i am giving a presentation on topic agentic AI using langchain. can you please help me with giving transcrive/speech to give a presentation.

the contents are:

What is Agent?

* The core idea of agents is to use a language model to choose a sequence of actions to take. In chains, a sequence of actions is hardcoded (in code). In agents, a language model is used as a reasoning engine to determine which actions to take and in which order.
* And as seen below, the agent creates a chain in real-time, reflects on the question, and goes through a process of action, observation, thought, action, observation, thought...until the final answer is reached.

Action Observation Thought Final Answer

* LangChain library functions parse the LLM's output assuming that it will use certain keywords.
* Example here uses Thought, Action, Observation as keywords for Chain-of-Thought Reasoning. (ReAct)
* In LangChain, an "Agent" is an Al entity that interacts with various

"Tools" to perform tasks or answer queries. Tools are essentially functions that extend the agent's capabilities by allowing it to perform specific actions, like retrieving the current time or accessing an external database.

Agents let the model use tools in a loop, so that it can decide how many times to use tools.

An agent with a Tool

* Can make decisions; act as reasoning engines.
* Interact with Tools.
* Autonomous, doesn't need constant human input.

As a consequence, it is able to autocorrect itself when any errors appear (if trained well).

Create prompt and tool for the agent

prompt = ChatPromptTemplate.from\_messages ( "system",

111111

You are a research assistant that will help generate a research paper. Answer the user query and use necessary tools.

Wrap the output in this format and provide no other text\n(format\_instructions}

,

("placeholder", "(chat\_history)"), ("human", "(query)"), ("placeholder", "(agentscratchpad)",

1) partial(format\_instructions=parser.get\_format\_instructions)) tools = [search\_tool, wiki\_tool]

agent = create\_tool\_calling\_agent lIm=Ilm,

prompt=prompt, tools=tools)

Run & Test the Agent

question = "Tell me about Action Park?"

agent\_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)

raw\_response = agent\_executor.invoke(|" query": question}) structured\_response = parser.parse(raw\_response.get|" output")[O]["text"]) print(structured\_response)

Types of Agents or Agentic Patterns

* Tool Calling
* Structure Output
* Human in Loop
* Map-Reduce
* MultiAgents
* Planning
* Reflection(Reflex Agents)
* ReAct Agent(Learning Agents)
* Hierarchical Agents

Benefits of Al Agents

* Enhanced Efficiency and Productivity
* Improved Decision-Making
* 24/7 Availability and Scalability
* Personalized Customer Experiences
* Cost Reduction and Increased Revenue
* Innovation and New Opportunities

Product Examples of Artificial Intelligent Agents

1. Intelligent Personal Assistants
2. Chatbots
3. Autonomous Robots
4. Game Playing Agents
5. Fraud Detection Agents

Transcript for Presentation on Agentic AI Using LangChain What is Agent?

An agent is an AI system that uses a language model to decide a sequence of actions it should take to achieve a goal.

In traditional AI chains, these actions are hardcoded —

. But in agents, the language model itself becomes a reasoning engine, which actions to take, in which order, and when to stop.

Here’s how it works in practice: the agent creates a chain in real time. It reflects on the question, performs an action, observes the result, thinks about what that result means, and repeats the cycle — Action → Observation → Thought — until it reaches the Final Answer.

Now, in LangChain, an Agent is an AI entity that can interact with different tools to perform tasks or answer questions.

Tools are basically functions that give the agent specific capabilities — like looking up information from Wikipedia, checking the current time, or querying.

Agents can use these tools in a loop,:

* They can make decisions like a reasoning engine.
* They can work autonomously without constant human guidance.
* And if they’re trained well, they can even self-correct when they encounter errors.

For example, when creating an agent in LangChain, we typically define a prompt and a set of tools. In this case, our prompt could tell the agent: “You are a research assistant who must answer queries using necessary tools, and wrap the output in a specific format.” We then provide those tools — maybe a search tool, a wiki tool — and connect them to the agent.

Once that’s ready, we test it. Suppose we ask it: “Tell me about Action Park?” The AgentExecutor runs the question through the agent, the tools are called as needed, and finally we get a structured, clean response parsed for output.

Now, there are different types of agents or agentic patterns in LangChain:

* Tool Calling Agents
* Structured Output Agents
* ReAct Agents (Learning Agents)
* Hierarchical Agents

So, why are AI agents valuable? The benefits are quite impressive:

* Enhanced efficiency and productivity — they can do repetitive work faster and more accurately.
* Better decision-making — thanks to reasoning with multiple inputs.
* 24/7 availability and scalability — perfect for customer service or monitoring tasks.
* Personalized customer experiences — they can tailor responses based on user data.
* Cost reduction and increased revenue through automation.
* They also open doors to innovation and new opportunities. We already see agents in action in many products:

1. Intelligent personal assistants like Siri or Alexa.
2. Customer support chatbots.
3. Autonomous robots in manufacturing.
4. Game-playing agents like AlphaZero.
5. Fraud detection agents used in banking.

To sum it all up — LangChain enables us to build AI agents that reason, plan, and act using tools, making them not only powerful but also adaptive to complex situations. This combination of reasoning engines with tool usage is paving the way for more autonomous, intelligent systems in the real world.

Thank you.

Would you like me to also make a shorter, more concise version of this speech so it’s under 3–4 minutes for quick presentations? That way you can choose between a full and short delivery.

Here’s a detailed explanation of each Agent Type or Agentic Pattern in LangChain and agentic AI systems. Tool Calling Agents

These are the most fundamental form of agents in LangChain. A Tool Calling Agent can decide which external tool to use, based on the input query and reasoning. Tools can be anything from a web search function, a calculator, or a database query API.

* + Core idea: The LLM uses reasoning to determine which tool to invoke and when.
  + Example: When asked “What’s the weather in New York right now?”, the agent decides to call the weather API tool, retrieves the data, and formats the output.
  + Use case: Useful in tasks where multiple specialized tools are integrated, and the model must dynamically choose between them.

Structured Output Agents

Structured Output Agents are designed to produce responses in a specific format or schema, rather than free-text. This ensures that the agent’s output can be programmatically parsed and used directly by other systems.

* + Core idea: Output follows a defined structure such as JSON, tables, or key-value pairs.
  + Example: An agent generating a summary report in JSON format so another application can easily interpret it.
  + Use case: Ideal for report generation, structured data extraction, and API-to-API communication.

Human-in-the-Loop Agents

These agents include human feedback at critical decision points. The agent does most of the reasoning and action autonomously, but humans intervene when significant judgment calls or high-risk outputs occur.

* + Core idea: The system leverages both AI automation and human supervision.
  + Example: A medical diagnosis agent that proposes a diagnosis but requires a doctor’s approval before finalizing the output.
  + Use case: Perfect for regulated or safety-critical domains like healthcare, finance, or legal reasoning.

Map-Reduce Agents

This pattern is inspired by parallel computing. The Map-Reduce Agent breaks down a large problem into smaller sub-tasks (Map phase), processes them simultaneously or sequentially, and then consolidates the results (Reduce phase) into a final answer.

* + Core idea: Divide and conquer for large data or complex problems.
  + Example: When summarizing multiple research papers, each paper is summarized individually (map), and then the summaries are combined into a unified abstract (reduce).
  + Use case: Large document analysis, distributed query answering, or scalable summarization. Multi-Agents (Collaborative Agents)

Multi-Agent systems consist of multiple AI agents interacting and collaborating to solve a common goal. Each agent has specialized expertise or roles.

* + Core idea: Agents communicate, exchange results, and coordinate actions—like a team of AI specialists.
  + Example: One agent handles data retrieval, another analyzes sentiment, and a third writes the final report.
  + Use case: Complex workflows such as research automation, multi-domain problem solving, or software development assistance.

Planning Agents

Planning Agents are capable of breaking down tasks into a sequence of steps or subtasks before execution. These agents plan, reason, predict dependencies, and execute actions accordingly.

* + Core idea: Task decomposition — the agent plans before acting.
  + Example: If the goal is to “build a website,” the agent first generates a plan: design layout → write HTML → add CSS → deploy site. It then executes these steps sequentially.
  + Use case: Project management automation, task schedulers, and workflow optimization systems.

Reflection or Reflex Agents

These agents possess a kind of self-monitoring or introspective ability. They can reflect on their previous performance, detect mistakes, and adjust future reasoning accordingly.

* + Core idea: Continuous learning and self-correction during or after execution.
  + Example: If a response is found to be inaccurate, the agent reviews the reasoning steps (thought, action, observation) and improves the next attempt.
  + Use case: Useful for adaptive learning systems or autonomous agents requiring high reliability.

ReAct Agents (Reason + Act Agents)

The ReAct (Reason + Act) pattern combines reasoning and acting steps in alternation. The agent reflects on its previous thought, takes an action, observes the outcome, and reasons again until it concludes the task.

* + Core idea: A dynamic cycle of “Thought → Action → Observation → Thought,” enabling chain-of-thought reasoning.
  + Example: When answering a complex query, the agent might first search the web, review results, adjust reasoning, and then produce the final answer.
  + Use case: General-purpose reasoning agents for search, question-answering, and decision-making.

Hierarchical Agents

Hierarchical Agents operate in a tiered structure, where a top-level (manager) agent assigns sub-tasks to lower-level agents and integrates their results.

* + Core idea: Divide responsibilities — high-level planning by a supervisor agent, and execution by worker agents.
  + Example: A project manager agent delegates “data gathering” to one agent and “report writing” to another, later synthesizing both outputs.
  + Use case: Complex multi-stage workflows, enterprise AI coordination, and large-scale multi-agent orchestration.

Would you like me to create a clear comparison table summarizing all these agent types — their key functions, advantages, and use cases — for your presentation slides?