📘 Lecture 1: Introduction to Agentic AI & Prompt Engineering

🎯 Overview

Agentic AI represents a shift from passive AI models to active, autonomous systems. These systems **plan**, **reason**, **invoke tools**, and **adapt** to fulfill user goals. Prompt engineering is the key to enabling this behavior.

🤖 What is Agentic AI?

Agentic AI refers to AI systems that can:

- **Understand a goal or objective**
- **Plan** a sequence of steps to achieve it
- **Act** (e.g., call tools, retrieve data, generate responses)
- **Reflect and adjust** based on outcomes

🧠 Example:

> "Book a meeting with John, get the weather for Monday, and summarize the plan in an email."

An agent must:

1. Understand the user's intent.
2. Query calendars and weather APIs.
3. Compose a contextual email.

🧰 Role of Prompt Engineering in Agentic AI

Prompt engineering enables:

Task	Prompt's Role
Define user intent	Guide the agent's goal interpretation
Tool invocation	Provide syntax and format to call functions or APIs
Multi-step reasoning	Structure tasks using step-wise planning
Context management	Pass relevant memory or history through prompts
Safety and constraint handling	Enforce limits, rules, and ethical boundaries

---

👩‍🏫 Real-Life Analogy

Imagine giving instructions to a junior employee:

- You don't just say, "Handle it."

- You outline: goals, steps, tools, and expectations.

Prompting an AI agent is the same. The clearer the prompt, the smarter the behavior.

💬 Prompt Examples

Passive Prompt (Standard)

> "Summarize this article."

Agentic Prompt

> "You are an executive assistant. First, summarize the meeting notes. Then, extract action items. Finally, draft a follow-up email using a friendly tone."

📦 Core Agentic Prompt Design Elements

- **Role**: Define who the agent is ("You are a scheduling assistant...")
- **Goal**: Specify what to achieve ("Book a 30-minute call...")
- **Context**: Provide input, memory, or history
- **Tools**: Mention available APIs or functions
- **Format**: Clarify expected output (JSON, bullet points, etc.)
- **Constraints**: Define safety or logic boundaries

📌 Key Takeaways

- Agentic AI systems are **active participants** in workflows
- Prompt engineering defines how these agents think, act, and decide
- Prompts become **modular**, **multi-step**, and **contextual**
- This lecture lays the foundation for building reliable, tool-using, reflective agents

---

📝 Quick Quiz (10 Q&A)

1. **Q:** What does "agentic AI" mean?
   A: AI that can plan, reason, and take actions autonomously

2. **Q:** What is one key feature of an agent that standard LLMs lack?
   A: The ability to invoke tools or external functions

3. **Q:** How does prompt engineering enable tool use?
   A: By embedding the tool syntax or instructions directly in the prompt

4. **Q:** Give an example of a goal for an AI agent.
   A: "Schedule a meeting and notify all participants"

5. **Q:** What element defines the persona of an agent?
   A: The **role** (e.g., “You are a travel planner.”)

6. **Q:** Why is context important in agent prompts?
   A: It helps the agent make decisions based on history or memory

7. **Q:** What does a constraint in a prompt do?
   A: Limits or controls the agent's behavior (e.g., "Don't send emails to unverified contacts.")

8. **Q:** What's the difference between a passive and an agentic prompt?
   A: Agentic prompts involve planning, acting, and reflecting; passive ones do not

9. **Q:** What is the first thing an agentic AI must do after receiving a prompt?
   A: Interpret the user's goal

10. **Q:** How is prompt engineering like managing a junior employee?
    A: You must give clear, actionable, step-by-step instructions

---

🟫 Lecture 2: Designing Prompts for Autonomous Decision-Making

🎯 Overview

In agentic AI, the model must **make decisions** on its own—whether that's choosing a tool, selecting a response path, or planning multiple steps. Prompt engineering guides this process by framing the environment, constraints, and reasoning patterns the agent should follow.

🤖 What is Autonomous Decision-Making?

Agentic AI systems are expected to:

- **Interpret** the user's objective
- **Choose** the best next action
- **Execute** the action (e.g., call a tool, return a response)
- **Reflect or revise** based on the results

To support these actions, prompt engineering must guide:

- Goal interpretation
- Action planning
- Prioritization and conditional logic
- Fallback or retry behavior

🔑 Prompt Design Principles for Decision-Making

1. Explicit Role and Task

"You are an assistant that can search the web and summarize news."

2. Structured Decision Chains

"First, determine if the question needs external data. If yes, call the web search tool. If not, answer directly."

3. Conditional Instructions

"If the user mentions weather, call the weather API. Else, ask a clarifying question."

4. Reasoning Aids

"Before answering, think aloud and explain the steps you are taking."

🔁 Prompt Pattern: Decide-Then-Act

Pattern:
You are an assistant agent. Given the user query:

1. Decide what the user's intent is.
2. Decide if a tool is needed (Yes/No).
3. If yes, specify the tool and its input.
4. Return a structured response with your reasoning.

This pattern simulates **agentic thinking** and helps models reason before acting.

🧠 Prompt Example for Decision Flow

Prompt:
You are an AI travel planner. When the user asks for a destination:

- Decide if they mentioned weather or budget.
- If yes, call the appropriate API.
- If not, return 3 general suggestions with explanations.

This prompt ensures structured reasoning and conditional tool use—hallmarks of decision-making agents.

🧰 Tools That Support Decision Prompts

Tool or Technique	Purpose
Decision templates	Provide predictable structure for reasoning
If/then branching	Enable conditional logic in prompt flow
Step-by-step reasoning	Support decomposing problems into actions
JSON output formatting	Machine-readable outputs for chaining

🌍 Real-Life Analogy

Prompting agentic AI for decisions is like giving a junior analyst a flowchart.
You’re telling them: “If this happens, do that. If not, try something else.”

📌 Key Takeaways

- Prompt engineering enables autonomous decision-making by providing structure
- Effective prompts include roles, tasks, conditions, and reasoning steps
- Decision-first prompts help agents plan actions before execution
- Using logic trees in prompts makes behavior more reliable and testable

📝 Quick Quiz (10 Q&A)

1. Q: What does autonomous decision-making mean for agents?
A: Making action choices independently based on the goal.

2. Q: What should prompts include to support decision-making?
A: Roles, goals, conditions, and reasoning structure.

3. Q: Give an example of a conditional prompt.
A: "If the user asks about news, search the web. Else, reply directly."

4. Q: What's a benefit of step-by-step planning in prompts?
A: Reduces errors and increases reasoning quality.

5. Q: True or False: Agents should always act immediately.
A: False – they should first decide what action is needed.

6. Q: What is the "decide-then-act" prompt pattern?
A: A format that helps agents decide before executing a task.

7. Q: Why use structured JSON responses?
A: They're easier for tools or follow-up agents to process.

8. Q: What do reasoning prompts encourage?
A: Transparency and explainability in agent actions.

9. Q: How do prompt conditions improve agent intelligence?
A: They enable context-aware branching behavior.

10. Q: Real-life analogy for prompting agent decisions?
A: Writing rules for a digital assistant to follow logically.

🛠️ Lecture 3: Tool Invocation and Function Calling in Prompts

🎯 Overview

One of the core features of agentic AI is its ability to **interact with external tools** like APIs, functions, or code modules. This lecture focuses on how to design prompts that **initiate, structure, and control tool use** reliably.

🛠️ What is Tool Invocation?

Tool invocation means guiding an LLM to call a specific tool or function based on context. It requires:

- Tool selection
- Input formatting
- Output handling

Examples include:

- Calling a weather API
- Running a calculator
- Searching a document index

🔑 Prompt Structure for Tool Use

A prompt designed for tool invocation must include:

1. **Tool Description:** What the tool does
2. **When to Use It:** Trigger conditions
3. **Input Format:** Expected syntax or parameters
4. **Output Handling:** How to report or act on results

🤖 Prompt Example: Weather API

Prompt:
"You have access to a weather API that takes a city name and returns the current temperature.
If the user asks about weather, call the API using this format:
`weather(city='CityName')`.
Return a response like:
'The temperature in CityName is X degrees.'"

⚙️ Prompt Template for Generic Tool Use

Prompt Template:
You are an agent. You have access to the following tools:

- `tool_name_1`: [description]
- `tool_name_2`: [description]

When answering user questions:

- Identify if a tool is needed
- Use correct syntax to call the tool
- Integrate tool output into your answer

Function Calling Format (OpenAI-style)

Many agent frameworks use a structured **function schema**, like:

{ "function": "get_weather", "parameters": { "city": "London" } }

Prompts can guide models to output JSON that downstream systems can parse.

Best Practices for Tool Prompts

Prompting Practice	Benefit
Explicit tool conditions	Reduces false calls
Input/output templates	Ensures consistent results
Add fallback logic	Supports retries or explanations
Provide tool examples	Improves accuracy and reliability

Real-Life Analogy

Prompting an agent to use a tool is like teaching a trainee to use a machine.
You don't just say "get the weather."
You say:

1. What the tool is
2. When to use it
3. What inputs it needs
4. What to do with the result

Key Takeaways

- Agents can call tools using carefully crafted prompts
- Prompt design must include tool context, syntax, and example outputs
- JSON or structured outputs make integration easier
- Tool prompts turn language models into **actionable systems**

Quick Quiz (10 Q&A)

1. Q: What is tool invocation?
A: Instructing an AI to call external tools like APIs or functions.

2. Q: Why define input format in a prompt?
A: To ensure the AI gives valid parameters to the tool.

3. Q: Give an example tool an agent might use.
A: A weather API or calculator.

4. Q: What does a tool-enabled prompt need?
A: Tool description, input format, and usage conditions.

5. Q: What is function calling format useful for?
A: Structuring responses that a system can execute automatically.

6. Q: Why include fallback logic in prompts?
A: To handle errors or unavailable tools gracefully.

7. Q: How does tool prompting differ from standard Q&A?
A: It requires structured, action-ready output.

8. Q: What's a good way to describe tool use in a prompt?
A: With examples and clear instructions.

9. Q: How can prompts integrate tool output?
A: By formatting a follow-up sentence using the tool's result.

10. Q: Real-life analogy for tool invocation?
A: Teaching someone how to operate a vending machine step-by-step.

Lecture 4: Prompt Chaining for Multi-Step Agents

Overview

Agentic AI often needs to complete tasks that span multiple reasoning steps or actions. This lecture explores how to design prompts for **multi-step planning and execution**, known as **prompt chaining**.

What is Prompt Chaining?

Prompt chaining involves breaking down a task into **smaller sub-tasks**, where:

- Each sub-task is handled by a separate prompt
- The output of one becomes the input of the next
- The agent follows a logical flow of execution

Why Use Chaining?

Challenge	Solution Through Chaining
Complex tasks	Break into simpler subtasks
Tool interaction + reasoning	Separate reasoning and tool usage
Sequential logic	Maintain state and flow through steps

Example: Job Application Agent

Task: Help a user apply for a job

1. **Prompt 1:** "Summarize the job description."
2. **Prompt 2:** "Extract required qualifications from the summary."
3. **Prompt 3:** "Check if the user meets those qualifications."
4. **Prompt 4:** "Draft a personalized cover letter."

Each step builds on the previous, allowing modular and verifiable reasoning.

Modular Prompting Format

Each module should include:

- **Clear goal**
- **Defined input format**
- **Expected output type**
- **Fallback or error handling** (optional)

Example Module Prompt:
"You are a resume expert. Given the following summary, extract all bullet-point qualifications."

Types of Prompt Chains

Type	Description
Linear	Simple step-by-step chain
Branching	Conditional steps based on earlier decisions
Looping	Repeat until a condition is met
Agent-directed	Model decides the next step (more autonomous)

Best Practices

- **Use intermediate memory** if chaining over long conversations
- **Log and verify** each output before passing it to the next prompt
- **Keep prompts focused** – one goal per prompt
- **Define formats** to keep structure consistent across steps

---

Real-Life Analogy

Prompt chaining is like a production line:

- Each station (prompt) does one job
- Output is passed to the next station
- Mistakes can be caught between stages

📌 Key Takeaways

- Prompt chaining decomposes complex tasks for better agent control
- Chains can be linear, branching, or agent-controlled
- Each prompt in the chain should have a specific, testable outcome
- This technique supports scalable and explainable agent workflows

---

📝 Quick Quiz (10 Q&A)

1. Q: What is prompt chaining?
A: Splitting a complex task into multiple prompts where each handles one step.

2. Q: Why is chaining useful in agentic AI?
A: It enables structured, multi-step reasoning and tool use.

3. Q: Give an example of a chained task.
A: Summarizing a job post → extracting skills → generating a cover letter.

4. Q: What is a linear chain?
A: A straight sequence of prompts executed step-by-step.

5. Q: What's a branching chain?
A: A chain with conditional logic that changes path based on outputs.

6. Q: What does modularity mean in this context?
A: Each prompt module handles one part of the task with clear input/output.

7. Q: When should you use memory in chaining?
A: When context needs to persist across multiple steps.

8. Q: How do you ensure chaining is robust?
A: Use structured formats and validate each step's output.

9. Q: What's a looping chain used for?
A: Repeating a step until a goal is achieved (e.g., search until found).

10. Q: Real-life analogy for prompt chaining?
A: A factory line where each machine (prompt) completes one operation.

🧯 Lecture 5: Error Handling and Refactor Prompts in Agents

🎯 Overview

In real-world use, agentic AI may produce incorrect, incomplete, or irrelevant outputs. This lecture covers how to build **error handling** and **self-correction** into prompts, making agents more robust and self-aware.

❗ Why Error Handling is Essential

Agents must gracefully handle:

- Incorrect tool calls
- Missing context or inputs
- Unexpected outputs
- User misunderstandings

Prompt engineering can help models **detect**, **report**, and **recover** from such issues.

🔁 Common Agent Errors and Fixes

Error Type	Example Scenario	Prompt Solution
Missing input	User asks to schedule but no time provided	Ask clarifying question
Tool failure	API call returns error or null	Retry or suggest alternative
Incomplete reasoning	Agent skips a step	Include checklist or plan structure
Contradictory behavior	Agent gives conflicting answers	Add consistency checks in prompt

---

🧠 Prompt Pattern: Verify-Then-Act

Prompt structure example:

1. Interpret the task
2. Validate if all inputs are available
3. If not, ask the user for missing info
4. Proceed only when all conditions are met

Example Prompt:
“You are a meeting assistant. If time, date, or attendees are missing, ask for them before booking.”

🔄 Refactoring Prompts for Reliability

If a prompt is producing inconsistent results:

- Break it into smaller sub-prompts
- Use clearer role and context statements
- Add example inputs and outputs
- Include explicit validation logic

Before:
“Book a meeting for tomorrow.”

After:
“You are a scheduling agent. First check if the user has provided time, date, and participants. If any are

missing, request them. Then confirm the booking."

Techniques for Error-Resilient Prompts

- Add "checklist" instructions
- Use conditionals like "if X not provided, then ask"
- Allow the agent to output a warning message
- Log and explain internal errors to user

Real-Life Analogy

Think of an AI agent like a junior assistant. If you say "book a call," they should ask:

- "When?"
- "With whom?"
- "Where?"

Instead of making assumptions, they ask to clarify—just like a well-designed agent prompt should.

Key Takeaways

- Prompts should anticipate missing or bad inputs
- Add fallback logic to ask, retry, or clarify
- Refactor prompts that don't perform reliably
- Error-resilient agents are more trustworthy and usable

Quick Quiz (10 Q&A)

1. Q: Why is error handling important in agentic AI?
A: To ensure robustness and avoid silent failures.

2. Q: What can a model do if input is missing?
A: Ask clarifying questions or defer the task.

3. Q: What is the verify-then-act pattern?
A: A structure that validates inputs before taking action.

4. Q: What's one fix for inconsistent agent behavior?
A: Add validation checks or refactor the prompt.

5. Q: Give an example of a refactored prompt.
A: From "book a meeting" to "check time/date/people first, then book."

6. Q: What prompt element can reduce skipped steps?
A: A checklist or plan structure.

7. Q: How can agents deal with tool failure?
A: Retry, suggest alternatives, or show error.

8. Q: What's the risk of skipping validation?
A: Producing wrong or incomplete actions.

9. Q: Why include example inputs in a prompt?
A: To guide the model toward correct formatting.

10. Q: Real-life analogy for error handling?
A: Like a junior staff member asking questions instead of guessing.

Lecture 6: Memory, Context, and State Management in Agentic AI

Overview

Effective agentic systems require **memory and state awareness**. This lecture focuses on how to use prompts to manage and simulate memory, pass context across interactions, and track task state reliably.

Why Memory Matters

In real-world tasks, agents need to:

- Remember past interactions
- Maintain user preferences
- Track task progress
- Refer back to earlier outputs

Prompt design plays a central role in **simulating** or **maintaining** memory.

Types of Memory in Agents

Type	Description
Short-term memory	Context window (previous prompt and response)
Long-term memory	Saved user data (name, goals, preferences)
Episodic memory	Sequence of actions or events in a session
Tool-state memory	Remembering last-used tools or settings

Contextual Prompting Techniques

1. **Context Window Inclusion**
   Include key previous messages in the current prompt.

2. **State Summarization**
   Add a summary of past steps or task goals before prompting.

3. **Memory Embedding**
   Use persistent storage or vector databases to fetch and insert relevant history.

🤔 Prompt Example: Context Carry-Forward

Prompt:
"You are helping Alex plan a trip. Previously, Alex selected Paris and a budget of $2,000. Continue suggesting affordable hotels and local attractions."

This ensures the agent maintains coherence across multiple turns.

🔁 State Tracking Patterns

- **Task Progress**
  Use checklists in prompts to mark completed items.
- **Input-Output Memory**
  Store earlier outputs and reuse them as input for future prompts.
- **Conditional Recall**
  "If the user has chosen a destination, suggest activities. Otherwise, ask."

🔍 Best Practices for Memory-Aware Prompts

- Keep context **compact but relevant**
- Summarize history instead of repeating full logs
- Use **labels** or **formatting** to separate memory from user input
- Clearly define what the agent knows or remembers

🌍 Real-Life Analogy

Imagine a travel agent who remembers your preferences from last week's call:
That's memory. Without it, every session would start from scratch.

📌 Key Takeaways

- Memory helps agents maintain coherence and relevance
- Prompt design must include relevant context or memory structures
- Use summaries, conditionals, and formatting to carry state
- State-aware agents are more personalized and effective

📝 Quick Quiz (10 Q&A)

1. Q: What is the role of memory in agentic AI?
A: To track user context, preferences, and task progress.

2. Q: What are the two types of memory most used in prompts?
A: Short-term and long-term memory.

3. Q: What is a context window?
A: Recent messages or turns used to maintain conversation flow.

4. Q: How can you simulate memory in a prompt?
A: By including summaries or previous outputs directly in the prompt.

5. Q: Give an example of stateful prompting.
A: "You previously suggested Paris. Now recommend nearby hotels."

6. Q: What's a good practice for long conversations?
A: Use brief summaries instead of full history to save space.

7. Q: Why is conditional logic helpful with memory?
A: It adapts behavior based on what's known vs. unknown.

8. Q: How can agents retrieve long-term memory?
A: Through embeddings or vector store lookups.

9. Q: What's a risk of poor memory use?
A: Redundant or incoherent responses.

10. Q: Real-world analogy of agent memory?
A: A personal assistant who remembers your previous meetings and choices.

📓 Lecture 7: Controlling Output Style, Format, and Tone

🎯 Overview

Agentic AI must generate outputs that are not only correct, but also **well-formatted, styled appropriately, and aligned with the intended tone**. This lecture explores prompt techniques that shape how agents communicate, not just what they say.

🧠 Why Output Control Matters

- Professional settings require formal tone
- APIs may require structured JSON
- Emails may need polite, persuasive wording
- Reports need bullet points, summaries, or formatting

Prompting helps enforce this consistently.

💼 Output Control Parameters

Output Feature	Controlled By Prompt?	Example
Format	☑ Yes	"Return result as a bullet list"
Style	☑ Yes	"Write in an academic tone"
Length	☑ Yes	"Keep answer under 100 words"
Structure	☑ Yes	"Respond in JSON format with keys: ..."

Prompt Templates for Style and Format

Formal Prompt Example:
"You are an executive assistant. Draft a formal email apologizing for the meeting reschedule."

JSON Output Prompt:
"Respond with JSON in the format: { 'destination': '', 'temperature': '', 'activities': [] }"

Bullet Point Prompt:
"List the pros and cons of remote work in bullet point format."

Friendly Tone Prompt:
"Use a casual and cheerful tone. Say 'Hi!' and thank the user warmly."

Post-Processing vs. Prompt Engineering

Method	When to Use
Prompt Engineering	When tone/format must come directly from the model
Post-Processing	When structure or validation is needed afterward

Prompting is best when style needs to be **visible to users immediately**.

Common Mistakes to Avoid

- Forgetting to specify format or tone leads to generic responses
- Overloading the prompt with too many conflicting style instructions
- Using vague terms like “make it good” instead of “make it persuasive”

Real-Life Analogy

Imagine telling a coworker, “Write this like a marketing email, but formal, concise, and enthusiastic.”
That's exactly what you’re asking the AI agent to do in a style prompt.

Key Takeaways

- Control tone, format, and style directly through the prompt

- Use specific phrases to guide tone (e.g. “formal”, “cheerful”, “bullet points”)
- JSON and markdown are common output formats for agents
- Clear prompts lead to consistent and user-friendly outputs

Quick Quiz (10 Q&A)

1. Q: Why is output style important?
A: It affects professionalism, usability, and communication clarity.

2. Q: What prompt phrase ensures bullet formatting?
A: “List as bullet points.”

3. Q: How do you enforce JSON output in a prompt?
A: “Respond in JSON format with keys: ...”

4. Q: What does tone refer to in outputs?
A: The attitude or voice of the response (e.g., formal, casual).

5. Q: When should you choose prompt control over post-processing?
A: When the response must be styled before it’s delivered.

6. Q: Give an example of a vague style instruction.
A: “Make it sound good.”

7. Q: What’s a better version of the vague prompt above?
A: “Make it persuasive and professional.”

8. Q: How can you set length constraints?
A: “Limit to 100 words.”

9. Q: Why is tone alignment important for agents?
A: It ensures the output matches the situation or audience.

10. Q: Real-life analogy for output control?
A: Giving a colleague specific instructions on how to write a message.

Lecture 8: Domain-Specific Prompt Engineering

Overview

Different domains—like law, medicine, coding, education, or customer service—have unique terminology, formats, and expectations. In this lecture, you'll learn how to tailor prompts to specific domains for **more relevant, accurate, and usable outputs**.

Why Customize Prompts by Domain?

- **Terminology**: Legal or medical language is specialized
- **Audience Expectations**: Academic tone vs. casual explanation

- **Data Types**: JSON for APIs, citations for academic content
- **Regulatory Requirements**: Compliance in finance, healthcare, etc.

Prompt Components That Should Be Specialized

Prompt Component	Domain Impact Example
Role Definition	"You are a legal assistant..."
Output Format	"Summarize case facts in bullet points."
Context Clues	"Use ICD-10 codes when referencing diagnoses."
Tone Control	"Use respectful tone appropriate for a client brief."

Example: Medical Domain

Prompt:
"You are a medical diagnosis assistant. Given a list of symptoms, suggest possible conditions using medical terminology and include ICD-10 codes."

Response Example:
"Symptoms: fatigue, weight loss, night sweats
Possible diagnosis: Tuberculosis (ICD-10: A15)"

Example: Legal Domain

Prompt:
"You are a junior legal clerk. Summarize the case and highlight precedents cited. Output should follow legal memo format."

Response Example:

- Case Summary: [summary]
- Precedents: Roe v. Wade, Brown v. Board of Education
- Legal Memo: [formatted result]

Example: Educational Tutoring

Prompt:
"You are a math tutor. Explain how to factor quadratic equations step-by-step in simple language for a 10th-grade student."

Response Example:

1. Look for two numbers that multiply to give ac and add to b
2. Split the middle term
3. Factor by grouping

Tips for Effective Domain Prompting

- Be explicit about the domain role (e.g., “scientific assistant”, “financial planner”)
- Include formatting requirements (e.g., "use APA citations")
- Define the audience level (e.g., “explain to a high school student”)
- Use real examples from the domain to guide the response

Real-Life Analogy

Using a generic prompt in a specialized domain is like asking a generalist to write a technical manual—they might get the words right, but not the depth or structure.

Key Takeaways

- Domain-specific prompts result in higher-quality, audience-appropriate outputs
- Tailor role, tone, structure, and language to the domain
- Use real-world terminology and context clues
- Clarify output formats expected in the professional field

Quick Quiz (10 Q&A)

1. Q: Why are domain-specific prompts important?
A: To ensure relevance, accuracy, and proper tone for the field.

2. Q: What does “ICD-10” refer to in medical prompts?
A: A standardized code for medical diagnoses.

3. Q: Give an example of a role in the legal domain.
A: “You are a junior legal clerk.”

4. Q: How can prompts differ in tone by domain?
A: Medical = clinical, Legal = formal, Education = friendly.

5. Q: What is a key element of educational prompts?
A: Explaining concepts clearly for the learner’s level.

6. Q: What’s an output format used in law?
A: Legal memos or case brief summaries.

7. Q: Why should we define the audience level?
A: So the explanation is neither too advanced nor too basic.

8. Q: What prompt addition helps with regulatory domains?
A: Compliance rules or coding systems like ICD-10.

9. Q: What domain might use APA formatting?
A: Academic or research writing.

10. Q: Real-world analogy for domain-specific prompting?
A: Giving a chef exact instructions for a vegan dish, not just "make dinner."

📓 Lecture 9: Advanced Prompting Techniques for Agentic AI

🎯 Overview

Once basic prompting is mastered, more sophisticated techniques can unlock deeper reasoning, better tool use, and autonomous behaviors. This lecture covers **advanced strategies** like self-reflection, planning, and ensemble reasoning.

🧠 What Makes Prompting "Advanced"?

Advanced prompting often involves:

- Multi-stage reasoning
- Use of internal validation
- Redirection or reflection mechanisms
- Simulated cognitive processes like planning or debating

These go beyond direct Q&A into **meta-thinking** and **self-supervision**.

🔍 Key Advanced Techniques

1. Chain-of-Thought (CoT)

Encourages step-by-step reasoning.

Prompt:
"Think step by step before giving your final answer."

2. Tree-of-Thought (ToT)

Explores multiple reasoning paths before selecting the best one.

Prompt Structure:
"List three different approaches to solving this, then select the most effective."

3. Self-Consistency

Generates multiple answers, then picks the most frequent or best one.

Use Case:
Mathematical problem solving or ambiguous queries.

4. Reflexion

Agent reflects on whether its response was good, then retries.

Prompt Template:
"Was your last answer correct? If not, revise it and try again."

5. Tool-Augmented Planning

Plan tool calls in a sequence, not just reactively.

Prompt Template:
"Plan your next 3 steps. If one includes using a tool, describe when and why."

Combining Techniques for Agent Behavior

Technique	Benefit
CoT	Transparent, explainable reasoning
ToT	Creative exploration of multiple paths
Reflexion	Improved accuracy via self-revision
Self-Consistency	More robust outputs via majority vote
Tool Planning	Efficient multi-step problem solving

Example: Reflexion Agent

Prompt:
"You are a research assistant. After answering, evaluate your response. If you detect an error, rewrite it."

Initial Answer:
"The Eiffel Tower is in Berlin."

Self-check:
"This is incorrect. The Eiffel Tower is in Paris."

Revised Answer:
"The Eiffel Tower is located in Paris, France."

Real-Life Analogy

Advanced prompts are like giving a person **not just instructions**, but the **ability to pause, reflect, correct, and plan** before acting.

Key Takeaways

- Advanced prompting techniques allow LLMs to simulate higher-level cognition
- Chain-of-thought and self-reflection improve accuracy and reasoning

- Tool planning and self-consistency add robustness and reliability
- These approaches are vital for complex agentic workflows

Quick Quiz (10 Q&A)

1. Q: What is chain-of-thought prompting?
A: A technique that guides the model to think step-by-step.

2. Q: What does tree-of-thought simulate?
A: Exploring multiple solutions before deciding.

3. Q: Why use self-consistency?
A: To improve reliability by generating multiple outputs.

4. Q: What does a reflexion agent do?
A: It evaluates and revises its own responses.

5. Q: What is tool-augmented planning?
A: Planning tool usage ahead of time instead of reactively.

6. Q: When is chain-of-thought useful?
A: For problems requiring multi-step reasoning.

7. Q: What's the benefit of reflexion?
A: Self-correction improves final output accuracy.

8. Q: How does ToT differ from CoT?
A: ToT explores multiple branches; CoT is linear.

9. Q: Can these techniques be combined?
A: Yes, combining them increases agent intelligence.

10. Q: Real-life analogy for advanced prompting?
A: Teaching someone not just what to do, but how to think, plan, and review.

Lecture 10: Tools, Frameworks, and the Future of Prompt Engineering

Overview

The field of prompt engineering is evolving rapidly with new tools, libraries, and best practices. This final lecture explores the landscape of available tools and frameworks and offers a look at the **future of prompt engineering** for agentic AI systems.

Popular Prompt Engineering Tools

Tool/Library	Description

Tool/Library	Description
LangChain	Framework for chaining prompts and tool use
LlamaIndex	Connects LLMs with data sources via indexes
Guidance	DSL for structured prompt programming
PromptLayer	Logging and version control for prompts
OpenAI Functions	Enables calling structured tools via prompts

---

🧰 Framework Capabilities

Capability	Example Tool	Benefit
Chaining	LangChain	Manage multi-step agent workflows
Memory Integration	LangChain, Haystack	Persist context and history
Tool Usage	OpenAI, LangChain	Structured tool calling
Prompt Debugging	PromptLayer	Track and improve prompt results

---

🚀 The Future of Prompt Engineering

1. Programmatic Prompting

Mixing code with prompt logic. Tools like Guidance and LMQL enable this.

2. Auto-prompting

Models that generate their own optimized prompts based on user intent.

3. Standardization

Emergence of prompt schemas and governance (e.g., PromptML, ISO-like formats).

4. Integrated Tool Ecosystems

More agents using plugins, APIs, and third-party tools in a composable way.

5. Multimodal Prompting

Expanding prompts to handle image, audio, and structured inputs, not just text.

🔮 Research Directions

- Prompt tuning vs. fine-tuning
- Learning from demonstrations and feedback
- Trustworthy prompting for critical use cases (e.g. medicine, law)
- Reducing hallucinations with context-aware prompting

🌍 Real-Life Analogy

Prompt engineering today is like web development in the early 2000s—powerful, growing fast, and starting to standardize. It's where creativity meets engineering rigor.

📌 Key Takeaways

- Prompt engineering tools enhance agent capabilities at scale
- Frameworks like LangChain help structure complex prompt workflows
- The future includes auto-prompting, multimodal agents, and standard formats
- Staying up-to-date with new tools is key for building powerful AI agents

📝 Quick Quiz (10 Q&A)

1. Q: What is LangChain used for?
A: Managing prompt chains, tools, and memory for agents.

2. Q: What does PromptLayer help with?
A: Logging and analyzing prompt performance.

3. Q: Name a tool that connects LLMs with external data.
A: LlamaIndex.

4. Q: What is auto-prompting?
A: Letting the model generate its own optimized prompts.

5. Q: How does programmatic prompting work?
A: Combining code with structured prompt logic.

6. Q: What is multimodal prompting?
A: Using inputs like text, image, or audio in prompts.

7. Q: What is a challenge addressed by prompt standardization?
A: Making prompts more consistent, interpretable, and sharable.

8. Q: What does OpenAI Functions allow?
A: Calling tools and APIs from within a prompt.

9. Q: Why is trust important in prompting?
A: Critical domains need accurate, consistent outputs.

10. Q: Real-life analogy for prompt engineering today?
A: Like early web development—rapid, creative, and growing fast.