QUIZ PREPARATION

OpenAI Agent SDK

Important links:

https://github.com/panaversity/learn-agentic-ai?tab=readme-ov-file#quiz-details ➔ Quiz detail of Panaversity Repo.

OPENAI AGENTS SDK, TOPICS COVERED IN QUIZ, PROMPT ENGINEERING

- Sir Qasim used 2 approaches:
  - Approach 01: Official documentation
  - Approach 02: Sir Qasim has made UML diagram of OpenAI Agent SDK for better understanding
- A UML (Unified Modeling Language) diagram is a standardized visual modeling language used to specify, visualize, construct, and document the artifacts of software and other systems.
- MCP, Voice agent, A2A and design pattern will be included in Advance MCQs, basic markdown included

PROMPT ENGINEERING

Discussion

- In this class, we'll focus on prompt engineering.
- We will learn:
  - Technical Depth (Level 01):
    - OpenAI SDK's architecture (agents, tools, handoffs, runner)
    - Pydantic models, async programming, prompt engineering
  - Conceptual topics (Level 02):
    - Dynamic instruction, context management, error handling, chain of thoughts prompting
  - Code Analysis
  - Domain Knowledge

PROMPT ENGINEERING

Discussion

- **Prompt engineering in Agentic AI** refers to the strategic design of prompts to guide autonomous AI agents—such as AI assistants or multi-agent systems—to perform tasks effectively, often with minimal human intervention. Prompt refers to “asking questions”.
- We are discussing OpenAI ChatGPT 4.1 model.
- In today’s world, there is a different approach to do prompt engineering of today’s agentic framework, previously while using generative AI, it didn’t support reasoning and context is not rich.

PROMPT ENGINEERING

Discussion

- LLMs give you 2 returns
  - Either final output
  - Or ask you to call some tool
- Previously, performance of LLMs respect to tool calling is not good.
- OpenAI work on 3 things to improve it and make it better for developers

PROMPT ENGINEERING

Discussion

- LLMs give you 2 returns
  - Either final output
  - Or ask you to call some tool
- Previously, performance of LLMs respect to tool calling is not good.
- OpenAI work on 3 things to improve it and make it better for developers:
  - First, they reached the **contexts**, now we can process up to 1 M token in 1 query, check link for detail https://cookbook.openai.com/examples/gpt4-1_prompting_guide#2-long-context

PROMPT ENGINEERING

Discussion

- Secondly, they improve **tool calling**, OpenAI gives guidelines in tool calling schema like we do things 1) when to used tool, define in system prompt and 2) when work tool will do? How you can use it? What are their examples? Define these in tool schema
  - OpenAI tried to make persistence with the help f prompt engineering, **Persistence** in Agentic AI using the **OpenAI SDK** means the AI can **remember things over time**—even after a task ends or the system restarts.
- COT (**Chain of thoughts**), GPT-4.1 is not a reasoning model, but prompting the model to think step by step (called “chain of thought”) can be an effective way for a model to break down problems into more manageable pieces, solve them, and improve overall output quality

STRUCTURED FLOWCHART REPRESENTATION OF THE GPT-4.1 PROMPTING GUIDE

Start

↓

[Context & Best Practices]

- Use examples (few-shot)
- Be specific & clear
- Encourage planning

↓

[Agentic Workflows?] — Yes ⟶ Use 3 System-Prompt Reminders:

1. Persistence ("keep going…")
2. Tool-calling ("don't guess, call tools…")
3. Planning (optional "plan before calling tools…")

↓

[Workflow Initiation]

- Agent enters multi-message exchange
- Uses tools via API definitions
- Optionally reasons via planning

↓

STRUCTURED FLOWCHART REPRESENTATION OF THE GPT-4.1 PROMPTING GUIDE

[Chain-of-Thought Prompting?] — Yes ⟶ Add “think step by step” cue

↓

[Instruction Literalness]

- GPT-4.1 follows instructions very literally
- Must repeat or restate key rules
- Front-load & back-load instructions

↓

[Common Failure Modes]

- Check for conflicting or ambiguous instructions
- Align examples with rules
- Avoid unnecessary caps or incentives

↓

[Iteration & Evaluation]

- Test prompts with evals
- Migrate old prompts (see “Prompt Migration Guide”)
- Refine and repeat

↓

End

GPT-4.1 PROMPTING GUIDE

Discussion

1. Agentic Workflow:

a) System Prompt Reminder:

i. **Persistence:** Remember stuff overtime, even after a task ends or the system restarts. We make agent persistence using prompt

ii. **Tool calling:** This encourages the model to make full use of its tools, and reduces its likelihood of hallucinating or guessing an answer.

Example:

If you are not sure about file content or codebase structure pertaining to the user's request, use your tools to read files and gather the relevant information: do NOT guess or make up an answer.

GPT-4.1 PROMPTING GUIDE

Discussion

iii. Planning: planning means chain of thoughts, You can ask the model to stop and think before using tools. Instead of just using one tool after another quickly, it will write out a short plan or explanation first –like saying what it wants to do, and why –before actually doing it.

You MUST plan extensively before each function call, and reflect extensively on the outcomes of the previous function calls. DO NOT do this entire process by making function calls only, as this can impair your ability to solve the problem and think insightfully.

GPT-4.1 follows instructions very carefully, especially in agent setups.

- By giving it three clear reminders at the start (about persistence, using tools, and planning), it works much better—about 20% better in tests.
- These reminders help the model act less like a basic chatbot and more like a smart, independent assistant that can move tasks forward on its own.

GPT-4.1 PROMPTING GUIDE

Discussion

b) Tool Calls:

- GPT-4.1 is better at using tools passed through the API.
- So, instead of putting tool details directly into your prompt and writing custom logic, you should use the tools field fields to pass tools — it's cleaner and causes fewer mistakes. In OpenAI's testing, this method worked 2% better than the old way.

Tips for Developers:

- Name your tools clearly so the model understands what they do.
- Write short but clear descriptions for each tool and its parameters.
- If tools are complex, add an # Examples section in your system prompt (not in the description itself).

GPT-4.1 PROMPTING GUIDE

Discussion

- This helps the model know:
  - When to use a tool
  - What to include
  - Which parameters to pick
- You can also use "Generate Anything" in the Prompt Playground to help write tool definitions easily.

GPT-4.1 PROMPTING GUIDE

Discussion

c) Prompting-Induced Planning & Chain-of-Thought

- GPT-4.1 doesn’t naturally “**think step-by-step**,” but you can tell it to plan out loud in your prompt.
- Instead of quickly calling tools one after another, you can prompt it to pause, explain what it’s doing, and then act.
- This is called "**Prompting-Induced Planning**" or using "**Chain-of-Thought**" prompting.
- It’s like making the model explain its thinking before doing something.
- In testing, adding this kind of step-by-step planning made the model perform 4% better on agent tasks.

GPT-4.1 PROMPTING GUIDE

Discussion

d) Sample Prompt: SWE-bench Verified:

- Link for sample prompt ➔ https://cookbook.openai.com/examples/gpt4-1_prompting_guide#sample-prompt-swe-bench-verified

2. Long Context:

GPT-4.1 can handle very long inputs — up to 1 million tokens.

This makes it great for tasks that involve a lot of information, like:

- Reading and understanding big documents
- Picking out the important parts and ignoring the rest

GPT-4.1 PROMPTING GUIDE

Discussion

- Re-ranking results (like choosing the best answers)
- Doing multi-step reasoning using info from different parts of the text

In short: GPT-4.1 is good at thinking over long content and making sense of it.

a) Optimal Context Size

- GPT-4.1 works well even with huge inputs (up to 1 million tokens) — like finding a small detail in a big pile of text (a “needle in a haystack”).
- It’s also good at tasks that mix useful and useless content, like sorting through big documents or code.
- **But there’s a limit:** If the task needs the model to remember and reason about everything at once (like doing a graph search or keeping track of many things across the whole input), its performance can start to drop.

GPT-4.1 PROMPTING GUIDE

Discussion

b) Tuning Context Reliance

- Tuning Context Reliance means deciding how much the model should rely on:
  - 📄 External context (what you give it in the prompt)
  - 🧠 Internal knowledge (what the model already knows)
- 📜 **Two instruction styles:**
  - ***Use ONLY the context given***→ "Only answer using the documents I give you. If it's not there, say you don't know — even if you think you do.
  - ***"Use both context and model knowledge***→ "Use the documents first, but if something's missing and it's basic knowledge you're sure about, go ahead and use it too."

GPT-4.1 PROMPTING GUIDE

Discussion

c) Prompt Organization

Where you put instructions in a long prompt matters.

- ✅ **Best performance:** Put instructions both before and after the context
- 👍 **Second best:** Put instructions before the context only
- ⚠️ **Not ideal:** Putting instructions only after the context

This helps GPT-4.1 follow your instructions better, especially when the prompt is long.

Tip: For long tasks, repeat key rules at the top and bottom to get the best results.

GPT-4.1 PROMPTING GUIDE

Discussion

3. Chain of Thought (CoT)

- **GPT-4.1 doesn’t automatically reason step by step**, but you can tell it to **“think things through”** by adding clear instructions to your prompt.
- This is called **Chain of Thought (CoT) prompting**. It helps the model **break problems into smaller steps**, solve them better, and give more accurate answers.
  However, it uses more tokens, so it can be a bit **slower and costlier**.

GPT-4.1 PROMPTING GUIDE

Discussion

How to Use It:

You can start with this kind of instruction at the end of your prompt:

> "First, think carefully step by step about what documents are needed to answer the query. Then, print out the TITLE and ID of each document. Then, format the IDs into a list."

To Improve Chain of Thought:

If the model makes mistakes:

- **Check where it went wrong** — maybe in understanding the question, analyzing the context, or reasoning through the steps.

GPT-4.1 PROMPTING GUIDE

Discussion

- **Add clearer instructions** to fix those mistakes.
- If it tries different strategies and one works well, **lock that in** with more specific prompting.

Example Reasoning Strategy:

- **Understand the question**
  - Figure out what it’s asking and use context to clear up confusion

GPT-4.1 PROMPTING GUIDE

Discussion

- **Analyze the documents**
  - For each one, explain why it may help or not
  - Rate it: high / medium / low / none
- **Summarize**
  - Pick out the most useful documents and explain why

This method helps the model **slow down, reason better, and make fewer mistakes.**

GPT-4.1 PROMPTING GUIDE

Discussion

5. General Advice

a) Prompt Structure

For reference, here is a good starting point for structuring your prompts.

```
# Role and Objective
# Instructions
## Sub-categories for more detailed instructions
# Reasoning Steps
# Output Format
# Examples
## Example 1
# Context
# Final instructions and prompt to think step by step
```

GPT-4.1 PROMPTING GUIDE

Discussion

- Reasoning steps are chain of thoughts, it includes:
  - Collection of code
  - Deep analyze it
  - Check problem in it and see relevant files
  - Give solution of those problems and update it in patch form
  - Then patch you have updated, run test on it
  - If any test will fail, check and rectify it
  - Run some test from your end apart from given/suggested test.

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- Initially LLMs give responses in markdown format, but now we can give input in markdown, xml, Json formats, among these markdown is easiest technique.
- https://www.markdownguide.org/cheat-sheet/
- https://www.markdownguide.org/basic-syntax/
- Markdown is for formatting. The Markdown elements outlined in the original design document.
- We can preview markdown using Ctrl + Shift + V on VS code and Cursor. Other ways are Colab text option and readme file in md format.

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

Heading Best Practices

- ✅ **Do this**
  - # Here's a Heading
  - Try to put a blank line before...
  - # Heading
  - ...and after a heading.

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

Heading Best Practices

- ❌ **Don't do this**
  - #Here's a Heading
  - Without blank lines, this might not look right.
  - # Heading
  - Don't do this!

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- To create **paragraphs**, use a blank line to separate one or more lines of text.
- Don't put tabs or spaces in front of your paragraphs. This can result in unexpected formatting problems.
- Keep lines left-aligned like this. Don't add tabs or spaces in front of paragraphs.
- To create a **line break** or new line (
), end a line with two or more spaces, and then type return.

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- Best practice of line break are:
  - First line with two spaces after. And the next line.
  - First line with the HTML tag after.
. And the next line.
- You can add **emphasis** by making text bold or italic.
  - To bold text, add two asterisks or underscores before and after a word or phrase.
    - I just love **bold text**.
    - I just love __bold text__.
    - Love**is**bold

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- To **italicize** text, add one asterisk or underscore before and after a word or phrase. To italicize the middle of a word for emphasis, add one asterisk without spaces around the letters.
  - * Italicized text is the *cat's meow*.
  - * Italicized text is the _cat's meow_.
  - * A*cat*meow
- To emphasize text with ****bold**** and ***italics*** at the same time, add three asterisks or underscores before and after a word or phrase. To bold and italicize the middle of a word for emphasis, add three asterisks without spaces around the letters.

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- This text is ***really important***.
- This text is ___really important___.
- This text is __*really important*__.
- This text is **_really important_**.
- This is really***very***important text.

Blockquotes

- To create a blockquote, add a > in front of a paragraph.
- **>** Dorothy followed her through many of the beautiful rooms in her castle.

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- Blockquotes can be nested. Add a >> in front of the paragraph you want to nest.
- > Dorothy followed her through many of the beautiful rooms in her castle.
- >
- >> The Witch bade her clean the pots and kettles and sweep the floor and keep the fire fed with wood.

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- You can organize items into **ordered and unordered lists.**
- To create an **ordered list**, add line items with numbers followed by periods. The numbers don’t have to be in numerical order, but the list should start with the number one.
- To create an **unordered list**, add dashes (-), asterisks (*), or plus signs (+) in front of line items. Indent one or more items to create a nested list.
- If you need to start an **unordered list item with a number** followed by a period, you can use a **backslash (\)** to escape the period.

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- Check more detail in below link
- https://www.markdownguide.org/basic-syntax/
- **Agents are stateless but when we send system prompt to it (using GPT 4.1) and used markdown in it, how it becomes stateful?**
  OpenAI made this mechanism and similarly they made some system on backend like managing requests from specific IP address for specific time, because according to persistence technique they have placed data somewhere as a temporarily

OPENAI AGENTS SDK, INTRODUCTION TO MARKDOWN

Discussion

- First use case of Prompt Engineering is in system prompt (agent instructions), other usage is in tool descriptions.
- We are using markdown instead of plain text in agent's instruction and in tool description because OpenAI, Google and other platform have trained their models on markdown instead of plain text.

OPENAI AGENTS SDK, AGENT HOOKS & RUNNER HOOKS

- Article → Multi-Agent Portfolio Collaboration with OpenAI Agents SDK → https://cookbook.openai.com/examples/agents_sdk/multi-agent-portfolio-collaboration/multi_agent_portfolio_collaboration
- https://notebooklm.google.com/ → you can mind map anything in it, even in quiz preparation.
- Focus on OOPs first (if you are weak in it)
  - Data classes
  - Generic
  - Async Programming
  - Understand conversation with LLMs
- LLMs understanding includes:
  - Prompt Engineering (from cookbook of GPT 4.1 non reasoning model)

LIFECYCLE

- https://openai.github.io/openai-agents-python/ref/lifecycle/
- **RunHooks (alias for RunHooksBase)**
- **Purpose**: Track lifecycle events across an entire agent run (including handoffs and tool usage).
- **Methods to override (async)**:
  - **on_agent_start(context, agent)**
    Before an agent begins executing; also fired on agent handoffs.
  - **on_agent_end(context, agent, output)**
    When an agent produces its final output.
  - **on_handoff(context, from_agent, to_agent)**
    Triggered when execution shifts from one agent to another.
  - **on_tool_start(context, agent, tool)**
    Before invoking a tool (e.g., search, function call).
  - **on_tool_end(context, agent, tool, result)**
    After the tool finishes and returns results.

LIFECYCLE (RUNHOOKS)

- RunHooks and AgentHooks are different, in Runner loop will run which run Agent, Agent is an instance of class.
- RunHooks connect with runner

1. Why do we write generic classes as MyClass[T] with []?

- In Python (since PEP 484 / PEP 585), when you define a generic class, the [] is used to bind the type variable (T) to a concrete type when you instantiate or annotate the class.

LIFECYCLE (RUNHOOKS)

2. What if each attribute should have a different type?

- In that case, **you do not need to use a generic class at all**, unless you want to make it generic over multiple types using **multiple type variables**.
- **Option A: Just define types directly (most common case):**

```python
from dataclasses import dataclass

@dataclass
class MyTest:
    id: int
    idsub: str
    abc: float
```

LIFECYCLE (RUNHOOKS)

- **Option B: Use multiple type variables (if you want it generic):**

```python
from typing import TypeVar, Generic
from dataclasses import dataclass

T1 = TypeVar('T1')
T2 = TypeVar('T2')
T3 = TypeVar('T3')

@dataclass
class MyTest(Generic[T1, T2, T3]):
    id: T1
    idsub: T2
    abc: T3
```

```python
test = MyTest[int, str, float](id=123, idsub="abc", abc=1.23)
```

LIFECYCLE (AGENTHOOKS)

AgentHooksBase

`Bases: Generic[TContext, TAgent]`

- A class that receives callbacks on various lifecycle events for a specific agent. You can set this on agent.hooks to receive events for that specific agent.
- Subclass and override the methods you need.
- Agenthook is present with every agent, similarly runhook present with Runner.

LIFECYCLE (AGENTHOOKS)

AgentHooksBase (alias for AgentHooksBase)

- **Purpose:** Attach lifecycle callbacks to a specific agent instance via agent.hooks.
- **Methods to override (async):**
  - **on_start(context, agent):** Just before this agent begins execution (or takes over after handoff).
  - **on_end(context, agent, output):** When this agent completes with its final output.
  - **on_handoff(context, agent, source):** When control is handed to this agent from another (source).
  - **on_tool_start(context, agent, tool):** Before this agent invokes a tool.
  - **on_tool_end(context, agent, tool, result):** After the tool invoked by this agent returns results.

RUNCONTEXTWRAPPER

- https://openai.github.io/openai-agents-python/ref/run_context/
- **Bases: Generic[TContext]**
- This wraps the context object that you passed to Runner.run(). It also contains information about the usage of the agent run so far.
- **NOTE:** Contexts are not passed to the LLM. They're a way to pass dependencies and data to code you implement, like tool functions, callbacks, hooks, etc.

TOOL_USE_BEHAVIOR

- https://openai.github.io/openai-agents-python/ref/agent/#agents.agent.Agent.tool_use_behavior

FOR EXAMS PERSPECTIVE

- https://github.com/panaversity/learn-agentic-ai/blob/main/certification.md#3-fundamentals-of-agentic-ai-level-2-professional-quiz
- **Covers**:
  - OpenAI Agents SDK architecture (Agents, Tools, Handoffs, Runner).
  - Pydantic models for typed inputs/outputs.
  - Async programming and multi-agent workflows.
  - Prompt engineering (Chain-of-Thought, Tree-of-thoughts, system message design, sensitive data handling).
  - Basic Markdown syntax.

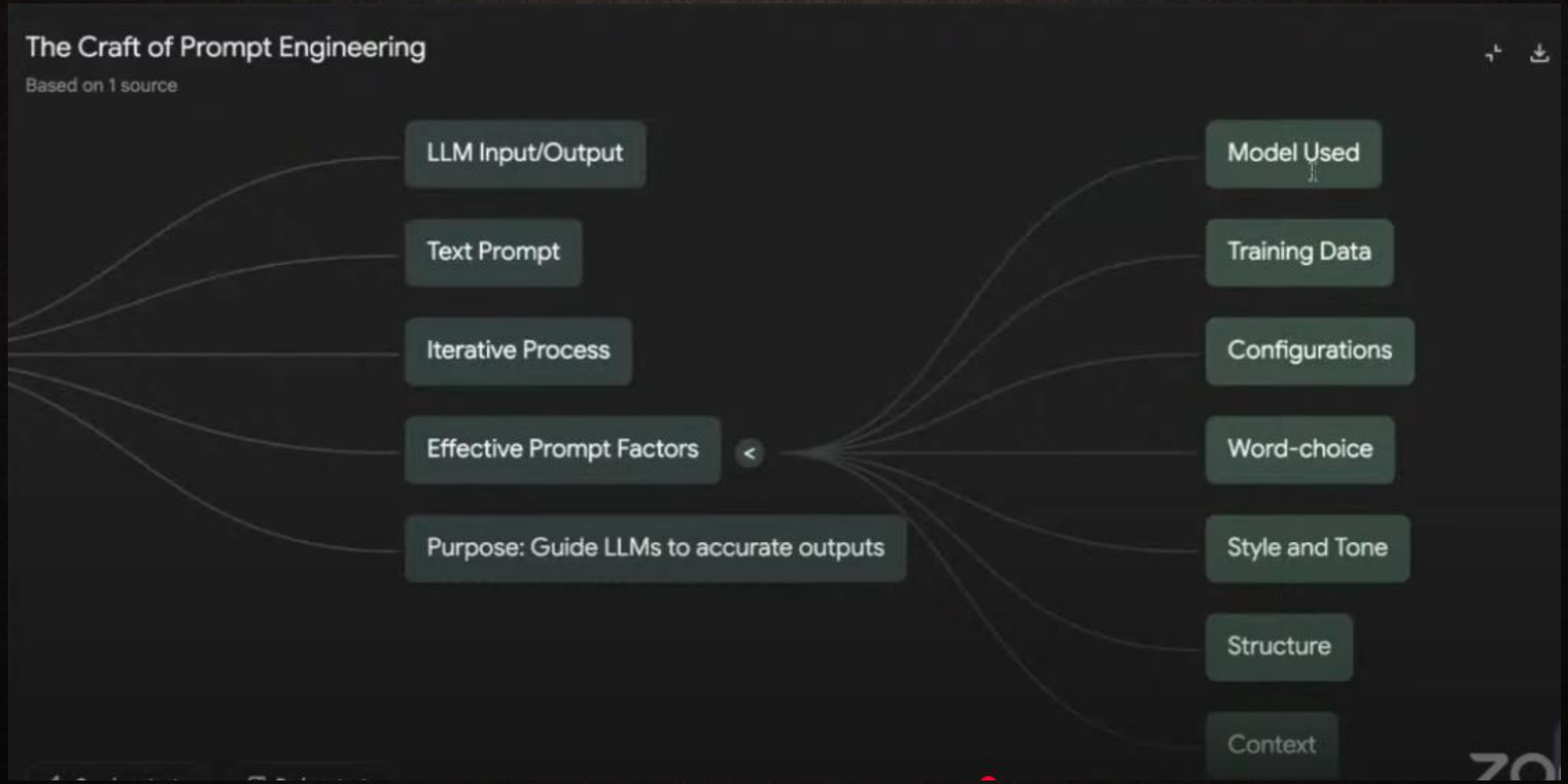
FOR EXAMS PERSPECTIVE

- Source
  - OpenAI Agents SDK Documentation
  - OpenAI Cookbook: Multi-Agent Portfolio Collaboration
  - OpenAI GPT-4.1 Prompting Guide
  - Kaggle Whitepaper on Prompt Engineering
  - Markdown Basic Syntax
  - Markdown Cheat Sheet
  - Colabs: https://github.com/panaversity/learn-agentic-ai/tree/main/01_ai_agents_first

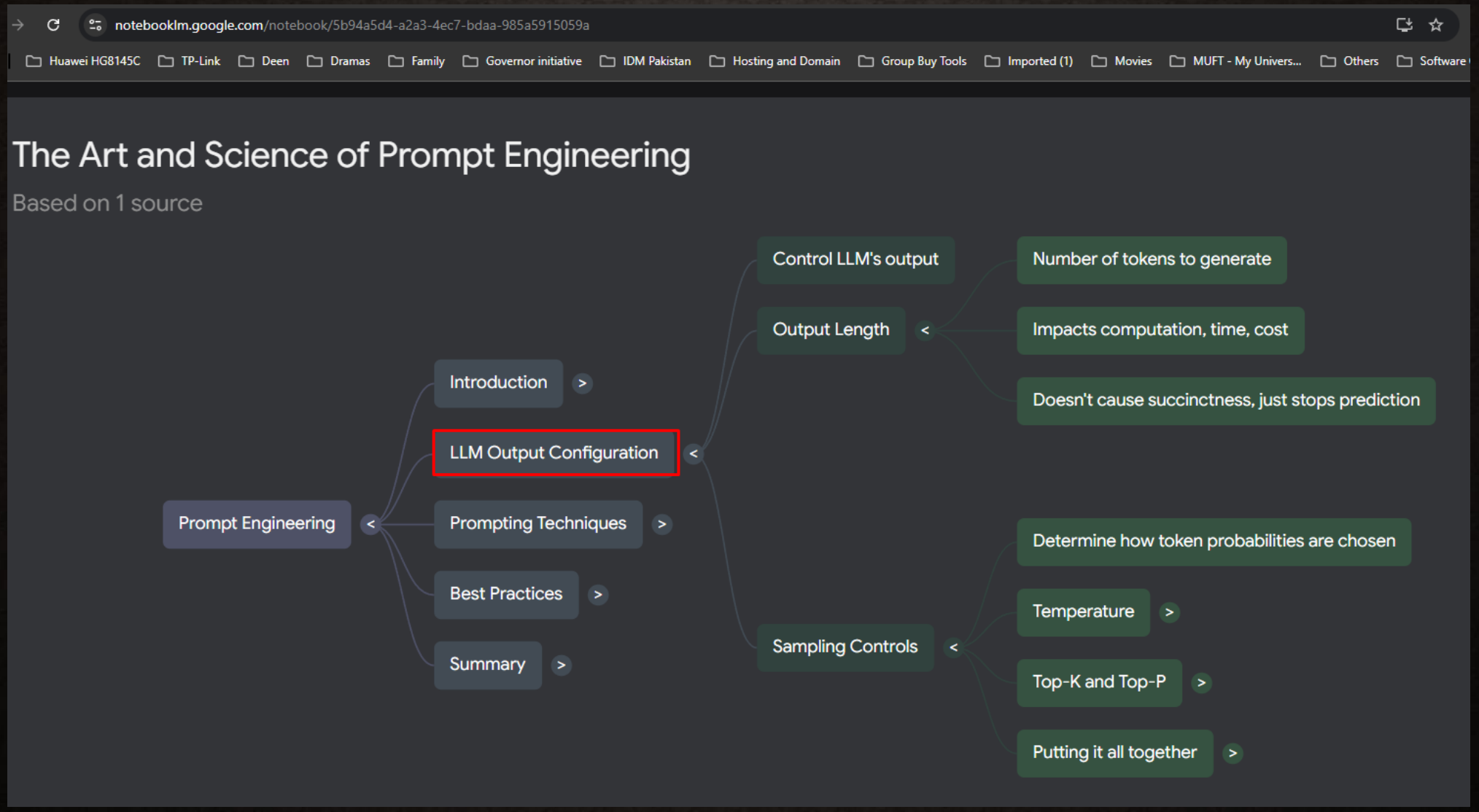
PROMPT ENGINEERING

- https://github.com/panaversity/learn-agentic-ai/blob/main/certification.md
- https://www.kaggle.com/whitepaper-prompt-engineering
- Kaggle whitepaper is a Google Gemini prompt engineering technique introduced in Feb 2025
- Give this Kaggle's PDF to notebook LLM to make graphical representation to understand it easily.
- We are discussion this whitepaper in below slides.

INTRODUCTION

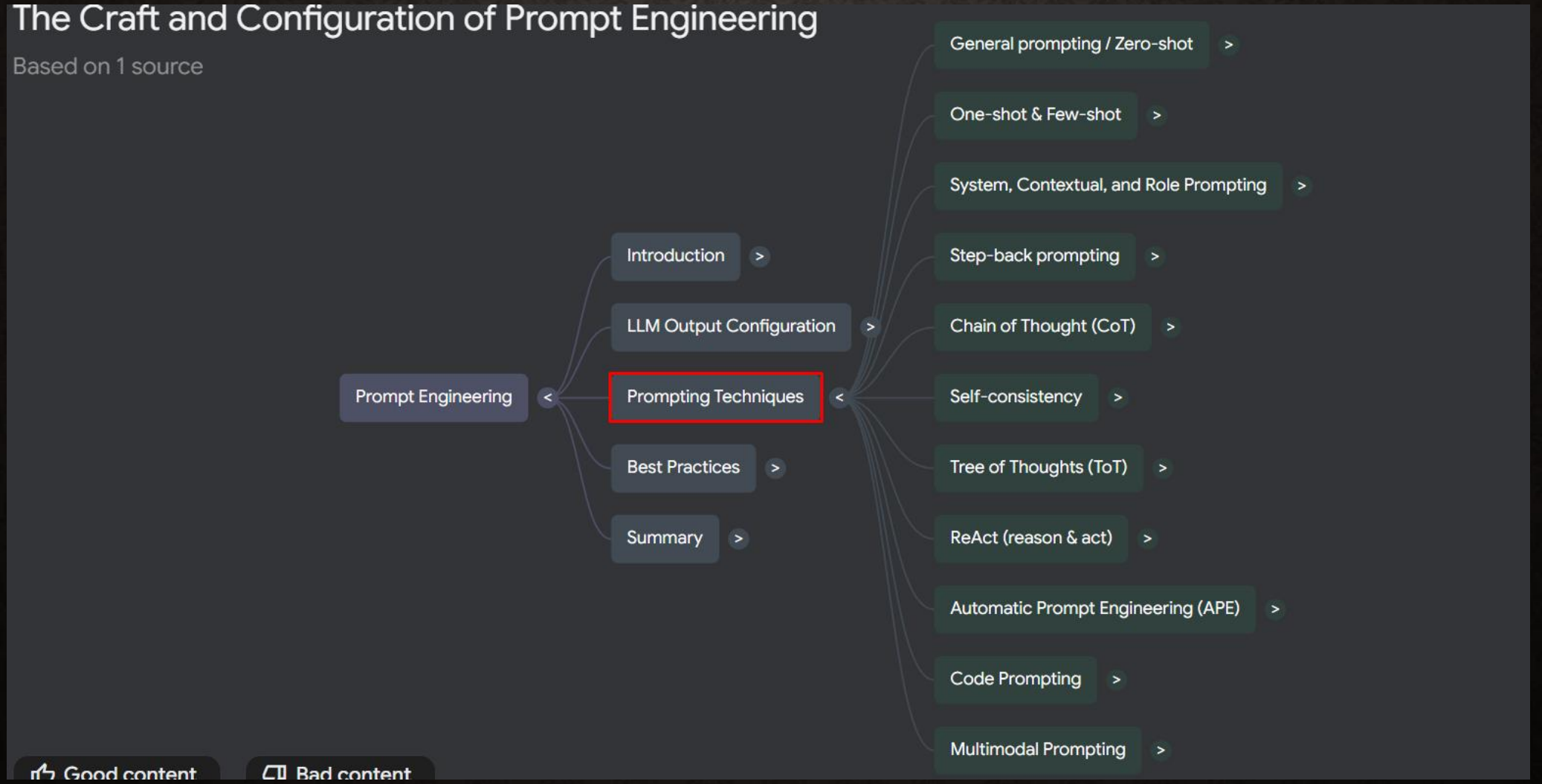


LLM OUTPUT CONFIGURATION



KAGGLE WHITEPAPER

Continuation of last class



KAGGLE WHITEPAPER

Continuation of last class