SUMMARY: BASICS OF PROMPT ENGINEERING

Objective:

By the end of this session, students will be able to:

- Understand what prompts are and why they are essential in communicating effectively with language models.
- Identify and apply the key components of a well-structured prompt, including role, instruction, input, and output format.
- Learn and practice core prompt engineering techniques such as Zero-Shot, One-Shot, Few-Shot, and Role-Based prompting.
- Recognize the importance of clarity, specificity, tone, and structure in crafting high-quality prompts.
- Explore common challenges like the "loss in the middle" issue and understand strategies to minimize its impact.
- Compare prompting strategies and understand when to use structuring, role assignment, or example-based prompting for different tasks.

In our previous sessions, we laid a strong foundation by exploring the core concepts of Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL), and Generative AI (GenAI). We discussed what Large Language Models (LLMs) are, how they are built and trained, and differentiated between open-source and closed-source models.

With that foundational knowledge in place, we now move into a crucial and exciting part of working with GenAI — **Prompt Engineering.**
"If LLMs are powerful engines, prompts are the controls — and from today onwards, we begin learning how to drive them like pros."

Prompt Engineering?

We often hear:

"The prompt should be clear."
"You need a better prompt."
"That's a poorly structured prompt."

But wait — **what *is* a prompt?**
Before diving into *prompt engineering*, we need to understand the *foundation* — the prompt itself.

A **prompt** is the instruction or input you give to a language model (like ChatGPT, Gemini, or Claude) to generate a response.

It's your **way of communicating** with the model, asking a question, giving a task, or requesting specific behavior.

For example: What is Artificial Intelligence?- It is a prompt

Now, let's connect it with a real-life interaction:

You walk into a restaurant and say to the waiter:
"Get me food."

What happens?
The waiter looks confused. *"What kind of food? Veg or non-veg? Spicy or mild? Starter or main course?"*

You didn't give enough detail, so you'll likely get something random… maybe not even what you wanted.

Now, instead, you say:

> "I'd like a spicy paneer tikka starter, served hot, and a glass of lemon soda. Make sure it's not too salty."

Suddenly, the waiter knows exactly what to bring. You get exactly what you imagined — maybe even better!

Similarly, you open ChatGPT or Gemini and type:
"Explain photosynthesis."
You get a basic textbook-style definition.

Now you rephrase it:
"Explain photosynthesis like I'm 12, using emojis and a simple story."
Suddenly, it turns into a fun, engaging explanation involving emojis, etc

So, Prompt Engineering is the craft of writing smart, structured, and context-aware instructions that guide AI models to produce high-quality, useful, and creative outputs.

It's not just about "asking" something — it's about knowing **how to ask**, **what to include**, and **how to guide the model** to understand your intent.

Why Prompt Engineering Matters?

Because no matter how powerful an LLM is, **its output depends on your input**.
Even the smartest models need the right directions.

Let's imagine you and your friend are using the **same AI model**, like ChatGPT.

You type:
"Tell me something about climate change."
You get a general, surface-level response.

Your friend types:
"Act as an environmental journalist. Write a 200-word report explaining the recent rise in global temperatures, with statistics and real examples."
They get a detailed, focused, and high-quality answer.

Same model, completely different outcomes. Why?
Because of **how the prompt was written**.
That's the power of Prompt Engineering — you're not changing the model, just the *way you talk to it*.

Prompt Structuring: The Building Blocks of a Great Prompt

*A prompt is not just a question. It's a **set of instructions** you're giving to the AI.*
*And just like people, AI models perform best when those instructions are **clear, complete, and well-organized**.*

You can have brilliant wording and perfect tone…
But if you forget to tell the model *who it is*, *what to do*, or *what format to follow* — you'll likely get a response that's off-track.

That's where **prompt structuring** comes in.

So now that we understand **why structure matters**,
let's break down a well-structured prompt into its **essential building blocks** —
the parts that help the AI know *who it is*, *what it needs to do*, *what information it should use*, and *how the output should look*.

These are called the **Prompt Components**.

1. Prompt Components

Think of a prompt like giving instructions to a super-smart assistant. If you're too vague, you get average results. But if you're clear, specific, and structured, the magic happens.

To get that magic, most great prompts include **4 essential components**. Let's explore each with examples.

a. Role – Who Should the AI Be?

Imagine you're walking into a theater rehearsal.
You say to an actor:
"Say something."

They might be confused — *as who? In what tone? What context?*

But now you say:
"You're playing the role of a strict judge in a courtroom."
Suddenly, the actor straightens up, changes posture, and speaks with authority.

That's exactly how **LLMs respond to role instructions.**
When you define a clear role, you're not just feeding a sentence — you're helping the model *"get into character."*

Why it matters:
When you say, "You are a senior UX designer," the model instantly adopts that lens, and the response reflects it.

Examples:

- "You are a travel blogger who writes short Instagram captions."
- "You are a strict grammar teacher correcting essays."
- "Act like a Sherlock Holmes-style detective solving a case."

b. Instruction – What Should the AI Do?

This is the heart of your prompt — the **specific task** or **goal** you're asking the AI to perform.

Why it matters:
Just like giving instructions to a human, unclear tasks confuse even the smartest systems.

Examples:

- "Summarize this text in two lines."
- "Translate this paragraph into Spanish."
- "Write a thank-you email to a recruiter"

c. Input – What Information Does It Need?

The AI works best when you feed it context — this is your **input**, like a piece of text, a dataset, a paragraph, or even a question.

Why it matters:

LLMs are not magic — they work based on patterns in context.
If you don't give them input, they'll guess based on general knowledge, which often leads to generic, vague, or wrong answers.

In other words:

No input = No context = Poor output

Just like humans give better answers when they know the background, AI models also perform better when they have the right information in front of them.

Examples:

- "Here's the resume: [paste content]"
- "This is the paragraph I wrote: [paste] — improve its tone."
- "Based on this job description: [paste JD], write a cover letter."

"The better your input, the better your output."

d. Output Format – How Should the Answer Look?

You've set the role, given the task, provided the input — now tell the AI **how to deliver the result**. Define tone, length, format, or structure.

Why it matters:
It avoids messy or unusable answers — you get exactly what you need.

Examples:

- "Reply in bullet points under 100 words."
- "Write it as a tweet, under 280 characters."
- "Present it in JSON format."
- "Write a paragraph in a formal tone with 3 key points."

Let's now combine everything we've learned — **Role, Instruction, Input, and Output Format** — into one powerful prompt:

Prompt:

"You are a professional career coach with 10+ years of experience.
Review the following resume and suggest improvements that would make it more appealing for product management roles.
Resume: [Insert resume text here]
Provide your feedback in bullet points, under 150 words, and use a friendly, constructive tone."

Role → *Sets the AI as a career coach.*
Instruction → *Clear task: review and suggest improvements.*
Input → *The resume.*
Output Format → *Bullet points, under 150 words, with a specific tone.*

So far, we've learned how to **structure a prompt** —
We define the **role**, give clear **instructions**, provide **input**, and shape the **output format**.

Structuring a prompt is like planning a message before sending it.
You think: *Who am I talking to? What do I want? What do they need to know?*

But here's a subtle and often overlooked truth:

Even when your prompt is **well-structured**, even when it has **all the right components**...

The **position** of information inside the prompt still matters.

2. Loss in the Middle:

> *Imagine reading a long WhatsApp message.*
> *You read the beginning carefully.*
> *You scroll down quickly.*
> *You focus again at the end.*
> *But what happened to the middle part? You skimmed it.*

That's exactly what happens in **LLMs**.

AI models (especially large ones) pay more attention to:

- The **start** (where instructions usually are)
- The **end** (where examples or final tasks are)

Anything in the **middle** risks being ignored or misunderstood, especially if the prompt is long.

Why this happens?
Because Language models are trained to process input **token by token** —
They pay the most attention to the **beginning and the end**.
But the **middle often gets less weight**, and sometimes, key instructions hidden there are **lost or ignored**.

This is called: **Loss in the Middle**

So, let's understand how you should **structure your prompts** to reduce this issue.

1. Place Critical Instructions Early

Put the most important part of the instruction at the **very beginning**.

Instead of:
"Let's write a blog. It should be about mental health. Use a friendly tone. Use emojis. Keep it under 200 words."

Do this:
"Write a short (under 200 words), friendly blog post about mental health using emojis."

2. Repeat Key Points at the End (Recency Bias)

Models remember the last part too. So, **restate the key instruction** at the end.

Example:

> "Summarize the following article in 3 bullet points...
> [long article]
> — Reminder: Summarize in just 3 short bullet points."

3. Use Clear Formatting

Use **numbered steps, bullet points, or bold words** to highlight important parts.

Your task:

*1. Translate the **text to French***
*2. Keep the tone **casual***
*3. Output **only the translation***

The model is more likely to notice this structure.

4. Break Long Prompts into Smaller Calls

Instead of one big prompt, use **multi-turn prompts** or **chain of smaller instructions**.

Example: Long Prompt (Less Effective)

> "Summarize the article, highlight the main points, suggest a tweet based on it, and explain why it matters — all in a friendly tone under 150 words."

Problem: Too many instructions in one go — the model may miss or blend parts.

Better Approach: Break It Down

Prompt 1:

"Summarize this article in 3 bullet points."
(Output) → [Model gives summary]

Prompt 2:

"Based on that summary, write a tweet in a friendly tone."
(Output) → [Tweet]

Prompt 3:

"Now explain in 2 lines why this topic is important."

Each task is handled clearly and step-by-step → **better control, cleaner output**.

So by keeping these points in mind, we can structure the prompt effectively,

Now, you have all the components in place —
But if your words are vague, the tone is off, or the instruction is dry…
You still won't get the best results.

That's where **Prompt Crafting** comes in.

So let's move from *what to say and where to place it*,
to *how to say it so the model gives you the best possible output*.

Prompt Crafting Techniques:(The Art of Prompt Wording)

So far, we've understood **how to structure a prompt** —
We've learned that giving the model a clear role, task, and context leads to more accurate and relevant responses.

But **structure alone isn't enough**.

Just like a well-planned house still needs smart interior design,
A well-structured prompt still needs thoughtful **crafting** —
how we **phrase**, **tone**, and **guide** the model to give us the best possible output.

Imagine you're speaking to a lawyer.

You start with a well-structured statement:

"I need legal advice regarding a property issue."

That's clear. The **role is defined**, the **topic is set**. Good structure!

But now consider two ways you craft your message:

Example 1: Poorly Crafted Prompt

> *"I want to know… is it bad if there's something wrong with the papers or like if it's fake or something?"*

The lawyer is confused — too vague, unsure what exactly you want.

Example 2: Well-Crafted Prompt

> *"Can you walk me through how to verify a land title legally in Delhi?*
> *Also, what common red flags should I watch for in the documentation?"*

Now the lawyer can:

- Focus precisely on **your goal**
- Understand the **jurisdiction and scope**
- Respond in an informed, structured way

Just like that — **AI performs better** when your prompt is not just structured, but **crafted with clarity, precision, and intent.**

Crafting is where you:

- Choose the right **tone** ("Explain like I'm five" vs "Be professional")
- Select the right **style** (list, summary, story, code block)
- Control the **depth** (surface-level vs in-depth)
- Include **constraints** (word limits, step-by-step, do not assume

It's not just about *what* you ask, but *how* you ask it.

Now that we know how to structure a prompt,
let's learn how to **craft it thoughtfully** —
so we can **guide the model's tone, behavior, and output** to get responses that are not just correct, but **truly useful.**

a. Clarity and Conciseness:

> *Be clear. Be direct. Remove ambiguity.*

LLMs respond best when your instructions are **specific and streamlined**. If the model has to guess what you're asking, you'll get vague or off-target results.

Do this:

- Use active language
- Avoid unnecessary filler words

- Focus on the *intent* of your task

Poor Prompt:
"Tell me something helpful."
(Model doesn't know what "helpful" means in this context.)

Better Prompt:
"Give me 3 quick tips to stay focused while studying for exams."

b. Specificity:

Add context: Who, what, why, how, and for whom.

Specific prompts give the model **boundaries and focus**. You're telling the model not just *what* you want, but *how it should frame* the response.

Do this:

- Mention the target audience
- Include the use-case or intent
- Specify format, style, or language

Poor Prompt:
"Write a message."

Better Prompt:
"Write a birthday message for my manager in a professional tone."
"Write a thank-you email after a job interview for a marketing role."
"Write a welcome message for new interns joining our design team."

c. Tone and Style:

Control how the model sounds — fun, serious, simple, formal, technical, etc.

Tone affects the **feel** of the response, while style affects its **format** and **depth**. You can ask the model to write like a teacher, a friend, a reporter, or even mimic a known personality.

Common tone modifiers:

- "Use a friendly, conversational tone"
- "Be concise and professional"
- "Use a formal business tone"
- "Make it playful and quirky"

Examples:

- "Explain machine learning like I'm 10 years old."
- "Summarize this privacy policy in a formal tone."
- "Describe our app in a fun, informal style for a Gen Z audience."

d. Depth and Constraints

Control how much detail you want and how it's formatted.

Sometimes, you want a deep explanation. Other times, you need a short, scannable response. Prompt constraints help **manage verbosity** and ensure consistency.

Types of constraints you can apply:

- Number of points
- Word or character limits
- Paragraph count
- Step-by-step breakdown
- Do's and don'ts

Examples:

- "List 5 pros and cons of electric cars, each under 20 words."
- "Write a 3-paragraph intro about AI's impact on education."
- "Give a step-by-step explanation of how blockchain works — in under 100 words."
- "Summarize this article in exactly 2 bullet points."

e. Intent Framing

Guide how the model should approach the task, not just what to do.

Intent framing helps the model **take on a mindset or behavioral style**, especially for open-ended tasks like ideation, analysis, or conversation.

You're telling the model:

"Here's not just what to do, but how to think while doing it."

Common intent frames:

- "You are brainstorming, not finalizing."
- "Give constructive but kind feedback."
- "Act like a critical reviewer."
- "Suggest ideas, but include pros and cons."

Examples:

- "Act like a mentor giving constructive but kind feedback on a student's portfolio."
- "Help me brainstorm startup ideas for sustainability — don't finalize anything yet."
- "Be creative and playful — suggest funny names for a productivity app."

This sets expectations for *how* the AI should behave, and is especially useful in creative or conversational use cases.

Now, let's go one level deeper and explore the **techniques** used to *refine* and *optimize* how we prompt the model.

Prompting Techniques:

So far, we've learned how to structure a prompt and how to craft it carefully —
Choosing the right tone, style, clarity, and depth to shape how the model responds.

But let me ask you this:

What happens when you don't just want the model to respond clearly?
But to learn a pattern, mimic a style, or reason through a problem?

Crafting helps you fine-tune how the model talks.
But sometimes, you need to teach it how to think.

That's where we now move from prompt wording to prompt strategy.

> "Knowing *what* to ask is good —
> But knowing *how* to ask it in different ways is *powerful*."

Before jumping into Techniques, let's relate this to **real life**.

For Instance:

Let's say you're teaching someone how to solve a math problem.

You could:

1. Just give them the problem and expect an answer → Zero-Shot Prompting
2. Give one solved example first, then a new problem → One-Shot Prompting
3. Show multiple examples before asking for a solution → Few-Shot Prompting
4. Tell them to act like a math teacher or examiner → Role-Based Prompting
5. Ask them to think aloud, step-by-step → Chain-of-Thought Prompting
6. Make them reason before acting → ReAct Prompting
7. Have them try multiple answers and pick the most consistent one → Self-Consistency
8. Or ask them to explore all branches of thought, like a decision tree → Tree of Thought

Each approach changes how students understand and solve the task.

This is exactly how **Prompt Engineering Techniques** work in Generative AI. These techniques define **how much context or example** we give to the model to help it perform better.

There are many Prompting techniques:

1. Zero Shot Prompting
2. One-Shot Prompting
3. Few-Shot Prompting
4. Role-based Prompting
5. Chain-of-Thought Prompting
6. ReAct Prompting (Reason + Act)
7. Self-Consistency Prompting
8. Tree of Thought (ToT) Prompting

In today's session, we'll focus on the core techniques:
Zero-Shot, **One-Shot**, and **Few-Shot Prompting, Role-based Prompting** — the foundational building blocks of prompt engineering.

The remaining advanced techniques — including **Chain-of-Thought**, **ReAct**, **Self-Consistency**, and **Tree of Thought** — will be covered in detail in the **next session**.

Now, let's understand all the techniques one by one;

1. Zero Shot Prompting:

Imagine you're managing a helpdesk, and a **new employee joins on their first day**.

You tell them:

> "Reply to this customer email."

But you don't give them any training, no examples, no previous replies — just the task.

They'll try to figure it out on their own, based on general knowledge and common sense.

This is exactly what **Zero-Shot Prompting** is.

You're giving the model a task **without any prior examples**, relying purely on its **pretrained knowledge** to generate a response.

Example:

Prompt:

"Translate the following English sentence to French:
'The weather is nice today.'"

There might be a chance that model produce wrong or vague output if the model has never seen your specific prompt before in their training

Output: “Il fait beau aujourd’hui.”

Advantages:

- **Simplicity:** Prompts are simple to construct and easy to understand. This approach allows for users to experiment with different prompts without deep prompt engineering knowledge.
- **Ease of use:** Zero-shot prompting doesn't require any additional data, making it valuable in cases where relevant data is difficult to source or scarce.
- **Flexibility:** Prompts are easy to adapt as needed. Improving a prompt or updating a prompt due to changing circumstances requires low effort.

Limitations:

- **Performance Can Be Unpredictable**
  Zero-shot prompting sometimes works well, but not always.
  If the task is very complex, needs expert-level knowledge, or expects a very specific answer — the model might get confused or give a weak response.
  It’s not always as good as a model trained specifically for that task.

- **Depends on What the Model Already Knows**
  Zero-shot prompting works only if the model has already “learned” about that topic during training.
  If the model hasn’t seen much about a certain subject, language, or situation, its answer may not be very accurate.
  The better and broader the model's training, the better the zero-shot result.

2. One Shot Prompting:

Imagine you're teaching a student how to write a thank-you note.

You say:

“Here’s an example of how it’s done:
‘Dear Aunt Maya, thank you so much for the birthday gift. I really loved it!’
Now, write a thank-you note for your uncle.”

The student now understands both the **format** and **tone** — and will likely write a better version than if you’d just said “Write a thank-you note” (Zero-Shot).

This is the essence of **One-Shot Prompting**:
You give the model **one clear example** of the task, so it can understand **what to do and how to do it.**

In **One-Shot Prompting**, you provide the model with **one example input-output pair** before asking it to perform the task on new input.

It helps the model:

- Understand the **format** of the desired output
- Mimic the **style** or logic in your example

For Example:

Prompt:
Convert the sentence to passive voice.

Example:
Input: "The chef cooked the meal."
Output: "The meal was cooked by the chef."

Now try this one:
Input: "The teacher praised the student."

Advantages of One-Shot Prompting

1. Saves Time and Data
You don't need to feed tons of examples to the model. Just **one example** is enough to help it understand the task.
Example: Instead of training on hundreds of emails to detect spam, one good example can show the pattern.

2. Faster to Use and Deploy
Since there's **no heavy training** involved, models using one-shot prompting can be set up and used quickly — great for situations that need **fast responses**.
Useful in: Customer support bots, quick feedback systems, or tools that need to handle new topics fast.

3. Works in Many Areas
One-shot prompting can be used in **a wide range of tasks** — like writing summaries, answering questions, generating code, or recommending products.
It also blends well with **few-shot or zero-shot** techniques depending on the need.

Limitations of One-Shot Prompting

1. Can Inherit Biases
Since the model relies mostly on **pretrained data** and just one example, it may pick up **biases** from the

training phase — especially if the example or topic is sensitive.
Impact: The responses might sometimes reflect unfair or inaccurate views without you realizing it.

2. May Not Be Perfect for Complex Tasks
For tasks that need **deep reasoning or detailed context**, one example might not be enough.
Result: The model might give inconsistent or **less accurate answers** compared to more heavily trained models.

3. Few Shot Prompting:

Imagine you're coaching someone to write Instagram captions for a travel page. Instead of giving them just one example, you give **3–4** good ones to show different styles:

> *"Exploring the streets of Rome — history in every step."*
> *"Sunsets in Bali hit different."*
> *"Woke up in the Swiss Alps. Coffee, calm, and clouds."*

Now, when they write the next caption, they'll likely get the **tone, vibe, and format** just right, better than with just one sample.

That's exactly how **Few-Shot Prompting** works:
You give the model **a few examples** of what you expect before asking it to generate a new response.

In **Few-Shot Prompting**, the prompt includes **2 to 5 example input-output pairs** followed by a new input.

This technique helps the model:

- Learn the **pattern** more clearly
- Generalize better across slightly varied inputs
- Improve performance on **complex or subtle tasks**

4 . Role-Based Prompting:

Imagine this scenario:
You're at a hospital, and you ask **three different people** the same question:

> "Can you explain this medical report to me?"

- **The doctor** gives a detailed, accurate medical explanation.
- **The teacher** simplifies it with analogies and diagrams.
- **The parent** explains it with care, focusing on your comfort.

Though you asked the same question, each person's **role** influenced how they responded.

This is the idea behind **Role-Based Prompting**:
We assign the AI a **specific role** (doctor, teacher, designer, recruiter, etc.) to **shape its response style, tone, and knowledge depth**.

Role-Based Prompting is a prompt engineering technique where we ask the AI to behave like a specific role — such as a teacher, lawyer, therapist, or chef — to generate more accurate, contextual, and user-appropriate outputs.

This approach helps **narrow the context** and **align the output tone and content** to what the user expects from someone in that role.

Example:
Prompt:

> "You are a professional career counselor. Suggest suitable job roles for someone with a background in mechanical engineering and good communication skills."

Output: AI will now act like a career counselor and respond with practical job advice, tone, and structure accordingly.

Compare Few Shots prompting and Role-based Prompting:

We've already seen how **Few-Shot Prompting** helps the AI learn from a few examples, and how **Role-Based Prompting** guides the model to think from a specific expert's perspective.

Now let's compare and contrast these two techniques — their **benefits, differences, and ideal use cases**.

Aspect	**Few-Shot Prompting**	**Role-Based Prompting**
Purpose	Teach the model with a few examples	Guide model behavior by assigning it a role
Style	Example-driven	Persona-driven
Example	"1. Hello → Bonjour 2. How are you? → Comment ça va?"	"You are a French teacher. Translate: What time is it?"
Best For	Tasks needing clear patterns (e.g., translation, classification)	Expert advice, tone control, and simulations
Strength	Learns structure and formatting	Adds depth, context, and realism to responses

When to use	When you want to teach the **task format** with minimal examples	When the task needs **domain knowledge**, tone, or a specific persona
Can Combine?	Yes – More powerful when used together	Yes – Use role + examples for optimal output

Advantage:

Few-Shot Prompting: Learns format & structure from minimal examples, enabling quick adaptability
Role-Based Prompting: Adds contextual depth & tone, improving relevance in responses

Limitations:

Few-Shot Prompting: Performance depends heavily on the quality and relevance of provided examples
Role-Based Prompting: May overfit the role, causing generic or rigid replies if not balanced

So far, we've looked at how to write great prompts — structured, well-worded, and using techniques like few-shot or role-based prompting.

But what happens when you need to do this **at scale**?

- What if your prompt needs to change based on user input?
- What if it needs to pull in data or a document?
- Or use the result of one response to build the next?

That's where **Dynamic Prompting** comes in.

Dynamic Prompting:Making Prompts Smarter with Context

Imagine you're a waiter at a busy restaurant.

Customers walk in one by one. You don't ask everyone the same question, like:

> "What do you want?"

Instead, based on what they say or how they look, you adjust your approach:

- A couple says, "We're celebrating!" → You suggest a special candlelight menu.
- A family with kids → You show them the kids' combo and booster seats.
- Someone looks like they're in a rush → You offer quick-serve items.

You're adapting in real time, personalizing your service based on context.
This flexibility is what makes the experience great.

This is what Dynamic Prompting is.

So, **Dynamic Prompting** is the **technique of automatically adjusting, generating, or modifying prompts** given to an AI based on real-time inputs or changing context, **rather than using fixed, one-size-fits-all instructions.**

How Does It Work?

Instead of using hardcoded prompts, dynamic prompting involves:

- Reading user input
- Extracting context, intent, or preferences
- **Generating a new prompt** based on that

Typically done using:

- Code logic (e.g., in JavaScript, Python)

```
function generatePrompt(topic, tone, audience) {
  return `Explain the topic '${topic}' in a ${tone} tone for a ${audience} audience.`;
}
```

- Templates + dynamic data (like inserting style, audience, tone, etc.)

Template Example:

> *"Summarize the following ${topic} in ${style} style for a ${audience} audience:"*

Dynamic Input:

- topic = "budget report"
- style = "concise"
- audience = "non-finance"

Where it is used?

1. Chatbots & Virtual Assistants

In multi-turn conversations, the user input keeps changing, and so must the prompt.

Example:
A customer support bot dynamically changes the prompt:
"Based on the customer's last message, where they reported an issue with the payment page, respond with troubleshooting steps."

Here, the prompt is generated using the user's previous input, not written once.

2. Personalized Experiences

In recommendation engines, resume coaches, or learning platforms — prompts often need to include **user-specific data**.

The system fills the prompt using **user history, preferences, or past selections**.

Example:
"Hi Rahul, based on your last booking to Jaipur, here are 3 hotel suggestions with better ratings and similar price."

The prompt pulls **past interaction data** and shapes the output.

3. Multi-Step Tasks and Chaining

In agent-based workflows, one prompt's output becomes the next prompt's input.

Example:

1. Prompt 1: "Summarize this resume."
2. Prompt 2: "Now, based on that summary, write a cover letter."
3. Prompt 3: "Suggest 3 interview questions based on the resume."

These chained prompts are dynamically created on the fly.

4. Prompt Templates in Applications

When building AI-powered apps, we often use **templates** with variables to generate custom prompts on the fly.

Example:
Template:
"You are a [role]. Based on this [input], write a [format] that achieves [goal]."

Dynamic prompt:
"You are a social media strategist. Based on this product launch brief, write a 3-line Instagram caption that increases engagement."

"Systems fill these variables based on user selections or input."

So as you can see — Dynamic Prompting isn't just a cool idea — it's how modern GenAI systems work behind the scenes.

Whether it's chatbots, agents, or product features — prompts are rarely static. They adapt.

Conclusion:

1. **Prompts are the Language of AI:**
   Just like instructions to a human, the way you speak to a model through prompts directly affects the output. Clear, structured prompts = better, more relevant results.

2. **Understand the Components Before Crafting:**
   A great prompt includes: a clear **role**, precise **instructions**, relevant **input**, and a well-defined **output format** — all working together to guide the model.

3. **Structure Enhances Clarity:**
   Using prompt structuring techniques like formatting, instruction-first placement, and bullet points reduces confusion and boosts model performance.

4. **Crafting Techniques Add Finesse:**
   Mastering clarity, specificity, tone, and style in your wording can turn a basic response into something polished and professional — small phrasing tweaks matter a lot.

5. **Prompting Techniques Shape Output Powerfully:**
   Techniques like **Zero-Shot**, **One-Shot**, **Few-Shot**, and **Role-Based Prompting** allow flexibility based on context and task complexity, and can even be combined for more powerful and accurate outputs.