CSCE 585: Machine Learning Systems

Lecture 3: Designing Agentic AI Systems

Pooyan Jamshidi

Fall 2024 | Course Website: https://pooyanjamshidi.github.io/mls/

UNIVERSITY OF South Carolina

What are agents?

- Agents can be defined as fully autonomous systems that operate **independently** over **extended periods**, using various tools to accomplish complex tasks.

Agents vs Workflow

- **Workflows** are systems where LLMs and tools are orchestrated through predefined code paths.
- **Agents**, on the other hand, are systems where LLMs dynamically direct their own processes and tool usage, maintaining control over how they accomplish tasks.

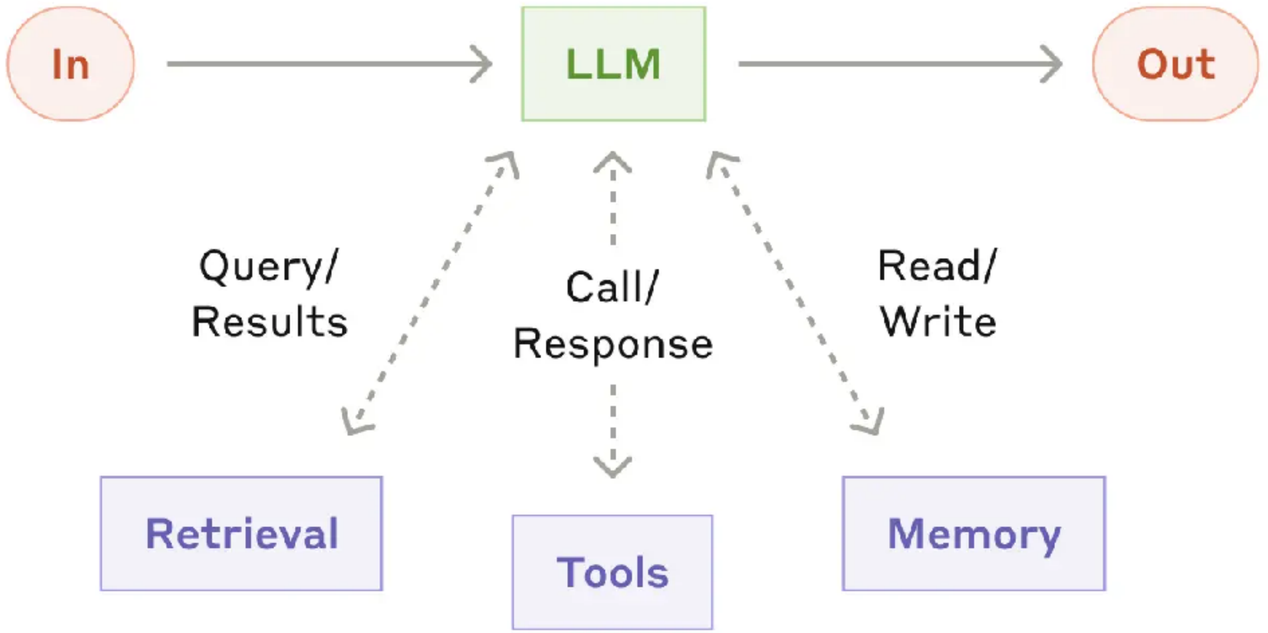
When (and when not) to use agents

- When building applications with LLMs, we recommend finding the **simplest solution** possible, and only **increasing complexity** when needed.
- Agentic systems often trade **latency** and **cost** for better **task performance**, and you should consider when this tradeoff makes sense.
- When more complexity is warranted, workflows offer **predictability** and **consistency** for well-defined tasks, whereas agents are the better option when **flexibility** and **model-driven decision-making** are needed at scale.
- For many applications, however, optimizing **single LLM calls** with retrieval and in-context examples is usually enough.

Design patterns for agentic systems

https://www.anthropic.com/engineering/building-effective-agents

Building block: The augmented LLM

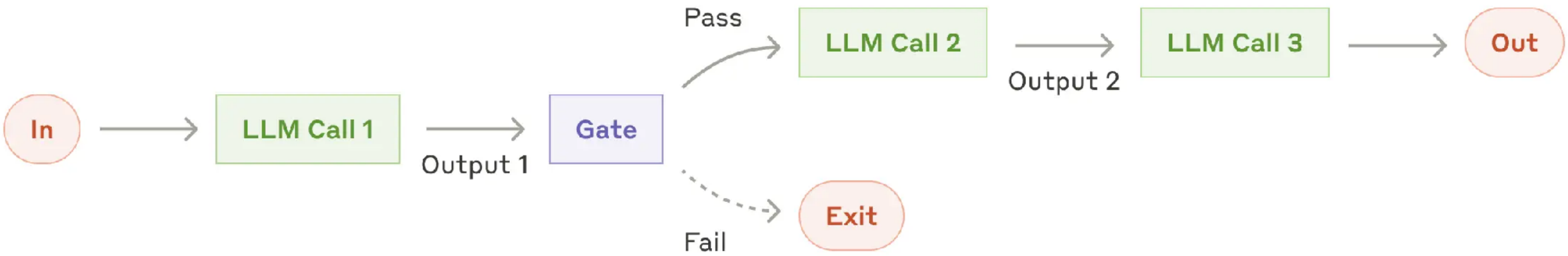


The basic building block of agentic systems is an LLM enhanced with augmentations such as retrieval, tools, and memory. LLMs can actively use these capabilities—generating their own search queries, selecting appropriate tools, and determining what information to retain.

How to implement these augmentations?

- **Messy way**: Direct function call by client
- **Clean way**: Model Context Protocol

Workflow: Prompt chaining

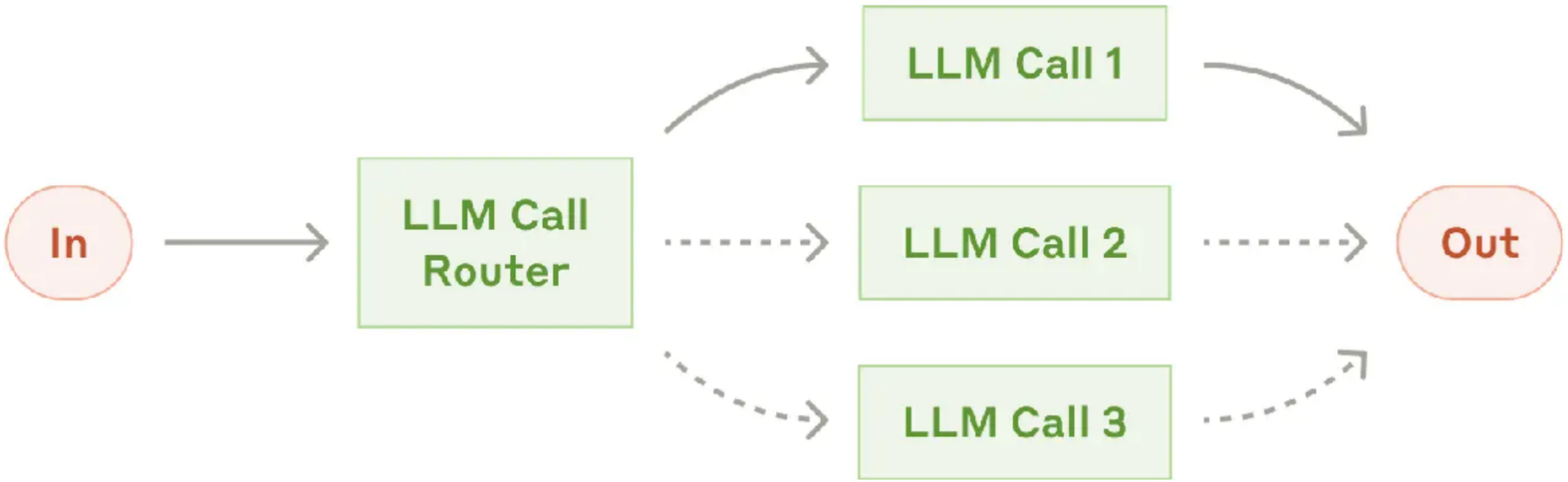


Prompt chaining decomposes a task into a sequence of steps, where each LLM call processes the output of the previous one.

When to use prompt chaining workflow

- This workflow is ideal for situations where the task can be easily and cleanly decomposed into fixed **subtasks**. The main goal is to trade off **latency** for higher **accuracy**, by making each LLM call an easier task.
- **Examples**:
  - Generating Marketing copy, then translating it into a different language.
  - Writing an outline of a document, checking that the outline meets certain criteria, then writing the document based on the outline.

Workflow: Routing

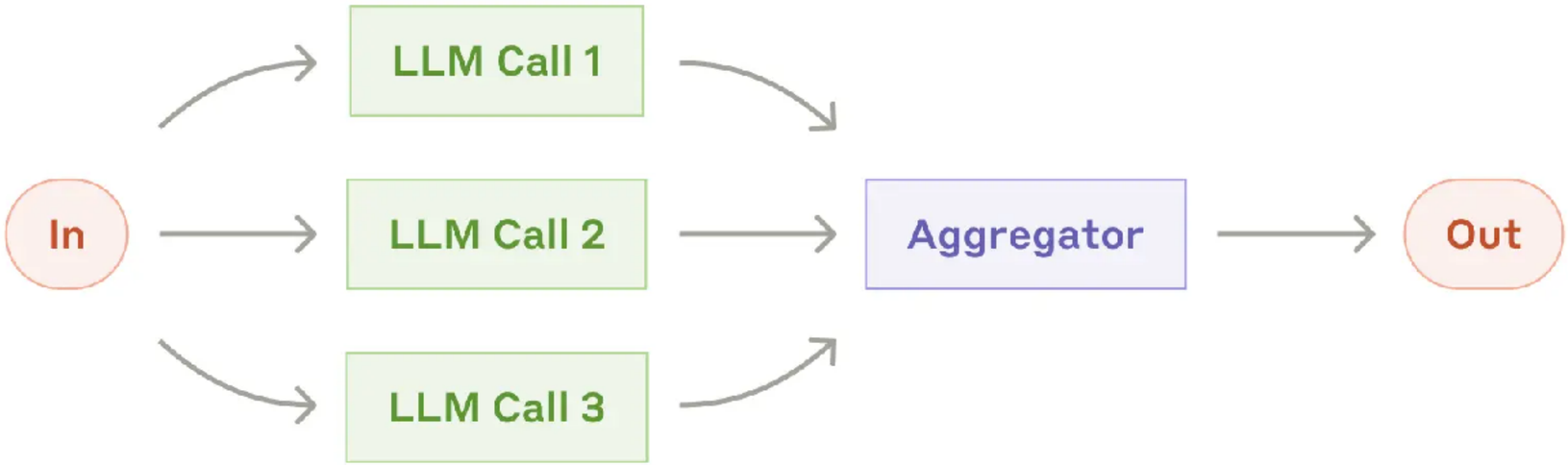


Routing classifies an input and directs it to a specialized followup task. This workflow allows for separation of concerns, and building more specialized prompts. Without this workflow, optimizing for one kind of input can hurt performance on other inputs.

When to use Routing workflow

- Routing works well for **complex tasks** where there are **distinct categories** that are better handled separately, and where classification can be handled accurately, either by an LLM or a more traditional classification model/algorithm.
- **Examples**:
  - Directing different types of customer service queries (general questions, refund requests, technical support) into different downstream processes, prompts, and tools.
  - Routing easy/common questions to smaller models like Claude 3.5 Haiku and hard/unusual questions to more capable models like Claude 3.5 Sonnet to optimize cost and speed.

Workflow: Parallelization



LLMs can sometimes work simultaneously on a task and have their outputs aggregated programmatically.

When to use this workflow

- Parallelization is effective when the divided subtasks can be parallelized for speed, or when multiple **perspectives** or **attempts** are needed for higher confidence results. For complex tasks with multiple considerations, LLMs generally perform better when each consideration is handled by a separate LLM call, allowing focused attention on each specific aspect.

Examples

- **Sectioning**:
  - Implementing guardrails where one model instance processes user queries while another screens them for inappropriate content or requests. This tends to perform better than having the same LLM call handle both guardrails and the core response.
  - Automating evals for evaluating LLM performance, where each LLM call evaluates a different aspect of the model's performance on a given prompt.
- **Voting**:
  - Reviewing a piece of code for vulnerabilities, where several different prompts review and flag the code if they find a problem.
  - Evaluating whether a given piece of content is inappropriate, with multiple prompts evaluating different aspects or requiring different vote thresholds to balance false positives and negatives.

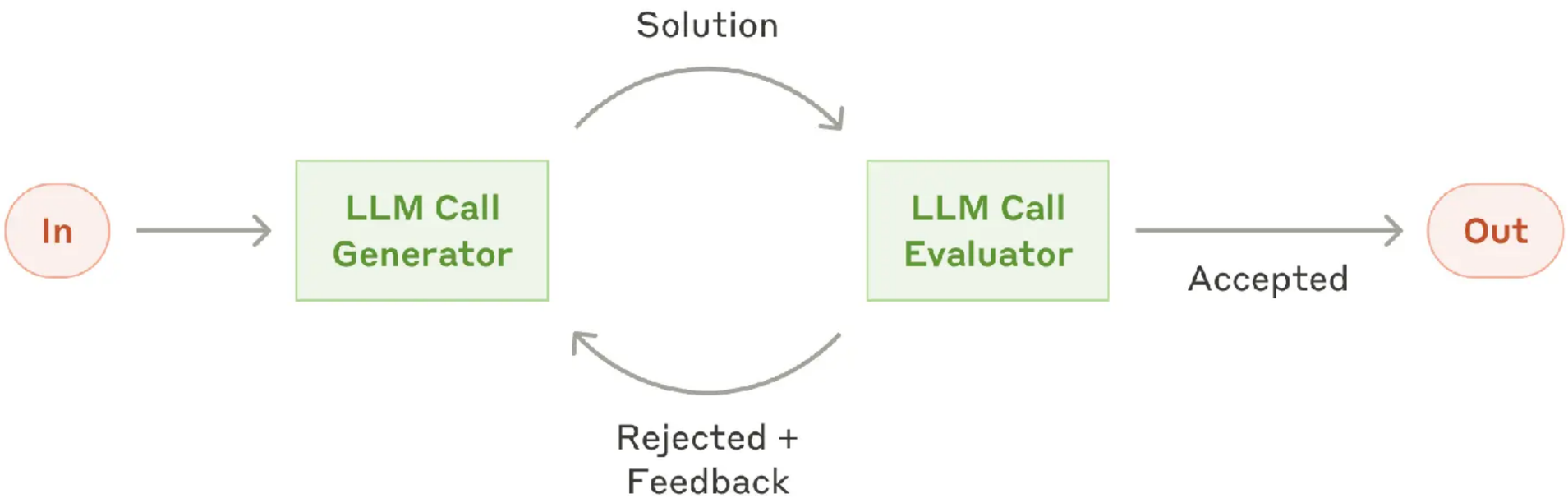
Workflow: Orchestrator-workers

In the orchestrator-workers workflow, a central LLM dynamically breaks down tasks, delegates them to worker LLMs, and synthesizes their results.

When to use this workflow

- This workflow is well-suited for complex tasks where you **can’t predict the subtasks** needed (in coding, for example, the number of files that need to be changed and the nature of the change in each file likely depend on the task). Whereas it’s topographically similar, the key difference from parallelization is its flexibility—**subtasks aren't pre-defined**, but determined by the orchestrator based on the specific input.
- **Examples**:
  - Coding products that make complex changes to multiple files each time.
  - Search tasks that involve gathering and analyzing information from multiple sources for possible relevant information.

Workflow: Evaluator-optimizer

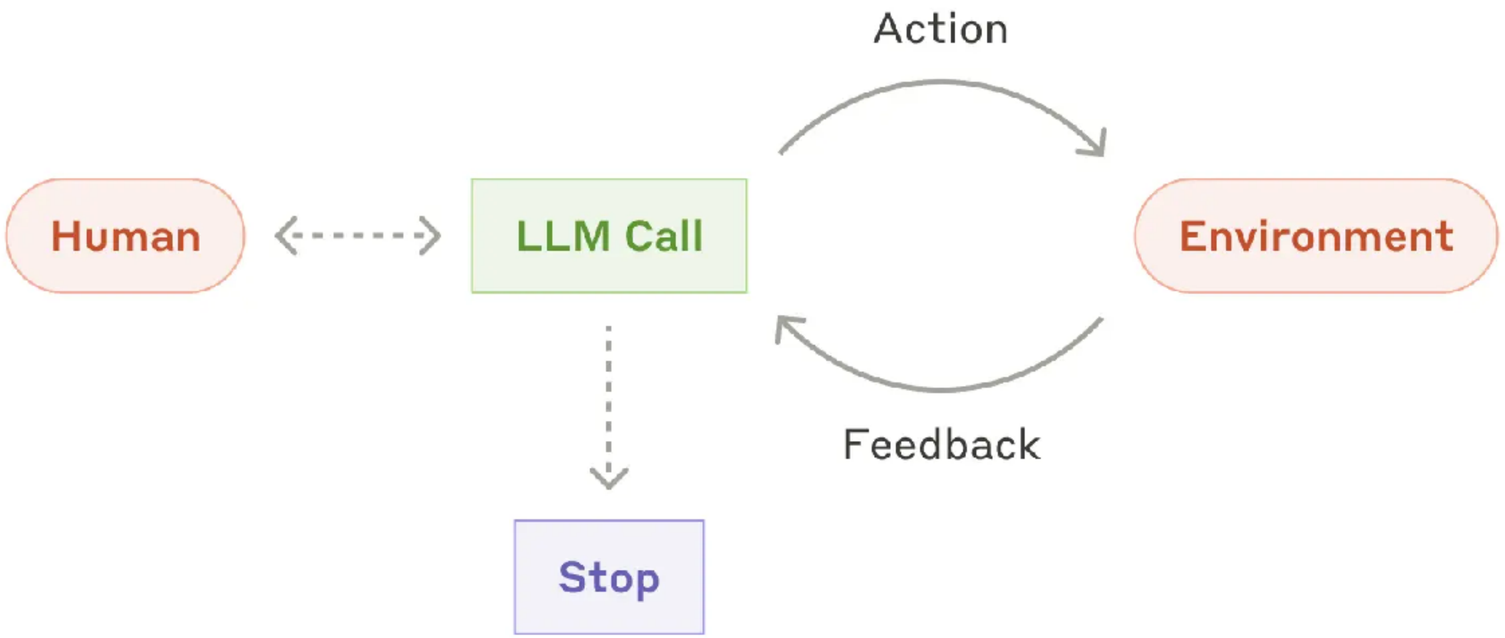


In the evaluator-optimizer workflow, one LLM call generates a response while another provides evaluation and feedback in a loop.

When to use this workflow

- This workflow is particularly effective when we have clear evaluation criteria, and when iterative refinement provides measurable value. The two signs of good fit are, first, that LLM responses can be demonstrably improved when a human articulates their feedback; and second, that the LLM can provide such feedback. This is analogous to the iterative writing process a human writer might go through when producing a polished document.
- Examples:
  - Literary translation where there are nuances that the translator LLM might not capture initially, but where an evaluator LLM can provide useful critiques.
  - Complex search tasks that require multiple rounds of searching and analysis to gather comprehensive information, where the evaluator decides whether further searches are warranted.

Agents



Agents begin their work with either a command from, or interactive discussion with, the human user. During execution, it's crucial for the agents to gain “ground truth” from the environment at each step (such as tool call results or code execution) to assess its progress. Agents can then pause for human feedback at checkpoints or when encountering blockers. The task often terminates upon completion, but it’s also common to include stopping conditions (such as a maximum number of iterations) to maintain control.

When to use Agents

- Agents can be used for open-ended problems where it’s difficult or impossible to predict the required number of steps, and where you can’t hardcode a fixed path. The LLM will potentially operate for many turns, and you must have some level of trust in its decision-making. Agents' autonomy makes them ideal for scaling tasks in trusted environments.

- The autonomous nature of agents means higher costs, and the potential for compounding errors. We recommend extensive testing in sandboxed environments, along with the appropriate guardrails.

High-level flow of a coding agent