Procesamiento de Lenguaje Natural Avanzado

Introduction to AI Agents

From Single Agents to Multi-Agent Systems

Dra. Helena Gómez Adorno
helena.gomez@iimas.unam.mx

Dr. Fazlourrahman Balouchzahi
fbalouc@iimas.unam.mx

Correo del curso:
pln.cienciadedatos@gmail.com

Where Do AI Agents Fit?

NLP / ML Foundations — Neural Networks — PLMs & LLMs — FSL · ZSL · RAG — AI Agents ← Now — Multi-Agent Systems

LLMs give agents a brain

Reasoning, planning, and language understanding — powered by the PLMs and LLMs you already know.

RAG gives agents memory

Retrieval-Augmented Generation becomes an agent's tool for accessing live, grounded knowledge.

Agents add action

Unlike a bare LLM, agents can perceive environments, call tools, plan across steps, and act autonomously.

What Is an AI Agent?

An AI agent is any system that perceives its environment and takes actions to achieve a goal.

PEAS Framework — Performance · Environment · Actuators · Sensors

P — Performance	E — Environment	A — Actuators	S — Sensors
How do we measure success? Accuracy · Speed · Cost · Safety User satisfaction	Where does the agent operate? Web · File system · Physical world Other agents · Databases	How does the agent act? API calls · Code execution Robot motors · Text output	How does the agent perceive? Text · Images · DB queries Tool outputs · User input

Example: A web-search agent — P: relevant results · E: internet · A: HTTP requests · S: query text + page content

Agent vs. Pure LLM — What's the Difference?

Pure LLM (e.g. ChatGPT)	AI Agent
✗ Receives a prompt → produces text	✓ Perceives environment continuously
✗ Stateless: no memory between calls	✓ Maintains state and memory over time
✗ Cannot take external actions	✓ Executes real actions (APIs, code, files)
✗ Does not observe results of outputs	✓ Observes feedback and adapts behaviour
✗ One-shot: input → output, done	✓ Iterative loop: think → act → observe
✗ No planning across multiple steps	✓ Plans and decomposes multi-step goals

VS

Think of an agent as an LLM wrapped in a perception-action loop with memory and tools.

The Perceive–Think–Act–Observe Loop

1 PERCEIVE

Receive input from environment
(text, image, sensor, API result, user message)

2 THINK

Reason using LLM or planner (Chain-of-Thought, ReAct, Tree-of-Thought)

3 ACT

Execute action in the world (call API, write code, send message, move)

4 OBSERVE

Receive feedback from environment
(result, error, new state, reward signal)

MEMORY

Stores context, past experiences
and external knowledge (RAG, vector DB)

GOAL / TASK

The objective guiding all decisions in the loop

Key Components Inside an AI Agent

🧠 Reasoning Engine (LLM)

The cognitive core. Interprets inputs, plans step-by-step (CoT / ReAct), makes decisions. This is the LLM you already know — now used as a planner.

💾 Memory

Short-term: conversation context window.
Long-term: vector DB, key-value store.
Episodic: past action traces for reflection.

🔧 Tools / Actions

Functions the agent can call: web search, code interpreter, REST APIs, file I/O, SQL queries. Tools extend beyond text generation.

📋 Planning

Decomposes a high-level goal into sub-tasks.
Strategies: CoT, ReAct, Tree-of-Thought, MCTS.
Full task graph or on-the-fly planning.

👁️ Perception

Processes raw inputs: text, images (vision), structured data, sensor streams. Multimodal agents combine all of these.

📊 Reflection & Evaluation

Self-critiques outputs, scores results against goal, re-plans on failure. Enables iterative improvement within a single task.

Agent Classification — A Complete Taxonomy

AI agents can be classified along two independent axes:

AXIS 1 — By Internal Architecture (How the agent decides what to do)

Simple Reflex
IF–THEN rules only

Model-Based
Internal world state

Goal-Based
Search & planning

Utility-Based
Maximise expected reward

Learning Agent
Improves from experience

AXIS 2 — By Action Scope (What the agent is designed to do)

Task Agent
Executes a specific defined task (code review, translation, summarisation)

Router / Orchestrator Agent
Classifies intent and delegates to the right specialist agent or tool

Conversational Agent
Maintains multi-turn dialogue, manages context and user state

Autonomous Agent
Pursues long-horizon goals with minimal human intervention

Five Canonical Types — Overview

Each type adds capability on top of the previous. Complexity increases left → right.

Simple Reflex	Model-Based Reflex	Goal-Based	Utility-Based	Learning Agent
IF cond → action	**Internal world state**	**Search & planning**	**Maximise reward**	**Improve over time**
No memory. Reacts to current percept only. Fast but brittle.	Tracks state. Handles partial observability.	Evaluates actions toward a goal. Multi-step capable.	Handles tradeoffs. Rational under uncertainty.	Updates its own policy from experience & feedback.

Increasing complexity & capability →

Simple Reflex & Model-Based Reflex Agents

TYPE 1 — Simple Reflex Agent

How:

IF <condition> THEN <action>. No memory of past states. Reacts to current percept only.

Pros:

Simple, fast, predictable. Easy to implement and debug. Low compute cost.

Cons:

Fails in partially observable environments. Cannot handle sequences.

Example:

Thermostat · Spam filter (rule-based) · Traffic light controller

LLM link:

A fixed prompt template mapping input type → output. Zero memory, one shot.

TYPE 2 — Model-Based Reflex Agent

How:

Maintains an internal world model. Updates model per percept, applies rules based on model state.

Pros:

Handles partial observability. Tracks changes over time. More robust than simple reflex.

Cons:

World model can become stale or incorrect. Still no goal-based reasoning.

Example:

Robot vacuum mapping a room · Dialogue manager tracking conversation state

LLM link:

LLM + conversation history or scratchpad tracking entity states across turns.

Goal-Based & Utility-Based Agents

TYPE 3 — Goal-Based Agent

How:

Evaluates possible actions by whether they lead toward a desired goal state. Uses search / planning.

Pros:

Handles multi-step tasks. Flexible: different goals → different behaviour.

Cons:

Planning can be expensive. Needs accurate world model. Goal conflicts not handled.

Example:

GPS navigation (goal = destination) · Chess agent (goal = checkmate)

LLM link:

ReAct framework: LLM checks each action against the task goal before executing.

TYPE 4 — Utility-Based Agent

How:

Assigns a utility score to each outcome. Chooses the action maximising expected utility — handles tradeoffs.

Pros:

Rational under uncertainty. Handles competing objectives (speed vs. safety).

Cons:

Misspecified utility = bad behaviour (reward hacking). Hard to define utility function.

Example:

Recommendation engine · Autonomous vehicle (speed/safety/comfort tradeoff)

LLM link:

LLM-as-judge: model scores candidate responses and picks the highest-utility one.

Learning Agent — The Most Capable Type

TYPE 5 — Learning Agent

Learning Element

Makes improvements by observing outcomes and adjusting the performance element. Learns from feedback.

Performance Element

Selects actions based on current knowledge. This is the agent's visible behaviour.

Critic

Evaluates how well the agent is doing vs. a performance standard. Provides the feedback signal.

Problem Generator

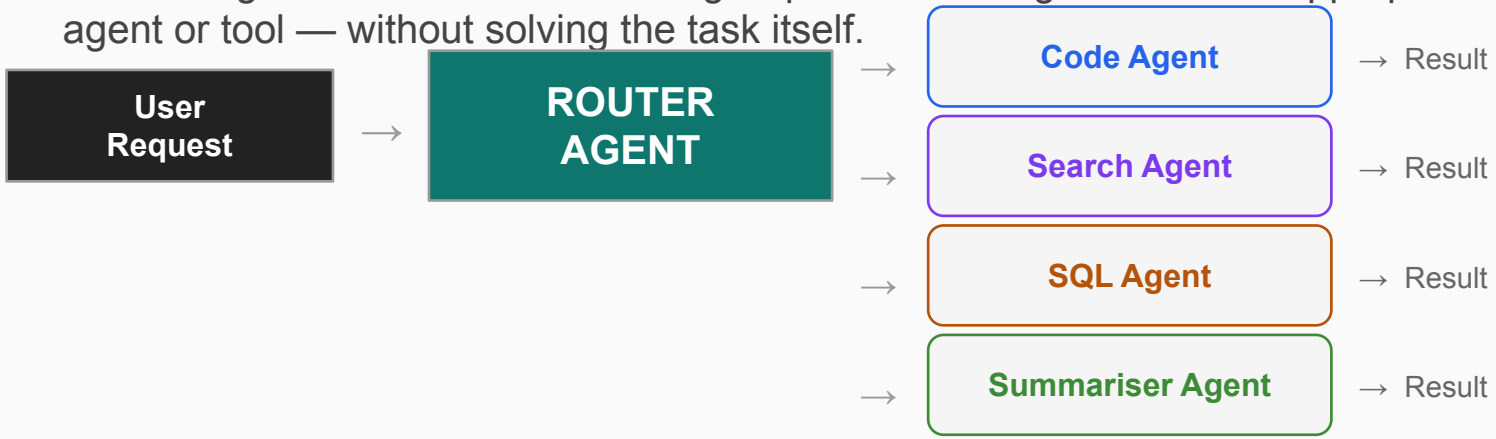
Suggests exploratory actions to discover new knowledge — even if suboptimal now.

Connection to RLHF (Reinforcement Learning from Human Feedback)

RLHF = Learning Agent in action: Performance Element = LLM | Critic = reward model trained on human ratings | Learning Element = PPO/GRPO optimizer. This is what happens during ChatGPT/Claude training.

Router Agents & Orchestrator Agents

A router agent classifies the incoming request and delegates it to the appropriate specialist agent or tool — without solving the task itself.



Intent Classification

Router uses LLM to classify the query intent (code, search, data, summarise) and route accordingly.

Tool Selection

Extends beyond agents to tools: the router chooses which API or function to call based on the input.

Fallback Handling

If no specialist matches, the router can invoke a general-purpose agent or ask the user for clarification.

Single-Agent Systems — Anatomy & ReAct Pattern

Anatomy of a Single LLM Agent

User / Environment

Provides task, query or observation as text input

LLM Core

Reasons, plans, decides next action (CoT or ReAct prompt)

Tool Router

Parses LLM output → identifies which tool and arguments

Tool Execution

Runs the tool and captures result (search, code, API)

Feedback Loop

Result appended to context; LLM reasons again until done

ReAct Trace — Example

Task:

What is the population of Mexico City in 2025?

Thought 1:

I need to search for current population data.

Action:

web_search("Mexico City population 2025")

Observation:

Result: approx. 22 million in metro area.

Thought 2:

I now have enough information to answer.

Answer:

Mexico City metro area: ~22 million (2025).

Planning Strategies in Agents

Chain-of-Thought (CoT)

How:

Step-by-step reasoning in a single forward pass. LLM writes out reasoning before giving an answer.

Best for:

Well-defined problems with a clear answer. No external info needed.

Limitation:

Single pass — cannot react to new info mid-reasoning.

Example:

Math word problems · Logical deductions · Code explanation

ReAct (Reason + Act)

How:

Interleaves Thought → Action → Observation cycles. Agent queries tools and reasons over results iteratively.

Best for:

Tasks requiring external information or multi-step execution.

Limitation:

Many LLM calls → higher latency and cost.

Example:

Research agent · Code debug loop · Web browsing

Tree-of-Thought (ToT)

How:

Explores multiple reasoning branches in parallel. Evaluates and prunes branches using heuristic scoring.

Best for:

Problems with multiple valid approaches; complex planning.

Limitation:

Expensive: many LLM calls for tree expansion.

Example:

Creative writing strategy · Game playing · Design search

What Are Multi-Agent Systems (MAS)?

A MAS consists of multiple interacting agents sharing an environment to solve tasks too large, complex, or parallel for a single agent.

Parallelism

Split a task into subtasks. Multiple agents work simultaneously → faster completion.

Specialisation

Each agent is an expert in its domain (coder, researcher, critic). Division of labour.

Robustness

If one agent fails, others continue. Redundancy improves system reliability.

Scalability

Add more agents as task size grows. No single bottleneck.

Single Agent

Sequential execution · Good for short focused tasks · Context window bottleneck · No parallelism

vs

Multi-Agent System

Parallel specialised agents · Long-horizon complex tasks · Communication overhead · Emergent intelligence

MAS Architectures — How Agents Are Organised

Centralised — Orchestrator–Worker

[Orchestrator]

↓ delegates ↓ delegates ↓

[Worker A] [Worker B] [Worker C]

Research · Code · Critique

↓ results ↓ results ↓

[Orchestrator synthesises → Final Output]

One orchestrator receives the task, decomposes it, delegates subtasks to specialised workers, collects results, and synthesises the final output.

Used in: crewAI, AutoGen, LangGraph

Strength: Clear control flow. Easy to debug and monitor.
Weakness: Single point of failure. Bottleneck at orchestrator.

Decentralised — Peer-to-Peer

[Agent 1] ↔ [Agent 2] ↔ [Agent 3]

↕ ↕ ↕

[Agent 4] ↔ [Agent 5]

Shared blackboard / message bus

← communicate via protocol →

Agents communicate directly using message-passing protocols (MCP, A2A, ACP). No single controller. Agents negotiate, share findings, and reach consensus.

Used in: Swarm AI, simulation environments

Strength: Robust to failure. Truly parallel.
Weakness: Harder to coordinate. Can diverge.

Agent Communication & Coordination in MAS

Communication Mechanisms

Shared Memory / Blackboard

Agents read/write to a common data store. Simple but can become a bottleneck.
Example: shared vector DB, shared context.

Message Passing

Agents send structured messages directly. Async or sync. Standard protocols: MCP, A2A, ACP. Analogous to microservices.

Broadcast

One agent broadcasts state updates to all. Others subscribe and react. Used in swarm coordination.

Coordination Strategies

Task Decomposition

Orchestrator breaks task into parallel subtasks, assigns to workers. Workers return results for synthesis.

Role Assignment

Each agent has a fixed role: Planner, Researcher, Coder, Critic, Reviewer. Reduces overlap and confusion.

Debate / Critique Loop

Agents generate competing solutions, critique each other's work, and converge to a consensus answer.

Agent Frameworks — What Developers Use

LangChain

Single + Multi

General-purpose pipelines. Chains of LLM calls + tool use. Large ecosystem of integrations.

```
agent = create_react_agent(llm, tools)
```

LangGraph

Single + Multi

Graph-based control flow. Nodes = agents, Edges = transitions. Great for complex loops.

```
graph.add_node('planner', planner_fn)
```

AutoGen (MSFT)

Multi-Agent

Conversational multi-agent. Agents chat with each other. Human-in-the-loop support.

```
AssistantAgent + UserProxy pattern
```

CrewAI

Multi-Agent

Role-based agent crews. Define agents by role, goal, backstory. Task + process flows.

```
Crew(agents=[researcher, coder])
```

OpenAI Assistants

Single

Managed API. Built-in tools: code interpreter, file search, function calling.

```
assistant.run(thread_id)
```

LangFlow

No-Code / GUI

Visual drag-and-drop builder for LangChain agents. Good for prototyping without coding.

```
Visual graph → JSON → deploy
```

Real-World Applications of AI Agents

Software Engineering

- GitHub Copilot Workspace: write, test, fix code end-to-end
- Devin / SWE-agent: resolve GitHub issues autonomously
- Code review agent: multi-agent lint + security + logic critique

Research & Information

- Deep Research (OpenAI): multi-step web research → reports
- Academic assistants: search papers, extract findings, summarise
- Patent analysis: claim comparison across thousands of docs

Customer Service

- Tier-1 support: answer FAQs, check orders, escalate to humans
- Personalised shopping assistants with preference memory
- Multi-language agents serving global customers 24/7

Healthcare

- Clinical summarisation: read records, generate structured summaries
- Drug interaction checker: retrieves + cross-references databases
- Appointment scheduling + follow-up coordination agents

Finance & Operations

- Automated financial report generation from raw data
- Fraud detection: monitor + flag + investigate anomalies
- Supply chain: monitor inventory, place orders, reroute shipments

Education

- Personalised tutoring agents adapting to student progress
- Automated grading with detailed, personalised feedback
- Curriculum design: tailored lesson plan generation

Challenges & Open Problems

Agents are powerful — but these unsolved problems keep researchers and engineers busy:

Hallucination & Trust

Agents acting on hallucinated outputs cascade into serious failures. Ground truth verification is hard.

Long-Horizon Planning

LLMs struggle to maintain coherent plans over 20+ steps. Error accumulation and context limits compound.

Tool Reliability

APIs fail, return unexpected formats, or have rate limits. Agents must handle errors gracefully.

Safety & Alignment

Agents with real actuators (file deletion, code execution) can cause irreversible harm if misaligned or prompt-injected.

Multi-Agent Coordination

Preventing duplicate work, contradictions, or feedback loops between agents is an open engineering problem.

Evaluation & Benchmarking

No standard benchmark for general agentic capability. Hard to measure performance on open-ended tasks.

Cost & Latency

Each agent loop costs LLM calls. Multi-agent systems multiply this. Quality vs cost tradeoff is non-trivial.

Human-in-the-Loop

When should an agent pause for human confirmation? Too often = useless. Too rarely = dangerous. Active research area.

What You Should Know from This Lecture

1 Agent = Perceive + Reason + Act

Any system sensing its environment and acting toward a goal. PEAS captures the essentials.

2 Agent Classification

By architecture (5 types) and by scope (task · router · conversational · autonomous). Both axes matter.

3 The Agent Loop

Perceive → Think → Act → Observe → repeat. ReAct formalises this. CoT / ToT handle planning.

4 LLMs + RAG = Brain + Memory

Everything you learned plugs into agents. LLM = reasoner, RAG = memory, FSL = fast skill acquisition.

5 Multi-Agent Systems

Multiple specialised agents > one generalist for complex tasks. Orchestrator-worker or peer-to-peer.

6 Router Agents Are Central

In any real MAS, a router/orchestrator classifies intent and delegates — the glue of multi-agent systems.

Discussion Questions

Q1 How does an AI agent differ from a traditional software program that takes inputs and produces outputs?

Q2 You need to build a system that writes and executes a research report. Design the agent architecture: which types of agents would you use, and how would they communicate?

Q3 A router agent mis-classifies a medical query and sends it to a general chat agent instead of a medical specialist agent. What safeguards would you design?

Q4 In a multi-agent debate loop, two agents give contradictory answers. How should the system resolve the conflict? Is there a parallel to human knowledge systems?