What is AI?

Artificial Intelligence builds agents that perceive reason act to achieve goals.

Preception:Vison,speech,sensors.

Reasoning & Learning: Search,planning,ML

Action:robitics,control

Application:healthcare(diagnolsis),finance(fraud),maps(routing),assistant(NLP),games(chess/Go),robotics(drones).

Classification of AI system by Environment:

In AI, the environment is Everything external to

the agent that it interacts with. Agents perceive

the enviroinment through sensors and act usig using actuators.

1.Observable:

**Fully Observable:**

The agent's sensor can access the complete state of the Environment at each Point in time.

Example:Chess or tic-tac-toe (the whole board is visible).

**Partially Observable:** The agent only perceives part of the state; the environment may be hidden or noisy.

Example: Self-driving cars (can’t see around corners, sensors may fail

2.Agents:

Single-Agent:Only one agent is operating,no competition or cooperation.

Example:Crossword puzzle Solver.

Multi-Agent:Multiple agent exist,action affect each other.

Example:

Cooperative: robots moving boxes together.

Competitive: chess, poker, online auctions.

3. Determinism

Deterministic: The next state is completely determined by the current state and action.

Example: Mathematical puzzles, solving a maze.

Stochastic (Non-deterministic): Outcomes are uncertain due to randomness or unknown factors.

Example: Delivery drones (affected by wind/weather).

4. Episodic vs Sequential

Episodic: Agent’s experience is divided into independent episodes. Each decision does not depend on previous ones.

Example: Image recognition (classifying one image at a time).

Sequential: Current decisions affect future actions.

Example: Chess, driving a car (past choices matter for future).

5. Static vs Dynamic

Static: The environment does not change while the agent is deciding.

Example: Crossword puzzles.

Dynamic: Environment may change during computation.

Example: Stock market trading, real-time games.

Semi-dynamic: Environment doesn’t change, but performance score does.

Example: Timed exam (questions stay same, but score reduces if you take too long).

6. Discrete vs Continuous

Discrete: States, actions, and perceptions are distinct and countable.

Example: Chess (finite moves).

Continuous: States, time, and actions are real-valued, infinite.

Example: Driving a car (continuous speed, steering angles, time).

7. Known vs Unknown

Known: Agent knows the rules, transitions, and effects of actions.

Example: Chess (rules are fixed).

Unknown: Agent must learn the effects of actions through interaction.

Example: Learning to play a new video game without rules provided

| Property | Type 1 | Type 2 | Example |

| --------------- | ------------- | ---------- | ---------------------------- |

| \*\*Observable\*\* | Fully | Partially | Chess vs Self-driving |

| \*\*Agents\*\* | Single | Multi | Puzzle vs Chess |

| \*\*Determinism\*\* | Deterministic | Stochastic | Maze vs Drone |

| \*\*Episodes\*\* | Episodic | Sequential | Image recognition vs Driving |

| \*\*Change\*\* | Static | Dynamic | Crossword vs Stock market |

| \*\*State\*\* | Discrete | Continuous | Chess vs Driving |

| \*\*Knowledge\*\* | Known | Unknown | Chess vs New game

|

Types of Intelligence:

1.simple reflex Agent

2.Model based reflex Agents

3.Goal based Agents

4.Utility based Agents

5.Learning Agents.

**3)Problem-Solving Agents & Search:**

Generic Tree Search

What it is:

generic tree search: explores a problem space structured like a tree. It assumes \*\*no repeated states\*\*, meaning each node is visited only once.

How it works:

- Starts at the \*\*root node\*\* (initial state).

- Expands nodes by generating \*\*successors\*\*.

- Uses a \*\*queue (fringe)\*\* to keep track of nodes to explore.

- Applies a strategy (e.g., DFS, BFS, UCS, A\*) to decide which node to expand next.

Limitations:

- Doesn’t check for \*\*repeated states\*\*, so it can revisit the same state multiple times if the problem space is cyclic.

- Can be inefficient or even infinite in cyclic graphs.

---

Generic Graph Search:

What it is:

A \*\*generic graph search\*\* is an enhancement of tree search that \*\*avoids revisiting states\*\* by keeping track of explored nodes.

How it works:

- Same basic structure as tree search.

- Adds an \*\*explored set\*\* to record visited states.

- Before expanding a node, it checks if the state has already been explored.

- Prevents cycles and redundant work.

Advantages:

- More efficient in cyclic or large graphs.

- Guarantees \*\*completeness\*\* (if a solution exists, it will find it).

- Avoids infinite loops.

---

## 🔁 Comparison Table

| Feature | Tree Search | Graph Search |

|------------------------|----------------------------------|----------------------------------|

| Repeated State Check | ❌ No | ✅ Yes |

| Explored Set | ❌ Not used | ✅ Used |

| Suitable for Cycles | ❌ No | ✅ Yes |

| Memory Usage | 🟡 Lower | 🔵 Higher |

| Completeness | 🟡 Depends on strategy | ✅ Guaranteed (with proper strategy) |

---

## 🧪 Example Use Case

Imagine solving a maze:

- \*\*Tree search\*\* might keep re-entering the same dead-end.

- \*\*Graph search\*\* remembers where it’s been and avoids wasting time.

Graph:

Input: problem

Frontier ← priority container as per strategy

Explored ← ∅

Insert initial node into Frontier

Loop:

if Frontier empty → failure

n ← pop(Frontier)

if Goal(n.state) → return solution

if n.state ∉ Explored:

add n.state to Explored

for each action a:

s' ← Result(...); g' ← n.g + cost(...)

if s' ∉ Explored and not in Frontier with lower g:

push/update s' into Frontier

Input: problem

Frontier ← {Node(state=initial, parent=∅, action=∅, g=0)}

Loop:

if Frontier empty → return failure

n ← select and remove a node from Frontier (strategy decides which)

if Goal(n.state) → return solution (trace parents)

for each action a in A(n.state):

s' ← Result(n.state, a); g' ← n.g + cost(n.state,a,s')

add Node(s', parent=n, action=a, g=g') to Frontier

An agent that stores facts about the world in Knowledge base(KB) and derive new facts

Using an inference engine.

 **Interface:**

* TELL(KB, sentence) → add knowledge
* ASK(KB, query) → derive whether KB ⊨ query (entailed)

 **Why logic?** Precise syntax + clear semantics (truth in a model) → sound (never proves false things) and, in restricted fragments, complete (can prove all true things).