

**00:22:53,889 --> 00:22:55,991**

or simulate various scenarios.

**341**

**00:22:56,141 --> 00:23:01,110**

AI-driven testing solutions allow for faster iteration

and more thorough coverage.

**342**

**00:23:01,360 --> 00:23:03,636**

Example: Generating Test Cases

**343**

**00:23:03,885 --> 00:23:09,365**

For a web application, developers might ask the AI

to generate test cases based on user input.

**344**

**00:23:09,615 --> 00:23:15,188**

Prompt: Generate three test cases for a login function

that requires a username and password.

**345**

**00:23:15,438 --> 00:23:16,878**

Response from the AI:

**346**

**00:23:17,128 --> 00:23:17,685**

**1.**

**347**

**00:23:17,835 --> 00:23:22,804**

Test with valid username and password: Ensure the

user is successfully logged in.

**348**

**00:23:23,054 --> 00:23:23,611**

**2.**

**349**

**00:23:23,761 --> 00:23:29,195**

Test with an invalid password: Ensure the login fails

and an error message is displayed.

**350**

**00:23:29,445 --> 00:23:30,095**

**3.**

**351**

**00:23:30,245 --> 00:23:36,433**

Test with an empty username or password field: Ensure

the system prompts the user to fill in the required

**352**

**00:23:36,433 --> 00:23:37,165**

fields.

**353**

**00:23:37,415 --> 00:23:43,533**

Impact: Prompt-engineered test cases can streamline

the QA process, ensuring comprehensive test coverage

**354**

**00:23:43,533 --> 00:23:45,402**

and faster bug identification.

**355**

**00:23:45,652 --> 00:23:51,585**

Prompt engineering plays a pivotal role in enhancing

the functionality and accuracy of AI systems across

**356**

**00:23:51,585 --> 00:23:53,083**

various applications.

**357**

**00:23:53,233 --> 00:23:59,642**

By crafting clear and structured prompts, developers

can guide AI models to deliver more relevant, useful,

**358**

**00:23:59,642 --> 00:24:05,818**

and actionable outputs in projects ranging from coding

and debugging to decision-making, content generation,

**359**

**00:24:05,818 --> 00:24:07,118**

and customer support.

**360**

**00:24:07,268 --> 00:24:13,781**

This approach helps unlock AI's full potential, making

it a valuable tool for solving complex problems in

**361**

**00:24:13,781 --> 00:24:15,720**

software development and beyond.

**362**

**00:24:15,970 --> 00:24:17,038**

Key Takeaways:

**363**

**00:24:17,288 --> 00:24:17,846**

**1.**

**364**

**00:24:17,996 --> 00:24:20,085**

Definition of Prompt Engineering:

**365**

**00:24:20,405 --> 00:24:25,444**

* · A process in software development to design and

refine inputs (prompts) for AI models.

**366**

**00:24:25,915 --> 00:24:31,127**

* · Optimizes model outputs by enhancing clarity, specificity,

and task alignment.

**367**

**00:24:31,377 --> 00:24:31,935**

**2.**

**368**

**00:24:32,085 --> 00:24:34,035**

Significance in Software Projects:

**369**

**00:24:34,366 --> 00:24:37,954**

* · Key for fine-tuning AI models within software projects.

**370**

**00:24:38,308 --> 00:24:43,126**

* · Enhances productivity by improving the accuracy

and relevance of model outputs.

**371**

**00:24:43,504 --> 00:24:46,674**

* · Reduces the need for model re-training or complex

coding.

**372**

**00:24:48,474 --> 00:24:49,124**

**3.**

**373**

**00:24:49,274 --> 00:24:51,224**

Applications in AI Projects:

**374**

**00:24:51,567 --> 00:24:55,329**

* · Improves task-specific performance by crafting tailored

prompts.

**375**

**00:24:55,660 --> 00:25:00,408**

* · Used in natural language processing, code generation,

and decision-making models.

**376**

**00:25:00,728 --> 00:25:06,603**

* · Techniques include prompt chaining, prompt tuning,

and feedback loops to refine results iteratively.

**377**

**00:25:06,853 --> 00:25:07,549**

**4.**

**378**

**00:25:07,699 --> 00:25:08,582**

Benefits:

**379**

**00:25:09,517 --> 00:25:12,547**

* · Boosts efficiency in AI-powered applications.

**380**

**00:25:12,913 --> 00:25:18,973**

* · Lowers resource and time costs for development teams

by refining AI behavior through effective prompting.

**381**

**00:25:19,623 --> 00:25:22,224**

**1.2 Basics of API Interaction**

**382**

**00:25:22,474 --> 00:25:29,080**

APIs (Application Programming Interfaces) play an

important role in prompt engineering by allowing developers

**383**

**00:25:29,080 --> 00:25:33,434**

to integrate and interact with AI models in real-world

applications.

**384**

**00:25:33,584 --> 00:25:39,969**

APIs provide a standardized way to send requests

(prompts) to an AI system and receive responses, making

**385**

**00:25:39,969 --> 00:25:46,111**

embedding AI functionality into various platforms,

applications, or services easier without directly

**386**

**00:25:46,111 --> 00:25:48,491**

handling the underlying model architecture.

**387**

**00:25:48,741 --> 00:25:54,778**

**1.2.1 API Requests and Responses in Different Programming**

Languages (Python, JavaScript)

**388**

**00:25:55,028 --> 00:26:01,251**

APIs are significant in prompt engineering because

of their flexibility, scalability, and accessibility.

**389**

**00:26:01,401 --> 00:26:07,473**

They enable developers to experiment with prompts,

fine-tune models, and optimize responses efficiently

**390**

**00:26:07,473 --> 00:26:11,293**

by providing an easy way to interact with powerful

AI tools.

**391**

**00:26:11,443 --> 00:26:17,817**

APIs often include features like rate limiting, usage

tracking, and version control, ensuring that the AI

**392**

**00:26:17,817 --> 00:26:22,263**

services are used effectively while maintaining system

performance and reliability.

**393**

**00:26:22,513 --> 00:26:29,943**

To use an API in prompt engineering, developers send

an HTTP request (typically POST) containing the prompt,

**394**

**00:26:29,943 --> 00:26:35,099**

context, and additional parameters (such as response

length, format, or specific constraints).

**395**

**00:26:35,249 --> 00:26:41,843**

The API processes the request and returns a generated

response, which can be analyzed, tested, and refined.

**396**

**00:26:41,993 --> 00:26:47,125**

This process facilitates rapid experimentation with

different prompt structures, helping to improve the

**397**

**00:26:47,125 --> 00:26:52,941**

quality and relevance of AI-generated outputs while

seamlessly integrating AI capabilities into a wide

**398**

**00:26:52,941 --> 00:26:54,114**

range of applications.

**399**

**00:26:54,364 --> 00:27:01,179**

APIs (Application Programming Interfaces) are the

backbone of communication between software applications,

**400**

**00:27:01,179 --> 00:27:03,559**

enabling them to interact and share data.

**401**

**00:27:03,709 --> 00:27:08,725**

Whether fetching information from a web service,

interacting with a cloud database, or integrating

**402**

**00:27:08,725 --> 00:27:14,158**

with third-party platforms, APIs provide structured

access to a program's features or data.

**403**

**00:27:14,308 --> 00:27:19,927**

Developers must understand how to send API requests

and handle responses in different programming languages.

**404**

**00:27:20,177 --> 00:27:22,778**

Why is API Required in Prompt Engineering?

**405**

**00:27:23,028 --> 00:27:29,530**

API (Application Programming Interface) plays a crucial

role in prompt engineering by facilitating communication

**406**

**00:27:29,530 --> 00:27:35,056**

between different software components and enabling

more dynamic, efficient, and customizable interactions

**407**

**00:27:35,056 --> 00:27:37,563**

with large language models like GPT.

**408**

**00:27:37,714 --> 00:27:41,661**

Here are key reasons why APIs are required in prompt

engineering:

**409**

**00:27:41,981 --> 00:27:48,204**

* · Automation and Scalability: APIs allow developers

to automate large-scale prompt generation and interaction

**410**

**00:27:48,204 --> 00:27:49,341**

with AI models.

**411**

**00:27:49,491 --> 00:27:55,122**

Rather than manually crafting each prompt, developers

can write code that programmatically generates prompts

**412**

**00:27:55,122 --> 00:27:57,386**

based on varying inputs and scenarios.

**413**

**00:27:57,536 --> 00:28:03,573**

This is particularly important for businesses and

applications that need high-volume or complex interactions

**414**

**00:28:03,573 --> 00:28:04,409**

with AI.

**415**

**00:28:04,775 --> 00:28:12,333**

* · Customizability and Flexibility: With APIs, developers

can design workflows where different parameters or

**416**

**00:28:12,333 --> 00:28:15,247**

constraints can be applied dynamically to the prompts.

**417**

**00:28:15,397 --> 00:28:20,796**

For instance, in a customer service chatbot, the

API can adjust the prompt based on the user's current

**418**

**00:28:20,796 --> 00:28:24,221**

context (e.g., a product purchase issue vs.

**419**

**00:28:24,371 --> 00:28:26,043**

a technical support query).

**420**

**00:28:26,193 --> 00:28:32,137**

This level of customization is difficult to achieve

manually but is easily implemented through APIs.

**421**

**00:28:32,457 --> 00:28:40,061**

* · Integration with External Data: APIs prompt engineers

to integrate external data sources (e.g., databases,

**422**

**00:28:40,061 --> 00:28:44,090**

weather services, or financial data) into the model's

responses.

**423**

**00:28:44,240 --> 00:28:50,300**

This allows the AI to provide up-to-date and relevant

information, making interactions more valuable and

**424**

**00:28:50,300 --> 00:28:51,160**

reliable.

**425**

**00:28:51,310 --> 00:28:57,858**

For example, a prompt could pull real-time stock

market data via an API and use it to answer user queries.

**426**

**00:28:58,189 --> 00:29:04,772**

* · Dynamic Input and Output Management: APIs can manage

input-output more efficiently when working with large

**427**

**00:29:04,772 --> 00:29:05,863**

language models.

**428**

**00:29:06,013 --> 00:29:11,713**

Developers can structure inputs to maximize the model's

ability to understand the prompt while controlling

**429**

**00:29:11,713 --> 00:29:13,769**

the length and complexity of responses.

**430**

**00:29:13,919 --> 00:29:20,026**

APIs enable easy adjustments to various model parameters

like temperature, token limits, or prompt-specific

**431**

**00:29:20,026 --> 00:29:20,699**

rules.

**432**

**00:29:22,603 --> 00:29:29,267**

* · Consistency and Security: APIs enforce consistent

interaction patterns, which are important when deploying

**433**

**00:29:29,267 --> 00:29:31,043**

AI in production environments.

**434**

**00:29:31,193 --> 00:29:37,311**

They also provide a secure way to expose the prompt

engineering workflows, restricting access through

**435**

**00:29:37,311 --> 00:29:40,249**

authentication and authorization protocols.

**436**

**00:29:40,399 --> 00:29:47,202**

In scenarios where multiple developers or clients

interact with the AI, APIs ensure security while enabling

**437**

**00:29:47,202 --> 00:29:48,619**

diverse use cases.

**438**

**00:29:48,939 --> 00:29:56,056**

* · Seamless Integration with AI Platforms: Many AI

providers, including OpenAI, offer API endpoints that

**439**

**00:29:56,056 --> 00:29:59,225**

allow developers to interact with models like GPT.

**440**

**00:29:59,375 --> 00:30:04,809**

These APIs simplify access to complex models without

needing to deploy them themselves.

**441**

**00:30:04,959 --> 00:30:11,414**

Prompt engineering through APIs becomes streamlined,

making it easier to prototype and scale applications.

**442**

**00:30:11,664 --> 00:30:18,502**

APIs are indispensable in prompt engineering because

they provide the automation, scalability, flexibility,

**443**

**00:30:18,502 --> 00:30:23,460**

and integration capabilities required to build robust

AI-powered applications.

**444**

**00:30:24,110 --> 00:30:26,617**

How to Use API in Prompt Engineering?

**445**

**00:30:26,867 --> 00:30:32,254**

Using an API in prompt engineering involves designing

prompts and then sending these prompts programmatically

**446**

**00:30:32,254 --> 00:30:35,598**

to an AI model, typically through a RESTful API.

**447**

**00:30:35,748 --> 00:30:40,531**

Here's a basic workflow for using an API in prompt

engineering with an example:

**448**

**00:30:40,851 --> 00:30:46,911**

* · Set up API Access: First, you need access to the

API of the AI model you want to interact with.

**449**

**00:30:47,061 --> 00:30:51,380**

For instance, if you're using OpenAI's GPT, you'll

need an API key.

**450**

**00:30:51,530 --> 00:30:55,431**

APIs typically require authentication headers for

secure access.

**451**

**00:30:55,797 --> 00:31:00,047**

* · Design the Prompt: Crafting the right prompt is

key to prompt engineering.

**452**

**00:31:00,197 --> 00:31:04,608**

The prompt is the input you send to the AI model

to generate the desired response.

**453**

**00:31:04,758 --> 00:31:08,566**

Effective prompts are clear, concise, and contextual

to guide the AI.

**454**

**00:31:08,979 --> 00:31:15,991**

* · Programmatic API Call: Once the prompt is ready,

send it to the API using an HTTP request (e.g., POST

**455**

**00:31:15,991 --> 00:31:16,758**

request).

**456**

**00:31:16,908 --> 00:31:22,597**

APIs often allow various parameters to fine-tune

the response, such as setting the response length,

**457**

**00:31:22,597 --> 00:31:26,985**

controlling randomness (temperature), and specifying

the number of responses.

**458**

**00:31:27,305 --> 00:31:32,297**

* · Handling the Response: The API will return a response,

typically in JSON format.

**459**

**00:31:32,447 --> 00:31:39,088**

The developer can parse this response, extract the

relevant data, and incorporate it into their application.

**460**

**00:31:39,238 --> 00:31:43,046**

Based on the response, additional prompts can be

generated dynamically.

**461**

**00:31:43,696 --> 00:31:46,343**

API Basics: Requests and Responses

**462**

**00:31:46,593 --> 00:31:49,983**

At the core of API communication are requests and

responses.

**463**

**00:31:50,133 --> 00:31:55,474**

A client (usually a software program) requests a

server to access data or functionality.

**464**

**00:31:55,624 --> 00:32:02,021**

The client sends data such as parameters, headers,

and authentication details, and the server responds

**465**

**00:32:02,021 --> 00:32:04,633**

with data, status codes, and other metadata.

**466**

**00:32:04,783 --> 00:32:12,527**

These interactions commonly happen over HTTP/HTTPS

and use different methods like GET, POST, PUT, DELETE,

**467**

**00:32:12,527 --> 00:32:13,328**

and PATCH.

**468**

**00:32:13,478 --> 00:32:17,518**

The following figure depicts the typical API server

request and response.

**469**

**00:32:18,168 --> 00:32:21,280**

Here's a brief description of HTTP methods:

**470**

**00:32:21,611 --> 00:32:23,759**

* · GET: Retrieve data from a server.

**471**

**00:32:24,241 --> 00:32:27,724**

* · POST: Send data to a server to create a new resource.

**472**

**00:32:28,055 --> 00:32:30,761**

* · PUT: Update or create a resource on the server.

**473**

**00:32:31,185 --> 00:32:33,565**

* · DELETE: Remove a resource from the server.

**474**

**00:32:33,815 --> 00:32:40,305**

Now, let's explore how to implement API requests and

handle responses using Python and JavaScript, two

**475**

**00:32:40,305 --> 00:32:41,802**

widely used languages.

**476**

**00:32:42,052 --> 00:32:44,560**

API Requests and Responses in Python

**477**

**00:32:44,810 --> 00:32:50,011**

Python's simplicity and extensive libraries make it

an ideal language for working with APIs.

**478**

**00:32:50,161 --> 00:32:54,666**

The most popular library for making HTTP requests

in Python is Requests.

**479**

**00:32:56,467 --> 00:32:58,603**

Error Handling and Best Practices

**480**

**00:32:58,853 --> 00:33:05,134**

When working with API requests and responses, error

handling is crucial to ensure reliability and security

**481**

**00:33:05,134 --> 00:33:06,237**

in communication.

**482**

**00:33:06,387 --> 00:33:08,709**

Key aspects of error handling include:

**483**

**00:33:09,354 --> 00:33:15,530**

* · Status Code Checks: Always check the HTTP status

code of the response to handle different situations

**484**

**00:33:15,530 --> 00:33:21,405**

**(e.g., 200 for success, 404 for not found, 500 for**

server errors).

**485**

**00:33:22,363 --> 00:33:27,831**

* · Retry Mechanisms: Implement retries for transient

errors (e.g., network timeouts).

**486**

**00:33:28,174 --> 00:33:31,471**

* · Timeouts: Set time limits to avoid hanging requests.

**487**

**00:33:31,953 --> 00:33:38,466**

* · Rate Limiting: Respect the API's rate limits by

handling responses that indicate limit breaches (typically

**488**

**00:33:38,466 --> 00:33:41,659**

**HTTP status 429).**

**489**

**00:33:42,309 --> 00:33:44,399**

API Request Prompt Engineering

**490**

**00:33:44,649 --> 00:33:49,897**

Prompt engineering can optimize requests' structure,

ensuring APIs are used efficiently.

**491**

**00:33:50,047 --> 00:33:54,226**

When designing prompts for API interaction, consider

these factors:

**492**

**00:33:54,476 --> 00:33:55,034**

**1.**

**493**

**00:33:55,184 --> 00:33:59,178**

Clarity and Conciseness: Make sure the request's

structure is clear.

**494**

**00:33:59,328 --> 00:34:04,204**

API calls that request too much data or irrelevant

information slow down the system.

**495**

**00:34:04,454 --> 00:34:05,011**

**2.**

**496**

**00:34:05,161 --> 00:34:09,805**

Structured Queries: Pass query parameters that narrow

the API's response.

**497**

**00:34:09,955 --> 00:34:14,971**

For example, when querying a database API, filter

for only necessary fields.

**498**

**00:34:15,221 --> 00:34:15,871**

**3.**

**499**

**00:34:16,021 --> 00:34:22,917**

Input Validation: Ensure all inputs are sanitized

and validated before sending them to the API, improving

**500**

**00:34:22,917 --> 00:34:24,473**

security and efficiency.

**501**

**00:34:25,123 --> 00:34:28,560**

**1.2.2 Making Requests to OpenAI's API**

**502**

**00:34:28,809 --> 00:34:35,009**

Setting up and making requests to OpenAI's API using

SDKs and libraries is a powerful way to integrate

**503**

**00:34:35,009 --> 00:34:37,865**

NLP capabilities into various applications.

**504**

**00:34:38,015 --> 00:34:44,575**

OpenAI offers multiple SDKs and libraries in different

programming languages to ease interaction with its

**505**

**00:34:44,575 --> 00:34:50,554**

models, such as GPT, for tasks like text generation,

summarization, translation, and more.

**506**

**00:34:50,704 --> 00:34:56,149**

Understanding how to set up this API, make requests,

and effectively use prompt engineering can help you

**507**

**00:34:56,149 --> 00:34:57,809**

harness its full potential.

**508**

**00:34:58,059 --> 00:34:59,731**

Setting Up OpenAI's API

**509**

**00:34:59,981 --> 00:35:05,832**

To begin using OpenAI's API, you must create an account

on OpenAI's platform and obtain an API key.

**510**

**00:35:05,982 --> 00:35:10,998**

This key acts as a credential that allows you to

authenticate your requests and access the various

**511**

**00:35:10,998 --> 00:35:12,902**

models that OpenAI provides.

**512**

**00:35:13,152 --> 00:35:20,025**

Once you have your API key, you can install the necessary

SDK or library to interact with OpenAI's API.

**513**

**00:35:20,175 --> 00:35:27,443**

The most common setup is the Python SDK, though OpenAI

also supports libraries in Node.js, Ruby, and other

**514**

**00:35:27,443 --> 00:35:28,720**

programming languages.

**515**

**00:35:28,970 --> 00:35:33,150**

For Python, you can install the OpenAI package with

the following command:

**516**

**00:35:33,400 --> 00:35:36,186**

Bash code:

**517**

**00:35:36,436 --> 00:35:37,922**

pip install openai

**518**

**00:35:38,172 --> 00:35:43,141**

After installing the package, you must configure your

environment to use the API key.

**519**

**00:35:43,291 --> 00:35:48,957**

This can be done by setting an environment variable

for the API key or directly specifying it in your

**520**

**00:35:48,957 --> 00:35:49,561**

code:

**521**

**00:35:49,811 --> 00:35:50,786**

Python code:

**522**

**00:35:51,036 --> 00:35:52,243**

import openai

**523**

**00:35:52,830 --> 00:35:54,165**

# Set up the API key

**524**

**00:35:54,415 --> 00:35:58,223**

openai.api\_key = "your-api-key"

**525**

**00:35:58,473 --> 00:36:03,071**

At this point, your setup is complete, and you are

ready to start making requests to the API.

**526**

**00:36:03,321 --> 00:36:04,807**

Making API Requests

**527**

**00:36:05,057 --> 00:36:10,955**

There are two primary methods for requesting OpenAI's

API: the completion and chat endpoints.

**528**

**00:36:11,105 --> 00:36:15,516**

Depending on your use case, you'll use one of these

endpoints to interact with the models.

**529**

**00:36:17,317 --> 00:36:17,874**

**1.**

**530**

**00:36:18,024 --> 00:36:21,460**

Completion Endpoint: This is used for text generation

tasks.

**531**

**00:36:21,610 --> 00:36:26,115**

You provide a prompt, and the model generates a continuation

of the text based on the prompt.

**532**

**00:36:26,765 --> 00:36:27,322**

**2.**

**533**

**00:36:27,472 --> 00:36:33,103**

Chat Endpoint: This is used for conversational tasks

where you send a series of messages in a chat-like

**534**

**00:36:33,103 --> 00:36:33,788**

format.

**535**

**00:36:33,938 --> 00:36:39,882**

**The GPT-4 models are optimized for this endpoint,**

making it perfect for interactive applications.

**536**

**00:36:40,532 --> 00:36:43,179**

Prompt Engineering: Optimizing API Requests

**537**

**00:36:43,429 --> 00:36:48,213**

Prompt engineering is crucial for obtaining the desired

outputs from OpenAI's models.

**538**

**00:36:48,363 --> 00:36:52,914**

It involves carefully crafting the input text (or

prompt) that you send to the model.

**539**

**00:36:53,064 --> 00:36:56,036**

Here are some prompt engineering techniques with

examples:

**540**

**00:36:56,286 --> 00:36:56,843**

**1.**

**541**

**00:36:56,993 --> 00:37:01,777**

Be Explicit: To guide the model, you should clearly

state your intent in the prompt.

**542**

**00:37:01,927 --> 00:37:06,524**

This helps reduce ambiguity and ensures the model

generates relevant responses.

**543**

**00:37:06,774 --> 00:37:07,703**

Example:

**544**

**00:37:07,953 --> 00:37:08,928**

Python code:

**545**

**00:37:09,178 --> 00:37:15,726**

**prompt = "Write a 50-word summary of the following**

text:\n\nArtificial intelligence is..."

**546**

**00:37:15,976 --> 00:37:21,444**

In this case, the prompt explicitly asks the model

**to provide a 50-word summary, ensuring the response**

**547**

**00:37:21,444 --> 00:37:23,035**

is concise and on-point.

**548**

**00:37:23,285 --> 00:37:23,842**

**2.**

**549**

**00:37:23,992 --> 00:37:30,099**

Provide Examples: When trying to achieve a specific

output format, giving examples in the prompt can help

**550**

**00:37:30,099 --> 00:37:31,887**

the model understand what you want.

**551**

**00:37:32,137 --> 00:37:33,066**

Example:

**552**

**00:37:33,316 --> 00:37:34,291**

Python code:

**553**

**00:37:34,541 --> 00:37:38,489**

prompt = """Translate the following phrases to French:

**554**

**00:37:38,739 --> 00:37:39,296**

**1.**

**555**

**00:37:39,446 --> 00:37:40,282**

Good morning

**556**

**00:37:40,532 --> 00:37:41,089**

**2.**

**557**

**00:37:41,239 --> 00:37:42,121**

How are you?

**558**

**00:37:42,371 --> 00:37:43,022**

**3.**

**559**

**00:37:43,172 --> 00:37:44,147**

What is your name?

**560**

**00:37:44,397 --> 00:37:45,093**

**4.**

**561**

**00:37:45,243 --> 00:37:46,126**

Thank you"""

**562**

**00:37:46,376 --> 00:37:52,088**

The model understands the pattern by providing examples

of English phrases and will likely translate similar

**563**

**00:37:52,088 --> 00:37:53,388**

phrases accurately.

**564**

**00:37:53,638 --> 00:37:54,288**

**3.**

**565**

**00:37:54,438 --> 00:38:00,266**

Set Constraints: By including these constraints directly

in the prompt, you can control the length, tone, or

**566**

**00:38:00,266 --> 00:38:01,590**

format of the output.

**567**

**00:38:01,840 --> 00:38:02,769**

Example:

**568**

**00:38:03,019 --> 00:38:03,994**

Python code:

**569**

**00:38:04,244 --> 00:38:08,935**

Prompt = "Write a formal email to a professor requesting

a meeting.

**570**

**00:38:09,085 --> 00:38:11,221**

**Limit the email to 100 words."**

**571**

**00:38:11,471 --> 00:38:16,626**

This prompt specifies the content and places a word

limit, ensuring the response is concise.

**572**

**00:38:16,876 --> 00:38:17,572**

**4.**

**573**

**00:38:17,722 --> 00:38:23,666**

Role-Based Prompts: You can influence the style and

tone of the output by assigning a specific role or

**574**

**00:38:23,666 --> 00:38:24,920**

perspective to the model.

**575**

**00:38:25,570 --> 00:38:26,824**

Advanced Features

**576**

**00:38:27,074 --> 00:38:27,632**

**1.**

**577**

**00:38:27,782 --> 00:38:33,285**

Temperature and Max Tokens: You can fine-tune the

model's behavior by adjusting parameters like temperature

**578**

**00:38:33,285 --> 00:38:34,376**

and max tokens.

**579**

**00:38:34,526 --> 00:38:41,120**

Temperature controls the randomness of the output,

**with higher values (e.g., 0.8) producing more creative**

**580**

**00:38:41,120 --> 00:38:47,204**

**responses and lower values (e.g., 0.2) resulting in**

more focused and deterministic answers.

**581**

**00:38:47,454 --> 00:38:48,383**

Example:

**582**

**00:38:48,633 --> 00:38:49,608**

Python code:

**583**

**00:38:49,858 --> 00:38:53,481**

response = openai.Completion.create(

**584**

**00:38:53,731 --> 00:38:56,888**

**engine="text-davinci-003",**

**585**

**00:38:57,138 --> 00:39:01,690**

prompt="Write a creative story about a dragon.",

**586**

**00:39:01,940 --> 00:39:04,447**

**temperature=0.7,**

**587**

**00:39:04,697 --> 00:39:07,251**

**max\_tokens=100**

**588**

**00:39:07,502 --> 00:39:08,059**

**2.**

**589**

**00:39:08,209 --> 00:39:14,153**

Stop Sequences: You can define stop sequences to

indicate where the model should halt in its generation.

**590**

**00:39:14,303 --> 00:39:19,272**

This is helpful when limiting the response length

or ending the output after a specific phrase.

**591**

**00:39:19,522 --> 00:39:20,451**

Example:

**592**

**00:39:20,701 --> 00:39:21,676**

Python code:

**593**

**00:39:21,926 --> 00:39:25,549**

response = openai.Completion.create(

**594**

**00:39:25,799 --> 00:39:28,956**

**engine="text-davinci-003",**

**595**

**00:39:29,206 --> 00:39:32,689**

prompt="Write a dialogue between a teacher and a student:",

**596**

**00:39:32,939 --> 00:39:34,751**

stop=["Student:", "Teacher:"],

**597**

**00:39:35,001 --> 00:39:37,555**

**max\_tokens=100**

**598**

**00:39:37,805 --> 00:39:43,865**

The model will generate text until it encounters the

stop sequences, ensuring the response remains within

**599**

**00:39:43,865 --> 00:39:45,282**

the desired structure.

**600**

**00:39:45,532 --> 00:39:51,906**

Setting up and making requests to OpenAI's API through

SDKs and libraries is straightforward, allowing developers

**601**

**00:39:51,906 --> 00:39:55,748**

to integrate powerful NLP capabilities into their

applications.

**602**

**00:39:55,898 --> 00:40:01,657**

By mastering prompt engineering, you can guide the

model to produce more accurate and relevant results.

**603**

**00:40:01,807 --> 00:40:07,647**

Techniques like setting clear instructions, providing

examples, and using advanced parameters like temperature

**604**

**00:40:07,647 --> 00:40:14,717**

and stop sequences allow you to tailor the outputs

to meet specific needs, making OpenAI's API a valuable

**605**

**00:40:14,717 --> 00:40:18,850**

tool for a wide range of applications, from chatbots

to content generation.

**606**

**00:40:19,100 --> 00:40:22,955**

Practical Example of Prompt Engineering in Software

Development

**607**

**00:40:23,205 --> 00:40:29,242**

Consider the following scenario where a developer

needs to implement a basic REST API using Python and

**608**

**00:40:29,242 --> 00:40:29,939**

Flask.

**609**

**00:40:30,089 --> 00:40:35,337**

Instead of manually writing the entire code from

scratch, they can employ prompt engineering to simplify

**610**

**00:40:35,337 --> 00:40:36,219**

the process.

**611**

**00:40:38,019 --> 00:40:39,087**

Key Takeaways:

**612**

**00:40:39,337 --> 00:40:39,894**

**1.**

**613**

**00:40:40,044 --> 00:40:41,948**

Basics of AI Interaction:

**614**

**00:40:42,279 --> 00:40:48,049**

* · AI interaction involves exchanging data between

applications and AI models via APIs.

**615**

**00:40:48,415 --> 00:40:53,083**

* · Developers use APIs to send prompts (inputs) and

receive outputs from AI models.

**616**

**00:40:53,333 --> 00:40:53,890**

**2.**

**617**

**00:40:54,040 --> 00:40:55,944**

API Requests and Responses:

**618**

**00:40:56,275 --> 00:41:01,813**

* · Python: Use libraries like requests and http.client

to handle API calls.

**619**

**00:41:02,167 --> 00:41:07,775**

* · JavaScript: Utilize fetch() or Axios to make HTTP

requests to APIs.

**620**

**00:41:08,129 --> 00:41:13,830**

* · Both languages handle GET and POST requests, with

proper JSON parsing and error handling.

**621**

**00:41:14,080 --> 00:41:14,730**

**3.**

**622**

**00:41:14,880 --> 00:41:16,738**

OpenAI API Requests:

**623**

**00:41:17,325 --> 00:41:23,861**

* · Requires API key and access to OpenAI's SDK or libraries

(openai in Python or Node.js).

**624**

**00:41:24,192 --> 00:41:28,244**

* · Set up requests with parameters like prompt, model,

and tokens.

**625**

**00:41:28,598 --> 00:41:33,278**

* · Responses include model-generated outputs, which

can be processed for integration.

**626**

**00:41:33,528 --> 00:41:34,224**

**4.**

**627**

**00:41:34,374 --> 00:41:35,907**

Practical Considerations:

**628**

**00:41:36,227 --> 00:41:40,801**

* · Ensure proper authentication and error handling

for reliable communication.

**629**

**00:41:41,283 --> 00:41:45,323**

* · Optimize request parameters for improved performance

and cost management.

**630**

**00:41:45,973 --> 00:41:48,806**

**1.3 Understanding Prompt Structures**

**631**

**00:41:49,056 --> 00:41:56,463**

In today's digital landscape, where artificial intelligence

(AI) and machine learning (ML) are transforming industries,

**632**

**00:41:56,463 --> 00:42:01,873**

understanding how to effectively communicate with

AI models is crucial for developers.

**633**

**00:42:02,023 --> 00:42:07,700**

One way to do this is through prompt engineering—the

art and science of designing prompts to guide AI models

**634**

**00:42:07,700 --> 00:42:11,497**

**like GPT-4 to generate specific, useful outputs.**

**635**

**00:42:11,647 --> 00:42:17,731**

This article provides a comprehensive overview of

prompt syntax and components for developers and practical

**636**

**00:42:17,731 --> 00:42:21,260**

examples to help you understand how to craft effective

prompts.

**637**

**00:42:21,510 --> 00:42:25,225**

**1.3.1 Basic Prompt Syntax and Components for Developers**

**638**

**00:42:25,475 --> 00:42:31,002**

A prompt is a set of instructions or inputs an AI

model provides to generate a desired output.

**639**

**00:42:31,152 --> 00:42:36,307**

Prompts can range from a single word to several sentences,

depending on the complexity of the task.

**640**

**00:42:36,457 --> 00:42:41,658**

The better and clearer your prompt, the more accurate

and relevant the AI's response will be.

**641**

**00:42:41,908 --> 00:42:46,598**

Prompt engineering, therefore, involves carefully

designing prompts to maximize the effectiveness of

**642**

**00:42:46,598 --> 00:42:47,713**

AI outputs.

**643**

**00:42:47,863 --> 00:42:53,865**

By understanding the basic syntax and components,

developers can control the AI model's behavior, improving

**644**

**00:42:53,865 --> 00:42:57,337**

the reliability and accuracy of the generated responses.

**645**

**00:42:57,587 --> 00:42:59,398**

Basic Components of a Prompt

**646**

**00:42:59,648 --> 00:43:00,205**

**1.**

**647**

**00:43:00,355 --> 00:43:04,674**

Instruction: Clear instructions define the task you

want the AI to complete.

**648**

**00:43:04,824 --> 00:43:08,307**

These are essential for ensuring the model performs

the right action.

**649**

**00:43:08,557 --> 00:43:09,486**

Example:

**650**

**00:43:09,817 --> 00:43:12,708**

* · Poor prompt: "List programming languages."

**651**

**00:43:13,028 --> 00:43:18,902**

* · Improved prompt with instruction: "List five popular

programming languages used for web development."

**652**

**00:43:19,152 --> 00:43:19,710**

**2.**

**653**

**00:43:19,860 --> 00:43:23,110**

Context: This is the background or setting for the

task.

**654**

**00:43:23,260 --> 00:43:28,322**

It provides the model with the necessary information

to understand the environment in which it operates.

**655**

**00:43:28,472 --> 00:43:34,649**

Including context in a prompt helps narrow the possibilities

and provides clarity, especially for ambiguous queries.

**656**

**00:43:34,899 --> 00:43:35,828**

Example:

**657**

**00:43:36,148 --> 00:43:39,328**

* · Poor prompt: "Write a paragraph about technology."

**658**

**00:43:39,671 --> 00:43:45,220**

* · Improved prompt with context: "Write a paragraph

about the impact of blockchain technology on modern

**659**

**00:43:45,220 --> 00:43:46,080**

finance."

**660**

**00:43:46,330 --> 00:43:46,980**

**3.**

**661**

**00:43:47,130 --> 00:43:52,610**

Input Query: In cases where the model needs specific

data to operate, provide it explicitly.

**662**

**00:43:52,760 --> 00:43:57,868**

This is especially important for translation, summarization,

or code-generation tasks.

**663**

**00:43:58,118 --> 00:43:59,047**

Example:

**664**

**00:43:59,436 --> 00:44:03,941**

* · "Translate the following sentence from English to

French: 'Hello, how are you?'

**665**

**00:44:04,295 --> 00:44:08,092**

**· "Summarize this article in 50 words: [Insert article]"**

**666**

**00:44:08,342 --> 00:44:14,333**

The input query must include constraints to define

limitations or specific criteria for the task.

**667**

**00:44:14,483 --> 00:44:18,012**

This helps refine the output according to the desired

requirements.

**668**

**00:44:18,262 --> 00:44:19,191**

Example:

**669**

**00:44:19,511 --> 00:44:21,670**

* · Without constraints: "Write a story."

**670**

**00:44:21,990 --> 00:44:27,725**

**· With constraints: "Write a 200-word story for children**

about a cat that learns to swim."

**671**

**00:44:27,975 --> 00:44:28,672**

**4.**

**672**

**00:44:28,822 --> 00:44:35,788**

Desired Output: In many cases, specifying the output

format can help ensure the result is as expected.

**673**

**00:44:35,938 --> 00:44:40,535**

This is important for generating structured data,

lists, codes, or documents.

**674**

**00:44:40,785 --> 00:44:41,714**

Example:

**675**

**00:44:42,034 --> 00:44:44,147**

* · "List the key points in bullet form."

**676**

**00:44:44,513 --> 00:44:49,319**

· "Generate a JSON object with the following fields:

name, age, and occupation."

**677**

**00:44:51,120 --> 00:44:53,813**

Key Strategies for Effective Prompt Engineering

**678**

**00:44:54,063 --> 00:44:54,620**

**1.**

**679**

**00:44:54,770 --> 00:45:00,343**

Clarity and Specificity: Ambiguity often leads to

inaccurate or irrelevant responses.

**680**

**00:45:00,493 --> 00:45:04,115**

Be as clear and specific as possible to get the best

results.

**681**

**00:45:04,365 --> 00:45:05,294**

Example:

**682**

**00:45:05,625 --> 00:45:07,773**

* · Ambiguous prompt: "Explain Python."

**683**

**00:45:08,151 --> 00:45:14,083**

* · Specific prompt: "Explain Python as a programming

language used for data science, highlighting its libraries

**684**

**00:45:14,083 --> 00:45:17,079**

for data analysis, like Pandas and NumPy."

**685**

**00:45:17,329 --> 00:45:17,886**

**2.**

**686**

**00:45:18,036 --> 00:45:23,888**

Iterative Refinement: Start with a basic prompt and

refine it iteratively based on the results you get.

**687**

**00:45:24,038 --> 00:45:28,868**

This approach allows you to understand the model's

capabilities better and adjust accordingly.

**688**

**00:45:29,118 --> 00:45:30,046**

Example:

**689**

**00:45:30,389 --> 00:45:33,965**

* · First attempt: "Write a Python function to sort

a list."

**690**

**00:45:34,331 --> 00:45:40,264**

* · Refinement: "Write a Python function to sort a list

of integers in ascending order using the merge sort

**691**

**00:45:40,264 --> 00:45:41,135**

algorithm."

**692**

**00:45:41,385 --> 00:45:42,035**

**3.**

**693**

**00:45:42,185 --> 00:45:48,605**

Chain of Thought: Encouraging a step-by-step reasoning

process can lead to more coherent and accurate outputs,

**694**

**00:45:48,605 --> 00:45:50,590**

especially for complex tasks.

**695**

**00:45:50,740 --> 00:45:54,827**

You can guide the model through logical steps by

breaking the task into parts.

**696**

**00:45:55,077 --> 00:45:56,006**

Example:

**697**

**00:45:57,603 --> 00:46:04,011**

* · "Explain the process of training a neural network

step by step, starting from data collection, preprocessing,

**698**

**00:46:04,011 --> 00:46:06,194**

and ending with model evaluation."

**699**

**00:46:06,444 --> 00:46:07,141**

**4.**

**700**

**00:46:07,291 --> 00:46:13,281**

Role-playing: By assigning the AI model a role, you

can guide it to respond in a particular manner.

**701**

**00:46:13,431 --> 00:46:18,122**

This technique helps in generating contextually relevant

and focused outputs.

**702**

**00:46:18,772 --> 00:46:19,701**

Example:

**703**

**00:46:20,090 --> 00:46:25,291**

* · "You are a software engineer explaining to a beginner

how object-oriented programming works."

**704**

**00:46:25,541 --> 00:46:26,284**

**5.**

**705**

**00:46:26,434 --> 00:46:33,063**

Temperature Control and Sampling: Adjusting parameters

like temperature and top-k sampling can help control

**706**

**00:46:33,063 --> 00:46:36,326**

the creativity or randomness of the AI's responses.

**707**

**00:46:36,476 --> 00:46:42,188**

A low temperature makes the output more deterministic,

while a high temperature introduces more randomness.

**708**

**00:46:42,438 --> 00:46:43,367**

Example:

**709**

**00:46:43,687 --> 00:46:49,631**

* · High temperature: "Tell me a story about a space

adventure with a surprising twist." (More creative

**710**

**00:46:49,631 --> 00:46:51,233**

and diverse results)

**711**

**00:46:51,587 --> 00:46:56,870**

* · Low temperature: "Explain what an API is." (Factual

and deterministic response)

**712**

**00:46:57,520 --> 00:46:58,867**

Practical Examples

**713**

**00:46:59,117 --> 00:46:59,674**

**1.**

**714**

**00:46:59,824 --> 00:47:01,031**

Code Generation

**715**

**00:47:01,375 --> 00:47:04,579**

* · Task: Generate Python code for reversing a string.

**716**

**00:47:05,038 --> 00:47:11,470**

* · Prompt: "Write a Python function that takes a string

as input and returns the string in reverse order."

**717**

**00:47:11,790 --> 00:47:12,602**

· Output:

**718**

**00:47:12,852 --> 00:47:13,781**

Python code

**719**

**00:47:14,031 --> 00:47:15,517**

def reverse\_string(s):

**720**

**00:47:15,767 --> 00:47:17,764**

**return s[::-1]**

**721**

**00:47:18,014 --> 00:47:18,571**

**2.**

**722**

**00:47:18,721 --> 00:47:19,882**

Summarization

**723**

**00:47:20,236 --> 00:47:23,383**

* · Task: Summarize an article about climate change.

**724**

**00:47:23,737 --> 00:47:29,809**

* · Prompt: "Summarize the following article on climate

change in three sentences: [Insert article]."

**725**

**00:47:30,059 --> 00:47:30,710**

**3.**

**726**

**00:47:30,860 --> 00:47:31,882**

Translation

**727**

**00:47:32,212 --> 00:47:35,614**

* · Task: Translate a sentence from English to Spanish.

**728**

**00:47:35,934 --> 00:47:41,066**

· Prompt: "Translate the following sentence to Spanish:

'The weather is beautiful today.'"

**729**

**00:47:41,316 --> 00:47:42,012**

**4.**

**730**

**00:47:42,162 --> 00:47:43,555**

Document Formatting

**731**

**00:47:43,886 --> 00:47:46,452**

* · Task: Create a markdown document outline.

**732**

**00:47:46,899 --> 00:47:53,970**

* · Prompt: "Generate a markdown-formatted outline for

a project proposal with the following sections: Introduction,

**733**

**00:47:53,970 --> 00:47:56,919**

Objectives, Methodology, Timeline, and Budget."

**734**

**00:47:57,169 --> 00:48:01,581**

Prompt engineering is a powerful tool for developers

working with AI models.

**735**

**00:48:01,731 --> 00:48:07,687**

Developers can design prompts that generate precise

and useful responses by mastering the basic components—context,

**736**

**00:48:07,687 --> 00:48:11,112**

instruction, input data, constraints, output format,

and tone.

**737**

**00:48:11,262 --> 00:48:17,322**

The key to effective prompt engineering lies in specificity,

iterative refinement, and an understanding of the

**738**

**00:48:17,322 --> 00:48:18,506**

AI's behavior.

**739**

**00:48:18,656 --> 00:48:25,413**

With practice, developers can leverage prompt engineering

to optimize their workflows, create efficient automation,

**740**

**00:48:25,413 --> 00:48:27,016**

and solve complex problems.

**741**

**00:48:27,266 --> 00:48:30,888**

Example of a Basic prompt syntax and components for

developers

**742**

**00:48:31,138 --> 00:48:36,804**

Here's an example illustrating basic prompt syntax

and components for developers in the context of a

**743**

**00:48:36,804 --> 00:48:37,918**

Python project.

**744**

**00:48:38,068 --> 00:48:44,291**

This example demonstrates how to guide an AI model

to assist in code generation, explanation, or debugging

**745**

**00:48:44,291 --> 00:48:45,777**

within a Python project.

**746**

**00:48:46,027 --> 00:48:46,584**

**1.**

**747**

**00:48:46,734 --> 00:48:48,824**

Basic Prompt for Code Generation

**748**

**00:48:49,074 --> 00:48:54,635**

Suppose you're developing a Python project and need

the AI to generate a function to perform a specific

**749**

**00:48:54,635 --> 00:48:55,390**

task.

**750**

**00:48:55,640 --> 00:48:56,848**

Prompt Example:

**751**

**00:48:57,098 --> 00:49:02,206**

"Write a Python function named Fibonacci that returns

the nth number in the Fibonacci sequence.

**752**

**00:49:02,356 --> 00:49:07,232**

The function should be efficient, and memorization

should be used to store intermediate results."

**753**

**00:49:07,482 --> 00:49:08,411**

Components

**754**

**00:49:08,754 --> 00:49:10,983**

* · Context: Write a Python function.

**755**

**00:49:11,314 --> 00:49:14,391**

* · Input: A function to compute the nth Fibonacci number.

**756**

**00:49:14,722 --> 00:49:17,892**

* · Constraints: Use memorization for efficiency.

**757**

**00:49:19,692 --> 00:49:23,128**

Best Practices for Prompt Design in Python Projects

**758**

**00:49:23,935 --> 00:49:31,551**

* · Be Specific: Clearly define the function name, expected

behavior, and any required optimizations (e.g., memoization).

**759**

**00:49:31,882 --> 00:49:38,523**

* · Include Constraints: Define the performance requirements,

framework (e.g., unit test for testing), or error

**760**

**00:49:38,523 --> 00:49:42,483**

handling (e.g., raising exceptions for invalid inputs).

**761**

**00:49:42,849 --> 00:49:49,327**

* · Modular Inputs: For large projects, break prompts

into smaller steps—such as generating functions, creating

**762**

**00:49:49,327 --> 00:49:50,674**

tests, and debugging.

**763**

**00:49:50,924 --> 00:49:57,530**

This setup enables developers to guide AI effectively

through structured prompts, creating efficient collaboration

**764**

**00:49:57,530 --> 00:49:59,237**

in Python-based projects.

**765**

**00:50:00,287 --> 00:50:03,538**

**1.3.2 Crafting Prompts for Development Tasks**

**766**

**00:50:03,787 --> 00:50:10,869**

In AI, prompt engineering has emerged as a critical

skill for leveraging large language models (LLMs),

**767**

**00:50:10,869 --> 00:50:14,887**

like GPT-based systems, for a variety of programming

tasks.

**768**

**00:50:15,037 --> 00:50:21,852**

These tasks include code completion, debugging, and

testing, where crafting effective prompts can significantly

**769**

**00:50:21,852 --> 00:50:24,510**

enhance productivity and ensure accuracy.

**770**

**00:50:24,660 --> 00:50:30,697**

This article delves into the importance of prompt

engineering, offering examples of how to create simple

**771**

**00:50:30,697 --> 00:50:34,041**

yet powerful prompts for different programming tasks.

**772**

**00:50:34,291 --> 00:50:37,356**

The Role of Prompt Engineering in Programming Tasks

**773**

**00:50:37,606 --> 00:50:44,119**

Prompt engineering refers to designing concise, clear,

and task-specific instructions to guide an AI model

**774**

**00:50:44,119 --> 00:50:46,290**

in generating the desired output.

**775**

**00:50:46,440 --> 00:50:52,524**

When working with LLMs, the prompt's quality directly

affects the response's usefulness and accuracy.

**776**

**00:50:52,674 --> 00:50:58,212**

This skill is particularly important in programming

for tasks such as code generation, debugging, and

**777**

**00:50:58,212 --> 00:50:59,454**

test case creation.

**778**

**00:50:59,604 --> 00:51:03,877**

Well-crafted prompts can save time, reduce errors,

and improve code quality.

**779**

**00:51:04,127 --> 00:51:06,263**

Crafting Prompts for Code Completion

**780**

**00:51:06,513 --> 00:51:11,819**

Code completion involves providing a partial code

snippet to the AI model and asking it to complete

**781**

**00:51:11,819 --> 00:51:12,550**

the rest.

**782**

**00:51:12,700 --> 00:51:17,402**

The prompt should be clear enough for the model to

understand the programming language and context in

**783**

**00:51:17,402 --> 00:51:18,691**

which the code is written.

**784**

**00:51:18,841 --> 00:51:21,906**

Here's how to craft a simple prompt for code completion:

**785**

**00:51:22,156 --> 00:51:24,942**

**Example 1: Completing a Function in Python**

**786**

**00:51:25,192 --> 00:51:34,735**

Prompt: Write a Python function called 'is\_prime'

that takes an integer as input and returns True if

**787**

**00:51:34,735 --> 00:51:37,545**

the number is a prime number and False otherwise.

**788**

**00:51:37,695 --> 00:51:41,550**

The function should check divisibility only up to

the square root of the number.

**789**

**00:51:41,800 --> 00:51:46,351**

This prompt's task is clear: complete the function

with the specified behavior.

**790**

**00:51:46,501 --> 00:51:53,119**

The prompt specifies the function's name (is\_prime),

its input (an integer), and the expected output (True

**791**

**00:51:53,119 --> 00:51:53,931**

or False).

**792**

**00:51:54,081 --> 00:51:59,364**

It also provides a hint about optimization (checking

up to the square root of the number), guiding the

**793**

**00:51:59,364 --> 00:52:01,465**

AI toward an efficient solution.

**794**

**00:52:01,715 --> 00:52:04,502**

A well-engineered prompt for code completion should:

**795**

**00:52:04,752 --> 00:52:05,309**

**1.**

**796**

**00:52:05,459 --> 00:52:09,453**

Clearly state the task (e.g., "Write a function that...").

**797**

**00:52:09,703 --> 00:52:10,260**

**2.**

**798**

**00:52:10,410 --> 00:52:17,376**

Specify the input/output types or behaviors (e.g.,

"takes an integer as input, returns True/False").

**799**

**00:52:17,626 --> 00:52:18,276**

**3.**

**800**

**00:52:18,426 --> 00:52:24,835**

Mention any performance considerations or constraints

(e.g., "check divisibility up to the square root").

**801**

**00:52:25,085 --> 00:52:27,825**

**Example 2: Completing a SQL Query**

**802**

**00:52:28,075 --> 00:52:34,170**

Prompt: Write a SQL query that selects the name and

age of employees from the table 'employees' who are

**803**

**00:52:34,170 --> 00:52:37,781**

**over 30 years old and sorts them in descending order**

by age.

**804**

**00:52:38,031 --> 00:52:42,164**

In this example, the prompt specifies the query's

goal and table structure.

**805**

**00:52:42,314 --> 00:52:48,943**

**By stating the condition (age greater than 30) and**

the desired output (name and age), the AI can effectively

**806**

**00:52:48,943 --> 00:52:50,162**

complete the task.

**807**

**00:52:50,412 --> 00:52:52,270**

Crafting Prompts for Debugging

**808**

**00:52:52,520 --> 00:52:55,771**

Debugging is another area where prompt engineering

shines.

**809**

**00:52:55,921 --> 00:53:02,004**

Here, the goal is to provide enough context for the

model to analyze the code and suggest potential fixes.

**810**

**00:53:02,154 --> 00:53:06,566**

A good debugging prompt typically includes the problem

description and the faulty code.

**811**

**00:53:09,167 --> 00:53:11,907**

Best Practices for Crafting Effective Prompts

**812**

**00:53:12,156 --> 00:53:12,714**

**1.**

**813**

**00:53:12,864 --> 00:53:17,461**

Be Specific: Clearly define the task, expected output,

and constraints.

**814**

**00:53:17,611 --> 00:53:23,451**

For example, instead of saying, "Write a function,"

specify the function name, input, output, and edge

**815**

**00:53:23,451 --> 00:53:24,206**

cases.

**816**

**00:53:24,456 --> 00:53:25,013**

**2.**

**817**

**00:53:25,163 --> 00:53:31,374**

Provide Context: When requesting debugging or code

improvements, include relevant portions of code and

**818**

**00:53:31,374 --> 00:53:33,754**

describe the issue or desired change.

**819**

**00:53:34,004 --> 00:53:34,655**

**3.**

**820**

**00:53:34,805 --> 00:53:40,633**

Keep It Concise: While providing enough detail is

essential, overly complex prompts can confuse the

**821**

**00:53:40,633 --> 00:53:41,260**

model.

**822**

**00:53:41,410 --> 00:53:44,661**

Focus on the key details necessary to complete the

task.

**823**

**00:53:44,911 --> 00:53:45,607**

**4.**

**824**

**00:53:45,757 --> 00:53:51,980**

Iterate: If the output isn't as expected, refine

the prompt by adding more details or rephrasing to

**825**

**00:53:51,980 --> 00:53:53,280**

clarify the task.

**826**

**00:53:53,530 --> 00:53:59,567**

Prompt engineering is robust in utilizing AI models

for programming tasks like code completion, debugging,

**827**

**00:53:59,567 --> 00:54:00,450**

and testing.

**828**

**00:54:00,600 --> 00:54:06,579**

By crafting clear, concise, and specific prompts,

developers can harness AI's full potential to generate

**829**

**00:54:06,579 --> 00:54:10,492**

accurate code, fix errors, and create robust test

cases.

**830**

**00:54:10,642 --> 00:54:16,551**

As AI models evolve, the ability to write effective

prompts will become even more critical in streamlining

**831**

**00:54:16,551 --> 00:54:19,372**

software development and improving productivity.

**832**

**00:54:19,622 --> 00:54:20,690**

Key Takeaways:

**833**

**00:54:20,940 --> 00:54:21,498**

**1.**

**834**

**00:54:21,648 --> 00:54:23,691**

Prompt Structure Overview:

**835**

**00:54:24,011 --> 00:54:28,632**

* · A prompt is an input designed to guide AI models

in generating relevant outputs.

**836**

**00:54:29,010 --> 00:54:34,315**

* · Key components include the task description, constraints,

and desired format for the response.

**837**

**00:54:34,565 --> 00:54:35,122**

**2.**

**838**

**00:54:35,272 --> 00:54:37,734**

Basic Prompt Syntax and Components:

**839**

**00:54:38,088 --> 00:54:42,256**

* · Task Definition: Clearly define what the AI is expected

to do.

**840**

**00:54:43,005 --> 00:54:48,450**

* · Input/Output Format: Specify the expected format

(e.g., code, text).

**841**

**00:54:48,781 --> 00:54:53,995**

* · Constraints: Provide limitations such as length,

language, or specific requirements.

**842**

**00:54:54,245 --> 00:54:54,895**

**3.**

**843**

**00:54:55,045 --> 00:54:57,320**

Crafting Prompts for Development Tasks:

**844**

**00:54:57,651 --> 00:55:02,307**

* · Code Completion: Use clear instructions and provide

partial code for completion.

**845**

**00:55:02,673 --> 00:55:07,758**

* · Debugging: Present problematic code and ask for

specific fixes or explanations.

**846**

**00:55:08,089 --> 00:55:13,628**

* · Testing: Instruct the AI to generate test cases

or suggest improvements to existing code.

**847**

**00:55:13,878 --> 00:55:14,574**

**4.**

**848**

**00:55:14,724 --> 00:55:16,628**

Techniques for Effective Prompts:

**849**

**00:55:17,041 --> 00:55:20,129**

* · Be specific and concise to improve model accuracy.

**850**

**00:55:20,472 --> 00:55:24,605**

* · Iteratively refine prompts based on model responses

to improve performance.

**851**

**00:55:26,405 --> 00:55:28,958**

**1.4 Case Studies and Best Practices**

**852**

**00:55:29,209 --> 00:55:33,342**

**1.4.1 Examples of Successful Prompt Engineering**

**853**

**00:55:33,592 --> 00:55:38,921**

Prompt engineering, the art of designing inputs to

guide language models effectively, has become a critical

**854**

**00:55:38,921 --> 00:55:40,790**

tool in development and AI.

**855**

**00:55:40,940 --> 00:55:46,745**

With the increasing deployment of AI models, prompt

engineering plays a pivotal role in extracting the

**856**

**00:55:46,745 --> 00:55:52,875**

best performance from these models in various domains,

such as natural language processing, software development,

**857**

**00:55:52,875 --> 00:55:54,500**

and automated systems.

**858**

**00:55:54,650 --> 00:56:01,303**

Here are some real-world examples and best practices

of successful prompt engineering in development, highlighting

**859**

**00:56:01,303 --> 00:56:03,985**

its role in optimizing AI workflows.

**860**

**00:56:04,235 --> 00:56:04,792**

**1.**

**861**

**00:56:04,942 --> 00:56:08,053**

Improving Customer Support Chatbots at Zendesk

**862**

**00:56:08,304 --> 00:56:15,142**

Zendesk, a customer service platform, has integrated

large language models (LLMs) to enhance its chatbot

**863**

**00:56:15,142 --> 00:56:15,920**

systems.

**864**

**00:56:16,070 --> 00:56:22,212**

Initially, these chatbots were rudimentary, relying

on keyword matching and predefined responses, limiting

**865**

**00:56:22,212 --> 00:56:24,429**

their ability to handle complex queries.

**866**

**00:56:24,579 --> 00:56:31,603**

With the introduction of AI models like GPT and Zendesk's

chatbots, they evolved into more dynamic systems capable

**867**

**00:56:31,603 --> 00:56:35,817**

of understanding nuanced queries and delivering personalized

responses.

**868**

**00:56:36,067 --> 00:56:42,534**

Prompt Engineering Use Case: Zendesk engineers fine-tuned

the model prompts to guide the chatbot's behavior,

**869**

**00:56:42,534 --> 00:56:46,052**

making them more responsive to customer sentiment

and context.

**870**

**00:56:46,202 --> 00:56:52,216**

By engineering prompts that included specific instructions

such as "provide a friendly and concise response"

**871**

**00:56:52,216 --> 00:56:57,812**

or "offer a step-by-step guide to the user," the chatbots'

performance improved dramatically.

**872**

**00:56:57,962 --> 00:57:04,406**

The specificity of these prompts allowed the chatbot

to deliver more helpful and coherent responses, improving

**873**

**00:57:04,406 --> 00:57:05,857**

customer satisfaction.

**874**

**00:57:06,107 --> 00:57:12,318**

Best Practice: When developing conversational agents,

ensure prompts guide the tone, context, and form of

**875**

**00:57:12,318 --> 00:57:13,259**

the responses.

**876**

**00:57:13,409 --> 00:57:19,876**

Including specific context, such as the user's problem

or the required resolution, helps the AI generate

**877**

**00:57:19,876 --> 00:57:21,675**

relevant and useful answers.

**878**

**00:57:21,925 --> 00:57:22,854**

Example:

**879**

**00:57:23,104 --> 00:57:28,479**

Let's walk through an example to understand prompt

engineering in the context of improving customer support

**880**

**00:57:28,479 --> 00:57:29,977**

chatbots at Zendesk.

**881**

**00:57:30,127 --> 00:57:35,839**

In this scenario, we want to refine prompts to guide

an AI-powered customer support chatbot to respond

**882**

**00:57:35,839 --> 00:57:37,789**

more effectively to user queries.

**883**

**00:57:38,039 --> 00:57:38,597**

**1.**

**884**

**00:57:38,747 --> 00:57:44,517**

Objective: We want the chatbot to handle common customer

queries, such as ticket status, troubleshooting, and

**885**

**00:57:44,517 --> 00:57:48,778**

account details, with better accuracy, speed, and

contextual understanding.

**886**

**00:57:49,028 --> 00:57:49,585**

**2.**

**887**

**00:57:49,735 --> 00:57:52,568**

Prompt Design Principles for Zendesk Chatbots:

**888**

**00:57:54,943 --> 00:57:59,366**

* · Be Specific: Define the context and intent of the

conversation clearly.

**889**

**00:57:59,686 --> 00:58:06,118**

* · Use Conversational Context: Refer to the past conversation

history or customer-specific information to improve

**890**

**00:58:06,118 --> 00:58:07,046**

responses.

**891**

**00:58:07,517 --> 00:58:13,403**

* · Set Constraints: Ensure responses are concise and

fit within the desired tone (e.g., professional but

**892**

**00:58:13,403 --> 00:58:14,123**

friendly).

**893**

**00:58:14,919 --> 00:58:21,583**

* · Dynamic Data: Use placeholders for personalized

customer responses based on available data (e.g.,

**894**

**00:58:21,583 --> 00:58:23,011**

ticket number, status).

**895**

**00:58:23,261 --> 00:58:23,911**

**3.**

**896**

**00:58:24,061 --> 00:58:26,615**

Basic Prompt Example for Ticket Status:

**897**

**00:58:26,865 --> 00:58:33,413**

Customer Query: "What is the status of my support

**ticket #12345?"**

**898**

**00:58:33,663 --> 00:58:35,428**

Basic Prompt for the Chatbot:

**899**

**00:58:35,678 --> 00:58:42,829**

"Respond to the user by retrieving the status of their

**support ticket #12345 from the database.**

**900**

**00:58:42,979 --> 00:58:45,812**

If the ticket is open, tell them it's being worked

on.

**901**

**00:58:45,962 --> 00:58:50,142**

If the ticket is closed, notify them that the issue

has been resolved."

**902**

**00:58:50,392 --> 00:58:51,321**

Components

**903**

**00:58:51,652 --> 00:58:54,868**

* · Context: Respond with the status of a support ticket.

**904**

**00:58:55,199 --> 00:58:57,997**

* · Input: Ticket number from the customer query.

**905**

**00:58:58,328 --> 00:59:01,823**

* · Constraints: Provide a friendly, professional response.

**906**

**00:59:02,177 --> 00:59:06,578**

* · Data Handling: Retrieve ticket data dynamically

from Zendesk's database.

**907**

**00:59:06,828 --> 00:59:07,896**

AI Response:

**908**

**00:59:08,146 --> 00:59:13,533**

**"Our team is currently working on your support ticket #12345.**

**909**

**00:59:13,683 --> 00:59:16,191**

We'll update you as soon as we have more information.

**910**

**00:59:16,341 --> 00:59:17,873**

Thank you for your patience!"

**911**

**00:59:18,123 --> 00:59:18,820**

**4.**

**912**

**00:59:18,970 --> 00:59:21,524**

Advanced Prompt Example with Troubleshooting:

**913**

**00:59:21,774 --> 00:59:24,700**

Customer Query: "I can't log into my account.

**914**

**00:59:24,850 --> 00:59:25,778**

Can you help me?"

**915**

**00:59:26,028 --> 00:59:27,886**

Advanced Prompt for the Chatbot:

**916**

**00:59:28,136 --> 00:59:32,780**

"Help the user troubleshoot their login issue by suggesting

the following steps.

**917**

**00:59:32,930 --> 00:59:35,949**

First, ask if they've tried resetting their password.

**918**

**00:59:36,099 --> 00:59:41,997**

Then, check for any active service outages or account

locks for the user based on their email address.

**919**

**00:59:42,147 --> 00:59:45,212**

Provide tailored solutions if any issues are found."

**920**

**00:59:45,462 --> 00:59:46,344**

Components:

**921**

**00:59:46,675 --> 00:59:49,891**

* · Context: Assist with troubleshooting a login issue.

**922**

**00:59:50,234 --> 00:59:53,020**

* · Input: Login issue details from the user.

**923**

**00:59:53,351 --> 00:59:59,029**

* · Constraints: Follow troubleshooting steps (password

reset, account lock, service outage).

**924**

**00:59:59,511 --> 01:00:04,016**

* · Data Handling: Use the customer's email to check

account-specific issues.

**925**

**01:00:04,266 --> 01:00:05,334**

AI Response:

**926**

**01:00:05,584 --> 01:00:08,324**

"I'm sorry to hear that you're having trouble logging

in.

**927**

**01:00:08,474 --> 01:00:10,285**

Have you tried resetting your password?

**928**

**01:00:10,435 --> 01:00:12,339**

If not, I can help you with that.

**929**

**01:00:12,489 --> 01:00:16,483**

Also, let me quickly check for any service issues

or your account is locked."

**930**

**01:00:18,284 --> 01:00:19,027**

**5.**

**931**

**01:00:19,177 --> 01:00:22,009**

Chained Prompts for Escalation to a Human Agent:

**932**

**01:00:22,259 --> 01:00:25,185**

Customer Query: "None of the solutions worked.

**933**

**01:00:25,335 --> 01:00:27,007**

Can I speak to a support agent?"

**934**

**01:00:27,257 --> 01:00:29,115**

Chained Prompt for the Chatbot:

**935**

**01:00:29,493 --> 01:00:33,637**

**· Step 1: Apologize for the inconvenience and offer**

to escalate the issue.

**936**

**01:00:34,003 --> 01:00:39,320**

**· Step 2: Collect any additional information (like**

error codes) to pass on to the support team.

**937**

**01:00:39,674 --> 01:00:42,961**

**· Step 3: Escalate the conversation to a human agent.**

**938**

**01:00:43,211 --> 01:00:44,418**

Prompt Example:

**939**

**01:00:44,668 --> 01:00:50,322**

"Apologize to the user, ask for additional details

about their issue, and notify them that the case will

**940**

**01:00:50,322 --> 01:00:52,191**

be escalated to a human agent.

**941**

**01:00:52,341 --> 01:00:56,010**

Pass along relevant troubleshooting details from

the conversation history."

**942**

**01:00:56,260 --> 01:00:57,328**

AI Response:

**943**

**01:00:57,578 --> 01:00:59,761**

"I'm sorry that none of the solutions worked.

**944**

**01:00:59,911 --> 01:01:05,588**

To help our support team resolve the issue quickly,

please provide any error messages or additional details

**945**

**01:01:05,588 --> 01:01:06,366**

you encountered.

**946**

**01:01:06,516 --> 01:01:10,278**

I'll escalate this case to one of our agents now,

and they'll be in touch shortly."

**947**

**01:01:10,528 --> 01:01:11,271**

**6.**

**948**

**01:01:11,421 --> 01:01:14,021**

Prompt with Customer Data Personalization:

**949**

**01:01:14,271 --> 01:01:20,587**

Scenario: A customer logs into Zendesk and asks about

their account details, such as subscription status

**950**

**01:01:20,587 --> 01:01:21,748**

or renewal date.

**951**

**01:01:21,998 --> 01:01:26,038**

Customer Query: "Can you tell me when my subscription

renews?"

**952**

**01:01:26,288 --> 01:01:28,146**

Personalized Prompt Example:

**953**

**01:01:28,396 --> 01:01:31,925**

"Retrieve the customer's subscription details based

on their account.

**954**

**01:01:32,075 --> 01:01:35,094**

If the subscription is active, respond with the renewal

date.

**955**

**01:01:35,244 --> 01:01:38,495**

If the subscription expires soon, I suggest renewing

it.

**956**

**01:01:38,645 --> 01:01:40,502**

Keep the tone friendly and helpful."

**957**

**01:01:40,752 --> 01:01:41,635**

Components:

**958**

**01:01:41,978 --> 01:01:45,136**

* · Context: Retrieve and provide subscription details.

**959**

**01:01:45,490 --> 01:01:48,683**

* · Input: Customer-specific data from their account.

**960**

**01:01:49,014 --> 01:01:51,626**

* · Constraints: Professional and friendly tone.

**961**

**01:01:51,876 --> 01:01:52,944**

AI Response:

**962**

**01:01:53,194 --> 01:01:58,071**

"Your subscription is active and will renew on October

**30th, 2024.**

**963**

**01:01:58,221 --> 01:02:01,889**

If you have any questions or want to make changes,

please let me know!"

**964**

**01:02:02,139 --> 01:02:02,882**

**7.**

**965**

**01:02:03,032 --> 01:02:06,005**

Using Prompt Variations for Improved Engagement:

**966**

**01:02:06,255 --> 01:02:12,385**

For customer support chatbots, it's important to vary

responses to keep the conversation engaging and natural.

**967**

**01:02:12,535 --> 01:02:16,018**

This can be achieved by introducing slight variations

in the prompts.

**968**

**01:02:16,268 --> 01:02:17,475**

Prompt Example:

**969**

**01:02:17,725 --> 01:02:20,558**

"When providing a ticket status, vary the response.

**970**

**01:02:20,708 --> 01:02:27,372**

For example, instead of always saying, 'Your ticket

is being worked on,' try other phrases like 'Our team

**971**

**01:02:27,372 --> 01:02:33,014**

is currently handling your request' or 'Your ticket

is in progress, and we'll update you soon.'"

**972**

**01:02:33,264 --> 01:02:34,611**

AI Variations:

**973**

**01:02:36,208 --> 01:02:40,388**

* · "Our team is currently handling your request and

will reach out soon with an update."

**974**

**01:02:40,708 --> 01:02:44,353**

* · "Your ticket is in progress, and we're working on

resolving the issue.

**975**

**01:02:44,503 --> 01:02:46,314**

We'll notify you once it's completed."

**976**

**01:02:46,564 --> 01:02:47,214**

**8.**

**977**

**01:02:47,364 --> 01:02:50,104**

Best Practices for Zendesk Chatbots:

**978**

**01:02:50,435 --> 01:02:55,880**

* · Use Clear and Direct Language: Avoid ambiguous or

overly technical language when addressing customer

**979**

**01:02:55,880 --> 01:02:56,716**

concerns.

**980**

**01:02:57,059 --> 01:03:04,095**

* · Contextual Awareness: Reference past interactions

or current issues based on conversation history or

**981**

**01:03:04,095 --> 01:03:07,276**

account details to improve the user experience.

**982**

**01:03:07,863 --> 01:03:17,069**

* · Handle Escalation Gracefully: Always allow customers

an option to escalate issues to human agents if they

**983**

**01:03:17,069 --> 01:03:19,740**

are unsatisfied with automated troubleshooting.

**984**

**01:03:20,083 --> 01:03:27,258**

* · Include Data Personalization: Leverage Zendesk's

CRM data to provide personalized responses that make

**985**

**01:03:27,258 --> 01:03:29,788**

the interaction more relevant to the customer.

**986**

**01:03:30,038 --> 01:03:36,040**

By crafting thoughtful and well-structured prompts,

Zendesk's customer support chatbots can become more

**987**

**01:03:36,040 --> 01:03:42,531**

effective in resolving issues, improving customer

satisfaction, and enhancing overall support efficiency.

**988**

**01:03:42,681 --> 01:03:48,625**

Developers and support teams can use prompt engineering

to fine-tune chatbot interactions, making them more

**989**

**01:03:48,625 --> 01:03:52,480**

context-aware, dynamic, and capable of handling complex

queries.

**990**

**01:03:53,130 --> 01:03:53,687**

**2.**

**991**

**01:03:53,837 --> 01:03:56,345**

Code Generation at GitHub Copilot

**992**

**01:03:56,595 --> 01:04:02,748**

GitHub Copilot is an AI-powered code completion tool

that suggests entire code blocks based on developer

**993**

**01:04:02,748 --> 01:04:03,421**

input.

**994**

**01:04:03,571 --> 01:04:09,422**

Copilot leverages models like OpenAI's Codex, which

can understand natural language prompts and generate

**995**

**01:04:09,422 --> 01:04:10,444**

code accordingly.

**996**

**01:04:10,694 --> 01:04:16,859**

Prompt Engineering Use Case: Engineers at GitHub used

prompt engineering to refine how Copilot interprets

**997**

**01:04:16,859 --> 01:04:18,032**

developer requests.

**998**

**01:04:18,182 --> 01:04:24,115**

The tool could provide more accurate and efficient

code completions by adjusting the prompts to include

**999**

**01:04:24,115 --> 01:04:30,674**

more detailed descriptions of the expected outcome

(e.g., "write a function that calculates the sum of

**1000**

**01:04:30,674 --> 01:04:31,835**

two integers").

**1001**

**01:04:31,985 --> 01:04:37,685**

Engineers also experimented with prompts referencing

specific libraries or programming paradigms, allowing

**1002**

**01:04:37,685 --> 01:04:41,598**

the AI to tailor its suggestions based on the development

environment.

**1003**

**01:04:41,848 --> 01:04:47,468**

Best Practice: Clear and detailed prompts are key

when working with AI in code generation.

**1004**

**01:04:47,618 --> 01:04:53,400**

Providing context about the desired function, expected

inputs/outputs, and even specific code structures

**1005**

**01:04:53,400 --> 01:04:56,859**

or libraries ensures the AI generates more usable

code.

**1006**

**01:04:57,009 --> 01:05:01,560**

Iteratively refining prompts based on feedback from

generated code helps improve results.

**1007**

**01:05:01,810 --> 01:05:02,739**

Example:

**1008**

**01:05:02,989 --> 01:05:09,212**

Here's an example to understand prompt engineering

in the context of Code Generation at GitHub Copilot.

**1009**

**01:05:09,362 --> 01:05:16,885**

It shows how to guide AI models like Copilot to generate

accurate, useful, and context-aware code for developers.

**1010**

**01:05:18,685 --> 01:05:19,242**

**1.**

**1011**

**01:05:19,392 --> 01:05:20,181**

Objective:

**1012**

**01:05:20,431 --> 01:05:27,095**

The goal is to help developers use GitHub Copilot

effectively to generate code snippets, automate repetitive

**1013**

**01:05:27,095 --> 01:05:29,766**

tasks, or build specific functions.

**1014**

**01:05:29,916 --> 01:05:35,999**

By carefully crafting prompts, we can guide Copilot

in generating code that fits the project requirements.

**1015**

**01:05:36,649 --> 01:05:37,207**

**2.**

**1016**

**01:05:37,357 --> 01:05:40,004**

Prompt Design Principles for Code Generation:

**1017**

**01:05:40,707 --> 01:05:46,616**

* · Clear Intent: Specify what you want the AI to do

(e.g., function name, functionality).

**1018**

**01:05:46,970 --> 01:05:53,182**

* · Contextual Information: Provide enough project context,

including relevant comments or code snippets.

**1019**

**01:05:53,525 --> 01:06:00,630**

* · Constraints: Set boundaries like coding best practices,

performance optimizations, or using specific libraries.

**1020**

**01:06:01,971 --> 01:06:08,496**

* · Dynamic Placeholders: Allow Copilot to fill in key

parts of the code based on the context, such as variable

**1021**

**01:06:08,496 --> 01:06:10,029**

names or input types.

**1022**

**01:06:10,679 --> 01:06:11,329**

**3.**

**1023**

**01:06:11,479 --> 01:06:14,265**

Basic Prompt for Generating a Python Function:

**1024**

**01:06:14,515 --> 01:06:20,157**

Scenario: You are building a Python project and need

a function to calculate the factorial of a number

**1025**

**01:06:20,157 --> 01:06:21,295**

using recursion.

**1026**

**01:06:21,545 --> 01:06:28,233**

Prompt Example: "Write a Python function called factorial

that recursively calculates the factorial of a number."

**1027**

**01:06:28,483 --> 01:06:29,365**

Components:

**1028**

**01:06:29,696 --> 01:06:32,169**

* · Context: Generate a Python function.

**1029**

**01:06:32,500 --> 01:06:35,484**

* · Input: The task of calculating a factorial.

**1030**

**01:06:35,815 --> 01:06:37,731**

* · Constraints: Use recursion.

**1031**

**01:06:38,381 --> 01:06:39,078**

**4.**

**1032**

**01:06:39,228 --> 01:06:41,503**

Advanced Prompt with Multiple Constraints:

**1033**

**01:06:41,753 --> 01:06:48,533**

Scenario: You're working on a Node.js project and

need a function that fetches data from an API using

**1034**

**01:06:48,533 --> 01:06:50,948**

Axios and handles errors gracefully.

**1035**

**01:06:51,098 --> 01:06:55,557**

You also want the function to retry fetching the

data three times in case of failure.

**1036**

**01:06:55,807 --> 01:07:02,727**

Prompt Example: "Write a Node.js function named fetchData

that uses the axios library to make a GET request

**1037**

**01:07:02,727 --> 01:07:03,841**

to an API.

**1038**

**01:07:03,991 --> 01:07:08,124**

If the request fails, retry up to three times before

throwing an error.

**1039**

**01:07:08,274 --> 01:07:10,132**

Handle all errors gracefully."

**1040**

**01:07:10,382 --> 01:07:11,264**

Components:

**1041**

**01:07:11,758 --> 01:07:14,486**

* · Context: Generate a Node.js function.

**1042**

**01:07:14,817 --> 01:07:17,662**

* · Input: API data fetching using Axios.

**1043**

**01:07:17,912 --> 01:07:21,999**

**Constraints: Retry mechanism (3 times), error handling.**

**1044**

**01:07:22,649 --> 01:07:23,392**

**5.**

**1045**

**01:07:23,542 --> 01:07:26,003**

Prompt for Context-Aware Code Generation:

**1046**

**01:07:26,253 --> 01:07:32,186**

Scenario: You are working on a React project and need

a function that takes a list of users and filters

**1047**

**01:07:32,186 --> 01:07:33,915**

**out users under 18.**

**1048**

**01:07:34,065 --> 01:07:37,781**

The users' data is an array of objects with name

and age properties.

**1049**

**01:07:38,031 --> 01:07:44,242**

Prompt Example: "Write a React component function

that filters an array of users only to include users

**1050**

**01:07:44,242 --> 01:07:46,622**

**who are 18 years old or older.**

**1051**

**01:07:46,772 --> 01:07:50,487**

The array of users contains objects with name and

age properties."

**1052**

**01:07:50,737 --> 01:07:51,620**

Components:

**1053**

**01:07:51,951 --> 01:07:56,188**

* · Context: Generate a React component function.

**1054**

**01:07:56,508 --> 01:07:58,807**

* · Input: An array of user objects.

**1055**

**01:07:59,150 --> 01:08:01,472**

* · Constraints: Filter based on age.

**1056**

**01:08:02,522 --> 01:08:03,265**

**6.**

**1057**

**01:08:03,415 --> 01:08:05,087**

Prompt for Code Refactoring:

**1058**

**01:08:05,337 --> 01:08:12,442**

Scenario: You have an existing Python function that

performs a database query but is poorly optimized.

**1059**

**01:08:12,592 --> 01:08:18,989**

You want Copilot to refactor it to improve performance

and follow best practices, such as using prepared

**1060**

**01:08:18,989 --> 01:08:21,323**

statements and closing database connections.

**1061**

**01:08:21,573 --> 01:08:26,820**

Prompt Example: "Refactor this Python function to

improve performance and security.

**1062**

**01:08:26,970 --> 01:08:32,392**

Use prepared statements for the database query and

ensure the connection is properly closed after the

**1063**

**01:08:32,392 --> 01:08:33,054**

query."

**1064**

**01:08:34,855 --> 01:08:35,598**

**7.**

**1065**

**01:08:35,748 --> 01:08:37,652**

Prompt for Test Case Generation:

**1066**

**01:08:37,902 --> 01:08:43,196**

Scenario: You've written a Python function and need

test cases to verify its functionality.

**1067**

**01:08:43,346 --> 01:08:46,457**

The function calculates the sum of all numbers in

a list.

**1068**

**01:08:46,707 --> 01:08:53,011**

Prompt Example: "Write unit tests using Python's unittest

framework for the following function that calculates

**1069**

**01:08:53,011 --> 01:08:54,695**

the sum of numbers in a list."

**1070**

**01:08:55,345 --> 01:08:56,227**

Components:

**1071**

**01:08:56,570 --> 01:08:59,914**

* · Context: Generate test cases for the provided function.

**1072**

**01:09:00,234 --> 01:09:04,204**

* · Input: A Python function calculates the sum of numbers

in a list.

**1073**

**01:09:04,547 --> 01:09:07,194**

* · Constraints: Use the unittest framework.

**1074**

**01:09:07,844 --> 01:09:11,420**

Best Practices for Code Generation with GitHub Copilot:

**1075**

**01:09:13,481 --> 01:09:19,611**

* · Be Specific: Clearly define the function's purpose,

including its name, expected behavior, and edge cases.

**1076**

**01:09:19,931 --> 01:09:26,270**

* · Provide Enough Context: If the code is part of a

larger project, provide additional context, such as

**1077**

**01:09:26,270 --> 01:09:28,685**

related functions or libraries being used.

**1078**

**01:09:29,051 --> 01:09:35,657**

* · Use Constraints: Define constraints such as performance

optimizations, security measures (like using prepared

**1079**

**01:09:35,657 --> 01:09:39,662**

statements), or coding style (such as using a specific

framework).

**1080**

**01:09:40,307 --> 01:09:45,798**

* · Leverage Comments: Use comments in your code to

guide Copilot on what each section of the code should

**1081**

**01:09:45,798 --> 01:09:46,646**

accomplish.

**1082**

**01:09:47,674 --> 01:09:54,907**

* · Dynamic Placeholders: Utilize placeholders for variable

names, parameters, or values that Copilot can infer

**1083**

**01:09:54,907 --> 01:09:57,531**

based on the context of the surrounding code.

**1084**

**01:09:57,781 --> 01:10:03,807**

Through thoughtful, prompt engineering, developers

can guide GitHub Copilot in generating code tailored

**1085**

**01:10:03,807 --> 01:10:04,747**

to their needs.

**1086**

**01:10:04,897 --> 01:10:10,679**

Whether writing new functions, refactoring existing

code, or generating test cases, providing clear and

**1087**

**01:10:10,679 --> 01:10:16,089**

detailed prompts can improve the quality of Copilot's

code suggestions, making it a more powerful development

**1088**

**01:10:16,089 --> 01:10:16,693**

tool.

**1089**

**01:10:16,943 --> 01:10:17,593**

**3.**

**1090**

**01:10:17,743 --> 01:10:20,715**

**Content Moderation at OpenAI's GPT-4**

**1091**

**01:10:20,965 --> 01:10:27,258**

OpenAI has utilized prompt engineering for content

moderation systems that detect harmful or inappropriate

**1092**

**01:10:27,258 --> 01:10:28,070**

content.

**1093**

**01:10:28,220 --> 01:10:33,433**

Given this application's sensitivity, the prompts

must be carefully designed to balance flagging harmful

**1094**

**01:10:33,433 --> 01:10:35,836**

content and avoiding false positives.

**1095**

**01:10:36,086 --> 01:10:42,193**

Prompt Engineering Use Case: By designing prompts

that asked the model to "analyze the content based

**1096**

**01:10:42,193 --> 01:10:49,066**

on specific community guidelines," engineers ensured

**that GPT-4 would flag content that violated terms**

**1097**

**01:10:49,066 --> 01:10:50,808**

of service more accurately.

**1098**

**01:10:50,958 --> 01:10:57,599**

Including prompts that categorized violations (e.g.,

"identify hate speech" or "detect offensive language")

**1099**

**01:10:57,599 --> 01:11:00,757**

enabled more precise content filtering and moderation.

**1100**

**01:11:01,007 --> 01:11:07,741**

Best Practice: When developing AI systems for sensitive

applications like content moderation, it's important

**1101**

**01:11:07,741 --> 01:11:12,942**

to provide detailed instructions in prompts that clearly

outline the boundaries of acceptable behavior.

**1102**

**01:11:13,092 --> 01:11:18,642**

Tailoring prompts to specific use cases, such as

different types of violations, helps improve the precision

**1103**

**01:11:18,642 --> 01:11:19,965**

of the AI's judgments.

**1104**

**01:11:20,215 --> 01:11:21,144**

Example:

**1105**

**01:11:21,394 --> 01:11:27,942**

Here's an example of prompt engineering in the context

**of Content Moderation at OpenAI's GPT-4.**

**1106**

**01:11:28,092 --> 01:11:34,396**

**This example shows how to guide GPT-4 to moderate**

content, ensuring user inputs and outputs adhere to

**1107**

**01:11:34,396 --> 01:11:39,098**

community guidelines, prevent harmful content, and

ensure responsible AI use.

**1108**

**01:11:39,348 --> 01:11:39,905**

**1.**

**1109**

**01:11:40,055 --> 01:11:47,114**

Objective: The goal is to develop a content moderation

**system using GPT-4 that can accurately classify and**

**1110**

**01:11:47,114 --> 01:11:53,941**

filter out harmful, offensive, or inappropriate content

while allowing appropriate content to pass through.

**1111**

**01:11:54,091 --> 01:11:58,828**

**By refining prompts, we can guide GPT-4 to handle**

moderation tasks efficiently.

**1112**

**01:11:59,078 --> 01:11:59,635**

**2.**

**1113**

**01:11:59,785 --> 01:12:02,525**

Prompt Design Principles for Content Moderation:

**1114**

**01:12:02,879 --> 01:12:08,893**

* · Explicit Instructions: Clearly define what types

of content should be flagged (e.g., hate speech, violence,

**1115**

**01:12:08,893 --> 01:12:10,159**

explicit material).

**1116**

**01:12:10,479 --> 01:12:17,967**

* · Contextual Awareness: Ensure the model understands

the content context (e.g., satire, educational material)

**1117**

**01:12:17,967 --> 01:12:19,604**

to avoid false positives.

**1118**

**01:12:19,947 --> 01:12:25,102**

* · Dynamic Handling: Adjust responses based on severity

(e.g., warning vs.

**1119**

**01:12:25,252 --> 01:12:25,948**

block).

**1120**

**01:12:26,349 --> 01:12:32,003**

* · Consistent Feedback: Provide detailed explanations

or alternatives when content is flagged.

**1121**

**01:12:32,253 --> 01:12:32,903**

**3.**

**1122**

**01:12:33,053 --> 01:12:36,258**

Basic Prompt for Identifying Inappropriate Content:

**1123**

**01:12:36,508 --> 01:12:42,405**

**Scenario: A platform uses GPT-4 to automatically moderate**

user comments on a forum.

**1124**

**01:12:42,555 --> 01:12:48,175**

The AI should flag any comment that contains hate

speech, explicit content, or threats of violence.

**1125**

**01:12:48,425 --> 01:12:54,242**

Basic Prompt Example: "Analyze the following comment

and determine whether it contains hate speech, explicit

**1126**

**01:12:54,242 --> 01:12:55,948**

content, or violent threats.

**1127**

**01:12:56,098 --> 01:13:02,518**

If it does, respond with 'Flagged: inappropriate

content detected.' If not, respond with 'Approved:

**1128**

**01:13:02,518 --> 01:13:04,132**

content is appropriate.'"

**1129**

**01:13:04,382 --> 01:13:08,887**

Comment Input: "I can't believe people from [ethnicity]

are allowed here.

**1130**

**01:13:09,037 --> 01:13:11,266**

They're so dumb and should all be kicked out."

**1131**

**01:13:11,516 --> 01:13:12,398**

Components:

**1132**

**01:13:12,741 --> 01:13:15,945**

* · Context: Content moderation for a user comment.

**1133**

**01:13:16,265 --> 01:13:19,678**

* · Input: A comment that might contain harmful content.

**1134**

**01:13:20,009 --> 01:13:24,572**

* · Constraints: Detect hate speech, explicit content,

or violent threats.

**1135**

**01:13:24,822 --> 01:13:29,931**

**GPT-4 Response: "Flagged: inappropriate content detected**

(contains hate speech)."

**1136**

**01:13:31,731 --> 01:13:32,427**

**4.**

**1137**

**01:13:32,577 --> 01:13:35,503**

Advanced Prompt for Nuanced Content Moderation:

**1138**

**01:13:35,753 --> 01:13:42,254**

Scenario: Some content might not be inherently harmful

but could be offensive in specific contexts.

**1139**

**01:13:42,404 --> 01:13:47,699**

AI should determine whether content is harmful based

on the context and explain it when necessary.

**1140**

**01:13:47,949 --> 01:13:54,683**

Advanced Prompt Example: "Analyze the text for inappropriate

content, including hate speech, explicit content,

**1141**

**01:13:54,683 --> 01:13:56,958**

violence, or harmful misinformation.

**1142**

**01:13:57,108 --> 01:14:02,263**

Consider the context (e.g., satire, humor, or educational

purpose).

**1143**

**01:14:02,413 --> 01:14:06,035**

If inappropriate, explain why and suggest a safer

alternative.

**1144**

**01:14:06,185 --> 01:14:08,461**

If appropriate, respond with 'Approved'."

**1145**

**01:14:08,711 --> 01:14:13,169**

Content Input: "If you don't agree with the government,

you should just overthrow it."

**1146**

**01:14:13,419 --> 01:14:14,301**

Components:

**1147**

**01:14:14,632 --> 01:14:19,288**

* · Context: Moderate based on context, especially for

satire or humor.

**1148**

**01:14:19,619 --> 01:14:22,417**

* · Input: A politically charged statement.

**1149**

**01:14:22,748 --> 01:14:26,847**

* · Constraints: Detect violent incitement and provide

alternatives.

**1150**

**01:14:27,097 --> 01:14:31,834**

**GPT-4 Response: "Flagged: inappropriate content detected**

(incites violence).

**1151**

**01:14:31,984 --> 01:14:37,220**

Suggested alternative: 'If you don't agree with the

government, you can voice your concerns through peaceful

**1152**

**01:14:37,220 --> 01:14:38,950**

protests or elections.'"

**1153**

**01:14:39,200 --> 01:14:39,943**

**5.**

**1154**

**01:14:40,093 --> 01:14:42,786**

Prompt for Educational Context Moderation:

**1155**

**01:14:43,036 --> 01:14:48,470**

Scenario: A user posts a comment about historical

violence in a classroom discussion forum.

**1156**

**01:14:48,620 --> 01:14:53,403**

**GPT-4 should determine if it's appropriate based**

on the educational context.

**1157**

**01:14:53,653 --> 01:14:59,771**

Prompt Example: "Evaluate whether the following content,

which discusses historical events, is appropriate

**1158**

**01:14:59,771 --> 01:15:01,548**

in an educational context.

**1159**

**01:15:01,698 --> 01:15:07,364**

If it includes graphic violence or sensitive topics

but is relevant to the subject matter, allow it.

**1160**

**01:15:07,514 --> 01:15:09,696**

Otherwise, flag it as inappropriate."

**1161**

**01:15:09,946 --> 01:15:16,575**

Content Input: "During the French Revolution, many

people were executed by guillotine, a method of execution

**1162**

**01:15:16,575 --> 01:15:18,863**

that was designed to be quick and efficient."

**1163**

**01:15:19,113 --> 01:15:19,995**

Components:

**1164**

**01:15:20,338 --> 01:15:23,774**

* · Context: Educational discussion of historical events.

**1165**

**01:15:24,094 --> 01:15:28,018**

* · Input: A fact-based historical statement that mentions

violence.

**1166**

**01:15:28,372 --> 01:15:33,702**

* · Constraints: Allow relevant discussion while monitoring

graphic or inappropriate content.

**1167**

**01:15:33,952 --> 01:15:40,639**

**GPT-4 Response: "Approved: content is appropriate**

(educational context discussing historical events)."

**1168**

**01:15:41,689 --> 01:15:42,432**

**6.**

**1169**

**01:15:42,582 --> 01:15:44,626**

Prompt for Handling Misinformation:

**1170**

**01:15:44,876 --> 01:15:50,216**

Scenario: A user submits content that potentially

contains harmful misinformation.

**1171**

**01:15:50,366 --> 01:15:56,171**

**GPT-4 should identify whether the content contains**

false claims and suggest an accurate alternative.

**1172**

**01:15:56,421 --> 01:16:00,740**

Prompt Example: "Review the following text for potential

misinformation.

**1173**

**01:16:00,890 --> 01:16:06,509**

If the content contains false or misleading information,

explain why and provide a factual correction.

**1174**

**01:16:06,659 --> 01:16:09,492**

If the content is accurate, respond with 'Approved'."

**1175**

**01:16:09,742 --> 01:16:14,665**

Content Input: "Vaccines cause autism and are unsafe

for children."

**1176**

**01:16:14,915 --> 01:16:15,797**

Components:

**1177**

**01:16:16,477 --> 01:16:20,552**

* · Context: Misinformation moderation for health-related

content.

**1178**

**01:16:20,872 --> 01:16:24,238**

* · Input: A statement containing a widely debunked

claim.

**1179**

**01:16:24,616 --> 01:16:29,411**

* · Constraints: Detect false information and provide

accurate alternatives.

**1180**

**01:16:29,661 --> 01:16:34,816**

**GPT-4 Response: "Flagged: inappropriate content detected**

(contains misinformation).

**1181**

**01:16:34,966 --> 01:16:39,006**

Correction: 'Vaccines have been extensively studied

and shown to be safe.

**1182**

**01:16:39,156 --> 01:16:42,825**

There is no scientific evidence linking vaccines

to autism.'"

**1183**

**01:16:43,075 --> 01:16:43,818**

**7.**

**1184**

**01:16:43,968 --> 01:16:46,290**

Prompt for Escalation and Human Review:

**1185**

**01:16:46,540 --> 01:16:50,441**

Scenario: Some content might be borderline and difficult

to judge.

**1186**

**01:16:50,591 --> 01:16:56,210**

AI should flag this content for human review if it's

unclear rather than making an automatic decision.

**1187**

**01:16:56,460 --> 01:16:59,386**

Prompt Example: "Analyze the following content.

**1188**

**01:16:59,536 --> 01:17:05,724**

If you are unsure whether it's appropriate or inappropriate

based on context, flag it for human review with the

**1189**

**01:17:05,724 --> 01:17:08,011**

note 'Uncertain: requires human review'.

**1190**

**01:17:08,602 --> 01:17:10,367**

If it's inappropriate, flag it.

**1191**

**01:17:10,517 --> 01:17:12,235**

If it's appropriate, approve it."

**1192**

**01:17:12,485 --> 01:17:18,731**

Content Input: "I think we need to talk about sensitive

issues like mental health more openly, but sometimes

**1193**

**01:17:18,731 --> 01:17:22,005**

people just use it as an excuse to avoid responsibilities."

**1194**

**01:17:22,255 --> 01:17:23,138**

Components:

**1195**

**01:17:23,469 --> 01:17:26,406**

* · Context: Handle sensitive topics with nuance.

**1196**

**01:17:26,726 --> 01:17:29,211**

* · Input: A comment discussing mental health.

**1197**

**01:17:29,542 --> 01:17:33,129**

* · Constraints: Flag uncertain content for human review.

**1198**

**01:17:33,379 --> 01:17:39,532**

**GPT-4 Response: "Uncertain: requires human review**

(discussion of mental health could be interpreted

**1199**

**01:17:39,532 --> 01:17:40,717**

in multiple ways)."

**1200**

**01:17:40,967 --> 01:17:41,617**

**8.**

**1201**

**01:17:41,767 --> 01:17:45,621**

Best Practices for Content Moderation with GPT-4:

**1202**

**01:17:46,092 --> 01:17:53,046**

* · Provide Clear Definitions: To avoid ambiguity, specify

what types of content are considered harmful (e.g.,

**1203**

**01:17:53,046 --> 01:17:55,438**

hate speech, violence, misinformation).

**1204**

**01:17:55,781 --> 01:18:02,886**

* · Leverage Context: Context is critical when moderating

content (e.g., educational discussions, satire).

**1205**

**01:18:03,036 --> 01:18:07,077**

Make sure prompts are designed to handle context-sensitive

moderation.

**1206**

**01:18:07,803 --> 01:18:13,759**

* · Use Escalation Mechanisms: When the AI is unsure,

it's important to have a fallback mechanism, such

**1207**

**01:18:13,759 --> 01:18:16,011**

as flagging content for human review.

**1208**

**01:18:16,807 --> 01:18:22,832**

* · Incorporate Dynamic Feedback: When flagging content,

provide alternative suggestions or corrections to

**1209**

**01:18:22,832 --> 01:18:24,759**

keep the interaction constructive.

**1210**

**01:18:25,137 --> 01:18:32,381**

* · Avoid Over-Moderation: Ensure that the moderation

system doesn't over-censor content by clearly distinguishing

**1211**

**01:18:32,381 --> 01:18:34,669**

between harmful and harmless content.

**1212**

**01:18:36,469 --> 01:18:42,134**

**Through effective, prompt engineering, GPT-4 can be**

a powerful tool for content moderation.

**1213**

**01:18:42,284 --> 01:18:48,983**

By crafting prompts that include context, constraints,

**and dynamic responses, GPT-4 can accurately moderate**

**1214**

**01:18:48,983 --> 01:18:54,834**

user-generated content, identify harmful material,

and suggest safer alternatives while handling complex

**1215**

**01:18:54,834 --> 01:18:56,077**

cases with nuance.

**1216**

**01:18:56,727 --> 01:18:57,423**

**4.**

**1217**

**01:18:57,573 --> 01:19:00,870**

Documentation Generation in Microsoft's Power Platform

**1218**

**01:19:01,121 --> 01:19:07,576**

Microsoft's Power Platform integrates AI tools to

assist users in automating workflows and generating

**1219**

**01:19:07,576 --> 01:19:09,712**

documentation for business processes.

**1220**

**01:19:09,862 --> 01:19:16,793**

With the addition of AI-driven documentation generators,

users can input high-level descriptions of their processes,

**1221**

**01:19:16,793 --> 01:19:21,472**

and the system generates detailed documents, complete

with workflows and action steps.

**1222**

**01:19:21,722 --> 01:19:27,701**

Prompt Engineering Use Case: Engineers at Microsoft

refined prompts to improve the clarity and usefulness

**1223**

**01:19:27,701 --> 01:19:29,385**

of generated documentation.

**1224**

**01:19:29,535 --> 01:19:36,582**

For example, asking the AI to "describe the workflow

in simple terms for a non-technical audience" ensured

**1225**

**01:19:36,582 --> 01:19:42,306**

that the resulting documentation was accessible to

business users who lacked technical expertise.

**1226**

**01:19:42,456 --> 01:19:49,050**

Further, prompts that specified the format, such

as "generate a step-by-step guide," improved the structure

**1227**

**01:19:49,050 --> 01:19:51,279**

and readability of the documentation.

**1228**

**01:19:51,529 --> 01:19:57,415**

Best Practice: In automated documentation systems,

prompts should be tailored to the target audience's

**1229**

**01:19:57,555 --> 01:19:58,913**

level of expertise.

**1230**

**01:19:59,063 --> 01:20:05,147**

Including instructions about format, tone, and complexity

helps the AI generate content more aligned with user

**1231**

**01:20:05,147 --> 01:20:06,215**

expectations.

**1232**

**01:20:06,465 --> 01:20:07,394**

Example:

**1233**

**01:20:07,644 --> 01:20:13,623**

Here's an example to understand prompt engineering

in the context of Document Generation in Microsoft's

**1234**

**01:20:13,704 --> 01:20:14,703**

Power Platform.

**1235**

**01:20:14,853 --> 01:20:20,867**

It shows how to guide an AI model to automatically

generate high-quality documentation for apps, workflows,

**1236**

**01:20:20,867 --> 01:20:26,184**

and automation created within the Power Platform (Power

Apps, Power Automate, Power BI).

**1237**

**01:20:26,434 --> 01:20:26,991**

**1.**

**1238**

**01:20:27,141 --> 01:20:32,876**

Objective: The goal is to help users of the Power

Platform generate detailed, clear, and structured

**1239**

**01:20:32,876 --> 01:20:36,104**

documentation automatically for their apps or workflows.

**1240**

**01:20:36,254 --> 01:20:42,233**

Well-engineered prompts can guide the AI to document

the steps, logic, data flows, and key features of

**1241**

**01:20:42,233 --> 01:20:45,449**

the app, making it easy to understand and maintain.

**1242**

**01:20:45,699 --> 01:20:46,256**

**2.**

**1243**

**01:20:46,406 --> 01:20:49,471**

Prompt Design Principles for Documentation Generation:

**1244**

**01:20:49,837 --> 01:20:57,175**

* · Structured Output: Guide the AI to generate documentation

with a clear structure (e.g., overview, purpose, key

**1245**

**01:20:57,175 --> 01:20:58,174**

components).

**1246**

**01:20:58,517 --> 01:21:04,542**

* · Include Context: Provide the AI with the app's context

or workflow details to tailor the documentation to

**1247**

**01:21:04,542 --> 01:21:05,947**

the actual content.

**1248**

**01:21:06,592 --> 01:21:11,398**

* · Dynamic Data: Use placeholders to capture app-specific

data and logic.

**1249**

**01:21:11,729 --> 01:21:18,893**

* · Concise but Detailed: Ensure the AI generates documentation

that is not overly verbose but contains enough detail

**1250**

**01:21:18,893 --> 01:21:21,493**

to be useful for developers and stakeholders.

**1251**

**01:21:21,743 --> 01:21:22,393**

**3.**

**1252**

**01:21:22,543 --> 01:21:25,655**

Basic Prompt for Documenting a Power Automate Flow:

**1253**

**01:21:25,905 --> 01:21:31,989**

Scenario: You've created an automation in Power Automate

that sends an approval request when a new item is

**1254**

**01:21:31,989 --> 01:21:33,568**

added to a SharePoint list.

**1255**

**01:21:33,718 --> 01:21:37,108**

You need to generate a detailed overview of your

team's workflow.

**1256**

**01:21:37,358 --> 01:21:43,569**

Prompt Example: "Generate documentation for a Power

Automate workflow that sends an approval request when

**1257**

**01:21:43,569 --> 01:21:45,903**

a new item is added to a SharePoint list.

**1258**

**01:21:46,053 --> 01:21:51,625**

The documentation should include a title, purpose,

trigger, steps involved and expected outcomes."

**1259**

**01:21:51,875 --> 01:21:52,758**

Components:

**1260**

**01:21:53,112 --> 01:21:55,190**

* · Context: A Power Automate flow.

**1261**

**01:21:55,521 --> 01:21:59,992**

* · Input: Details about the automation logic (trigger,

approval request).

**1262**

**01:22:00,358 --> 01:22:04,142**

* · Constraints: Structure the documentation in a standard

format.

**1263**

**01:22:04,392 --> 01:22:06,296**

AI-Generated Documentation

**1264**

**01:22:06,547 --> 01:22:09,519**

Title: SharePoint Item Approval Workflow

**1265**

**01:22:09,769 --> 01:22:15,864**

Purpose: This Power Automate workflow is designed

to streamline the approval process for new items added

**1266**

**01:22:15,864 --> 01:22:21,809**

to the SharePoint list titled "Project Submissions."

The workflow automates the process by sending an approval

**1267**

**01:22:21,809 --> 01:22:25,651**

request to designated approvers when a new item is

added.

**1268**

**01:22:25,901 --> 01:22:32,078**

Trigger: The workflow is triggered when a new item

is created in the SharePoint list "Project Submissions."

**1269**

**01:22:32,328 --> 01:22:33,396**

Steps Involved:

**1270**

**01:22:33,646 --> 01:22:34,203**

**1.**

**1271**

**01:22:34,353 --> 01:22:38,486**

Trigger: When a new item is created in the "Project

Submissions" list.

**1272**

**01:22:38,736 --> 01:22:39,293**

**2.**

**1273**

**01:22:39,443 --> 01:22:46,107**

Approval Request: The flow sends an approval request

to the assigned approvers (Team Manager and Project

**1274**

**01:22:46,107 --> 01:22:49,428**

Lead) using the "Start and wait for an approval" action.

**1275**

**01:22:49,678 --> 01:22:50,328**

**3.**

**1276**

**01:22:50,478 --> 01:22:55,668**

Condition: If the approval is granted, the item's

status is updated to "Approved" in the SharePoint

**1277**

**01:22:55,668 --> 01:22:56,237**

list.

**1278**

**01:22:56,487 --> 01:22:57,183**

**4.**

**1279**

**01:22:57,333 --> 01:23:02,767**

Notification: A confirmation email is sent to the

item creator, informing them of the approval.

**1280**

**01:23:03,017 --> 01:23:03,760**

**5.**

**1281**

**01:23:03,910 --> 01:23:10,307**

Rejection Path: If the request is rejected, the status

is updated to "Rejected," and the item creator is

**1282**

**01:23:10,307 --> 01:23:12,501**

notified of the decision via email.

**1283**

**01:23:12,751 --> 01:23:18,649**

Expected Outcomes: The workflow automates the item

approval process, reducing manual effort and ensuring

**1284**

**01:23:18,649 --> 01:23:20,692**

all new submissions are reviewed promptly.

**1285**

**01:23:22,493 --> 01:23:23,189**

**4.**

**1286**

**01:23:23,339 --> 01:23:26,172**

Advanced Prompt for Documenting a Power App:

**1287**

**01:23:26,422 --> 01:23:30,276**

Scenario: You've built a Power App to track employee

expenses.

**1288**

**01:23:30,426 --> 01:23:36,417**

You want to generate detailed technical documentation

for the app, including data sources, key UI components,

**1289**

**01:23:36,417 --> 01:23:38,182**

logic, and formulas used.

**1290**

**01:23:38,432 --> 01:23:44,051**

Prompt Example: "Generate technical documentation

for a Power App to track employee expenses.

**1291**

**01:23:44,201 --> 01:23:50,981**

The documentation should include an overview, data

sources, key screens and UI components, business logic,

**1292**

**01:23:50,981 --> 01:23:54,604**

formulas used, and integration points with other systems."

**1293**

**01:23:54,854 --> 01:23:55,736**

Components:

**1294**

**01:23:56,067 --> 01:23:59,283**

* · Context: A Power App for tracking expenses.

**1295**

**01:23:59,614 --> 01:24:03,248**

* · Input: The app's features (data sources, logic,

UI).

**1296**

**01:24:06,052 --> 01:24:10,464**

* · Constraints: Provide technical details for developers

and stakeholders.

**1297**

**01:24:10,714 --> 01:24:12,618**

AI-Generated Documentation

**1298**

**01:24:12,868 --> 01:24:15,979**

Title: Employee Expense Tracker Power App

**1299**

**01:24:16,230 --> 01:24:22,407**

Overview: The Employee Expense Tracker is a custom

Power App designed to help employees submit and track

**1300**

**01:24:22,407 --> 01:24:23,660**

their expense reports.

**1301**

**01:24:23,810 --> 01:24:30,207**

The app integrates with SharePoint and SQL Server

to store and retrieve expense data while offering

**1302**

**01:24:30,207 --> 01:24:33,934**

an intuitive user interface for entering and reviewing

expenses.

**1303**

**01:24:34,184 --> 01:24:35,299**

Data Sources:

**1304**

**01:24:35,677 --> 01:24:39,775**

* · SharePoint List: Stores employee information and

department details.

**1305**

**01:24:40,095 --> 01:24:45,923**

* · SQL Server: Stores the expense reports, including

expense categories, amounts, and approval statuses.

**1306**

**01:24:46,173 --> 01:24:48,355**

Key Screens and UI Components:

**1307**

**01:24:48,675 --> 01:24:53,621**

* · Home Screen: This screen summarizes submitted expense

reports and their current status.

**1308**

**01:24:53,771 --> 01:24:58,229**

It contains a gallery component that lists expenses

filtered by the user.

**1309**

**01:24:58,630 --> 01:25:03,030**

* · New Expense Screen: This screen allows employees

to submit a new expense report.

**1310**

**01:25:03,180 --> 01:25:06,988**

It includes input fields for date, category, amount,

and description.

**1311**

**01:25:07,138 --> 01:25:10,203**

The submit button triggers the expense submission

process.

**1312**

**01:25:10,523 --> 01:25:15,887**

* · Expense Detail Screen: This screen shows detailed

information for a selected expense report.

**1313**

**01:25:16,037 --> 01:25:22,306**

It includes conditional formatting to display status

(e.g., "Pending," "Approved," or "Rejected").

**1314**

**01:25:22,556 --> 01:25:23,810**

Business Logic:

**1315**

**01:25:24,153 --> 01:25:30,167**

* · Formulas for Expense Submission: When the user submits

an expense report, the formula Patch(ExpenseReports,

**1316**

**01:25:30,167 --> 01:25:34,973**

Defaults(ExpenseReports), ) is used to create a new

record in the SQL Server database.

**1317**

**01:25:35,362 --> 01:25:41,354**

* · Validation Logic: The app uses If conditions to

check if all required fields are filled before submission.

**1318**

**01:25:41,685 --> 01:25:48,384**

* · Status Updates: A flow integrated with Power Automate

updates the status of an expense report based on approvals.

**1319**

**01:25:48,634 --> 01:25:49,795**

Integration Points:

**1320**

**01:25:50,149 --> 01:25:56,616**

* · Power Automate: The app is integrated with a Power

Automate flow to send approval requests to managers

**1321**

**01:25:56,616 --> 01:25:59,240**

after an employee submits an expense report.

**1322**

**01:25:59,571 --> 01:26:05,492**

* · Email Notifications: An email is sent to the employee

using Power Automate's email connector upon approval

**1323**

**01:26:05,492 --> 01:26:06,409**

or rejection.

**1324**

**01:26:06,659 --> 01:26:07,402**

**5.**

**1325**

**01:26:07,552 --> 01:26:10,710**

Prompt for Generating User-Friendly Documentation:

**1326**

**01:26:10,960 --> 01:26:18,123**

Scenario: You want to generate a user guide for non-technical

users of a Power BI dashboard that visualizes sales

**1327**

**01:26:18,123 --> 01:26:19,180**

performance data.

**1328**

**01:26:19,430 --> 01:26:26,024**

Prompt Example: "Generate user-friendly documentation

for a Power BI dashboard that visualizes sales performance

**1329**

**01:26:26,024 --> 01:26:26,675**

data.

**1330**

**01:26:26,825 --> 01:26:32,491**

Include a dashboard overview, key visuals, how to

interact with filters, and how to export data."

**1331**

**01:26:32,741 --> 01:26:33,623**

Components:

**1332**

**01:26:34,036 --> 01:26:36,381**

* · Context: A Power BI sales dashboard.

**1333**

**01:26:36,701 --> 01:26:40,717**

* · Input: Features of the dashboard (key visuals, interactivity).

**1334**

**01:26:41,048 --> 01:26:45,518**

* · Constraints: Tailor the documentation for non-technical

users.

**1335**

**01:26:45,768 --> 01:26:47,719**

AI-Generated Documentation:

**1336**

**01:26:47,969 --> 01:26:51,080**

Title: Sales Performance Dashboard User Guide

**1337**

**01:26:51,330 --> 01:26:57,820**

Overview: The Sales Performance Dashboard in Power

BI provides a comprehensive overview of sales data

**1338**

**01:26:57,820 --> 01:27:00,340**

across regions, products, and periods.

**1339**

**01:27:00,490 --> 01:27:05,598**

It allows users to track sales performance, identify

trends, and analyze KPIs.

**1340**

**01:27:05,848 --> 01:27:06,916**

Key Visuals:

**1341**

**01:27:07,247 --> 01:27:11,253**

* · Sales by Region: A visual map displaying sales totals

by region.

**1342**

**01:27:11,403 --> 01:27:14,654**

Hover over each region to see detailed sales data.

**1343**

**01:27:15,647 --> 01:27:19,362**

* · Sales Over Time: A line chart showing monthly sales

trends.

**1344**

**01:27:19,512 --> 01:27:21,973**

Use the date filter to adjust the time range.

**1345**

**01:27:22,304 --> 01:27:27,053**

**· Top 5 Products: A bar chart that ranks the top 5**

products by total sales.

**1346**

**01:27:27,303 --> 01:27:28,836**

Interacting with Filters:

**1347**

**01:27:29,167 --> 01:27:34,798**

* · Region Filter: Located on the right side of the

dashboard, this filter allows you to view data for

**1348**

**01:27:34,798 --> 01:27:36,144**

specific regions.

**1349**

**01:27:36,615 --> 01:27:43,175**

* · Period Filter: Use this dropdown to select a custom

**period, such as "Last 30 Days" or "Year-to-Date."**

**1350**

**01:27:43,553 --> 01:27:48,672**

* · Product Filter: Select one or more products from

this list to focus on specific items.

**1351**

**01:27:48,922 --> 01:27:55,087**

How to Export Data: To export data from any visual,

click on the three dots in the top-right corner and

**1352**

**01:27:55,087 --> 01:27:57,885**

select "Export data" from the dropdown menu.

**1353**

**01:27:58,035 --> 01:28:01,379**

You can choose to export data in CSV or Excel format.

**1354**

**01:28:03,180 --> 01:28:03,923**

**6.**

**1355**

**01:28:04,073 --> 01:28:06,905**

Prompt for Auto-Generating Data Flow Diagrams:

**1356**

**01:28:07,155 --> 01:28:13,645**

Scenario: You need documentation that includes a data

flow diagram for a complex Power Automate workflow

**1357**

**01:28:13,645 --> 01:28:15,793**

that processes customer orders.

**1358**

**01:28:16,043 --> 01:28:22,638**

Prompt Example: "Generate documentation for a Power

Automate workflow that processes customer orders.

**1359**

**01:28:22,788 --> 01:28:28,407**

Include a workflow description and generate a data

flow diagram showing how data moves between systems

**1360**

**01:28:28,407 --> 01:28:32,447**

(e.g., CRM, SQL Server, and Email)."

**1361**

**01:28:32,697 --> 01:28:33,580**

Components:

**1362**

**01:28:33,911 --> 01:28:37,916**

* · Context: A complex workflow that processes customer

orders.

**1363**

**01:28:38,236 --> 01:28:41,696**

* · Input: Workflow details and system integration points.

**1364**

**01:28:42,039 --> 01:28:45,847**

* · Constraints: Include a data flow diagram for visual

clarity.

**1365**

**01:28:46,097 --> 01:28:48,001**

AI-Generated Documentation

**1366**

**01:28:48,251 --> 01:28:50,759**

Title: Customer Order Processing Workflow

**1367**

**01:28:51,008 --> 01:28:57,974**

Workflow Description: This Power Automate workflow

processes customer orders placed through the CRM system.

**1368**

**01:28:58,124 --> 01:29:03,720**

Once an order is created, it updates the SQL Server

database, sends confirmation emails to customers,

**1369**

**01:29:03,720 --> 01:29:05,833**

and triggers a fulfillment process.

**1370**

**01:29:06,083 --> 01:29:07,012**

Data Flow:

**1371**

**01:29:07,355 --> 01:29:10,931**

**· Step 1: Customer places an order in the CRM system.**

**1372**

**01:29:11,320 --> 01:29:17,578**

**· Step 2: The workflow retrieves order details from**

the CRM and updates the SQL Server database with the

**1373**

**01:29:17,578 --> 01:29:19,076**

new order information.

**1374**

**01:29:19,442 --> 01:29:23,970**

**· Step 3: The email connector sends A confirmation**

email to the customer.

**1375**

**01:29:24,371 --> 01:29:30,362**

**· Step 4: The fulfillment team is notified, and the**

order status is updated in the SQL Server database

**1376**

**01:29:30,362 --> 01:29:31,557**

as "Processing."

**1377**

**01:29:31,807 --> 01:29:32,550**

**7.**

**1378**

**01:29:32,700 --> 01:29:36,741**

Best Practices for Documentation Generation in Power

Platform:

**1379**

**01:29:37,095 --> 01:29:44,445**

* · Structured Documentation: Define a clear structure

(overview, steps, logic, etc.) to ensure the documentation

**1380**

**01:29:44,445 --> 01:29:45,814**

is easy to follow.

**1381**

**01:29:46,134 --> 01:29:52,345**

* · Tailor for Different Audiences: Adjust the complexity

of the documentation depending on the target audience

**1382**

**01:29:52,345 --> 01:29:53,727**

(technical vs.

**1383**

**01:29:53,877 --> 01:29:54,945**

non-technical).

**1384**

**01:29:55,265 --> 01:30:01,325**

* · Include Visuals and Diagrams: To enhance clarity,

use data flow diagrams, screenshots, or other visuals

**1385**

**01:30:01,325 --> 01:30:02,347**

where applicable.

**1386**

**01:30:02,713 --> 01:30:09,842**

* · Leverage Dynamic Data: Incorporate dynamic content

based on the app or workflow's logic and flow to make

**1387**

**01:30:09,842 --> 01:30:11,467**

the documentation relevant.

**1388**

**01:30:11,821 --> 01:30:18,451**

* · Iterate and Refine: Continuously refine the prompts

based on feedback to improve the clarity and detail

**1389**

**01:30:18,451 --> 01:30:20,401**

of the generated documentation.

**1390**

**01:30:20,651 --> 01:30:26,769**

Through well-crafted prompt engineering, AI models

can significantly streamline documentation generation

**1391**

**01:30:26,769 --> 01:30:29,335**

for Microsoft Power Platform applications.

**1392**

**01:30:29,485 --> 01:30:35,801**

Developers and users can automatically generate comprehensive

and user-friendly documentation by defining clear

**1393**

**01:30:35,801 --> 01:30:41,281**

structures, specifying the necessary components, and

tailoring the output for different audiences.

**1394**

**01:30:41,531 --> 01:30:42,274**

**5.**

**1395**

**01:30:42,424 --> 01:30:45,071**

AI-assisted Game Development at Roblox

**1396**

**01:30:45,321 --> 01:30:51,950**

Roblox, a gaming platform that allows users to create

and share games, has integrated AI tools to help game

**1397**

**01:30:51,950 --> 01:30:54,191**

developers build new experiences.

**1398**

**01:30:54,341 --> 01:31:00,018**

AI-assisted game development can range from helping

**with 3D modeling to generating game narratives based**

**1399**

**01:31:00,018 --> 01:31:01,075**

on user prompts.

**1400**

**01:31:01,325 --> 01:31:07,188**

Prompt Engineering Use Case: Roblox developers used

carefully crafted prompts to generate engaging game

**1401**

**01:31:07,188 --> 01:31:07,966**

narratives.

**1402**

**01:31:08,116 --> 01:31:14,467**

For example, prompts like "create a quest for a medieval

fantasy game where the player must find a lost artifact"

**1403**

**01:31:14,467 --> 01:31:18,379**

gave the AI enough context to produce creative, coherent

storylines.

**1404**

**01:31:18,529 --> 01:31:26,226**

By iterating on the prompt (e.g., "make the quest

more challenging" or "introduce a plot twist midway"),

**1405**

**01:31:26,226 --> 01:31:30,789**

developers could fine-tune the AI's outputs to match

their vision for the game.

**1406**

**01:31:31,039 --> 01:31:37,436**

Best Practice: When using AI for creative tasks, such

as narrative or content generation, prompts should

**1407**

**01:31:37,436 --> 01:31:41,256**

balance providing specific guidance with allowing

room for creativity.

**1408**

**01:31:41,406 --> 01:31:46,328**

Iterative refinement of prompts ensures the AI's

outputs align with the desired outcomes.

**1409**

**01:31:46,578 --> 01:31:52,557**

The success of prompt engineering in these real-world

cases demonstrates its importance in AI-driven development

**1410**

**01:31:52,557 --> 01:31:54,241**

across various industries.

**1411**

**01:31:54,391 --> 01:32:00,161**

Whether improving customer service chatbots, automating

code generation, or assisting game developers, the

**1412**

**01:32:00,161 --> 01:32:05,537**

quality and specificity of prompts directly influence

the effectiveness of AI systems.

**1413**

**01:32:05,687 --> 01:32:11,469**

The best practices from these case studies highlight

the need for clarity, context, and iterative refinement

**1414**

**01:32:11,469 --> 01:32:13,767**

in prompts to achieve optimal outcomes.

**1415**

**01:32:13,917 --> 01:32:20,419**

As AI evolves, prompt engineering will remain essential

for developers looking to harness its full potential.

**1416**

**01:32:20,669 --> 01:32:27,495**

Example: Here's an example of prompt engineering in

the context of AI-assisted Game Development at Roblox.

**1417**

**01:32:27,645 --> 01:32:34,216**

It showcases how to guide AI to assist developers

in creating game elements such as characters, scripts,

**1418**

**01:32:34,216 --> 01:32:36,841**

and environments using well-designed prompts.

**1419**

**01:32:37,091 --> 01:32:37,648**

**1.**

**1420**

**01:32:37,798 --> 01:32:44,474**

Objective: The goal is to help developers on the

Roblox platform leverage AI to generate in-game content

**1421**

**01:32:44,474 --> 01:32:49,733**

(e.g., code, environments, NPC behavior) and streamline

the game development process.

**1422**

**01:32:49,883 --> 01:32:55,363**

Prompt engineering can instruct AI to assist with

scripting, asset creation, and gameplay logic.

**1423**

**01:32:57,163 --> 01:32:57,720**

**2.**

**1424**

**01:32:57,870 --> 01:33:01,446**

Prompt Design Principles for AI-assisted Game Development:

**1425**

**01:33:01,800 --> 01:33:08,337**

* · Specific Instructions: Provide clear, detailed prompts

that specify what type of game element or functionality

**1426**

**01:33:08,337 --> 01:33:09,033**

you need.

**1427**

**01:33:09,353 --> 01:33:14,345**

* · Iteration: Allow room for adjustments and iterations

of the generated assets or scripts.

**1428**

**01:33:14,688 --> 01:33:20,795**

* · Creativity with Constraints: Enable the AI to be

creative within predefined boundaries to maintain

**1429**

**01:33:20,795 --> 01:33:22,908**

consistency with the game's theme.

**1430**

**01:33:23,158 --> 01:33:28,975**

Dynamic Context: To ensure relevance, use prompts

that reflect the game's unique world, mechanics, and

**1431**

**01:33:28,975 --> 01:33:29,613**

rules.

**1432**

**01:33:30,263 --> 01:33:30,913**

**3.**

**1433**

**01:33:31,063 --> 01:33:34,825**

Basic Prompt for Generating a Simple NPC Behavior

Script:

**1434**

**01:33:35,075 --> 01:33:41,588**

Scenario: You're developing a simple platformer game

in Roblox, and you want an NPC to patrol between two

**1435**

**01:33:41,588 --> 01:33:42,273**

points.

**1436**

**01:33:42,423 --> 01:33:45,999**

The AI needs to generate a Lua script that handles

this behavior.

**1437**

**01:33:46,249 --> 01:33:52,460**

Prompt Example: "Generate a Lua script for an NPC

in Roblox that patrols between two points (A and B)

**1438**

**01:33:52,460 --> 01:33:53,447**

in a straight line.

**1439**

**01:33:53,597 --> 01:33:58,427**

**The NPC should wait 2 seconds at each point before**

turning around and walking to the other point.

**1440**

**01:33:58,577 --> 01:34:01,502**

Ensure the NPC repeats this behavior in a loop."

**1441**

**01:34:01,752 --> 01:34:02,635**

Components:

**1442**

**01:34:02,966 --> 01:34:06,043**

* · Context: NPC behavior in a platformer game.

**1443**

**01:34:06,363 --> 01:34:09,776**

* · Input: NPC patrol logic (moving between two points).

**1444**

**01:34:10,130 --> 01:34:13,323**

* · Constraints: Include pauses and looping behavior.

**1445**

**01:34:13,973 --> 01:34:14,670**

**4.**

**1446**

**01:34:14,820 --> 01:34:18,024**

Advanced Prompt for Generating a Customizable Terrain:

**1447**

**01:34:18,274 --> 01:34:24,056**

Scenario: You're building a fantasy game world in

Roblox, and you want to generate a terrain map that

**1448**

**01:34:24,056 --> 01:34:26,354**

includes mountains, forests, and rivers.

**1449**

**01:34:26,504 --> 01:34:31,381**

The terrain should be customizable regarding height,

tree density, and river size.

**1450**

**01:34:31,631 --> 01:34:37,808**

Prompt Example: "Generate a Lua script for creating

a customizable terrain in Roblox with mountains, forests,

**1451**

**01:34:37,808 --> 01:34:38,643**

and rivers.

**1452**

**01:34:38,793 --> 01:34:44,528**

The user should be able to adjust the terrain's height,

tree density, and river width through variables in

**1453**

**01:34:44,528 --> 01:34:45,341**

the script."

**1454**

**01:34:45,591 --> 01:34:46,473**

Components:

**1455**

**01:34:46,851 --> 01:34:50,346**

* · Context: Procedural terrain generation for a fantasy

world.

**1456**

**01:34:50,666 --> 01:34:55,704**

* · Input: Terrain features (mountains, forests, rivers)

with customizable parameters.

**1457**

**01:34:56,035 --> 01:35:00,738**

* · Constraints: Allow adjustments to terrain height,

tree density, and river width.

**1458**

**01:35:01,788 --> 01:35:02,531**

**5.**

**1459**

**01:35:02,681 --> 01:35:05,931**

Prompt for Generating Character Customization Options:

**1460**

**01:35:06,181 --> 01:35:11,893**

Scenario: You want to implement character customization

options in your Roblox RPG game.

**1461**

**01:35:12,043 --> 01:35:18,336**

The AI needs to generate a script that allows players

to customize their character's appearance, including

**1462**

**01:35:18,336 --> 01:35:20,217**

hair, outfit, and accessories.

**1463**

**01:35:20,467 --> 01:35:26,713**

Prompt Example: "Generate a Lua script that allows

players to customize their character's appearance

**1464**

**01:35:26,713 --> 01:35:27,758**

in Roblox.

**1465**

**01:35:27,908 --> 01:35:33,620**

The customization options should include hairstyle,

outfit, and accessories, with different color choices

**1466**

**01:35:33,620 --> 01:35:34,642**

for each option."

**1467**

**01:35:34,892 --> 01:35:35,774**

Components:

**1468**

**01:35:36,105 --> 01:35:39,646**

* · Context: Character customization for an RPG game.

**1469**

**01:35:39,977 --> 01:35:44,958**

* · Input: Customization options (hair, outfit, accessories)

with color choices.

**1470**

**01:35:45,608 --> 01:35:46,351**

**6.**

**1471**

**01:35:46,501 --> 01:35:49,009**

Prompt for AI-assisted Gameplay Mechanics:

**1472**

**01:35:49,259 --> 01:35:54,600**

Scenario: You are designing a battle mechanic for

your Roblox game, where the player fights an enemy

**1473**

**01:35:54,600 --> 01:35:55,436**

NPC.

**1474**

**01:35:55,586 --> 01:36:01,286**

The AI should generate a combat system where the

player can attack, and the NPC has a health bar and

**1475**

**01:36:01,286 --> 01:36:02,319**

counter-attacks.

**1476**

**01:36:02,569 --> 01:36:07,260**

Prompt Example: "Generate a Lua script for a simple

battle system in Roblox.

**1477**

**01:36:07,410 --> 01:36:13,598**

The player should be able to attack an enemy NPC,

reduce the NPC's health, and the NPC should counter-attack

**1478**

**01:36:13,598 --> 01:36:15,072**

after each player attack.

**1479**

**01:36:15,222 --> 01:36:18,613**

Display the health bars for both the player and the

NPC."

**1480**

**01:36:18,863 --> 01:36:19,745**

Components:

**1481**

**01:36:20,065 --> 01:36:22,874**

* · Context: Battle mechanic for a Roblox game.

**1482**

**01:36:23,205 --> 01:36:26,654**

* · Input: Player attacks, NPC health bar, counter-attacks.

**1483**

**01:36:26,985 --> 01:36:30,340**

* · Constraints: Health management and simple combat

logic.

**1484**

**01:36:32,141 --> 01:36:32,884**

**7.**

**1485**

**01:36:33,034 --> 01:36:36,934**

Best Practices for AI-assisted Game Development in

Roblox:

**1486**

**01:36:37,254 --> 01:36:43,581**

* · Iterative Prompts: AI-generated game elements might

require multiple iterations, so provide feedback and

**1487**

**01:36:43,581 --> 01:36:45,265**

refine the generated output.

**1488**

**01:36:45,619 --> 01:36:50,205**

* · Modular Prompts: Break down game development tasks

into smaller, modular prompts.

**1489**

**01:36:50,355 --> 01:36:57,042**

For example, generate one prompt for character customization,

another for NPC behavior, and a separate one for combat

**1490**

**01:36:57,042 --> 01:36:57,832**

mechanics.

**1491**

**01:36:58,233 --> 01:37:04,839**

* · Balance Creativity with Constraints: While AI can

be creative, set clear boundaries in prompts to ensure

**1492**

**01:37:04,839 --> 01:37:08,996**

that the generated content fits the game's mechanics,

style, and theme.

**1493**

**01:37:09,350 --> 01:37:16,630**

* · Use Placeholders for Flexibility: Employ placeholders

in scripts (e.g., for character names or parameters

**1494**

**01:37:16,630 --> 01:37:21,227**

like health and damage) that can be easily customized

later in development.

**1495**

**01:37:21,547 --> 01:37:27,363**

* · Adapt the Prompt to Skill Level: If the prompts

are for beginner Roblox developers, they should guide

**1496**

**01:37:27,363 --> 01:37:31,508**

the AI to generate more detailed and beginner-friendly

code with comments.

**1497**

**01:37:31,658 --> 01:37:36,906**

For advanced developers, prompts can request more

sophisticated logic and optimizations.

**1498**

**01:37:37,156 --> 01:37:43,832**

By effectively using prompt engineering, Roblox developers

can harness AI to streamline various aspects of game

**1499**

**01:37:43,832 --> 01:37:49,602**

development, from character scripting and terrain

generation to combat mechanics and user customization.

**1500**

**01:37:49,752 --> 01:37:54,733**

Developers can create immersive game experiences

efficiently through well-crafted prompts, iterating

**1501**

**01:37:54,733 --> 01:37:57,321**

on AI-generated assets and scripts as needed.

**1502**

**01:37:57,571 --> 01:38:01,565**

**1.4.2 Avoiding Common Pitfalls in Prompt Engineering**

**1503**

**01:38:01,815 --> 01:38:07,469**

Prompt engineering is an emerging discipline in AI

development that involves designing inputs (or "prompts")

**1504**

**01:38:07,469 --> 01:38:11,800**

to effectively elicit useful responses from language

models like GPT.

**1505**

**01:38:11,950 --> 01:38:17,894**

While it offers enormous potential, prompt engineering

also comes with its own set of challenges and pitfalls.

**1506**

**01:38:18,044 --> 01:38:23,953**

In this analysis, we explore some common pitfalls

in prompt engineering and propose strategies to avoid

**1507**

**01:38:23,953 --> 01:38:28,633**

them, using real-world examples to illustrate how

these challenges can be addressed.

**1508**

**01:38:28,883 --> 01:38:29,440**

**1.**

**1509**

**01:38:29,590 --> 01:38:30,890**

Overly Broad Prompts

**1510**

**01:38:31,140 --> 01:38:36,852**

One of the most common pitfalls in prompt engineering

is crafting overly broad or ambiguous prompts.

**1511**

**01:38:37,002 --> 01:38:42,726**

Broad prompts often lead to outputs that are either

irrelevant or lack the necessary specificity to be

**1512**

**01:38:42,726 --> 01:38:43,411**

useful.

**1513**

**01:38:43,561 --> 01:38:49,029**

For example, consider the prompt: "Tell me about

the weather." This could lead to general information

**1514**

**01:38:49,029 --> 01:38:53,406**

about weather phenomena or irrelevant details about

worldwide climate conditions.

**1515**

**01:38:53,656 --> 01:38:54,585**

Example:

**1516**

**01:38:56,936 --> 01:38:59,525**

* · Broad Prompt: "Explain machine learning."

**1517**

**01:38:59,775 --> 01:39:05,952**

o Result: The output might include general information

about algorithms, historical context, and various

**1518**

**01:39:05,952 --> 01:39:10,457**

subfields without focusing on any specific aspect

the user is interested in.

**1519**

**01:39:10,707 --> 01:39:14,422**

o Solution: To avoid this, narrow down the scope of

the prompt.

**1520**

**01:39:14,572 --> 01:39:18,194**

Provide context, target specifics, and ask clear

questions.

**1521**

**01:39:18,514 --> 01:39:24,644**

* · Improved Prompt: "Explain how supervised learning

differs from unsupervised learning in machine learning,

**1522**

**01:39:24,644 --> 01:39:25,735**

with examples."

**1523**

**01:39:25,985 --> 01:39:31,871**

This modified prompt offers direction to the model,

reducing ambiguity and leading to more useful, targeted

**1524**

**01:39:31,871 --> 01:39:32,765**

responses.

**1525**

**01:39:33,015 --> 01:39:33,573**

**2.**

**1526**

**01:39:33,723 --> 01:39:36,277**

Neglecting to Set Boundaries or Constraints

**1527**

**01:39:36,527 --> 01:39:41,891**

Another common issue in prompt engineering arises

from not setting clear boundaries or constraints on

**1528**

**01:39:41,891 --> 01:39:44,654**

the kind of information the model should generate.

**1529**

**01:39:44,804 --> 01:39:50,644**

Without constraints, the model might produce overly

detailed or off-topic responses, wasting time and

**1530**

**01:39:50,644 --> 01:39:51,538**

resources.

**1531**

**01:39:51,788 --> 01:39:52,716**

Example:

**1532**

**01:39:53,036 --> 01:39:57,192**

* · Unconstrained Prompt: "Summarize the benefits of

cloud computing."

**1533**

**01:39:57,442 --> 01:40:05,047**

o Result: The output might delve into various areas

like SaaS, IaaS, and PaaS, with detailed explanations

**1534**

**01:40:05,047 --> 01:40:07,845**

that are unnecessary for a general overview.

**1535**

**01:40:08,095 --> 01:40:14,179**

o Solution: Setting explicit constraints within the

prompt limits the model's focus on relevant content.

**1536**

**01:40:14,499 --> 01:40:21,070**

* · Improved Prompt: "Summarize the benefits of cloud

computing in relation to cost efficiency and scalability

**1537**

**01:40:21,070 --> 01:40:23,206**

**in no more than 100 words."**

**1538**

**01:40:23,456 --> 01:40:29,064**

This prompt guides the model towards a succinct and

focused response, ensuring that the output aligns

**1539**

**01:40:29,064 --> 01:40:31,025**

with the user's expectations.

**1540**

**01:40:32,826 --> 01:40:33,476**

**3.**

**1541**

**01:40:33,626 --> 01:40:35,576**

Ignoring Contextual Information

**1542**

**01:40:35,826 --> 01:40:40,842**

Another pitfall in prompt engineering is failing to

provide sufficient context for the model to generate

**1543**

**01:40:40,842 --> 01:40:42,049**

a helpful response.

**1544**

**01:40:42,199 --> 01:40:47,830**

Language models work best when they provide specific

details, context, or background information that help

**1545**

**01:40:47,830 --> 01:40:49,351**

them understand the request.

**1546**

**01:40:49,501 --> 01:40:54,098**

The model might produce vague, off-topic, or incorrect

responses without this.

**1547**

**01:40:54,348 --> 01:40:55,277**

Example:

**1548**

**01:40:55,620 --> 01:40:58,778**

* · Context-Less Prompt: "Write an email to a client."

**1549**

**01:40:59,028 --> 01:41:05,437**

o Result: The response might be too generic or irrelevant,

lacking any personalized touch or necessary information

**1550**

**01:41:05,437 --> 01:41:06,830**

that the client needs.

**1551**

**01:41:07,080 --> 01:41:13,709**

o Solution: Provide key details and context in the

prompt, such as the nature of the client relationship,

**1552**

**01:41:13,709 --> 01:41:16,554**

specific concerns, or the tone you want to convey.

**1553**

**01:41:16,874 --> 01:41:23,248**

* · Improved Prompt: "Write a formal email to a long-term

client, informing them about a delay in project delivery

**1554**

**01:41:23,248 --> 01:41:27,392**

due to unforeseen technical issues, and assure them

of a revised deadline."

**1555**

**01:41:27,642 --> 01:41:33,551**

By adding this context, the model can tailor its response

appropriately, generating a more accurate and useful

**1556**

**01:41:33,551 --> 01:41:34,190**

email.

**1557**

**01:41:34,440 --> 01:41:35,137**

**4.**

**1558**

**01:41:35,287 --> 01:41:37,330**

Lack of Iteration and Refinement

**1559**

**01:41:37,580 --> 01:41:40,691**

Prompt engineering requires iteration and refinement.

**1560**

**01:41:40,841 --> 01:41:44,510**

A common mistake is expecting the perfect prompt

on the first attempt.

**1561**

**01:41:44,660 --> 01:41:50,094**

However, AI models respond best to prompts that have

been fine-tuned based on previous outputs.

**1562**

**01:41:50,344 --> 01:41:51,272**

Example:

**1563**

**01:41:51,592 --> 01:41:56,399**

* · You might ask the model: "What are the best practices

for software development?"

**1564**

**01:41:56,649 --> 01:42:02,408**

o Result: The model generates a list of practices

that may not align with your specific context or field

**1565**

**01:42:02,408 --> 01:42:05,055**

of interest (e.g., front-end vs.

**1566**

**01:42:05,204 --> 01:42:06,179**

back-end).

**1567**

**01:42:06,429 --> 01:42:10,191**

o Solution: Iteratively refine the prompt based on

the output.

**1568**

**01:42:10,341 --> 01:42:16,970**

For example, if the initial output focuses too much

on agile methodologies but you are interested in continuous

**1569**

**01:42:16,970 --> 01:42:21,812**

integration, modify the prompt to steer the model

toward your preferred topics.

**1570**

**01:42:22,132 --> 01:42:28,517**

* · Improved Prompt: "What are the best practices for

software development in continuous integration and

**1571**

**01:42:28,517 --> 01:42:29,353**

delivery?"

**1572**

**01:42:29,603 --> 01:42:34,154**

By iterating on the prompt, you ensure the output

becomes more focused and relevant.

**1573**

**01:42:34,404 --> 01:42:35,147**

**5.**

**1574**

**01:42:35,297 --> 01:42:36,969**

Overcomplicating the Prompt

**1575**

**01:42:37,219 --> 01:42:42,327**

Sometimes, developers overcomplicate prompts by covering

too much information in one question.

**1576**

**01:42:42,477 --> 01:42:46,750**

This leads to either incomplete answers or confusion

in the generated response.

**1577**

**01:42:46,900 --> 01:42:52,333**

Long and complex prompts can overwhelm the model,

making it harder to produce clear outputs.

**1578**

**01:42:52,583 --> 01:42:53,512**

Example:

**1579**

**01:42:53,843 --> 01:42:59,602**

* · Overcomplicated Prompt: "Explain the differences

between supervised learning, unsupervised learning,

**1580**

**01:42:59,602 --> 01:43:05,535**

and reinforcement learning, and give examples of each,

and also discuss the advantages and disadvantages

**1581**

**01:43:05,535 --> 01:43:07,276**

of each type of learning model."

**1582**

**01:43:07,526 --> 01:43:13,145**

o Result: The output may be too scattered, failing

to adequately address any one part of the question.

**1583**

**01:43:13,395 --> 01:43:18,271**

o Solution: Break down complex prompts into smaller,

more manageable pieces.

**1584**

**01:43:18,421 --> 01:43:23,065**

Addressing each question individually helps the model

focus on one topic at a time.

**1585**

**01:43:23,385 --> 01:43:30,084**

* · Improved Prompt: "Explain the differences between

supervised and unsupervised learning with examples."

**1586**

**01:43:30,084 --> 01:43:35,343**

Follow-up: "Now explain reinforcement learning, with

its advantages and disadvantages."

**1587**

**01:43:35,593 --> 01:43:40,655**

This approach avoids overwhelming the model and allows

for more precise, detailed responses.

**1588**

**01:43:40,905 --> 01:43:41,648**

**6.**

**1589**

**01:43:41,798 --> 01:43:44,492**

Assuming Model Expertise in Niche Domains

**1590**

**01:43:44,742 --> 01:43:49,757**

Language models are trained on diverse datasets but

might not be experts in every niche domain.

**1591**

**01:43:49,907 --> 01:43:55,236**

A common mistake in prompt engineering is assuming

the model has deep, specialized knowledge in areas

**1592**

**01:43:55,236 --> 01:43:57,570**

requiring domain-specific expertise.

**1593**

**01:43:57,820 --> 01:43:58,749**

Example:

**1594**

**01:43:59,069 --> 01:44:04,943**

* · Assumption-Based Prompt: "Explain the latest advancements

in quantum computing algorithms developed in the past

**1595**

**01:44:04,943 --> 01:44:05,547**

month."

**1596**

**01:44:05,797 --> 01:44:12,775**

o Result: The model might generate outdated, incorrect,

or overly generalized information since it doesn't

**1597**

**01:44:12,775 --> 01:44:15,410**

have real-time access to the latest research.

**1598**

**01:44:15,660 --> 01:44:21,639**

o Solution: For niche or highly specialized topics,

frame the question in a way that allows the model

**1599**

**01:44:21,639 --> 01:44:25,784**

to give broader information and cross-reference with

up-to-date sources.

**1600**

**01:44:26,603 --> 01:44:32,349**

* · Improved Prompt: "Give an overview of quantum computing

algorithms and their development.

**1601**

**01:44:32,499 --> 01:44:38,258**

For the latest research, please cross-reference with

recent academic publications or industry reports."

**1602**

**01:44:38,508 --> 01:44:43,616**

This helps avoid reliance on the model for highly

specialized or time-sensitive information.

**1603**

**01:44:45,417 --> 01:44:46,160**

**7.**

**1604**

**01:44:46,310 --> 01:44:48,028**

Ignoring Model Limitations

**1605**

**01:44:48,278 --> 01:44:54,501**

Lastly, failing to recognize the model's limitations

can lead to unrealistic expectations of the output.

**1606**

**01:44:54,651 --> 01:45:01,013**

For example, AI models are not infallible and can

occasionally produce biased, incorrect, or nonsensical

**1607**

**01:45:01,013 --> 01:45:01,803**

results.

**1608**

**01:45:02,053 --> 01:45:02,981**

Example:

**1609**

**01:45:03,324 --> 01:45:08,665**

* · Expecting a model to write code that can directly

run without human intervention is one such pitfall.

**1610**

**01:45:08,915 --> 01:45:15,440**

o Solution: Always double-check the output, especially

in scenarios that require precise or critical results,

**1611**

**01:45:15,440 --> 01:45:17,738**

like coding or legal documentation.

**1612**

**01:45:17,888 --> 01:45:22,904**

Use prompts to ask for explanations rather than direct

execution to allow for review.

**1613**

**01:45:23,584 --> 01:45:26,916**

* · Improved Prompt: "Generate Python code to achieve

X.

**1614**

**01:45:27,066 --> 01:45:30,920**

Explain how the code works and possible optimizations

that can be made."

**1615**

**01:45:31,170 --> 01:45:34,792**

This ensures a balance between automation and human

review.

**1616**

**01:45:35,042 --> 01:45:41,416**

Prompt engineering is an iterative process that requires

thoughtful design, refinement, and understanding of

**1617**

**01:45:41,416 --> 01:45:43,587**

the model's strengths and limitations.

**1618**

**01:45:43,737 --> 01:45:49,914**

By avoiding common pitfalls such as overly broad

prompts, lack of context, or ignoring model limitations,

**1619**

**01:45:49,914 --> 01:45:54,000**

developers can ensure more accurate, relevant, and

useful outputs.

**1620**

**01:45:54,150 --> 01:45:59,538**

Crafting precise, well-defined prompts is key to

effectively leveraging AI models' full potential.

**1621**

**01:46:02,138 --> 01:46:07,014**

One of the most critical aspects of effective prompt

engineering is avoiding common pitfalls.

**1622**

**01:46:07,164 --> 01:46:12,829**

Let's take a closer look at these common mistakes

and explore practical strategies to overcome them.

**1623**

**01:46:13,079 --> 01:46:14,565**

First, vague prompts.

**1624**

**01:46:14,715 --> 01:46:20,613**

When instructions are unclear or too general, the

AI often struggles to generate the desired results.

**1625**

**01:46:20,763 --> 01:46:23,550**

To address this, always aim for specificity.

**1626**

**01:46:23,700 --> 01:46:29,737**

Define the expected output, provide formatting guidelines,

and set clear constraints to ensure precision.

**1627**

**01:46:29,987 --> 01:46:32,727**

Next, avoid creating overly complex prompts.

**1628**

**01:46:32,877 --> 01:46:37,521**

Complicated instructions can confuse the AI and lead

to suboptimal responses.

**1629**

**01:46:37,671 --> 01:46:41,200**

Instead, break down tasks into smaller, modular steps.

**1630**

**01:46:41,350 --> 01:46:46,226**

By simplifying each prompt, you'll enable the AI

to focus on one objective at a time.

**1631**

**01:46:46,476 --> 01:46:49,263**

Another frequent issue is providing no context.

**1632**

**01:46:49,413 --> 01:46:53,035**

Without background information, the AI may misinterpret

the request.

**1633**

**01:46:53,185 --> 01:46:59,547**

Always include relevant details, such as the goal,

use case, or scenario, to give the AI a clear understanding

**1634**

**01:46:59,547 --> 01:47:00,616**

of what's required.

**1635**

**01:47:00,866 --> 01:47:06,763**

Similarly, failing to set constraints or guidelines

can result in inconsistent or unusable outputs.

**1636**

**01:47:06,913 --> 01:47:12,184**

Whether it's a character limit, tone, or specific

formatting rules, be sure to define these upfront

**1637**

**01:47:12,184 --> 01:47:13,879**

to guide the AI effectively.

**1638**

**01:47:14,129 --> 01:47:17,659**

When writing prompts, beware of including too many

instructions in one.

**1639**

**01:47:17,809 --> 01:47:21,849**

Overloading a single prompt with multiple objectives

can dilute its clarity.

**1640**

**01:47:21,999 --> 01:47:26,318**

To avoid this, separate complex tasks into distinct

prompts or iterations.

**1641**

**01:47:26,568 --> 01:47:30,330**

Unclear or inconsistent terminology is another common

challenge.

**1642**

**01:47:30,480 --> 01:47:35,170**

Using ambiguous language can confuse the AI, leading

to misinterpretations.

**1643**

**01:47:35,320 --> 01:47:40,196**

Always use clear and consistent vocabulary to maintain

coherence and accuracy.

**1644**

**01:47:40,446 --> 01:47:44,161**

A key point to remember is not to ignore the AI's

limitations.

**1645**

**01:47:44,311 --> 01:47:46,773**

While powerful, AI systems have boundaries.

**1646**

**01:47:46,923 --> 01:47:52,821**

Understanding what the AI can and cannot do will

help you craft prompts that align with its capabilities,

**1647**

**01:47:52,821 --> 01:47:55,328**

ensuring realistic expectations.

**1648**

**01:47:55,578 --> 01:47:59,108**

Additionally, effective prompt engineering requires

iteration.

**1649**

**01:47:59,258 --> 01:48:01,348**

Rarely is the first attempt perfect.

**1650**

**01:48:01,498 --> 01:48:07,198**

Continuously refine your prompts based on the output,

testing and tweaking until you achieve the desired

**1651**

**01:48:07,198 --> 01:48:07,953**

results.

**1652**

**01:48:08,203 --> 01:48:11,593**

Avoid the trap of expecting one-size-fits-all prompts.

**1653**

**01:48:11,743 --> 01:48:18,059**

Each context, whether it's coding, content creation,

or data analysis, demands tailored instructions.

**1654**

**01:48:18,209 --> 01:48:22,760**

Customizing prompts for specific tasks improves both

relevance and effectiveness.

**1655**

**01:48:23,010 --> 01:48:26,771**

Another invaluable strategy is including examples

or templates.

**1656**

**01:48:26,921 --> 01:48:32,772**

Providing sample inputs and desired outputs sets

clear expectations and helps guide the AI toward the

**1657**

**01:48:32,772 --> 01:48:33,934**

intended outcome.

**1658**

**01:48:34,184 --> 01:48:38,456**

In collaborative environments, neglecting user feedback

can hinder progress.

**1659**

**01:48:38,606 --> 01:48:43,390**

Stakeholder insights are vital for refining prompts

and improving overall output quality.

**1660**

**01:48:43,640 --> 01:48:46,565**

And remember, don't rely solely on the first output.

**1661**

**01:48:46,715 --> 01:48:51,127**

AI-generated results often benefit from exploration

and refinement.

**1662**

**01:48:51,277 --> 01:48:55,457**

Generate multiple outputs, compare them, and adjust

your prompts as needed.

**1663**

**01:48:55,707 --> 01:48:57,425**

Testing is equally important.

**1664**

**01:48:57,575 --> 01:49:01,708**

Underestimating the importance of testing can lead

to unreliable results.

**1665**

**01:49:01,858 --> 01:49:07,617**

Evaluate your prompts with various inputs and scenarios

to ensure consistency and reliability.

**1666**

**01:49:07,867 --> 01:49:11,164**

For dynamic tasks, consider using conditional logic.

**1667**

**01:49:11,314 --> 01:49:17,165**

Incorporating this approach allows you to handle

variations in user input and generate more adaptive

**1668**

**01:49:17,165 --> 01:49:18,094**

responses.

**1669**

**01:49:18,344 --> 01:49:21,688**

Lastly, never forget to specify the output format.

**1670**

**01:49:21,838 --> 01:49:28,200**

Whether you need JSON, a list, or plain text, explicitly

defining the format ensures the output meets your

**1671**

**01:49:28,200 --> 01:49:28,850**

needs.

**1672**

**01:49:29,000 --> 01:49:31,972**

And don't overlook error handling or edge cases.

**1673**

**01:49:32,122 --> 01:49:37,509**

Providing instructions for unexpected inputs or scenarios

ensures robustness in your prompts.

**1674**

**01:49:37,759 --> 01:49:43,192**

By addressing these common mistakes and implementing

these strategies, you can unlock the full potential

**1675**

**01:49:43,192 --> 01:49:47,140**

of prompt engineering and achieve consistently high-quality

outputs.

**1676**

**01:49:47,790 --> 01:49:48,858**

Key Takeaways:

**1677**

**01:49:49,108 --> 01:49:49,666**

**1.**

**1678**

**01:49:49,816 --> 01:49:52,324**

Examples of Successful Prompt Engineering

**1679**

**01:49:52,574 --> 01:49:53,131**

**2.**

**1680**

**01:49:53,281 --> 01:49:54,999**

Avoiding Common Pitfalls:

**1681**

**01:49:55,423 --> 01:49:59,986**

* · Vague Prompts: Ambiguous or unclear prompts lead

to irrelevant outputs.

**1682**

**01:50:00,136 --> 01:50:04,083**

Solution: Be specific in describing tasks and expected

outcomes.

**1683**

**01:50:04,437 --> 01:50:09,024**

* · Overly Complex Prompts: Complex prompts confuse

models, resulting in errors.

**1684**

**01:50:09,174 --> 01:50:12,935**

Solution: Break down tasks into simpler, manageable

steps.

**1685**

**01:50:13,255 --> 01:50:18,480**

* · Ignoring Iteration: Not refining prompts based on

initial results can hinder performance.

**1686**

**01:50:18,630 --> 01:50:22,995**

Solution: Use iterative feedback to improve prompt

clarity and precision.

**1687**

**01:50:23,245 --> 01:50:23,895**

**3.**

**1688**

**01:50:24,045 --> 01:50:25,392**

Best Practices:

**1689**

**01:50:25,735 --> 01:50:28,753**

* · Test prompts thoroughly in development environments.

**1690**

**01:50:29,084 --> 01:50:33,554**

* · Continuously monitor and adjust prompts to align

with evolving project needs.

**1691**

**01:50:34,106 --> 01:50:38,495**

* · Leverage pre-existing successful templates as a

foundation for new projects.

**1692**

**01:50:40,295 --> 01:50:42,244**

**1.5 Hands-on Exercise**

**1693**

**01:50:42,495 --> 01:50:47,696**

Setting Up Your Development Environment and Making

Basic Api Calls With Code Examples

**1694**

**01:50:47,946 --> 01:50:54,622**

In natural language processing (NLP), prompt engineering

has emerged as a critical method for optimizing large

**1695**

**01:50:54,622 --> 01:50:57,559**

**language models (LLMs) like GPT-4.**

**1696**

**01:50:57,709 --> 01:51:03,514**

Prompt engineering involves designing and refining

input prompts to get a model's most accurate, creative,

**1697**

**01:51:03,514 --> 01:51:05,743**

or contextually relevant responses.

**1698**

**01:51:05,893 --> 01:51:11,606**

To get started, setting up a development environment

and learning how to make API calls is essential.

**1699**

**01:51:11,756 --> 01:51:18,432**

This guide will walk you through the setup process

and demonstrate basic API calls in Python, focusing

**1700**

**01:51:18,432 --> 01:51:20,626**

on prompt engineering principles.

**1701**

**01:51:20,876 --> 01:51:23,709**

**Step 1: Setting Up Your Development Environment**

**1702**

**01:51:23,958 --> 01:51:29,891**

It's crucial to set up a proper development environment

before making API calls to interact with language

**1703**

**01:51:29,891 --> 01:51:30,553**

models.

**1704**

**01:51:30,703 --> 01:51:34,464**

This guide will use Python due to its widespread

use and simplicity.

**1705**

**01:51:34,714 --> 01:51:35,829**

Prerequisites:

**1706**

**01:51:36,149 --> 01:51:37,751**

**· Python 3.x installed.**

**1707**

**01:51:38,094 --> 01:51:42,738**

* · An IDE (Integrated Development Environment) like

VSCode or PyCharm.

**1708**

**01:51:43,058 --> 01:51:46,749**

* · Access to the OpenAI API (or any LLM API).

**1709**

**01:51:47,092 --> 01:51:50,250**

* · Basic understanding of Python and API requests.

**1710**

**01:51:50,900 --> 01:51:51,457**

**1.**

**1711**

**01:51:51,607 --> 01:51:57,412**

Install Python and Pip: If you haven't installed

Python, download it from the official website.

**1712**

**01:51:57,562 --> 01:52:02,578**

Pip, the Python package installer, comes pre-installed

**with Python 3.x versions.**

**1713**

**01:52:03,228 --> 01:52:03,785**

**2.**

**1714**

**01:52:03,935 --> 01:52:10,297**

Set Up a Virtual Environment: A virtual environment

allows you to isolate project dependencies, ensuring

**1715**

**01:52:10,297 --> 01:52:12,945**

that different projects don't interfere with each

other.

**1716**

**01:52:13,595 --> 01:52:14,245**

**3.**

**1717**

**01:52:14,395 --> 01:52:21,431**

Install Required Libraries: Once the virtual environment

is active, install the required Python libraries,

**1718**

**01:52:21,431 --> 01:52:23,868**

such as requests for making API calls:

**1719**

**01:52:24,518 --> 01:52:25,215**

**4.**

**1720**

**01:52:25,365 --> 01:52:31,124**

Create a Python File: In your IDE, create a new Python

file (main.py) to make API calls.

**1721**

**01:52:31,374 --> 01:52:34,114**

**Step 2: Obtaining API Credentials**

**1722**

**01:52:34,515 --> 01:52:38,265**

You'll need an API key to interact with a model like

**GPT-4.**

**1723**

**01:52:38,415 --> 01:52:41,201**

For the OpenAI API, follow these steps:

**1724**

**01:52:41,451 --> 01:52:42,008**

**1.**

**1725**

**01:52:42,158 --> 01:52:46,849**

Sign up for OpenAI: Create an account at OpenAI and

obtain an API key.

**1726**

**01:52:47,099 --> 01:52:47,656**

**2.**

**1727**

**01:52:47,806 --> 01:52:53,239**

Store the API Key Securely: Never hardcode the API

key directly in your scripts.

**1728**

**01:52:53,389 --> 01:52:57,244**

Use environment variables or a configuration file

to keep it secure.

**1729**

**01:52:57,894 --> 01:53:00,588**

**Step 3: Making Basic API Calls**

**1730**

**01:53:00,837 --> 01:53:05,435**

With your environment set up, let's make a simple

**API call to GPT-4.**

**1731**

**01:53:05,585 --> 01:53:11,436**

For this guide, we'll use OpenAI's API and demonstrate

a basic prompt engineering example.

**1732**

**01:53:12,086 --> 01:53:13,665**

Output Example:

**1733**

**01:53:13,915 --> 01:53:14,937**

vbnet code:

**1734**

**01:53:15,187 --> 01:53:21,561**

Prompt engineering in AI is crucial because it allows

developers to guide and optimize the responses of

**1735**

**01:53:21,561 --> 01:53:23,686**

**language models like GPT-4.**

**1736**

**01:53:23,836 --> 01:53:30,047**

By carefully crafting prompts, developers can obtain

more accurate, relevant, and context-aware answers,

**1737**

**01:53:30,047 --> 01:53:34,006**

improving the efficiency and utility of AI-driven

applications.

**1738**

**01:53:34,656 --> 01:53:37,860**

**Step 4: Advanced Prompt Engineering Example**

**1739**

**01:53:38,110 --> 01:53:42,429**

In prompt engineering, refining the prompt can lead

to more precise responses.

**1740**

**01:53:42,579 --> 01:53:45,551**

Here's how you can tweak the prompt to request a

bulleted list.

**1741**

**01:53:46,201 --> 01:53:47,780**

Output Example:

**1742**

**01:53:48,030 --> 01:53:48,866**

Diffcode:

**1743**

**01:53:49,186 --> 01:53:51,670**

* · Improves response relevance and accuracy.

**1744**

**01:53:52,048 --> 01:53:55,032**

* · Enhances user control over the model's output.

**1745**

**01:53:55,421 --> 01:53:57,836**

* · Reduces ambiguity in the generated text.

**1746**

**01:53:58,167 --> 01:54:00,826**

* · Allows fine-tuning for specific use cases.

**1747**

**01:54:01,157 --> 01:54:04,838**

* · Saves time by reducing the need for multiple API

calls.

**1748**

**01:54:05,088 --> 01:54:06,992**

Explanation of Key Parameters:

**1749**

**01:54:07,312 --> 01:54:11,514**

* · Temperature: This parameter controls the "creativity"

of the model.

**1750**

**01:54:11,664 --> 01:54:22,206**

**Lower values (e.g., 0.2) make the output more deterministic,**

**while higher values (e.g., 0.8) introduce randomness.**

**1751**

**01:54:22,549 --> 01:54:25,475**

* · Max Tokens: Defines the length of the output.

**1752**

**01:54:25,625 --> 01:54:30,454**

You can limit this if you expect a short response

to prevent unnecessary token usage.

**1753**

**01:54:32,255 --> 01:54:34,995**

**Step 5: Debugging and Error Handling**

**1754**

**01:54:35,244 --> 01:54:38,170**

You may encounter errors while making API calls.

**1755**

**01:54:38,320 --> 01:54:44,009**

Here's a basic structure for handling common issues,

such as reaching the token limit or providing invalid

**1756**

**01:54:44,009 --> 01:54:44,915**

credentials.

**1757**

**01:54:45,565 --> 01:54:47,701**

**Step 6: Optimizing Prompts**

**1758**

**01:54:47,951 --> 01:54:53,094**

Effective prompt engineering involves iterating over

different versions of your prompt to get the most

**1759**

**01:54:53,094 --> 01:54:54,174**

useful output.

**1760**

**01:54:54,324 --> 01:54:55,531**

Here are some tips:

**1761**

**01:54:55,781 --> 01:54:56,339**

**1.**

**1762**

**01:54:56,489 --> 01:55:00,529**

Be specific: The more detailed your prompt, the better

the response.

**1763**

**01:55:00,679 --> 01:55:07,320**

Instead of asking, "Explain AI," try, "Explain how

AI improves healthcare, particularly in diagnostics."

**1764**

**01:55:07,570 --> 01:55:08,127**

**2.**

**1765**

**01:55:08,277 --> 01:55:12,549**

Set explicit instructions: When you need structured

output, state it.

**1766**

**01:55:12,699 --> 01:55:18,980**

**For example, "Summarize this text in 3 sentences,"**

or "List the benefits of AI in healthcare as bullet

**1767**

**01:55:18,980 --> 01:55:19,665**

points."

**1768**

**01:55:19,915 --> 01:55:20,566**

**3.**

**1769**

**01:55:20,716 --> 01:55:24,988**

Use examples: To guide the model's behavior, you

can provide examples.

**1770**

**01:55:25,138 --> 01:55:29,318**

For instance, "Here's an example of a good summary:

[Insert example].

**1771**

**01:55:29,468 --> 01:55:31,000**

Now summarize this text."

**1772**

**01:55:31,250 --> 01:55:33,015**

**Step 7: Conclusion**

**1773**

**01:55:33,265 --> 01:55:39,395**

Setting up a development environment and making basic

API calls are foundational steps in prompt engineering.

**1774**

**01:55:39,545 --> 01:55:42,703**

Through this hands-on experiment, we've explored

how to:

**1775**

**01:55:43,197 --> 01:55:44,718**

* · Set up a Python environment.

**1776**

**01:55:45,072 --> 01:55:50,215**

* · Obtain API credentials.

**1777**

**01:55:50,558 --> 01:55:53,902**

**· Make basic API calls with OpenAI's GPT-4.**

**1778**

**01:55:54,640 --> 01:55:56,938**

* · Improve prompts through iterations.

**1779**

**01:55:57,188 --> 01:56:04,131**

Prompt engineering enhances the utility of LLMs by

optimizing the interaction between the user and the

**1780**

**01:56:04,131 --> 01:56:07,637**

model, making the output more relevant and useful.

**1781**

**01:56:07,787 --> 01:56:13,174**

Following these steps, you can build and refine your

applications using API-driven NLP models.

**1782**

**01:56:13,424 --> 01:56:14,492**

Key Takeaways:

**1783**

**01:56:14,742 --> 01:56:15,300**

**1.**

**1784**

**01:56:15,450 --> 01:56:17,122**

Setting Up Development Environment:

**1785**

**01:56:17,442 --> 01:56:25,684**

* · Install necessary tools like Python/Node.js, text

editors (VS Code), and package managers.

**1786**

**01:56:26,038 --> 01:56:29,139**

* · Create a virtual environment to manage dependencies.

**1787**

**01:56:29,389 --> 01:56:29,946**

**2.**

**1788**

**01:56:30,096 --> 01:56:31,953**

Making Basic API Calls:

**1789**

**01:56:32,296 --> 01:56:39,633**

* · In Python, use requests or OpenAI SDK to make API

calls, providing the API key and required parameters

**1790**

**01:56:39,633 --> 01:56:41,399**

(e.g., prompt, model).

**1791**

**01:56:41,719 --> 01:56:46,803**

* · In JavaScript, fetch() or Axios sends API requests

and handles responses.

**1792THESE ELEMENTS INCLUDE**

**01:56:47,053 --> 01:56:47,704**

**3.**

**1793**

**01:56:47,854 --> 01:56:49,061**

Code Examples:

**1794**

**01:56:49,392 --> 01:56:55,058**

* · Example code snippets demonstrate API setup, sending

requests, and processing responses for Python and

**1795**

**01:56:55,058 --> 01:56:56,556**

JavaScript environments.