SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES, CHITTOOR
(AUTONOMOUS)

Approved by AICTE, New Delhi, Affiliated to JNTUA, Ananthapuramu.

LECTURE NOTES

Course Name : AIML for Mechanical Engineering

Course Code : 23MEC361T

Year / Branch : III Year – Mechanical

Regulation : R23

Prepared By : Dr.S.Rajesh

Designation : Professor & COE

UNIT 1

INTRODUCTION TO ARTIFICIAL INTELLIGENCE AND PROBLEM- SOLVING AGENT

"The goal of AI is to make machines think, learn, and solve problems like humans."

— Andrew Ng

Artificial Intelligence (AI)

Definition:

Artificial Intelligence (AI) is a branch of computer science that enables machines to perform tasks that require human intelligence such as learning, reasoning, problem-solving, perception, and decision-making.

Key Characteristics:

• Learning – acquiring knowledge from data
• Reasoning – drawing logical conclusions
• Problem-solving – finding solutions
• Perception – interpreting sensory input
• Decision-making – choosing best actions

Major Problems Addressed in AI:

1. **Problem Solving & Search:**

AI explores possible solutions using algorithms like BFS, DFS, and A*. Example: games and pathfinding.

2. **Knowledge Representation:**

Organizing knowledge using logic, semantic networks, and ontologies. Example: expert systems.

3. **Reasoning & Inference:**

Drawing conclusions using deductive and inductive reasoning. Example: medical diagnosis.

4. **Machine Learning:**

Systems learn from data. Types: supervised, unsupervised, reinforcement learning.

5. **Natural Language Processing:**

Understanding human language. Example: chatbots, translation systems.

6. **Computer Vision:**

Interpreting images and videos. Example: face recognition, self-driving cars.

7. **Planning & Decision Making:**

Choosing actions to achieve goals. Example: scheduling, navigation systems.

8. **Robotics:**

Combining AI with machines to perform tasks. Example: industrial robots.

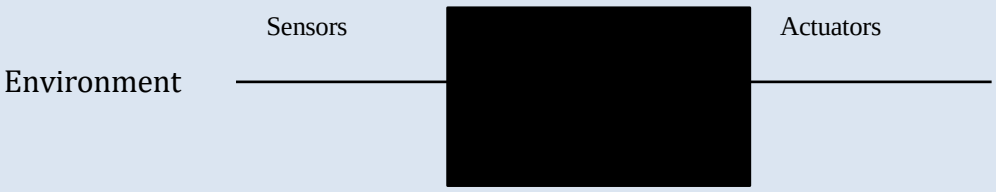
9. **Handling Uncertainty:**

Using probability and fuzzy logic to deal with incomplete data.

Intelligent Agents

Definition:

An Intelligent Agent perceives its environment through sensors and acts upon it using actuators to achieve goals.



Types of Intelligent Agents:

1. Simple Reflex Agent:

Acts based on current percept using condition-action rules. Example: thermostat.

2. Model-Based Agent:

Maintains internal state using past percepts.

3. Goal-Based Agent:

Acts to achieve goals using planning.

4. Utility-Based Agent:

Chooses actions based on utility function.

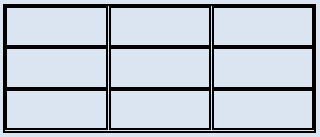
5. Learning Agent:

Improves performance using experience.

Tic-Tac-Toe as an AI Problem

Problem Definition:

Tic-Tac-Toe is a two-player game (X and O) played on a 3x3 grid. The goal is to place three marks in a row (horizontal, vertical, or diagonal).



Representation in AI:

• State: Configuration of the board

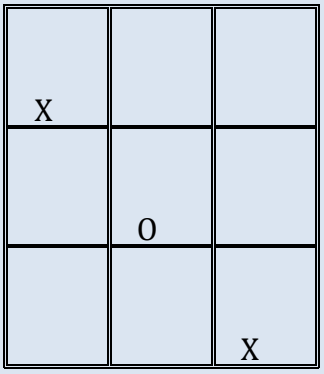
• Initial State: Empty board

• Players: X and O

• Actions: Place mark in empty cell

• Goal Test: Three in a row

• Utility: +1 (win), 0 (draw), -1 (loss)

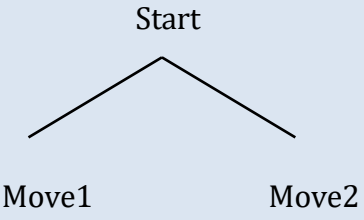


How AI Solves It:

• Use Game Tree: All possible moves are explored

• Minimax Algorithm: Maximizes winning chances and minimizes opponent advantage

• Terminal States: Win, Loss, or Draw



Key Concept:

Minimax assumes opponent plays optimally and selects best move accordingly.

Nature of Environments in AI

Definition:

An environment in AI is the surroundings in which an agent operates and interacts.

Types of Environments:

1. Deterministic Environment:

Next state is completely determined by current state and action. No randomness. Example: Chess.

2. Stochastic Environment:

Outcome involves randomness and uncertainty. Example: Weather prediction.

3. Static Environment:

Environment does not change while agent is thinking. Example: Crossword puzzle.

4. Dynamic Environment:

Environment changes continuously even if agent is not acting. Example: Self-driving cars.

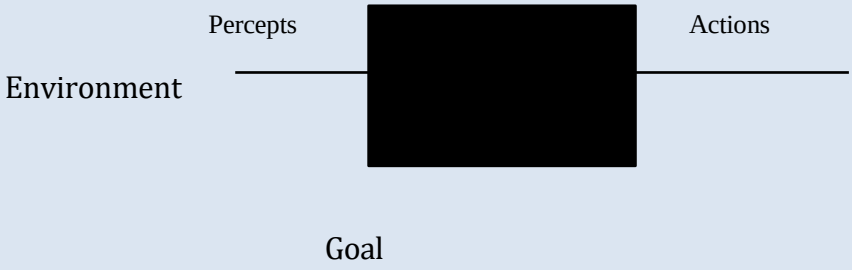
Deterministic vs Stochastic

Static vs Dynamic

Goal-Based Agent

Definition:

A Goal-Based Agent selects actions based on achieving specific goals. It evaluates future states and chooses actions that lead to goal achievement.



How it Operates:

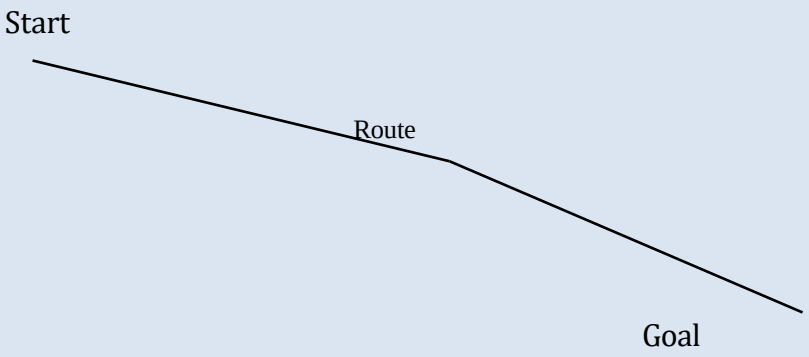
1. Perceives environment through sensors
2. Maintains internal model of world
3. Defines goal to achieve
4. Uses search/planning to evaluate actions
5. Selects action that leads closer to goal

Real-World Example: GPS Navigation System

The agent takes current location as input and goal as destination.

It evaluates multiple routes and selects the best path based on

distance/time. If traffic changes, it updates plan dynamically.



Key Point:

Goal-based agents are flexible and can adapt to different goals by planning actions accordingly.

Water Jug Problem - State Space Search

Problem Statement:

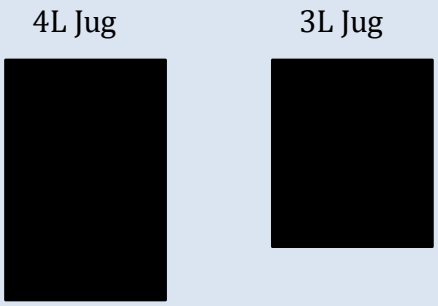
Given two jugs of capacity 4L and 3L, measure exactly 2L using allowed operations.

State Space Representation:

State: (x, y) where x = water in 4L jug, y = water in 3L

jug Initial State: (0, 0)

Goal State: (2, y) or (x, 2)



Operators (Actions):

• Fill a jug completely

• Empty a jug

• Pour water from one jug to another

Sample Solution Steps:

(0,0) → (4,0) → (1,3) → (1,0) → (0,1) → (4,1) → (2,3)

(0,0) _____ (4,0) _____ (1,3) _____ ...

State Space Search:

All possible states form a graph. Search algorithms like BFS/DFS are used to find path to goal state.

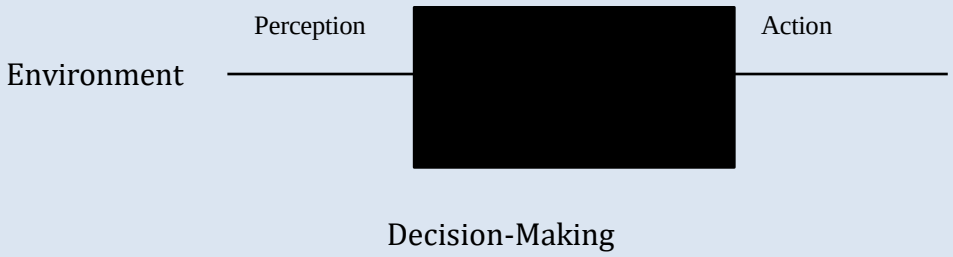
Key Point:

Water Jug problem is a classic example of problem formulation using state space search in AI.

Structure of Intelligent Agents

Concept:

An intelligent agent consists of perception, decision-making, and action components that work together to achieve goals.



1. Perception:

The agent collects information from the environment through sensors. This information is called percepts.

2. Decision-Making:

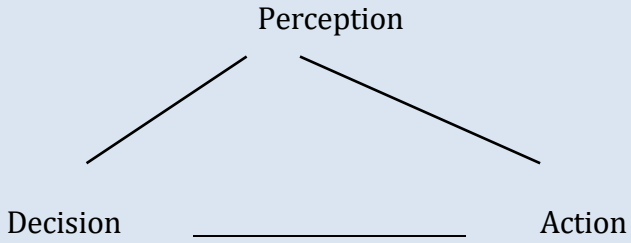
The agent processes percepts using internal logic, knowledge, or learning to choose the best action.

3. Action:

The agent performs actions through actuators to influence the environment.

Interconnection:

Perception provides input to decision-making. Decision-making selects appropriate actions. Actions change the environment, leading to new percepts, forming a continuous loop.



Key Point:

This continuous perception-action loop enables intelligent behavior in dynamic environments.

Goal-Based vs Utility-Based Agents

Goal-Based Agent:

Selects actions to achieve a specific goal using search and planning.

Utility-Based Agent:

Selects actions based on a utility function to maximize overall satisfaction.

Aspect	Goal-Based Agent	Utility-Based Agent
Decision	Goal achieved or not	Maximize utility value
Output	Binary	Quantitative
Flexibility	Limited	High
Uncertainty	Poor handling	Handles well
Complexity	Simple	Complex

Advantages of Goal-Based Agents:

• Simple design

• Clear objective

Limitations:

• Cannot compare best solutions

• Poor in uncertainty

Advantages of Utility-Based Agents:

• Handles trade-offs

• Suitable for real-world problems

Limitations:

• Hard to design utility function

• Computationally expensive

Conclusion:

Utility-based agents extend goal-based agents by enabling better decision-making in complex environments.

AI Techniques vs Traditional Programming

Introduction:

Artificial Intelligence (AI) techniques and traditional programming differ in how they solve problems. AI focuses on learning and adaptability, while traditional programming relies on predefined rules.

Aspect	AI Techniques	Traditional Programming
Approach	Learning from data	Explicit rules
Adaptability	High	Low
Handling Complexity	Effective for complex, uncertain problems	Best for well-defined problems
Data Dependency	Requires large datasets	Less dependent on data
Examples	Self-driving cars, NLP	Sorting, calculators

Effectiveness of AI Techniques:

• Handles unstructured data (images, text)

• Learns and improves over time

• Suitable for uncertain and dynamic environments

• Enables automation of complex decision-making

Limitations of AI:

• Requires large data and computation

• Lack of transparency (black-box models)

• May produce errors if trained poorly

Strengths of Traditional Programming:

• High reliability and predictability

• Easier to debug and understand

• Suitable for deterministic problems

Conclusion:

AI techniques are more effective for complex real-world problems involving uncertainty and large data, while traditional programming is better for simple, well-defined tasks.

Design Issues in Search Programs

Introduction:

Search algorithms are fundamental in AI. Their design involves evaluating performance based on completeness, optimality, time complexity, and space complexity.

1. Completeness:

A search algorithm is complete if it is guaranteed to find a solution if one

exists. Example: Breadth-First Search is complete.

2. Optimality:

An algorithm is optimal if it finds the best (lowest cost)

solution. Example: Uniform Cost Search is optimal.

3. Time Complexity:

Measures the amount of time taken to find a solution.

Depends on branching factor (b) and depth (d). Example: BFS has O(b^d).

4. Space Complexity:

Measures memory usage during search.

BFS requires large memory, DFS uses less memory.

Criteria	Meaning	Example
Completeness	Finds solution if exists	BFS
Optimality	Best solution	Uniform Cost
Time Complexity	Execution time	O(b^d)
Space Complexity	Memory usage	DFS low

Critical Analysis:

• Trade-off between time and space complexity

• Complete algorithms may consume more memory

• Optimal algorithms may take more time

• Choice depends on problem requirements

Conclusion:

Designing efficient search programs requires balancing completeness, optimality, time, and space constraints based on application needs.

UNIT –2

SEARCH TECHNIQUES

"AI is concerned with intelligent behavior in artifacts."

- *Nils J. Nilsson*

Problem-Solving Agents in AI

A problem-solving agent is a type of intelligent agent that decides what actions to take by searching for solutions to well-defined problems. It operates by identifying a goal and then exploring possible sequences of actions to reach that goal.

Definition

A problem-solving agent is an agent that:

- Formulates a problem based on its goals
- Searches through possible actions
- Selects a sequence of actions that leads to the desired goal state

Key Components of a Problem-Solving Agent

1. Initial State
   The starting point of the agent.
2. State Space
   All possible states reachable from the initial state.
3. Actions (Operators)
   The set of possible actions available to the agent.
4. Transition Model
   Describes the result of applying an action to a state.
5. Goal State
   The desired outcome the agent wants to achieve.
6. Path Cost
   The cost associated with a sequence of actions.

Process of Searching for Solutions in AI

The problem-solving process involves several well-defined steps:

1. Goal Formulation
   - The agent determines what it wants to achieve.
   - Example: Reach a destination in a map.
2. Problem Formulation
   - The agent translates the goal into a formal problem.
   - Defines states, actions, and constraints.
3. Search
   - The agent explores possible sequences of actions.
   - Uses search algorithms to find a path from the initial state to the goal state.
4. Solution Execution
   - Once a solution is found, the agent executes the sequence of actions.

Search Techniques in AI

1. Uninformed (Blind) Search
   - No domain knowledge is used.
   - Examples:
     - Breadth-First Search (BFS)
     - Depth-First Search (DFS)
     - Uniform Cost Search
2. Informed (Heuristic) Search
   - Uses additional knowledge (heuristics) to guide the search.
   - Examples:
     - Greedy Best-First Search
     - A* Search

General Search Framework

The search process can be visualized as:

1. Start with the initial state
2. Expand possible actions → generate new states
3. Check if the goal is reached
4. If not, continue exploring
5. Stop when a goal state is found

Example: Route-Finding Problem

- Initial State: Starting city
- Goal State: Destination city
- Actions: Travel between cities
- Solution: Shortest path between cities

Properties of Search Algorithms

When evaluating search strategies, we consider:

- Completeness: Will it find a solution if one exists?
- Optimality: Does it find the best solution?
- Time Complexity: How fast is the search?
- Space Complexity: How much memory is required?

Uninformed (Uniform) Search Strategies in AI

Uninformed search strategies (also called blind search) explore the search space without using domain-specific knowledge. They rely only on the problem definition (initial state, actions, goal test, and path cost).

Breadth-First Search (BFS)

Description

- Explores the search tree **level by level**.
- Expands all nodes at the current depth before moving to the next level.
- Uses a **queue (FIFO)** data structure.

Key Features

- Finds the **shortest path** (in terms of number of steps)
- Explores shallow nodes first

Properties

- **Completeness:** Yes (if branching factor is finite)
- **Optimality:** Yes (for equal step costs)
- **Time Complexity:** O(bd)O(b^d)O(bd)
- **Space Complexity:** O(bd)O(b^d)O(bd)

Advantage

- Guarantees optimal solution

Disadvantage

- High memory consumption

Depth-First Search (DFS)

Description

- Explores as far as possible along a branch before backtracking.
- Uses a **stack (LIFO)** or recursion.

Key Features

- Goes deep into the search tree first

Properties

- **Completeness:** No (may get stuck in infinite loops)
- **Optimality:** No
- **Time Complexity:** O(bm)O(b^m)O(bm)
- **Space Complexity:** O(bm)O(bm)O(bm)

Advantage

- Requires less memory

Disadvantage

- May not find the optimal solution
- Risk of infinite depth

Depth-Limited Search (DLS)

Description

- A variation of DFS with a **depth limit (l)**.
- Prevents infinite descent by restricting depth.

Key Features

- Avoids infinite loops in deep or infinite trees

Properties

- **Completeness:** Yes (if l≥dl \geq dl≥d)
- **Optimality:** No
- **Time Complexity:** O(bl)O(b^l)O(bl)
- **Space Complexity:** O(bl)O(bl)O(bl)

Advantage

- Controls depth, avoids infinite paths

Disadvantage

- May miss the solution if depth limit is too small

Bidirectional Search

Description

- Runs **two simultaneous searches**:
  - Forward from initial state
  - Backward from goal state
- Stops when the two searches meet

Key Features

- Reduces search space significantly

Properties

- **Completeness:** Yes
- **Optimality:** Yes (if BFS is used in both directions)
- **Time Complexity:** O(bd/2)O(b^{d/2})O(bd/2)
- **Space Complexity:** O(bd/2)O(b^{d/2})O(bd/2)

Advantage

- Much faster than BFS for large problems

Disadvantage

- Requires ability to generate predecessor states
- More complex to implement

Strategy	Complete	Optimal	Time Complexity	Space Complexity	Key Idea
BFS	Yes	Yes	(O(b^d))	(O(b^d))	Level-wise search
DFS	No	No	(O(b^m))	(O(bm))	Deep search first
DLS	Yes*	No	(O(b^l))	(O(bl))	Depth-limited DFS
Bidirectional Search	Yes	Yes	(O(b^{d/2}))	(O(b^{d/2}))	Two-way search

Breadth First Search (BFS)

Working Principle

Breadth First Search explores the search space **level by level**, starting from the root node. It visits all nodes at one depth before moving to the next level.

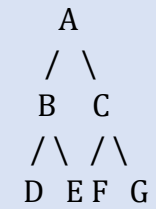
- Uses a **Queue (FIFO)**
- Nodes are expanded in the order they are discovered
- Guarantees the **shortest path** in unweighted graphs

Algorithm Steps (BFS)

1. Start with the **initial node** and add it to the queue
2. Remove the front node from the queue
3. Check if it is the goal
4. If not, add all its **unvisited neighbors** to the queue
5. Repeat until goal is found or queue becomes empty

Example (BFS Traversal)

Consider the tree:



Traversal Steps

- Start at **A** → Queue: [A]
- Visit A → add B, C → Queue: [B, C]
- Visit B → add D, E → Queue: [C, D, E]
- Visit C → add F, G → Queue: [D, E, F, G]

BFS Order: A → B → C → D → E → F → G

Advantages of BFS

- Complete and optimal (for equal cost problems)
- Finds shortest path

Disadvantages of BFS

- High **memory usage**
- Slow for deep trees

Depth First Search (DFS)

Working Principle

Depth First Search explores the search space by going **as deep as possible** along a branch before backtracking.

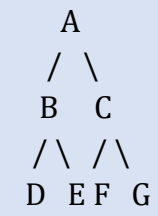
- Uses a **Stack (LIFO)** or recursion
- Explores one branch fully before moving to another

Algorithm Steps (DFS)

1. Start with the **initial node**
2. Push it onto the stack
3. Pop a node and check if it is the goal
4. If not, push its **unvisited neighbors** onto the stack
5. Repeat until goal is found or stack is empty

Example (DFS Traversal)

Using the same tree:



Traversal Steps

- Start at **A**
- Go deep: A → B → D
- Backtrack: D → B → E
- Backtrack: B → A → C → F → G

DFS Order: A → B → D → E → C → F → G

Advantages of DFS

- Uses less memory
- Simple to implement

Disadvantages of DFS

- Not guaranteed to find shortest path
- May get stuck in deep/infinite branches

Heuristic Search Strategies in AI

Concept of Heuristic Search

A **heuristic search strategy** is an approach in Artificial Intelligence that uses **additional knowledge (heuristics)** to guide the search process toward the goal more efficiently.

- A **heuristic function**, usually denoted as **h(n)**, estimates the **cost or distance from a node to the goal**.
- It helps the agent choose **more promising paths**, reducing unnecessary exploration.

Definition

A **heuristic search** is a search method that uses **problem-specific knowledge** to improve efficiency by prioritizing nodes that appear closer to the goal.

How Heuristic Search Works

1. Start from the **initial state**
2. Evaluate possible next states using a **heuristic function h(n)**
3. Select the node with the **best (lowest or highest) heuristic value**
4. Continue until the **goal state is reached**

Examples of Heuristic Search Algorithms

1. Greedy Best-First Search

- Selects the node with the **lowest heuristic value h(n)**
- Focuses only on estimated distance to goal

Advantage: Fast

Disadvantage: Not always optimal

2. A Search Algorithm*

- Uses both:
  - **g(n):** cost from start to current node
  - **h(n):** estimated cost to goal
- Evaluation function:
  $f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)$

Advantage:

- Complete and optimal (if heuristic is admissible)

Disadvantage:

- Higher memory usage

Characteristics of Heuristics

- **Admissible Heuristic:** Never overestimates the true cost
- **Consistent Heuristic:** Follows triangle inequality
- **Good heuristic:** Leads to faster and optimal solutions

Uninformed (Blind) Search

Uninformed search strategies **do not use any additional knowledge** beyond the problem definition.

They explore the search space **systematically** without guidance.

Examples

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Uniform Cost Search (UCS)

A* Search Algorithm – Path Finding Example

What is A* Search?

A* is a **heuristic search algorithm** that finds the **optimal path** by combining:

- Actual cost from start → **g(n)**
- Estimated cost to goal → **h(n)**

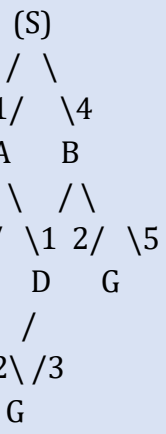
It evaluates nodes using:

$f(n)=g(n)+h(n)$f(n) = g(n) + h(n)$f(n)=g(n)+h(n)$

Example Problem

Find the shortest path from **Start node S** to **Goal node G**.

Graph with Costs & Heuristics



Heuristic Values h(n)

Node	h(n)
S	7
A	6
B	2
C	4
D	1
G	0

Step-by-Step Execution of A*

Step 1: Start at S

- g(S) = 0
- f(S) = 0 + 7 = **7**

Open List: S

Step 2: Expand S

Neighbors: A, B

Node	g(n)	h(n)	f(n)
A	1	6	7
B	4	2	6

Choose **B** (lowest f = 6)

Step 3: Expand B

Neighbors: D, G

Node	g(n)	h(n)	f(n)
D	6	1	7
G	9	0	9

Open List: A (7), D (7), G (9)

Choose **A** or **D** (tie → pick A)

Step 4: Expand A

Neighbors: C, D

Node	g(n)	h(n)	f(n)
C	4	4	8
D	2	1	3

Update D with lower cost

Open List: D (3), C (8), G (9)

hoose **D**

Step 5: Expand D

Neighbor: G

Node	g(n)	h(n)	f(n)
G	5	0	5

Update G with better cost

Step 6: Reach Goal G

- Path found: **S → A → D → G**
- Total cost = **5**

Final Optimal Path

S → A → D → G
Total Cost = 5 (Optimal)

Why A* Gives Optimal Solution

A* guarantees optimality when the heuristic is:

1. Admissible

- Never overestimates the true cost
- h(n)≤actual costh(n) \leq \text{actual cost}h(n)≤actual cost

2. Consistent

- Follows triangle inequality
- Ensures no need to revisit nodes

Key Observations

- A* avoids unnecessary paths (e.g., S → B → G with cost 9)
- It updates nodes when a **better path is found**
- Combines advantages of:
    - **Uniform Cost Search (g(n))**
    - **Greedy Search (h(n))**

Greedy Best-First Search (GBFS) – Route Finding Example

Concept

Greedy Best-First Search is a **heuristic search strategy** that always selects the node that appears **closest to the goal** using a heuristic function:

f(n)=h(n)f(n) = h(n)f(n)=h(n)

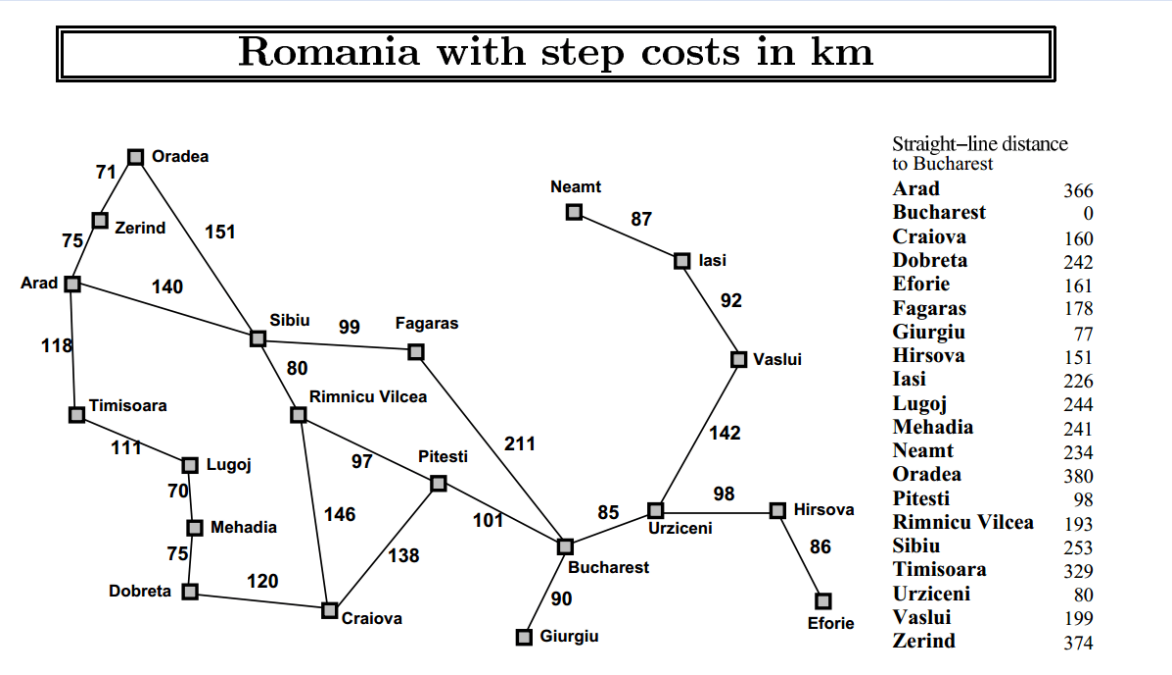
- It **ignores the path cost (g(n))**
- Focuses only on **estimated distance to the goal**

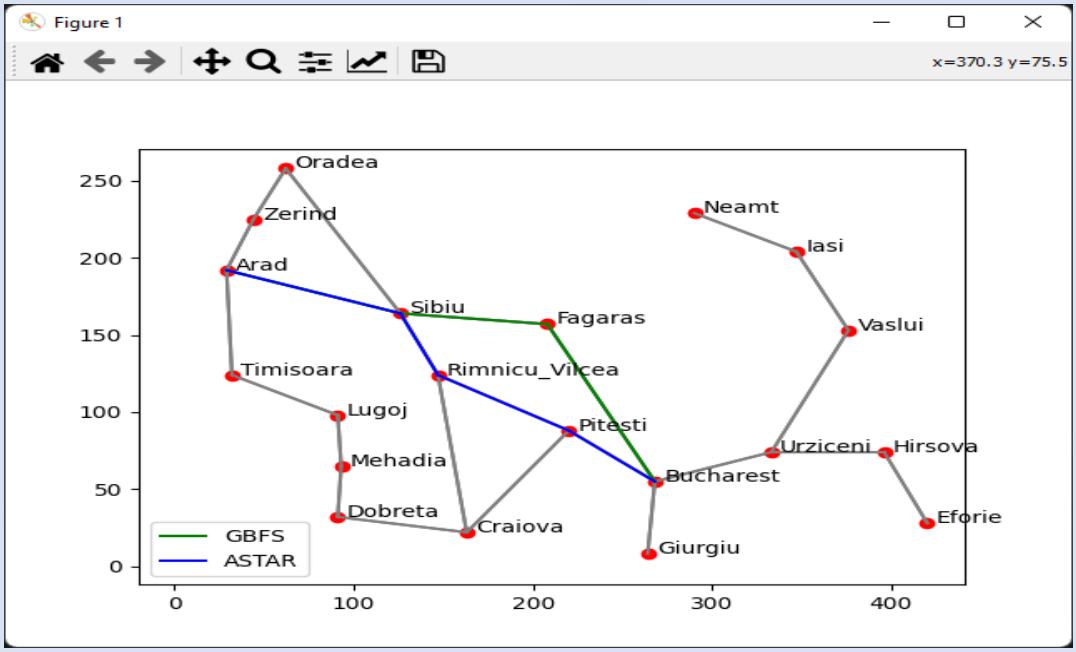
Real-World Scenario: Route Finding

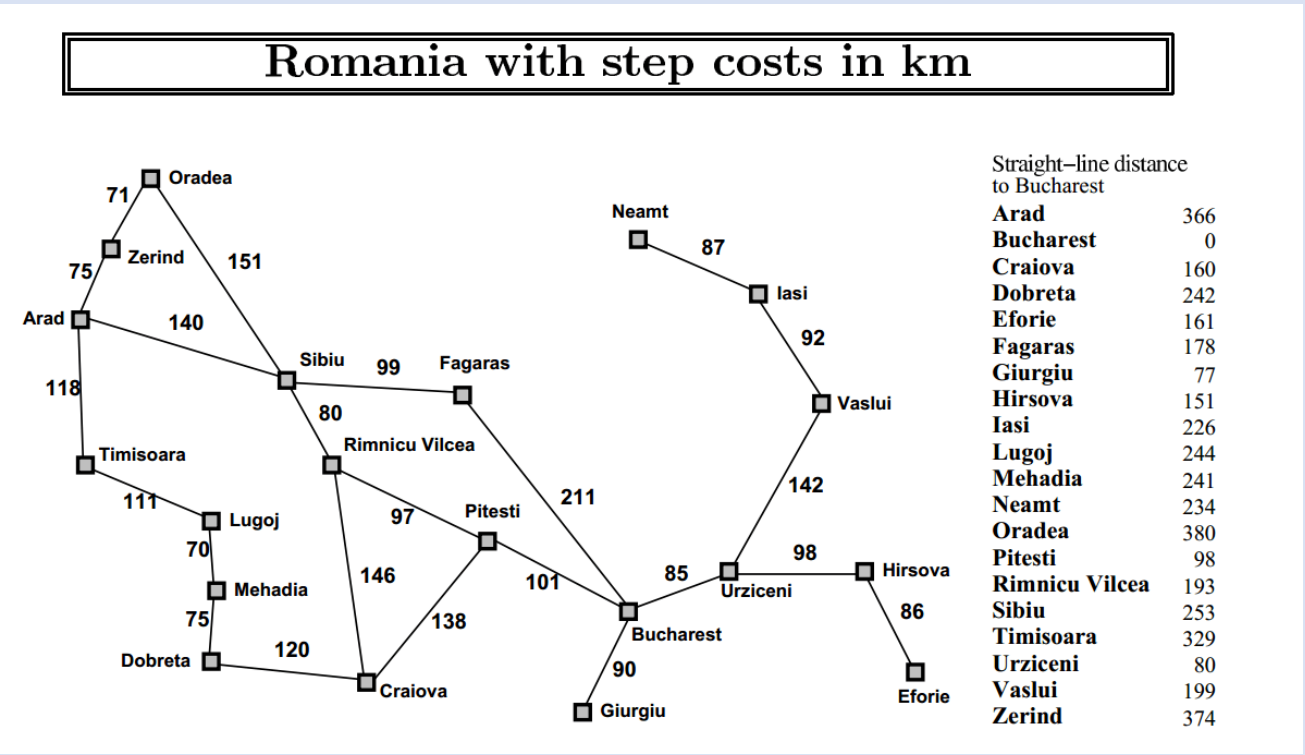
Problem

Find a path from **Arad (Start)** to **Bucharest (Goal)** using a map (classic AI example).

Map Representation (Romania Map)







Heuristic Values (Straight-Line Distance to Bucharest)

City	h(n)
Arad	366
Sibiu	253
Timisoara	329
Zerind	374
Fagaras	176
Rimnicu V.	193
Bucharest	0

Step-by-Step Execution of Greedy Best-First Search

Step 1: Start at Arad

- h(Arad) = 366
- Expand neighbors: Sibiu (253), Timisoara (329), Zerind (374)

Choose **Sibiu** (lowest h)

Step 2: From Sibiu

Neighbors:

- Fagaras (176)
- Rimnicu Vilcea (193)

Choose **Fagaras** (lowest h)

Step 3: From Fagaras

Neighbor:

- Bucharest (0)

Choose **Bucharest**

Final Path Found

Arad → Sibiu → Fagaras → Bucharest

Analysis of the Solution

Path Cost (Actual Distance)

- Arad → Sibiu = 140
- Sibiu → Fagaras = 99
- Fagaras → Bucharest = 211

Total Cost = 450

Is it Optimal?

✕ No

A better path exists:

- Arad → Sibiu → Rimnicu Vilcea → Pitesti → Bucharest
  Cost = **418 (shorter)**

Key Observations

- Greedy search **quickly reaches the goal**
- It chooses paths that look promising **locally**
- It may **miss the optimal path** because it ignores actual cost

Advantages

- Fast and efficient
- Easy to implement
- Works well when heuristic is very accurate

Disadvantages

- Not optimal
- Can get trapped in local minima
- Ignores path cost (g(n))

Greedy Best-First Search vs A* Search

Both **Greedy Best-First Search (GBFS)** and **A*** are **heuristic search strategies**, but they differ significantly in how they evaluate nodes and guarantee solutions.

Core Idea

- **Greedy Best-First Search:**
  Chooses the node that appears **closest to the goal** using only heuristic:
  $f(n)=h(n)f(n) = h(n)f(n)=h(n)$
- **A* Search:**
  Considers both **actual cost** and **estimated cost**:
  $f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)$

Key Differences

Aspect	Greedy Best-First Search	A* Search
Evaluation Function	$f(n)=h(n)f(n) = h(n)f(n)=h(n)$	$f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)$
Focus	Goal-directed (fast)	Balanced (cost + goal)
Optimality	✗ Not guaranteed	✓ Guaranteed (if heuristic valid)
Completeness	✗ Not always	✓ Yes
Speed	Very fast	Slower than Greedy
Memory Usage	Less	High
Accuracy	May choose wrong path	Finds best path

Working Behavior Comparison

Greedy Best-First Search

- Expands nodes that look **closest to goal**
- Ignores actual cost already incurred
- Can be misled by poor heuristics

Example behavior:

- Chooses shortest-looking path but may be **expensive overall**

A* Search

- Balances:
  - Cost so far (**g(n)**)
  - Estimated remaining cost (**h(n)**)
- Continuously updates paths if better ones are found

Example behavior:

- Avoids costly shortcuts and finds **optimal path**

Advantages

Greedy Best-First Search

- Faster in many cases
- Requires less computation
- Simple to implement

A* Search

- Complete and optimal
- Efficient when using good heuristics
- Widely applicable (navigation, robotics)

Limitations

Greedy Best-First Search

- Not optimal
- Can get stuck in **local minima**
- Heavily depends on heuristic accuracy

A* Search

- High **memory consumption**
- Slower than greedy in large spaces
- Performance depends on heuristic quality

When to Use Which?

- Use **Greedy Best-First Search** when:
  - Speed is more important than accuracy
  - Approximate solutions are acceptable
- Use **A*** when:
  - Optimal solution is required
  - Cost matters (e.g., shortest path, least time)

Key Insight

- **Greedy = "Go where it looks closest"**
- **A* = "Go where it is cheapest overall"**

Comparison of Uninformed (Uniform) Search Strategies

Uninformed search strategies explore the search space **without heuristic knowledge**.
The most common ones are:

- **Breadth-First Search (BFS)**
- **Depth-First Search (DFS)**
- **Depth-Limited Search (DLS)**
- **Bidirectional Search**

Evaluation Criteria

- **Completeness:** Will it find a solution if one exists?
- **Optimality:** Does it find the best (least-cost) solution?
- **Time Complexity:** Number of nodes generated
- **Space Complexity:** Memory required

Let:

- **b** = branching factor
- **d** = depth of shallowest goal
- **m** = maximum depth of tree
- **l** = depth limit

Comparison Table

Strategy	Completeness	Optimality	Time Complexity	Space Complexity
BFS	Yes (if b is finite)	Yes (for equal step cost)	O(bd)O(b^d)O(bd)	O(bd)O(b^d)O(bd)
DFS	No (fails in infinite depth)	No	O(bm)O(b^m)O(bm)	O(bm)O(bm)O(bm)
DLS	Yes (if l≥dl \ge dl≥d)	No	O(bl)O(b^l)O(bl)	O(bl)O(bl)O(bl)
Bidirectional	Yes	Yes (if BFS used in both directions)	O(bd/2)O(b^{d/2})O(bd/2)	O(bd/2)O(b^{d/2})O(bd/2)

Detailed Analysis

1. Breadth-First Search (BFS)

- **Completeness:** Guaranteed if branching factor is finite
- **Optimality:** Yes (finds shortest path in unweighted graphs)
- **Time & Space:** Very high, grows exponentially

Insight: Reliable but memory-intensive

2. Depth-First Search (DFS)

- **Completeness:** Not guaranteed (can go infinitely deep)
- **Optimality:** Not optimal
- **Time:** May explore entire tree
- **Space:** Very efficient

Insight: Good for memory-limited problems but risky

3. Depth-Limited Search (DLS)

- **Completeness:** Yes, if depth limit is sufficient
- **Optimality:** Not guaranteed
- **Time/Space:** Controlled by limit

Insight: Prevents infinite loops but may miss solution

4. Bidirectional Search

- **Completeness:** Yes
- **Optimality:** Yes (with BFS)
- **Time/Space:** Much lower than BFS

☞ **Insight:** Most efficient among uninformed methods, but harder to implement

Overall Comparison Insight

- **BFS:** Best when optimal shallow solution is needed
- **DFS:** Best when memory is limited
- **DLS:** Balanced control over DFS depth
- **Bidirectional:** Most efficient but requires reverse search capability

Memory-Bounded Heuristic Search Algorithms

Concept

Memory-bounded heuristic search algorithms are designed to **limit memory usage** while still using heuristics to guide the search.

They are especially useful when **A*** becomes impractical due to its **high space Complexity**.

Motivation:

- **A*** stores all generated nodes → memory grows as O(bd)O(b^d)O(bd)
- For large-scale problems, this quickly becomes infeasible

Major Memory-Bounded Algorithms

1. Iterative Deepening A* (IDA*)

- Combines **A*** with **iterative deepening**
- Uses a **threshold on f(n)** instead of depth

Working:

- Perform DFS with a cost limit (f-value)
- Increase limit iteratively until goal is found

Performance:

- **Completeness:** Yes
- **Optimality:** Yes (if heuristic is admissible)
- **Time:** Higher (repeated exploration)
- **Space:** Very low O(bd)O(bd)O(bd)

Best for: Large problems with limited memory

2. Recursive Best-First Search (RBFS)

- Uses recursion and keeps track of the **best alternative path**
- Stores only a **limited number of nodes**

Working:

- Expands best node recursively
- Backtracks when memory limit is reached

Performance:

- **Completeness:** Yes
- **Optimality:** Yes
- **Time:** May re-expand nodes
- **Space:** Linear O(bd)O(bd)O(bd)

Advantage: More memory-efficient than A*

Limitation: Repeated computations

3. Simplified Memory-Bounded A* (SMA*)

- Uses **all available memory**
- When memory is full:
  - Deletes worst nodes (highest f-value)
  - Stores backup values

Performance:

- **Completeness:** Yes (if memory sufficient)
- **Optimality:** Yes (if optimal path fits memory)
- **Time:** Efficient compared to IDA*
- **Space:** Limited by memory

Best balance between memory and optimality

Performance Evaluation

1. Memory Efficiency

- Major strength:
  - Use **linear or bounded memory** instead of exponential
- Suitable for:
  - Large graphs
  - Real-world navigation
  - Robotics

2. Time Complexity Trade-off

- Reduced memory → **increased computation time**
- Frequent **re-expansion of nodes**
- IDA* and RBFS may revisit states multiple times

3. Optimality and Completeness

- Maintain optimality **if heuristic is admissible**
- SMA* depends on available memory

4. Scalability

- Highly scalable compared to A*
- Can handle **very large state spaces**
- Still limited if branching factor is extremely high

Suitability for Large-Scale Problems

Highly Suitable When:

- Memory is limited (embedded systems, robotics)
- State space is very large
- Approximate or delayed solutions are acceptable

Applications:

- Route planning (GPS systems)
- Game AI (large search trees)
- Robotics path planning
- Puzzle solving (e.g., 8-puzzle, 15-puzzle)

Limitations in Large-Scale Problems

- Slower due to repeated exploration
- Performance depends heavily on heuristic quality
- SMA* may degrade if memory is too small

Local Search Algorithms in Optimization Problems

Concept

Local search algorithms are optimization techniques that **start with an initial solution and iteratively improve it** by making small, local changes.

- They **do not build full search trees**
- Operate on a **single current state**
- Aim to find the **best (optimal or near-optimal) solution**

Widely used for **large-scale optimization problems** where systematic search is infeasible.

How Local Search Works

1. Start with an **initial solution**
2. Evaluate neighboring solutions
3. Move to a **better neighbor**
4. Repeat until:
    - No improvement is possible, or
    - A satisfactory solution is found

Common Local Search Algorithms

1. Hill Climbing

- Moves to the **best neighboring state**
- Variants: Simple, Steepest-Ascent, Stochastic

Pros:

- Fast and simple
- Low memory usage

Cons:

- Gets stuck in:
    - **Local maxima**
    - **Plateaus**
    - **Ridges**

2. Simulated Annealing

- Allows occasional **worse moves** to escape local optima
- Controlled by a **temperature parameter**

Pros:

- Can reach global optimum
- Avoids local traps

Cons:

- Slow convergence
- Requires careful parameter tuning

3. Genetic Algorithms

- Inspired by **natural selection**

- Works with a **population of solutions**

Key operations:

- Selection
- Crossover
- Mutation

Pros:

- Explores large search spaces
- Good for complex optimization

Cons:

- Computationally expensive
- May converge slowly

Performance Evaluation

1. Efficiency

- Very **fast and memory-efficient**
- Suitable for **large-scale problems**

Only stores current state, not full paths

2. Optimality

- Often finds **near-optimal solutions**
- Not guaranteed to find global optimum (except with advanced methods like simulated annealing)

3. Scalability

- Highly scalable
- Works well with:
    - High-dimensional problems
    - Continuous optimization

4. Sensitivity to Initial State

- Strongly dependent on starting point
- Different initial states → different solutions

Strengths of Local Search

- ✓ Requires **very little memory**
- ✓ Efficient for **NP-hard problems**
- ✓ Easy to implement
- ✓ Works well in real-time systems

Limitations of Local Search

- ✗ Can get stuck in **local optima**
- ✗ No guarantee of **global optimality**

- ✕ Performance depends on **heuristic design**
- ✕ May cycle or stagnate

Applications in Real-World Optimization

- **Scheduling problems** (timetabling, job scheduling)
- **Route optimization** (Traveling Salesman Problem)
- **Machine learning** (parameter tuning)
- **Engineering design optimization**
- **Robotics and control systems**

Critical Analysis

When Local Search Works Well

- When the search space is **huge**
- When **approximate solutions are acceptable**
- When **time and memory are limited**

When It Struggles

- When the problem has many **local optima**
- When **exact optimal solutions are required**
- When the landscape is **flat or deceptive**

Hybrid Search Approach: Combining Heuristic Search and Local Search

Idea

A **hybrid search approach** integrates:

- **Heuristic (global) search** → guides exploration toward promising regions
- **Local search (optimization)** → refines solutions within those regions

Goal: **Balance exploration (global) + exploitation (local)** to efficiently solve complex optimization problems.

Why Hybridization is Needed

- Pure heuristic methods (like A*) → **optimal but memory-intensive**
- Pure local search → **fast but may get stuck in local optima**

Hybrid approach overcomes both limitations.

Proposed Hybrid Strategy

Step 1: Initial Heuristic Search (Global Guidance)

- Use algorithms like:
  - **A*** or **Greedy Best-First Search**
- Purpose:
  - Identify **promising regions of the search space**
  - Generate **good initial solutions**

Step 2: Local Search Optimization

- Apply:
  - **Hill Climbing** or **Simulated Annealing**
- Purpose:
  - Improve the solution locally
  - Reduce cost further

Step 3: Restart / Diversification

- If stuck in local optimum:
  - Restart from a new heuristic-guided state
  - Or use **randomization**

Step 4: Iterative Refinement

- Repeat:
  - Global search → Local improvement
- Keep best solution found

Algorithm Framework (Hybrid Search)

1. Initialize using heuristic search (A* / Greedy)
2. Generate initial solution S
3. Apply local search to improve S → S'
4. If S' is better, update best solution
5. If stuck:
   - Restart or explore new region using heuristic
6. Repeat until termination condition
7. Return best solution

Example: Route Optimization (Real-World)

Problem

Find the **shortest and fastest route** in a large city network.

Hybrid Solution

1. **Heuristic Phase:**
   - Use A* with straight-line distance → find a near-optimal route
2. **Local Search Phase:**
   - Apply local optimization:
     - Try alternate nearby roads
     - Reduce traffic delays
3. **Refinement:**
   - Re-route dynamically using updated heuristics

☞ Used in:

- GPS navigation systems
- Traffic-aware routing

Advantages of Hybrid Search

1. Improved Solution Quality

- Combines **global optimal guidance + local refinement**
- Produces **better solutions than either method alone**

2. Avoids Local Optima

- Local search limitations are reduced using:
  - Heuristic restarts
  - Diversification

3. Better Scalability

- Handles **large and complex search spaces**
- Reduces memory compared to pure A*

4. Faster Convergence

- Heuristics guide search quickly
- Local search fine-tunes efficiently

5. Flexibility

- Can combine multiple techniques:
  - A* + Simulated Annealing
  - Greedy + Genetic Algorithm

Limitations

- Increased algorithm complexity
- Requires **careful tuning** (heuristics + parameters)
- Performance depends on:
  - Quality of heuristic
  - Local search strategy

Applications

- **Vehicle routing problems (VRP)**
- **Traveling Salesman Problem (TSP)**
- **Robotics path planning**
- **Machine learning optimization (hyperparameter tuning)**
- **Scheduling and resource allocation**

UNIT 3

CONSTRAINT SATISFACTION PROBLEMS AND GAME THEORY

"AI techniques enable systems to learn, reason, and adapt."

—D. W. Patterson

Constraint Satisfaction Problems (CSPs) – Definition

A **Constraint Satisfaction Problem (CSP)** is a problem in Artificial Intelligence where the goal is to find values for a set of variables such that all given constraints are satisfied.
In simple terms:

Assign values to variables without violating any rules (constraints).

Formal Definition

A CSP is defined as a triple:
CSP=(X,D,C)\text{CSP} = (X, D, C)CSP=(X,D,C)
Where:

- **X** → Set of variables {X1,X2,...,Xn}\{X_1, X_2, ..., X_n\}{X1,X2,...,Xn}
- **D** → Set of domains (possible values for each variable)
- **C** → Set of constraints (rules restricting values)

Key Components of CSP

1. Variables

These are the unknowns that need to be assigned values.
Example:

- In map coloring: Regions like {A, B, C}
- In Sudoku: Each cell is a variable

2. Domains

Each variable has a domain (set of possible values it can take).
Example:

- Map coloring: {Red, Green, Blue}
- Sudoku: {1, 2, 3, ..., 9}

3. Constraints

Constraints define restrictions on the values that variables can take.
Types of Constraints:

- **Unary Constraint** → Applies to one variable
- **Binary Constraint** → Between two variables
- **Global Constraint** → Involves many variables

Example:

- Map coloring: Adjacent regions must not have the same color
- Sudoku: No repeated numbers in a row/column

Example 1: Map Coloring Problem

Problem:
Color a map such that no adjacent regions have the same color.
Components:

- **Variables**: A, B, C
- **Domain**: {Red, Green, Blue}
- **Constraints**:
  - A ≠ B
  - B ≠ C
  - A ≠ C

Valid Solution:

- A = Red
- B = Green
- C = Blue

Example 2: Sudoku

Components:

- **Variables**: Each cell in the grid
- **Domain**: {1–9}
- **Constraints**:
    - No repeated number in any row
    - No repeated number in any column
    - No repeated number in any 3×3 box

Goal:

Fill the grid satisfying all constraints.

Example 3: N-Queens Problem

Problem:

Place N queens on an N×NN \times NN×N chessboard so that no queen attacks another.

Components:

- **Variables**: Each queen (or each row)
- **Domain**: Column positions
- **Constraints**:
    - No two queens in same column
    - No two queens on same diagonal

Adversarial Search – Definition

Adversarial search is a search technique used in Artificial Intelligence for decision-making in environments where multiple agents compete against each other.

It is mainly used in **games**, where one player's gain is another player's loss (zero-sum scenarios).
In simple terms:
An agent chooses actions while considering the possible counter-moves of an opponent.

Key Idea

Unlike standard search problems (like pathfinding), adversarial search involves:

- **Multiple players (agents)**
- **Conflicting goals**
- **Strategic decision-making**

Role in Game-Playing AI Systems

Adversarial search plays a **central role** in designing intelligent game-playing systems.

1. Modeling Competitive Environments

Game-playing AI systems model games as:

- **States** → Current configuration of the game
- **Actions** → Possible moves
- **Players** → Usually MAX (AI) vs MIN (opponent)

Example:

- In chess: Board positions represent states
- Moves like pawn or knight movement represent actions

2. Decision Making Using Game Trees

AI constructs a **game tree**:

- Nodes → Game states
- Edges → Moves
- Leaves → Final outcomes (win/loss/draw)

The AI explores this tree to choose the best move.

3. Minimax Algorithm (Core of Adversarial Search)

The most fundamental algorithm used is **Minimax**:

- **MAX player (AI)** → Tries to maximize score
- **MIN player (Opponent)** → Tries to minimize score

The AI assumes the opponent plays optimally.

Working:

1. Expand game tree
2. Evaluate terminal states
3. Backpropagate values
4. Choose the optimal move

4. Alpha-Beta Pruning (Optimization)

To improve efficiency:

- **Alpha-Beta pruning** eliminates branches that won't affect the final decision
- Reduces computation time significantly

5. Handling Complex Games

In real-world games:

- Full search is impossible due to huge state space
- AI uses:
    - **Heuristic evaluation functions**
    - **Depth-limited search**

Example:

- Chess AI evaluates board positions instead of searching all moves

Examples of Adversarial Search in Games

1. Chess
    - AI predicts opponent strategies
    - Uses Minimax + heuristics
2. Tic-Tac-Toe
    - Simple example of adversarial search
    - Full game tree can be explored
3. Checkers / Go
    - Requires advanced pruning and heuristics

Characteristics of Adversarial Search Problems

- Multi-agent environment
- Competitive (zero-sum)
- Sequential decision-making
- Requires prediction of opponent behaviour

Local Search Techniques for Constraint Satisfaction Problems (CSPs)

Local search methods solve CSPs by starting with a *complete assignment* (all variables assigned, possibly violating constraints) and then **iteratively improving** it by reducing conflicts.

Instead of building a solution step-by-step, they **modify an existing solution** until all constraints are satisfied (or nearly satisfied).

Key Idea of Local Search in CSP

- Start with a **random assignment**
- Count how many constraints are violated
- Move to a "better" neighboring assignment
- Repeat until:
  - A valid solution is found ✓
  - Or no improvement is possible ✗

Common Local Search Techniques

1. Hill Climbing

- Chooses the neighbor with the **fewest conflicts**
- Moves "uphill" toward better solutions

Example:

- In N-Queens, move a queen to reduce attacks

Limitation:

- Can get stuck in:
  - Local maxima
  - Plateaus

2. Min-Conflicts Algorithm (Most Popular for CSPs)

- Select a variable involved in conflicts
- Assign it a value that **minimizes conflicts**

📌 Example:

- In N-Queens:
  - Pick a conflicting queen
  - Move it to the column with minimum conflicts

☞ Very efficient for large problems like N-Queens (even millions of queens!)

3. Simulated Annealing

- Sometimes accepts worse moves to escape local optima
- Inspired by cooling process in metallurgy

✓ Helps avoid getting stuck

✗ Slower than greedy methods

4. Random Restart

- Run local search multiple times with different initial states

✓ Increases probability of finding global solution

Example: N-Queens Using Min-Conflicts

- **Variables**: Queens (one per row)
- **Domain**: Column positions
- **Goal**: No two queens attack each other

Steps:

1. Place queens randomly
2. Pick a queen in conflict
3. Move it to position with fewer conflicts
4. Repeat until solved

Advantages of Local Search

✓ Uses **very little memory**

✓ Works well for **large-scale CSPs**

✓ Often finds solutions **quickly in practice**

Limitations

✗ Not guaranteed to find a solution

✗ May get stuck in local optima

✗ Cannot prove optimality or completeness easily

Minimax Algorithm – Description

The **Minimax algorithm** is a fundamental technique in **adversarial search** used by AI to make optimal decisions in **two-player, zero-sum games**.

It assumes:

- One player (**MAX**) tries to **maximize** the score
- The other player (**MIN**) tries to **minimize** the score
- Both players play **optimally**

Core Idea

Minimax explores all possible future moves (game tree) and selects the move that leads to the **best worst-case outcome**.

✓ MAX chooses the move with the **maximum value**

✓ MIN chooses the move with the **minimum value**

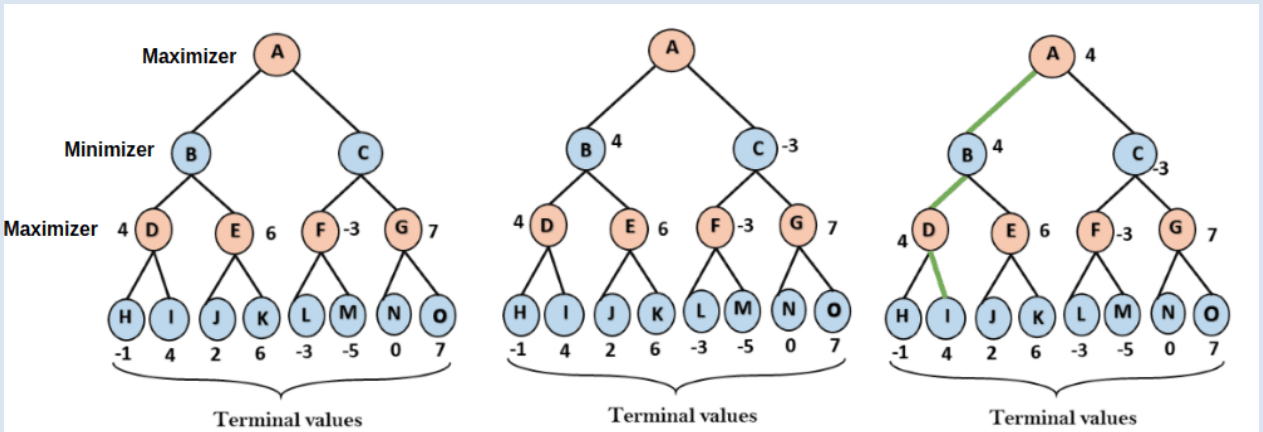
How Minimax Works

Step-by-Step Process

1. **Generate the Game Tree**
    - Nodes → Game states
    - Edges → Possible moves
2. **Assign Utility Values**
    - Terminal states are evaluated:
        - Win → +1
        - Loss → −1
        - Draw → 0
3. **Backpropagation**
    - Values are propagated upward:
        - MAX picks the **maximum** value
        - MIN picks the **minimum** value
4. **Choose Optimal Move**
    - The root node selects the move with the best value

Illustrative Example

Game Tree Evaluation



Explanation:

- Leaf nodes: values (e.g., 3, 5, 2, 9)
- MIN level: chooses minimum of children
- MAX level: chooses maximum of children

Final decision is the move leading to the **highest guaranteed value**

Mathematical Representation

Minimax(s)={Utility(s),if s is terminalmax⁡a∈Actions(s)Minimax(Result(s,a)),if MAX's turnmin⁡a∈Actions(s)Minimax(Result(s,a)),if MIN's turn\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{if } s \text{ is terminal} \\ \max_{a \in Actions(s)} \text{Minimax(Result(s,a))}, & \text{if MAX's turn} \\ \min_{a \in Actions(s)} \text{Minimax(Result(s,a))}, & \text{if MIN's turn} \end{cases}Minimax(s)= ⎩⎨⎧Utility(s),maxa∈Actions(s)Minimax(Result(s,a)),mina∈Actions(s)Minimax(Result(s,a)),if s is terminalif MAX's turnif MIN's turn

Example: Tic-Tac-Toe

- AI evaluates all possible moves
- Assumes opponent will block or counter optimally
- Chooses move that ensures:
  - Win if possible
  - Otherwise, at least a draw

How It Determines Optimal Decisions

Minimax ensures optimal decisions by:

1. Considering Opponent Moves

- It simulates **both players' actions**
- Assumes opponent is rational and optimal

2. Exploring Future Outcomes

- Looks ahead into future states
- Evaluates consequences of each move

3. Choosing Best Worst-Case Outcome

- Selects move with:
  - **Maximum payoff under worst-case opponent response**

This is called the **maximin strategy**

4. Guaranteeing Optimal Play

- If the entire game tree is explored:

  ✓ The decision is **optimal**

  ✓ No better move exists

Limitations

✗ High time complexity: O(bd)O(b^d)O(bd)

- bbb = branching factor
- ddd = depth

✗ Not feasible for large games (e.g., chess)

Alpha–Beta Pruning – Demonstration

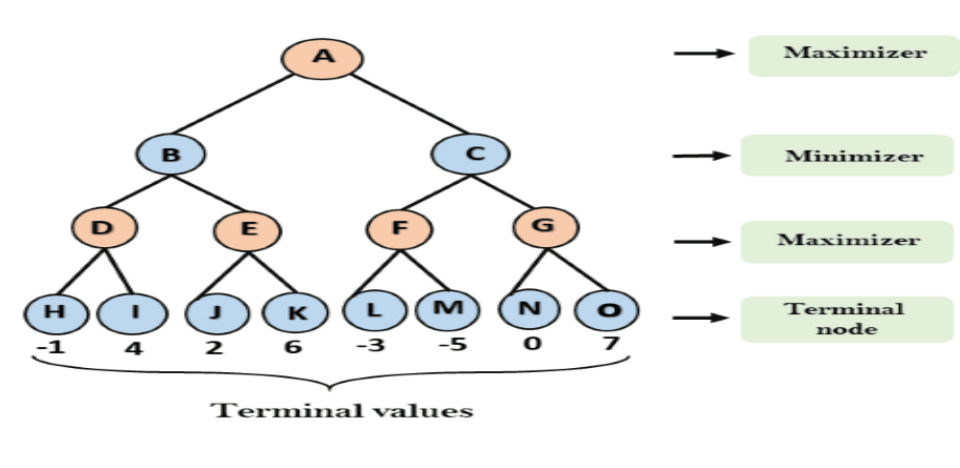
Alpha–Beta pruning is an optimization of the **Minimax algorithm** that avoids exploring parts of the game tree that **cannot influence the final decision**.
It keeps track of two values:

- **Alpha (α)** → Best value found so far for **MAX**
- **Beta (β)** → Best value found so far for **MIN**

Key Pruning Condition

- **If α ≥ β → Stop exploring (Prune the branch)**

Example Game Tree



Step-by-Step Demonstration

Consider a 3-level tree:

Level 0 (Root – MAX)

- Goal: Maximize value

Level 1 (MIN nodes)

- Goal: Minimize value

Level 2 (Leaf nodes)

- Terminal values:
  Example values → **3, 5, 6, 9, 1, 2, 0, -1**

Step 1: Evaluate Left Subtree

- First MIN node:
  - Evaluate leaves: **3, 5**
  - MIN chooses → **3**

☞ Update at root (MAX):

- α = max(-∞, 3) = **3**

Step 2: Move to Next Subtree

- Second MIN node:
  - First leaf = **6**
  - β = 6

Now compare:

- α = 3, β = 6 → continue
- Next leaf = **9**
  - MIN chooses → **6**

Root updates: α = max(3, 6) = **6**

Step 3: Third Subtree (Pruning Happens)

- Third MIN node:
  - First leaf = **1**
  - β = 1

Now check:

- α = 6, β = 1
- Since **α ≥ β → PRUNE**

Remaining leaves (**2, 0, -1**) are **NOT evaluated**

Final Decision

- Root (MAX) chooses the best value = **6**

Node Evaluation Comparison

Method	Nodes Evaluated
Minimax (no pruning)	8 leaf nodes
Alpha-Beta Pruning	5 leaf nodes

Saved evaluations: 3 nodes (≈ 37.5% reduction)

Why Pruning Works

- If a move is already worse than a previously explored option,
  ☞ it will **never be chosen**

So:

✓ No need to explore further

✓ Saves computation time

Applying the Minimax Algorithm – Worked Example

Let's solve a simple **two-player game tree** step by step using **Minimax**.

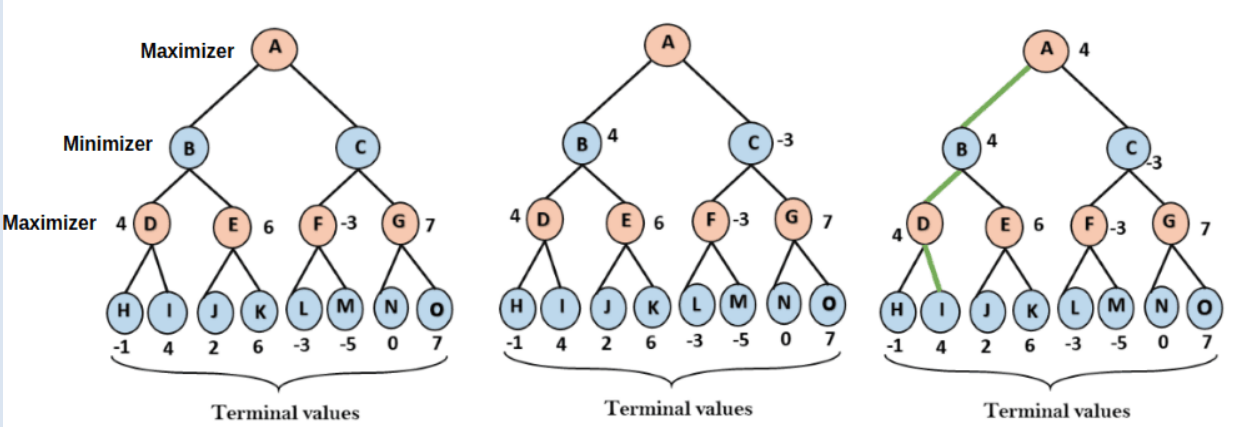
Game Tree Structure

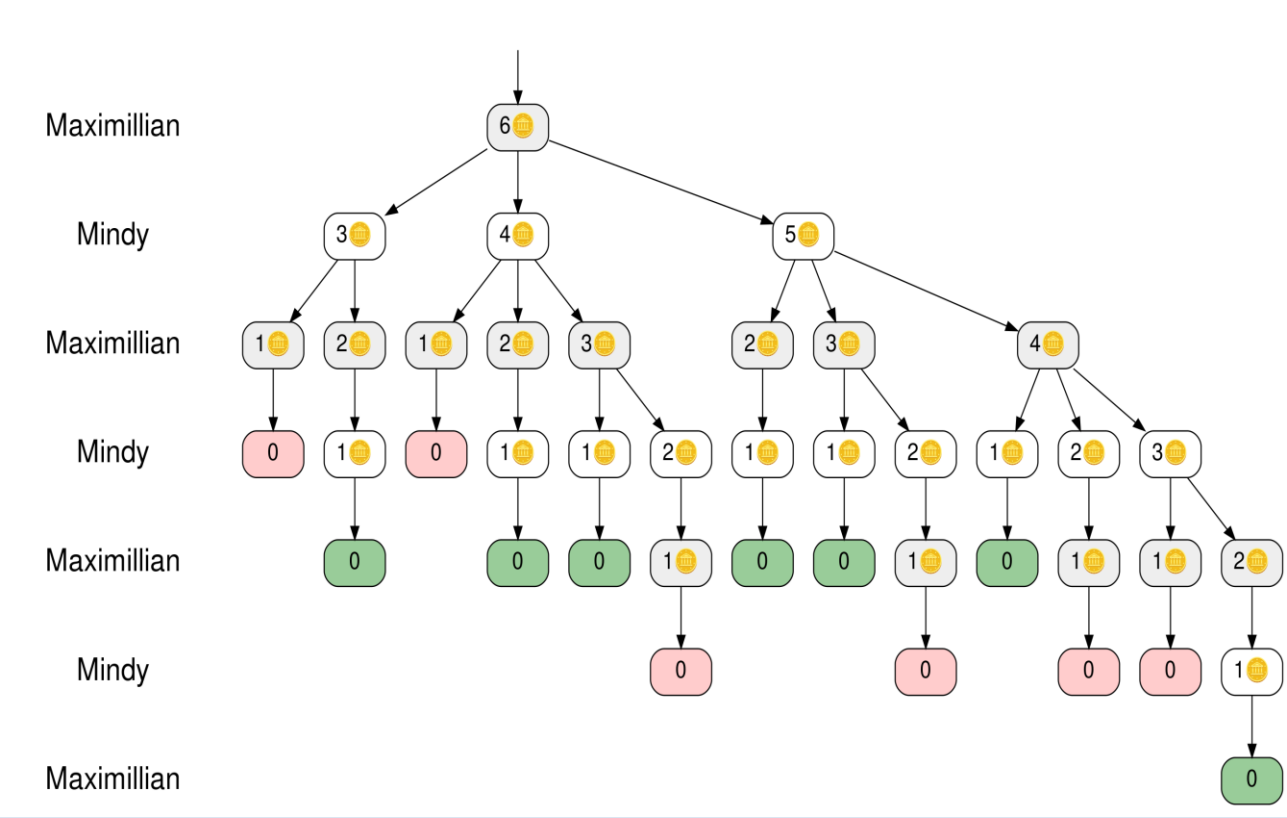
- **Root (MAX)** → AI's turn
- **Level 1 (MIN)** → Opponent's turn
- **Level 2 (Leaves)** → Terminal states with values

Leaf values:

```
      MAX
     /   \
   MIN    MIN
  / \    / \
 3   5  2   9
```

Visual Representation





Step-by-Step Minimax Evaluation

Step 1: Evaluate Leaf Nodes

Leaf values are given:

- Left subtree → (3, 5)
- Right subtree → (2, 9)

Step 2: MIN Level (Opponent's Turn)

MIN chooses the **minimum** value in each subtree:

- Left MIN node → min(3, 5) = **3**
- Right MIN node → min(2, 9) = **2**

Step 3: MAX Level (AI's Turn)

MAX chooses the **maximum** among MIN results:

- max(3, 2) = **3**

Final Decision

Optimal value = 3

Optimal move = Choose the LEFT subtree

Explanation

- If MAX goes left:
  - Opponent forces outcome = 3
- If MAX goes right:
  - Opponent forces outcome = 2

Since MAX assumes the opponent plays optimally,
it selects the move that gives the **best worst-case outcome**.

Key Insight

✓ Minimax ensures:

- Worst-case scenario is considered
- Decision is **optimal under perfect play**

Iterative Deepening in Adversarial Search

Iterative Deepening (ID) is a search strategy where the AI runs **depth-limited Minimax repeatedly**, increasing the depth limit step by step:
Depth=1 → 2 → 3 → ⋯\text{Depth} = 1 \;\rightarrow\; 2 \;\rightarrow\; 3 \;\rightarrow\; \cdotsDepth=1→2→3→⋯
At each iteration, the algorithm computes the **best move so far** using Minimax (often with **alpha–beta pruning**).

How It Works

1. Start with a shallow depth (e.g., 1 ply)
2. Run Minimax → get a move
3. Increase depth (2, 3, …)
4. Re-run search with deeper lookahead
5. Stop when:
    - Time runs out
    - Maximum depth reached

The **last completed iteration** provides the decision.

Why Iterative Deepening is Important

1. Anytime Decision-Making (Critical Under Time Constraints)

- ID is an **anytime algorithm**:
    - It can return a valid move **at any moment**
    - The move improves as more time is available

In real-time games (like chess):

- If time expires suddenly, AI still has the **best move from the last iteration**

2. Better Move Ordering (Boosts Alpha–Beta Pruning)

- Earlier shallow searches identify **promising moves**
- These moves are explored first in deeper searches

This improves pruning efficiency:

- More branches are cut earlier
- Search becomes much faster

3. Progressive Refinement of Decisions

Each iteration refines the evaluation:

- Depth 1 → Quick, rough decision
- Depth 3 → Better strategic insight
- Depth 5+ → More accurate prediction

Decisions become **increasingly reliable**

4. Efficient Use of Time

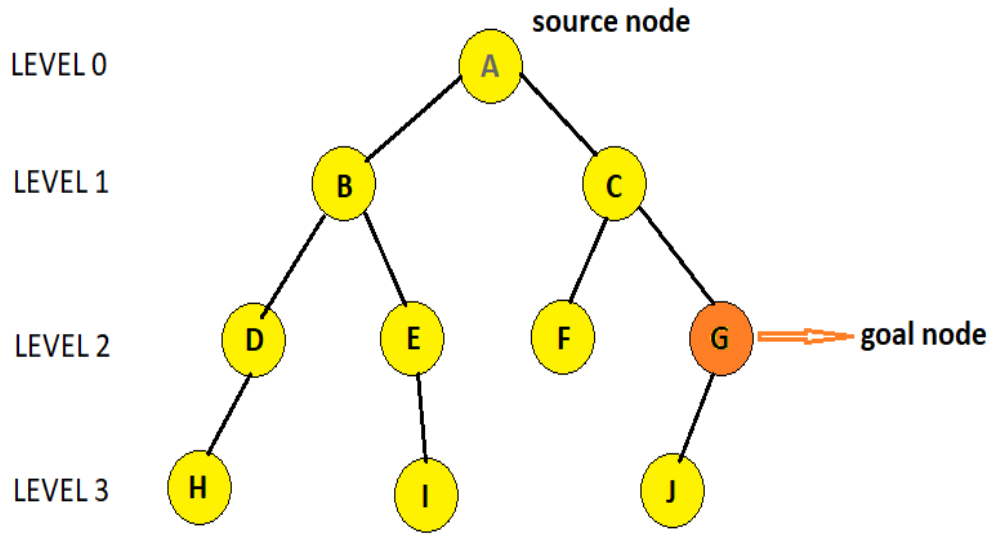
Instead of attempting one deep search (which may fail midway), ID:

- Uses time **incrementally**
- Avoids incomplete deep searches

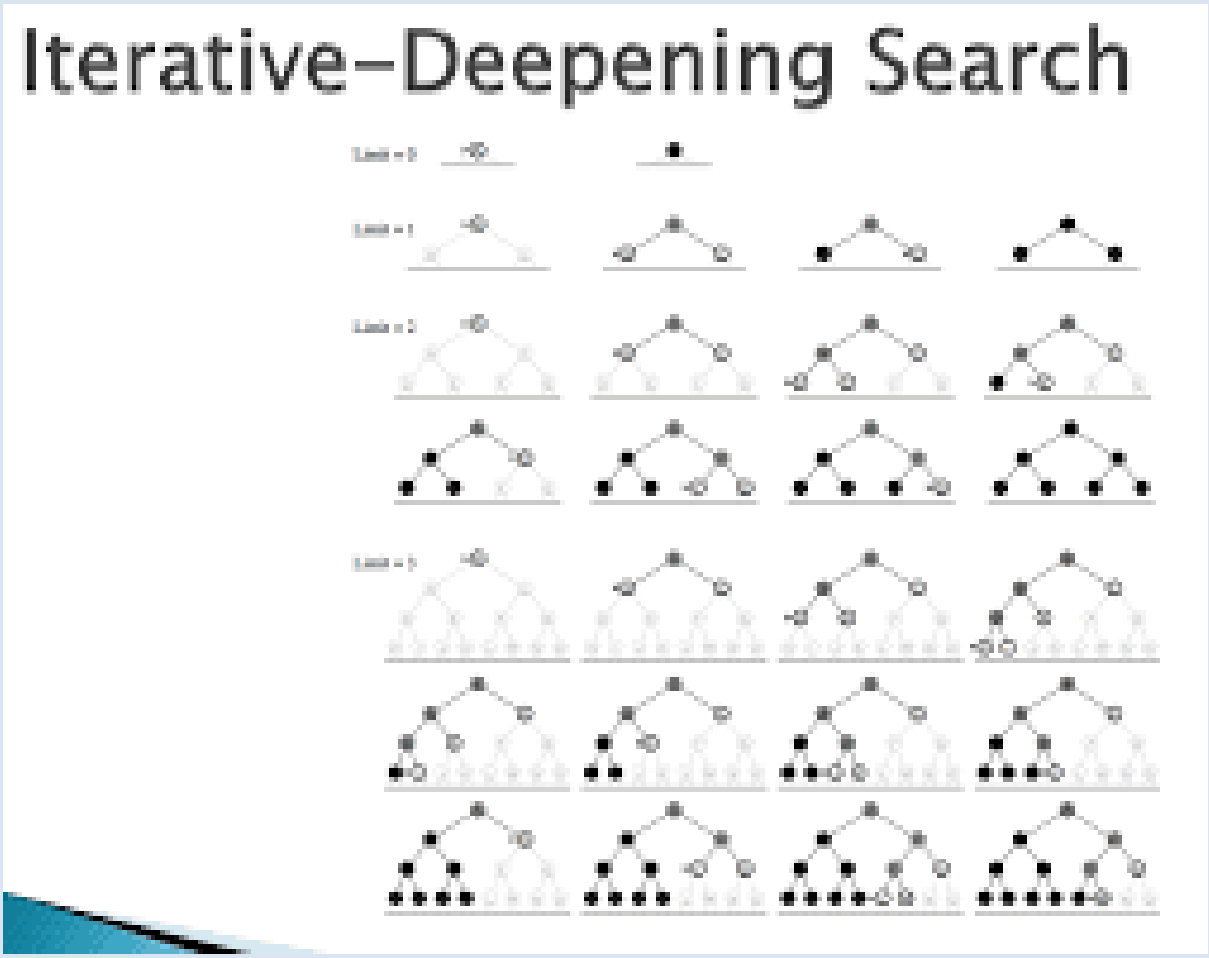
✓ Guarantees a usable result

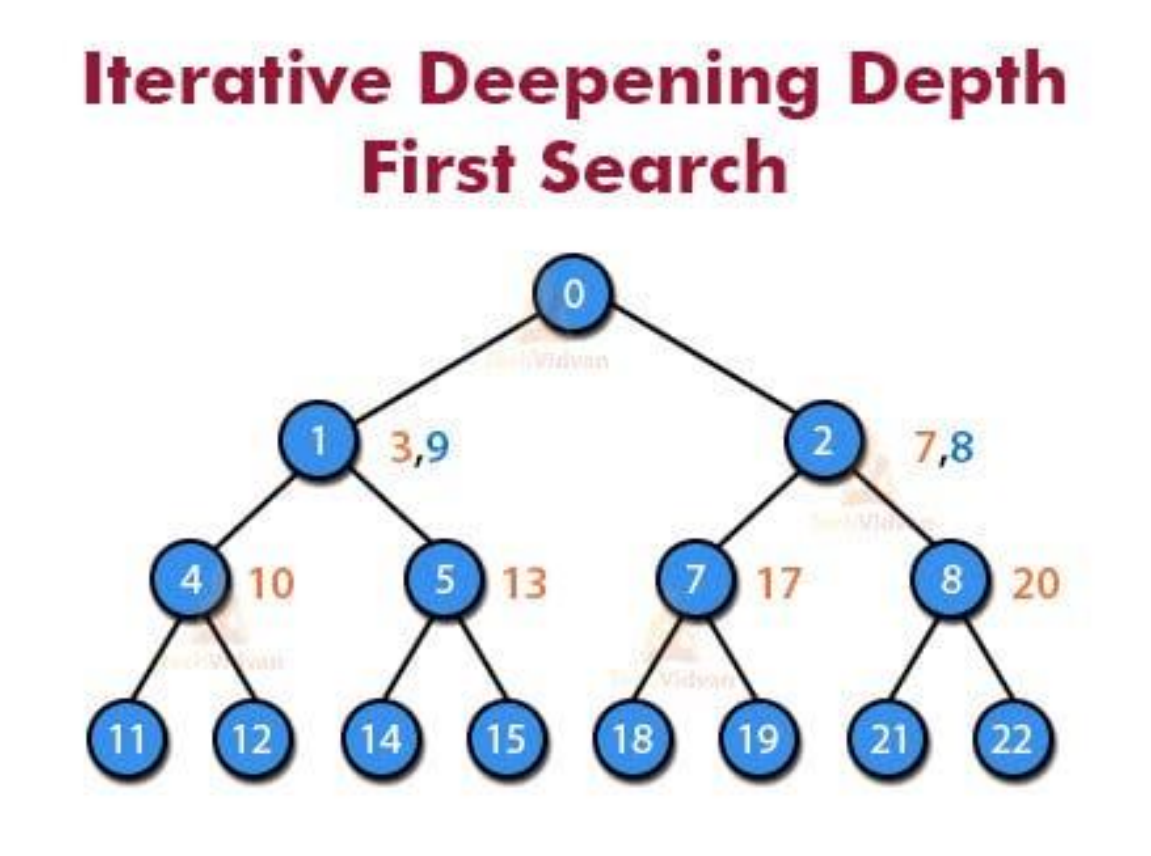
✓ Avoids wasted computation

Illustrative Concept



IDDFS with max depth-limit = 3
Note that iteration terminates at depth-limit=2
Iteration 0: A
Iteration 1: A->B->C
Iteration 2: A->B->D->E->C->F->G





Example in Game-Playing AI

In a chess engine:

- Iteration 1 → Evaluate immediate moves
- Iteration 2 → Consider opponent responses
- Iteration 3 → Look ahead further

If time runs out at depth 3:

☞ Use the best move found at depth 3

Performance Analysis of Minimax vs Alpha–Beta Pruning

Both **Minimax** and **Alpha–Beta pruning** are core algorithms in adversarial search. They produce the **same optimal decision**, but differ significantly in **efficiency**.

1. Minimax Algorithm – Time Complexity

Complexity

O(bd)O(b^d)O(bd)

Where:

- bbb = branching factor (average number of moves per state)
- ddd = depth of the game tree

Analysis

- Explores **every node** in the game tree
- No pruning → evaluates all possible outcomes
- Time grows **exponentially**

Example:

- Chess: b≈35b \approx 35b≈35, d=6d = 6d=6
- Nodes explored ≈ 356≈1.835^6 \approx 1.8356≈1.8 billion

Impractical for large games

2. Alpha–Beta Pruning – Time Complexity

Best Case

O(bd/2)O(b^{d/2})O(bd/2)

Worst Case

O(bd)O(b^d)O(bd)

Analysis

- Avoids exploring branches that **cannot affect the decision**
- Performance depends on **move ordering**

Cases Explained

Best Case (Perfect Ordering)

- Best moves explored first
- Maximum pruning occurs

Effective depth is **halved**

Example:

- Instead of bdb^dbd, complexity becomes bd/2b^{d/2}bd/2
- Huge reduction in computation

Worst Case (Poor Ordering)

- Worst moves explored first
- No pruning occurs

Same as Minimax: O(bd)O(b^d)O(bd)

Average Case

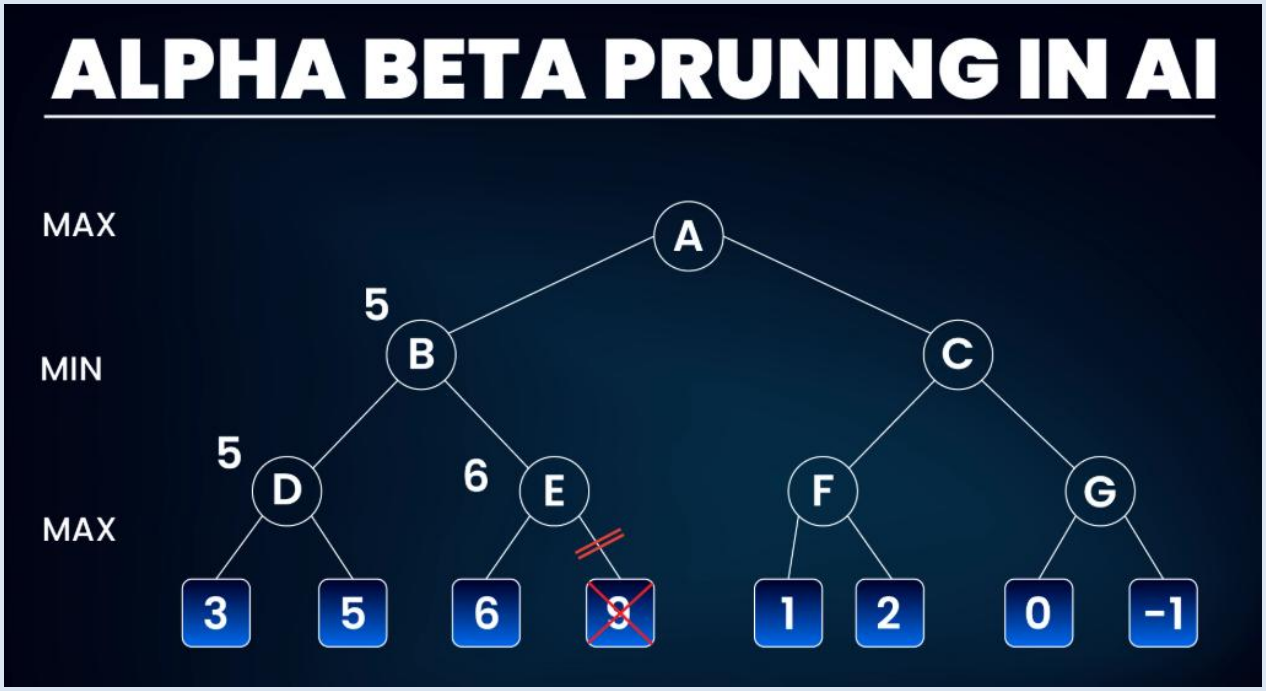
- Better than Minimax but not optimal
- Depends on heuristic quality

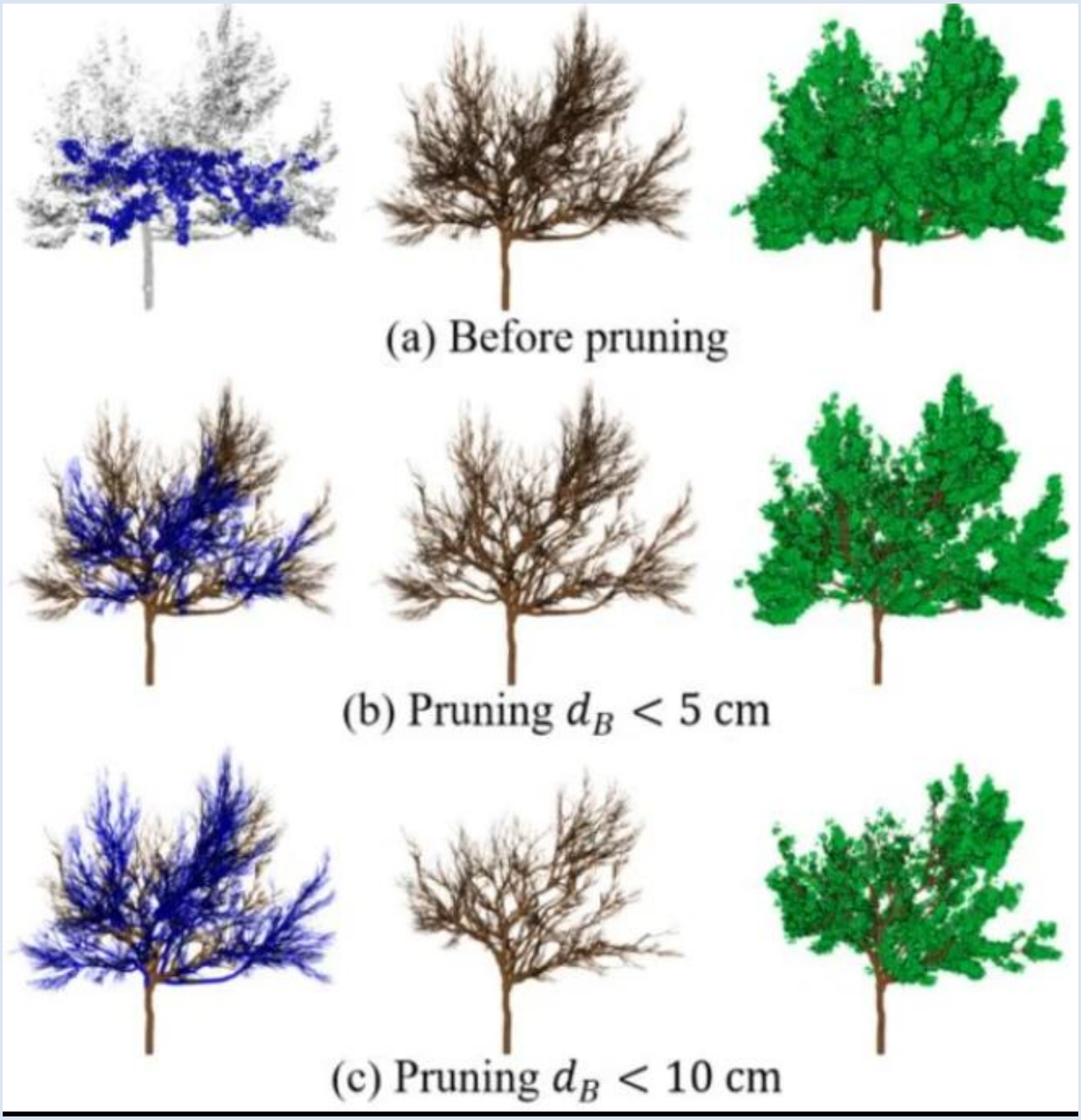
3. Visual Intuition

Minimax with Alpha Beta Pruning

- The **minimax algorithm** is a way of finding an optimal move in a two player game. ***Alpha-beta pruning*** is a way of finding the optimal minimax solution while avoiding searching subtrees of moves which won't be selected.
- In the search tree for a two-player game, there are two kinds of nodes, nodes representing ***your*** moves and nodes representing ***your opponent's*** moves.

1





(a) Before pruning

(b) Pruning $d_B < 5$ cm

(c) Pruning $d_B < 10$ cm

Strategies in Game-Playing AI – Evaluation

Game-playing AI systems use a combination of search and learning strategies to make decisions in competitive environments. Below is a clear evaluation of the major approaches.

1. Minimax Search

Idea: Explore all possible moves assuming an optimal opponent.

✓ Strengths

- Guarantees **optimal decision** (if full tree explored)
- Simple and foundational

Weaknesses

- **Exponential time complexity** O(bd)O(b^d)O(bd)
- Not practical for complex games (e.g., chess, Go)

2. Alpha–Beta Pruning

Idea: Optimize Minimax by pruning unnecessary branches.

✓ Strengths

- Same result as Minimax but **much faster**
- Can search deeper levels
- Widely used in real systems

Weaknesses

- Performance depends on **move ordering**
- Still limited for extremely large state spaces

3. Heuristic Evaluation Functions

Idea: Estimate the value of non-terminal states.

✓ Strengths

- Enables **depth-limited search**
- Makes large problems tractable

Weaknesses

- Quality depends on **heuristic accuracy**
- May lead to suboptimal decisions

Example:

- Chess: piece values, board control

4. Iterative Deepening

Idea: Repeatedly apply depth-limited search with increasing depth.

✓ Strengths

- Works under **time constraints**
- Improves move ordering → better pruning

✗ Weaknesses

- Repeats computations (though manageable)

5. Monte Carlo Tree Search (MCTS)

Idea: Use random simulations to evaluate moves.

✓ Strengths

- Handles **huge search spaces**
- Does not require strong heuristics
- Balances exploration vs exploitation

Weaknesses

- Results are **probabilistic**
- Needs many simulations for accuracy

Used in:

- Go-playing AI (e.g., AlphaGo)

6. Reinforcement Learning (RL) & Deep Learning

Idea: Learn strategies from experience using rewards.

✓ Strengths

- Learns **complex patterns automatically**
- Can surpass human-level performance
- Adapts over time

Weaknesses

- Requires **large training data & computation**
- Hard to interpret decisions

7. Hybrid Approaches (Modern AI)

Idea: Combine multiple techniques:

- Alpha–Beta + Heuristics
- MCTS + Neural Networks

✓ Strengths

- Leverages strengths of different methods
- Achieves **state-of-the-art performance**

Local Search Methods for CSPs – Critical Evaluation

Local search methods (e.g., **Hill Climbing, Min-Conflicts, Simulated Annealing, Random Restart**) approach a Constraint Satisfaction Problem by **starting with a complete assignment** and **iteratively reducing constraint violations** rather than constructing solutions step-by-step.

How They Work (Recap)

- Begin with a **random complete assignment**
- Evaluate number of **conflicts (violated constraints)**
- Move to a **neighboring assignment** with fewer conflicts
- Repeat until:
  - A valid solution is found ✓
  - Or no improvement is possible ✗

Advantages of Local Search Methods

1. High Scalability for Large Problems

- Can handle **millions of variables** (e.g., large N-Queens)
- Do not require storing the full search tree

Ideal for **large-scale real-world CSPs**

2. Low Memory Consumption

- Only maintains the **current state**, not the entire search space

✓ Space complexity is very low compared to backtracking

3. Fast in Practice

- Often finds solutions **quickly** even for complex problems
- Particularly effective with **Min-Conflicts heuristic**

4. Simplicity and Flexibility

- Easy to implement
- Can be adapted with:
  - Randomization
  - Heuristics
  - Restart strategies

5. Robust for Approximate Solutions

- Can produce **good enough solutions** when exact solutions are unnecessary

Useful in:

- Scheduling
- Resource allocation
- Network optimization

Limitations of Local Search Methods

1. Incompleteness

- No guarantee of finding a solution even if one exists

May get stuck in:

- Local optima
- Plateaus
- Ridges

2. No Guarantee of Optimality

- Finds **a solution**, not necessarily the best one

3. Sensitive to Initial State

- Poor starting points may lead to poor performance

4. Difficulty with Highly Constrained Problems

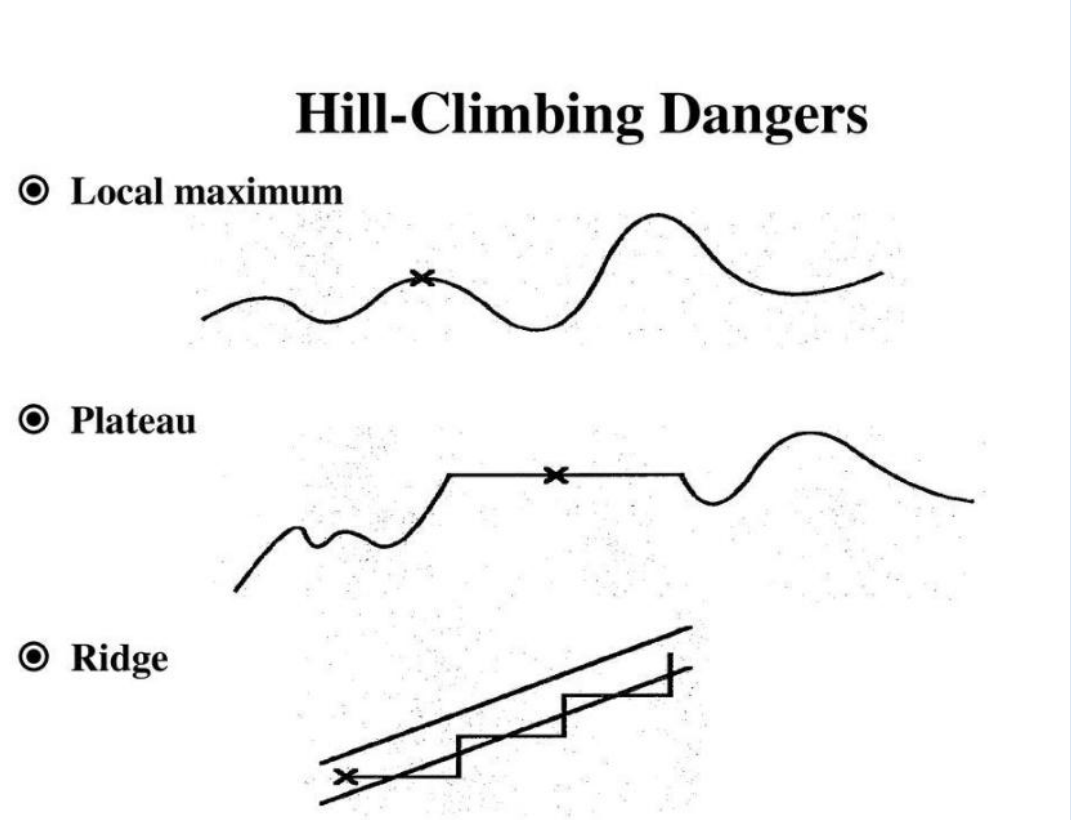
- When constraints are tight:
  - Fewer valid solutions
  - Harder to escape conflicts

5. Lack of Systematic Exploration

- Does not explore entire search space
- Cannot prove:

  ✓ Completeness

  ✓ Optimality



CSP Model for a Real-World Problem: University Timetable Scheduling

Let's model a practical and commonly studied problem as a **Constraint Satisfaction Problem (CSP)** and solve it using a **local search strategy**.

1. Problem Description

A university needs to schedule:

- Multiple **courses**
- Limited **time slots**
- Limited **classrooms**
- Assigned **faculty**

Create a timetable with **no conflicts**

2. CSP Formulation

(a) Variables (X)

Each lecture/session is a variable:

X={C1,C2,C3,...,Cn}X = \{C_1, C_2, C_3, ..., C_n\}X={C1,C2,C3,...,Cn}

Where each CiC_iCi represents a **course session**

(b) Domains (D)

Each variable takes a value:

Di={(TimeSlot,Room)}D_i = \{(TimeSlot, Room)\}Di={(TimeSlot,Room)}

Example:

- (Monday 9AM, Room 101)
- (Tuesday 11AM, Room 202)

(c) Constraints (C)

Hard Constraints (Must be satisfied)

1. **No Room Conflict**
    - No two classes in the same room at the same time
2. **No Faculty Clash**
    - A faculty cannot teach two classes simultaneously
3. **No Student Clash**
    - Students enrolled in multiple courses should not have overlapping classes

Soft Constraints (Preferences)

4. Prefer **morning slots**
5. Avoid **back-to-back classes**
6. Spread classes evenly

3. Objective Function (for Local Search)

Convert CSP into an optimization problem:

Cost=Number of constraint violations\text{Cost} = \text{Number of constraint violations}Cost=Number of constraint violations

Goal: **Minimize conflicts (cost = 0 ideally)**

4. Suitable Local Search Strategy

✅ **Min-Conflicts Algorithm**

Why Min-Conflicts? (Justification)

1. Highly Effective for Scheduling Problems

- Timetabling has:
  - Large variable space
  - Structured constraints

☞ Min-conflicts performs extremely well in such domains

2. Fast Convergence

- Quickly reduces conflicts by:
  - Picking a conflicting variable
  - Assigning value with minimum conflicts

3. Scales to Large Systems

- Works efficiently even with:
  - Hundreds of courses
  - Thousands of constraints

4. Simple and Practical

- Easy to implement
- Widely used in:
  - School timetables
  - Exam scheduling

5. Algorithm Steps

1. Start with a **random timetable**
2. Repeat until solution found:
   - Select a **conflicting class**
   - Assign it a **time/room with minimum conflicts**
3. If stuck:
   - Use **random restart**

6. Example Scenario

Initial assignment:

Course	Time	Room
C1	Mon 9AM	R1
C2	Mon 9AM	R1 ❌ (conflict)

Conflict detected (same room)

Fix using Min-Conflicts:

- Move C2 → Mon 11AM, R2 ✓

7. Enhancements

To improve performance:

- **Random Restart** → Avoid local optima
- **Weighted Constraints** → Prioritize hard constraints
- **Heuristic Initialization** → Better starting point

8. Critical Evaluation of the Approach

Advantages

✓ Handles **large-scale scheduling problems**

✓ Produces solutions **quickly**

✓ Low memory usage

Limitations

✗ No guarantee of optimal solution

✗ May struggle with very tight constraints

9. Conclusion

The **University Timetabling Problem** can be effectively modeled as a CSP using:

- Variables → Course sessions
- Domains → Time-slot & room pairs
- Constraints → Resource and scheduling rules

Min-Conflicts local search is the best choice because it:

- Efficiently handles large, structured problems
- Quickly minimizes conflicts
- Is widely proven in real-world scheduling systems

Final Insight

✓ CSP provides a **structured modelling framework**

✓ Local search (Min-Conflicts) provides a **practical solving strategy**

UNIT 4

KNOWLEDGE &REASONING: STATISTICAL REASONING

"Knowledge acquisition is the bottleneck in building expert systems"

—Giarratano & Riley

Significance in statistical reasoning

1. Definition of Probability

Probability is a measure of the likelihood that a particular event will occur. It ranges from **0 to 1**, where:

- **0** → the event will not occur
- **1** → the event will certainly occur

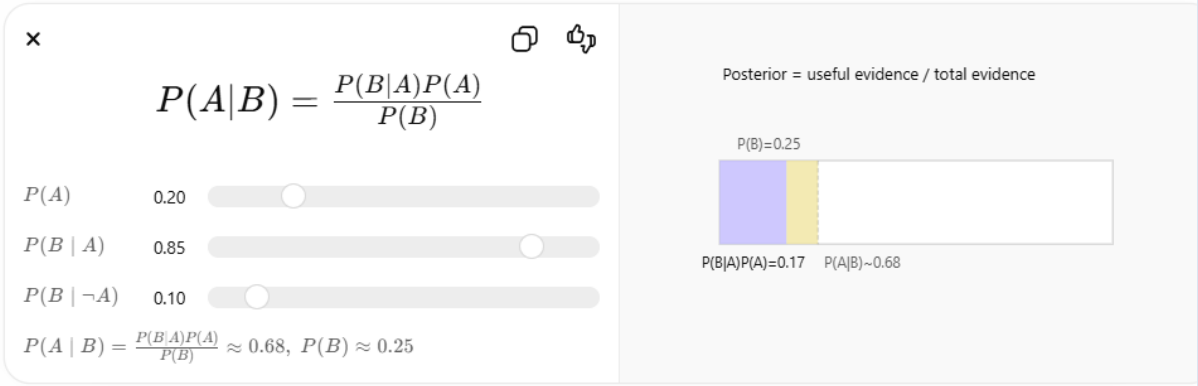
For an event *A*, probability is defined as:

$$P(A) = \frac{\text{Number of favorable outcomes}}{\text{Total number of possible outcomes}}$$

It provides a mathematical framework to quantify uncertainty and randomness.

2. Bayes' Theorem

Bayes' Theorem is a fundamental result in probability that allows us to update the probability of an event based on new evidence.



Where:

- $P(A|B)$: Posterior probability (probability of A given B)
- $P(B|A)$: Likelihood (probability of B given A)
- $P(A)$: Prior probability (initial belief about A)
- $P(B)$: Marginal probability of B

3. Significance in Statistical Reasoning

a) Handling Uncertainty

- Probability helps in analyzing uncertain events (e.g., weather forecasting, risk analysis).
- It forms the basis for all statistical inference methods.

b) Decision Making

- Used in making rational decisions under uncertainty (e.g., quality control, finance, engineering systems).

c) Updating Beliefs (Bayesian Thinking)

- Bayes' theorem allows revising predictions when new data becomes available.
- Widely used in fields like medical diagnosis, machine learning, and robotics.

d) Data Interpretation

- Helps interpret experimental and survey data by quantifying variability and reliability.

e) Real-world Applications

- **Medical diagnosis:** Estimating disease probability given test results
- **Spam filtering:** Classifying emails based on learned probabilities
- **Industrial automation:** Fault detection and predictive maintenance

Role in rule-based systems.

Certainty Factors (CFs) – Definition

Certainty Factors are numerical values used to represent the **degree of belief or confidence** in a hypothesis or conclusion under conditions of uncertainty.

- They typically range from **–1 to +1**:
  - **+1** → complete certainty (true)
  - **0** → no information / uncertainty
  - **–1** → complete disbelief (false)

A certainty factor expresses how strongly evidence supports or refutes a rule's conclusion.

Role of Certainty Factors in Rule-Based Systems

Rule-based systems (such as expert systems) often deal with **incomplete, vague, or uncertain knowledge**. Certainty factors help manage this uncertainty effectively.

1. Handling Uncertainty

- Real-world knowledge is rarely absolute.
- CFs allow rules to produce **probabilistic-like conclusions** instead of rigid true/false outputs.

2. Associating Confidence with Rules

- Each rule is assigned a certainty factor indicating its reliability.
- Example:
  - IF symptom = fever THEN disease = flu (CF = 0.7)

3. Combining Evidence

- When multiple rules support the same conclusion, their CFs are **combined** using specific formulas.
- This enables the system to aggregate evidence from different sources.

4. Conflict Resolution

- If different rules lead to conflicting conclusions, CFs help determine **which conclusion is stronger** based on higher confidence.

5. Incremental Reasoning

- As new evidence is added, the system updates certainty values dynamically.
- This is similar to belief updating in intelligent systems.

6. Practical Applications

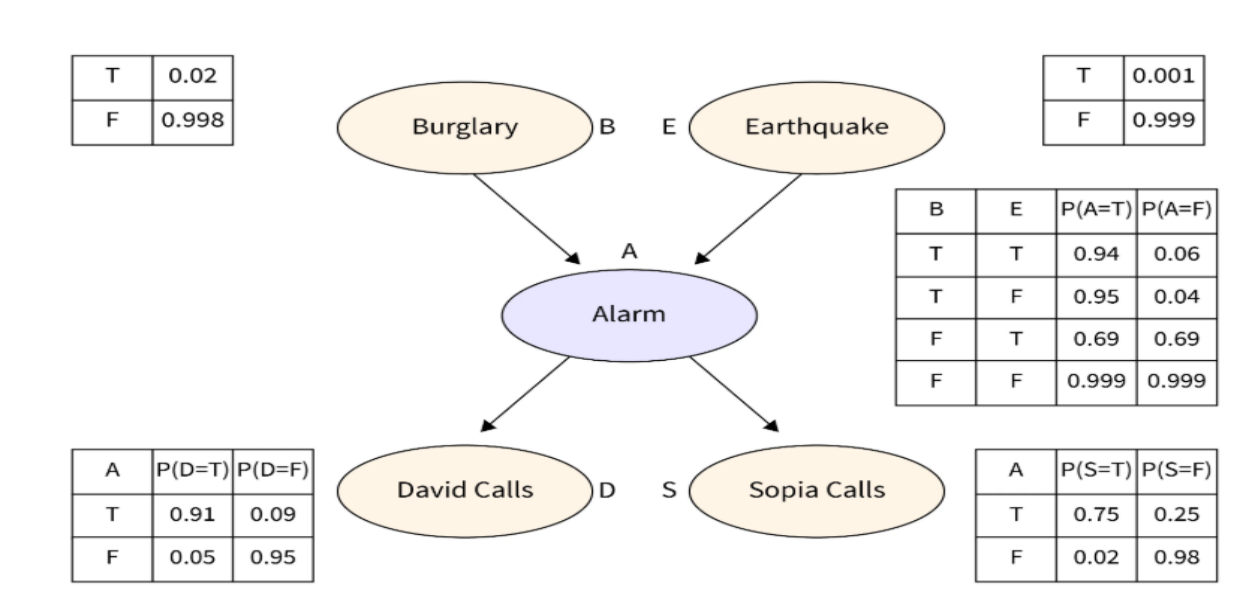
- Medical diagnosis systems (e.g., early expert systems like MYCIN)
- Fault detection in engineering systems
- Decision support systems

Explain Bayesian Networks. How do they represent uncertain knowledge in AI systems?

Bayesian Networks – Explanation

A **Bayesian Network (BN)** is a graphical model used in AI to represent **probabilistic relationships among variables**. It combines **graph theory** and **probability theory** to model uncertainty.

- It is also called a **Belief Network** or **Probabilistic Graphical Model**
- It consists of:
  - **Nodes** → represent random variables
  - **Directed edges (arrows)** → represent dependencies or causal relationships
  - It forms a **Directed Acyclic Graph (DAG)** (no loops)



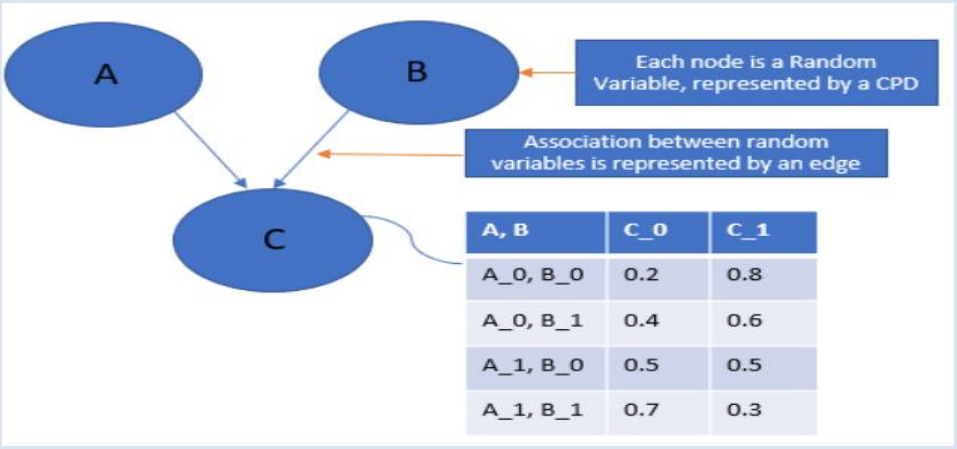
T	0.02
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P(D=T)	P(D=F)
T	0.91	0.09
F	0.05	0.95

A	P(S=T)	P(S=F)
T	0.75	0.25
F	0.02	0.98



A, B	C_0	C_1
A_0, B_0	0.2	0.8
A_0, B_1	0.4	0.6
A_1, B_0	0.5	0.5
A_1, B_1	0.7	0.3

Each node has an associated **Conditional Probability Table (CPT)** that defines the probability of the node given its parent nodes.

Example:

- Rain → Sprinkler → Wet Grass
- The probability of *Wet Grass* depends on both *Rain* and *Sprinkler*

How Bayesian Networks Represent Uncertain Knowledge

1. Probabilistic Representation

- Instead of fixed true/false values, BNs use **probabilities** to represent uncertainty.
- Each variable is described by a probability distribution.

2. Conditional Dependencies

- Relationships between variables are captured using **conditional probabilities**.
- This reduces complexity by modeling only **relevant dependencies**.

3. Joint Probability Modeling

- A BN represents the **joint probability distribution** of all variables as a product of conditional probabilities:

P(X1,X2,...,Xn)=∏P(Xi|Parents(Xi))P(X_1, X_2, ..., X_n) = \prod P(X_i \mid Parents(X_i))P(X1,X2,...,Xn)=∏P(Xi|Parents(Xi))

4. Efficient Inference

- BNs allow reasoning under uncertainty using inference techniques such as:
  - **Prediction** (cause → effect)
  - **Diagnosis** (effect → cause)
  - **Intercausal reasoning**

5. Updating Beliefs

- When new evidence is observed, probabilities are updated using **Bayes' Theorem**.
- This enables dynamic and adaptive decision-making.

6. Handling Incomplete Data

- BNs can still make predictions even when some variables are unknown.

Applications in AI Systems

- Medical diagnosis (predicting diseases from symptoms)
- Fault detection in engineering systems
- Robotics and autonomous systems
- Natural language processing
- Decision support systems

Discuss Fuzzy Logic. How does it differ from classical (crisp) logic?

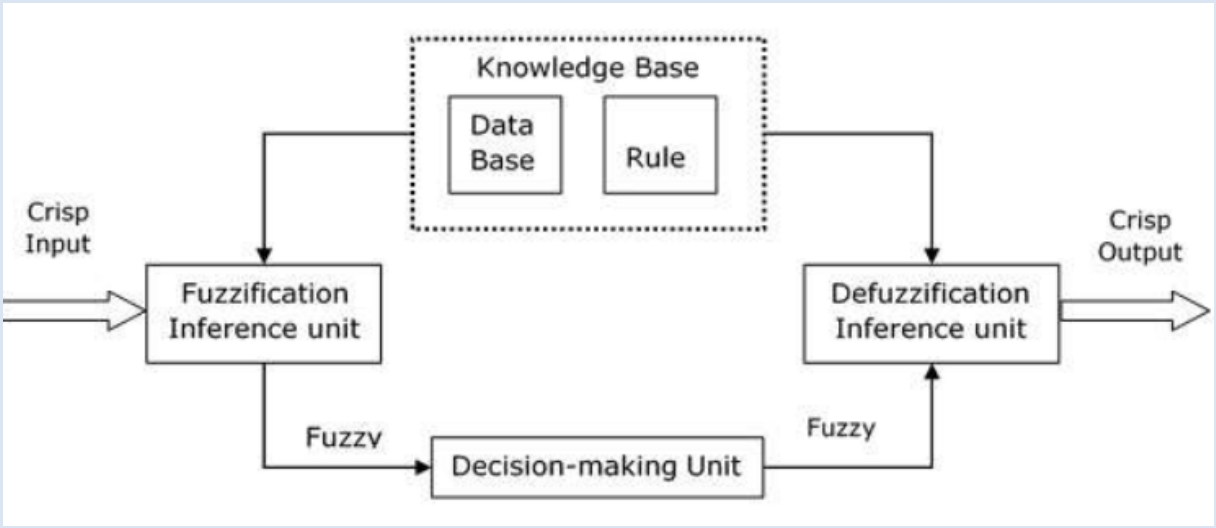
Fuzzy Logic – Explanation

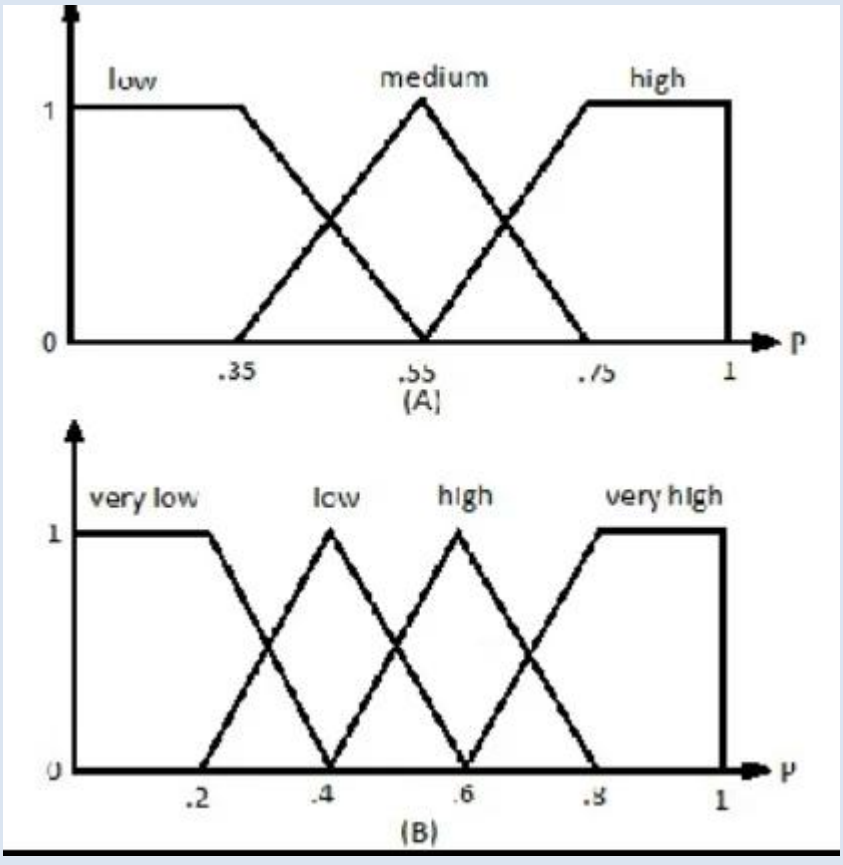
Fuzzy Logic is a form of reasoning that allows variables to have **degrees of truth** rather than only true or false values. It was introduced by Lotfi A. Zadeh in 1965.

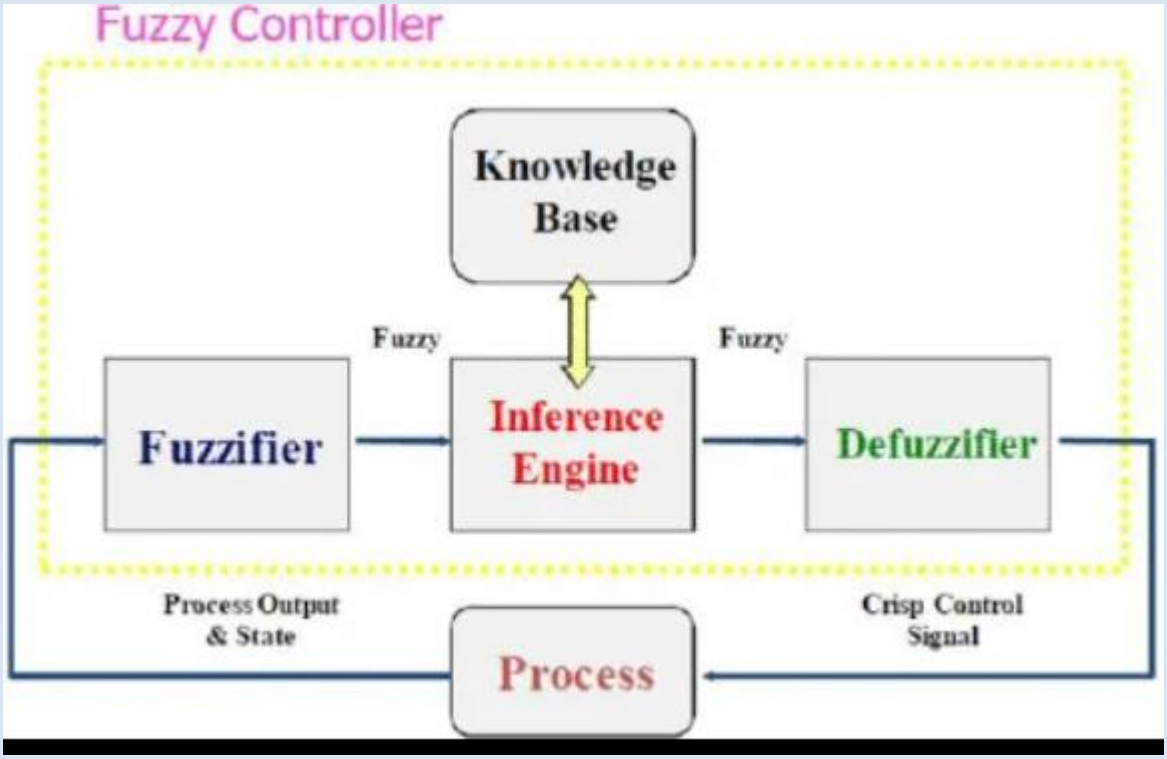
- In fuzzy logic, truth values range from **0 to 1**
- It is based on the concept of **Fuzzy Set Theory**, where elements can partially belong to a set

Example:

- Temperature can be **"cold" (0.2), "warm" (0.6), "hot" (0.8)** simultaneously







A fuzzy logic system typically involves:

1. **Fuzzification**
   - Converts crisp inputs into fuzzy values using membership functions
2. **Rule Base**
   - Contains IF–THEN rules (e.g., IF temperature is high THEN fan speed is fast)
3. **Inference Engine**
   - Applies rules and combines results
4. **Defuzzification**
   - Converts fuzzy output back to a crisp value

Significance of Fuzzy Logic

- Models **human reasoning and linguistic variables**
- Useful in systems where data is **imprecise or vague**
- Widely used in:
  - Control systems (washing machines, air conditioners)
  - Decision-making systems
  - Robotics and automation

Aspect	Fuzzy Logic	Classical (Crisp) Logic
Truth Values	Continuous (0 to 1)	Binary (0 or 1)
Nature of Reasoning	Approximate reasoning	Exact reasoning
Set Membership	Partial membership	Full or no membership
Handling Uncertainty	Handles vagueness effectively	Cannot handle ambiguity well
Real-world Suitability	Suitable for complex, human-like reasoning	Suitable for precise systems
Example	“Speed is high (0.7)”	“Speed is high (true/false)”

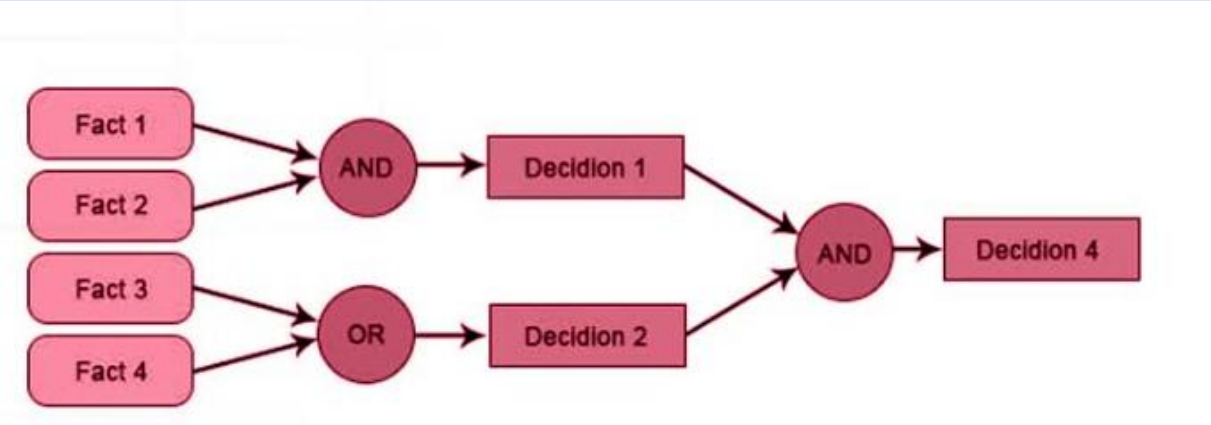
Illustrate forward and backward reasoning using a rule-based system with suitable examples.

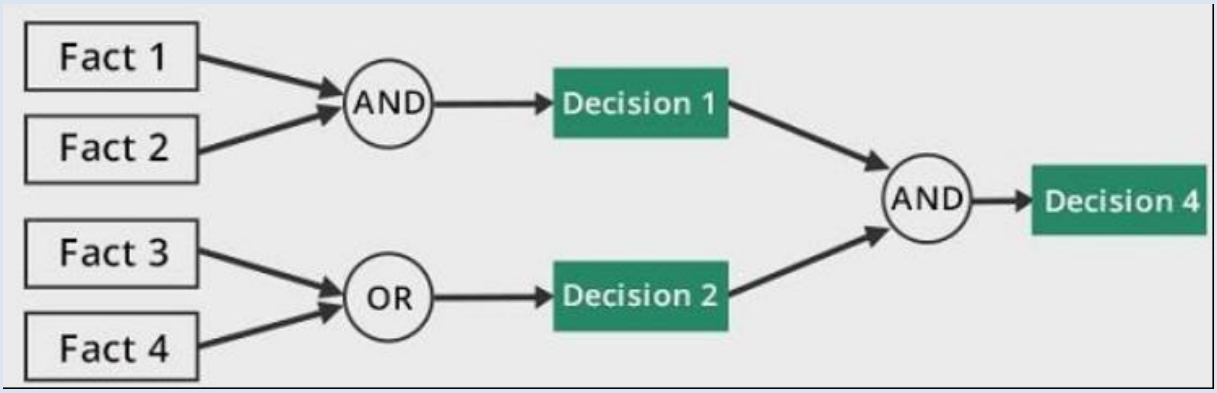
1. Forward Reasoning (Forward Chaining)

- Also called **data-driven reasoning**
- Starts from **known facts** and applies rules to reach conclusions

Process

1. Begin with available facts
2. Match facts with rule conditions (IF part)
3. Fire applicable rules
4. Add new facts
5. Continue until a goal is reached or no more rules apply





Example

Rules:

- R1: IF it is raining THEN ground is wet
- R2: IF ground is wet THEN shoes get dirty

Given Fact:

- It is raining

Execution:

- Apply R1 → Ground is wet
- Apply R2 → Shoes get dirty

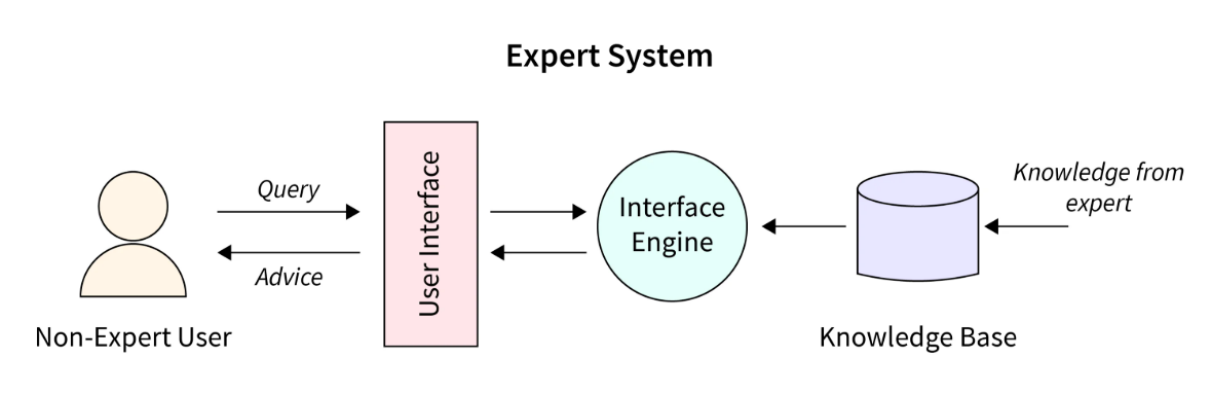
Conclusion: Shoes get dirty

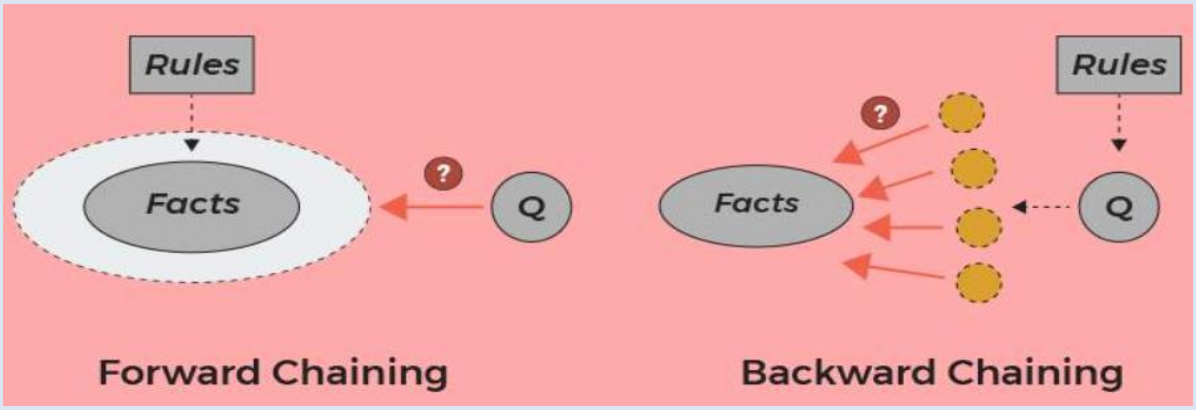
2. Backward Reasoning (Backward Chaining)

- Also called **goal-driven reasoning**
- Starts from a **goal (hypothesis)** and works backward to verify it

Process

1. Identify the goal
2. Find rules that can produce the goal
3. Check if conditions of those rules are true
4. If not, treat conditions as sub-goals
5. Continue until facts are verified





Example

Rules:

- R1: IF fever THEN flu
- R2: IF body pain THEN fever

Goal:

- Determine if patient has flu

Execution:

- To prove flu → need fever (from R1)
- To prove fever → check body pain (from R2)
- If body pain is true → fever is true → flu is true

Conclusion: Patient has flu

Apply Bayes' Theorem to a real-world problem involving uncertainty and interpret the results

Let's consider a **medical diagnosis scenario**, a classic example of reasoning under uncertainty.

Problem Statement

A disease affects **1% of a population**.
A diagnostic test has:

- **99% sensitivity** → correctly detects disease
- **95% specificity** → correctly identifies healthy people

If a person tests **positive**, what is the probability they actually have the disease?

Step 1: Use Bayes' Theorem

$$P(D|T) = \frac{P(T|D)P(D)}{P(T)}$$

Where:

- D: Disease present
- T: Test is positive

Step 2: Assign Values

- $P(D) = 0.01$ (1% prevalence)
- $P(T|D) = 0.99$ (true positive rate)
- $P(T|negD) = 0.05$ (false positive rate = 1 – specificity)

Step 3: Compute Total Probability of Positive Test

$$P(T) = P(T|D)P(D) + P(T|\neg D)P(\neg D)$$

$$P(T) = (0.99 \times 0.01) + (0.05 \times 0.99)$$

$$P(T) = 0.0099 + 0.0495 = 0.0594$$

Step 4: Apply Bayes' Theorem

$$P(D|T) = \frac{0.99 \times 0.01}{0.0594}$$

$$P(D|T) \approx 0.1667 \ (16.67\%)$$

Interpretation of Results

- Even after testing **positive**, the probability of actually having the disease is only **~16.7%**
- This may seem surprising, but it happens because:
  - The disease is **rare (low prior probability)**
  - False positives significantly affect the result

Analyze the differences between procedural and declarative knowledge. How are they represented in AI?

In Artificial Intelligence, knowledge is broadly categorized into **procedural** and **declarative** forms based on *how it is stored and used*.

1. Declarative Knowledge ("Know What")

Definition:
Declarative knowledge represents **facts, concepts, and relationships** about the world.

- It answers **"what is true?"**
- Knowledge is stored explicitly and can be easily queried

Examples

- "Paris is the capital of France"
- "Water boils at 100°C"
- "A robot has joints and degrees of freedom"

Representation in AI

Declarative knowledge is commonly represented using:

- **Predicate Logic** (First-order logic)
- **Semantic networks** (nodes and relationships)
- **Frames** (structured knowledge with attributes)
- **Knowledge graphs**

2. Procedural Knowledge ("Know How")

Definition:
Procedural knowledge represents **how to perform tasks or processes**.

- It answers **"how to do something?"**
- Often embedded in rules, procedures, or algorithms

Examples

- Steps to solve a math problem
- How a robot picks and places an object
- Troubleshooting a machine

Representation in AI

Procedural knowledge is represented using:

- **Production rules (IF–THEN rules)**
- Algorithms and programs
- Scripts and plans
- Rule-based systems (expert systems)

Aspect	Declarative Knowledge	Procedural Knowledge
Meaning	"Know what"	"Know how"
Nature	Static facts	Dynamic processes
Representation	Logic, semantic networks, frames	Rules, algorithms, procedures
Flexibility	Easy to modify and query	Harder to modify
Usage	Knowledge-based reasoning	Task execution

Examine the Dempster–Shafer Theory. How does it handle uncertainty differently compared to probability theory?

Dempster–Shafer Theory (DST) – Overview

Dempster–Shafer Theory, also known as the **Theory of Evidence**, is a framework for reasoning under uncertainty developed by Arthur P. Dempster and Glenn Shafer.

- It generalizes classical probability theory
- Allows representation of **partial belief** without requiring exact probabilities
- Works with **sets of hypotheses** rather than single events

Key Concepts in DST

1. Frame of Discernment (Θ)

- The set of all possible hypotheses
- Example: Θ = {Disease A, Disease B}

2. Basic Probability Assignment (BPA) / Mass Function

- Assigns belief to **subsets** of Θ
- Denoted as $m(A)$, where:
  - $m(\emptyset) = 0$
  - $\sum m(A) = 1$

👉 Unlike probability, belief can be assigned to **combinations of outcomes** (e.g., {A, B})

3. Belief (Bel) and Plausibility (Pl)

- **Belief (Bel)**: Minimum support for a hypothesis
- **Plausibility (Pl)**: Maximum possible support

$$Bel(A) \leq P(A) \leq Pl(A)$$

👉 This creates an **interval of uncertainty**, not a single value

4. Dempster's Rule of Combination

- Combines evidence from multiple sources
- Handles conflicting information intelligently

Aspect	Dempster–Shafer Theory	Probability Theory
Uncertainty Representation	Uses **belief intervals (Bel–Pl)**	Uses **single probability value**
Ignorance Handling	Can explicitly represent **ignorance**	Must distribute probability across events
Assignment of Belief	Can assign belief to **sets of hypotheses**	Assigns probability only to **individual events**
Evidence Combination	Uses Dempster's rule	Uses Bayes' rule
Flexibility	More flexible with incomplete data	Requires precise probabilities
Interpretation	Distinguishes between **uncertainty and lack of knowledge**	Treats both similarly

Illustrative Example

Suppose we are diagnosing a fault:

- DST assignment:
  - $m(A) = 0.6$ (evidence supports A)
  - $m(A, B) = 0.4$ (uncertainty between A and B)

👉 This means:

- We are **60% confident** in A
- But **40% of belief is uncertain**, not committed to either A or B

In probability theory, that 0.4 must be **split artificially**, even if we lack information.

Significance in AI

- Sensor fusion (combining uncertain sensor data)
- Expert systems
- Fault diagnosis
- Decision-making under incomplete information

Summary

- **Dempster–Shafer Theory** provides a **more flexible and expressive way** to model uncertainty.
- Unlike probability theory, it:
  - Represents **ignorance explicitly**
  - Uses **belief intervals instead of exact probabilities**
- This makes it highly useful in AI systems where **information is incomplete or ambiguous**.

Critically evaluate certainty factors and fuzzy logic. Discuss their advantages and limitations in real-world applications

Critical Evaluation of Certainty Factors and Fuzzy Logic

Both **certainty factors (CFs)** and **fuzzy logic** are approaches for handling uncertainty in AI, but they differ in philosophy, representation, and application.

1. Certainty Factors (CFs)

Overview

- Represent **degree of belief** in a hypothesis (range: –1 to +1 or 0 to 1)
- Widely used in early expert systems like MYCIN

Advantages

- **Simple and intuitive**: Easy to assign confidence values to rules
- **Efficient reasoning**: Works well in rule-based systems
- **Combines evidence**: Multiple rules can support or refute a conclusion
- **Low computational cost**: Suitable for real-time systems

Limitations

- **Lack of strong theoretical foundation** compared to probability theory
- **Subjectivity**: CF values often depend on expert judgment
- **Inconsistency in combination rules**: Different systems may use different formulas
- **Limited expressiveness**: Cannot clearly distinguish between uncertainty and ignorance

2. Fuzzy Logic

Overview

- Based on **Fuzzy Set Theory**
- Handles **vagueness and imprecision** using degrees of membership (0 to 1)

Advantages

- **Handles vague concepts effectively** (e.g., “high temperature”, “low speed”)
- **Human-like reasoning**: Uses linguistic rules (IF–THEN)
- **Robust in control systems**: Performs well even with noisy or imprecise data
- **Widely applied** in consumer electronics, automation, and robotics

Limitations

- **No learning capability (basic systems)** unless combined with AI/ML
- **Rule explosion**: Large systems require many rules
- **Membership function design is subjective**
- **Not probabilistic**: Cannot model randomness or stochastic uncertainty

Aspect	Certainty Factors	Fuzzy Logic
Type of Uncertainty	Degree of belief	Vagueness/imprecision
Basis	Heuristic	Mathematical (fuzzy sets)
Representation	Confidence values in rules	Membership functions
Reasoning Style	Rule-based inference	Approximate reasoning
Interpretability	Moderate	High (linguistic terms)
Flexibility	Limited	High for real-world ambiguity

4. Real-World Applications

Certainty Factors

- Medical diagnosis systems
- Fault detection in engineering
- Expert systems for decision support

☞ Works best when **expert knowledge is available but uncertain**

Fuzzy Logic

- Washing machines, air conditioners
- Automotive control systems (e.g., braking, gear control)
- Industrial automation and robotics

☞ Works best when **inputs are vague or linguistic**

5. Critical Perspective

- **Certainty Factors** are useful for **quick, rule-based decision-making**, but suffer from **lack of rigor and scalability issues**.
- **Fuzzy Logic** excels in **handling imprecision and human reasoning**, but cannot handle **random uncertainty or probabilistic inference**.

☞ In modern AI systems, both are often **combined with probabilistic methods or machine learning** to overcome their limitations.

Evaluate following knowledge representation techniques: Rule based, logic programming and Bayesian networks. Which is most effective under uncertainty and why?

Evaluation of Knowledge Representation Techniques

In AI, different knowledge representation (KR) techniques are suited to different types of problems—especially when **uncertainty** is involved. Here's a critical comparison of **rule-based systems**, **logic programming**, and **Bayesian networks**.

1. Rule-Based Representation

Overview

- Knowledge represented as **IF–THEN rules**
- Example: IF fever THEN disease = flu

Strengths

- **Simple and intuitive**
- Easy to implement and understand
- Suitable for **expert systems**
- Supports reasoning via forward/backward chaining

Limitations

- Poor handling of uncertainty (unless extended with certainty factors)
- **Rule explosion** in complex systems
- Difficult to maintain and scale
- Lacks probabilistic foundation

2. Logic Programming

Overview

- Based on formal logic, especially **First-Order Logic**
- Programs consist of facts and rules (e.g., Prolog systems)

Strengths

- **Mathematically precise and rigorous**
- Supports complex reasoning and inference
- Good for problems requiring **symbolic reasoning**
- Declarative (focus on *what*, not *how*)

Limitations

- Assumes **binary truth (true/false)**
- Not suitable for uncertain or incomplete knowledge
- Extensions for uncertainty are complex
- Computationally expensive for large problems

3. Bayesian Networks

Overview

- Probabilistic graphical models using **Bayes' Theorem**
- Represent variables and dependencies via a directed acyclic graph

Strengths

- **Strong theoretical foundation in probability**
- Naturally handles **uncertainty and incomplete data**
- Supports **belief updating** with new evidence
- Efficient representation of complex dependencies
- Widely used in AI, medical diagnosis, and decision systems

Limitations

- Requires **prior probabilities and conditional probabilities**
- Can be computationally expensive for very large networks
- Model construction can be complex

Feature	Rule-Based Systems	Logic Programming	Bayesian Networks
Representation	IF–THEN rules	Formal logic	Probabilistic graph
Uncertainty Handling	Limited (CF-based)	Very poor	Excellent
Reasoning Type	Heuristic	Deductive	Probabilistic
Flexibility	Moderate	Low for uncertainty	High
Real-world Applicability	Moderate	Limited	Very high

Which is Most Effective Under Uncertainty?

☞ **Bayesian Networks are the most effective under uncertainty**

Why?

- Provide a **quantitative framework** for uncertainty
- Use probabilities instead of binary or heuristic values
- Allow **updating beliefs dynamically** when new evidence arrives
- Can model **causal relationships and dependencies**
- Handle **missing or incomplete data gracefully**

Critical Insight

- **Rule-based systems** are useful for structured knowledge but struggle with uncertainty
- **Logic programming** is powerful for exact reasoning but unsuitable for uncertain environments
- **Bayesian networks** offer the **most robust and realistic approach** for uncertain, real-world AI problems

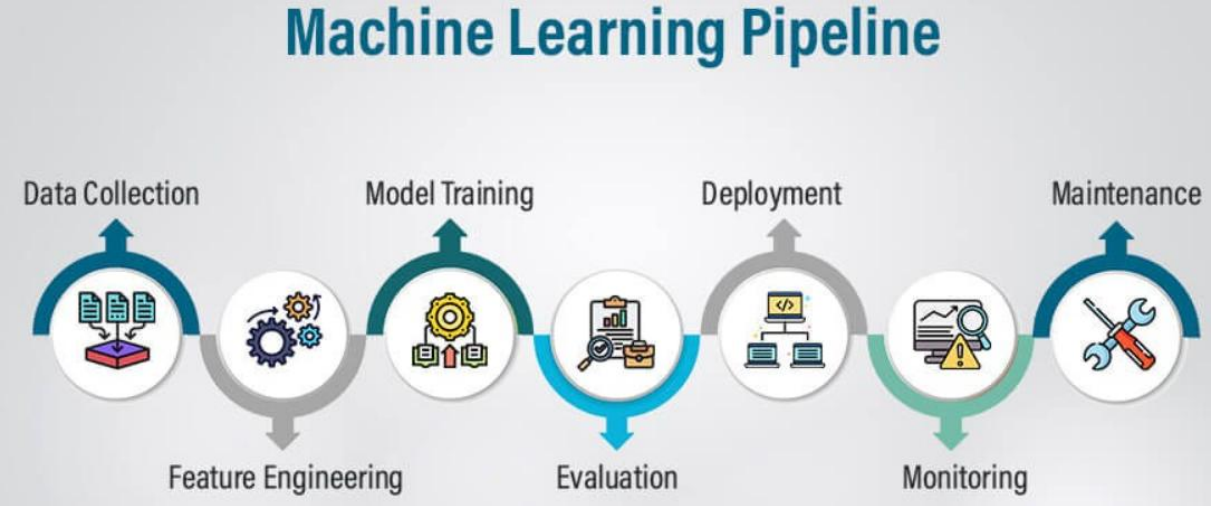
UNIT 5

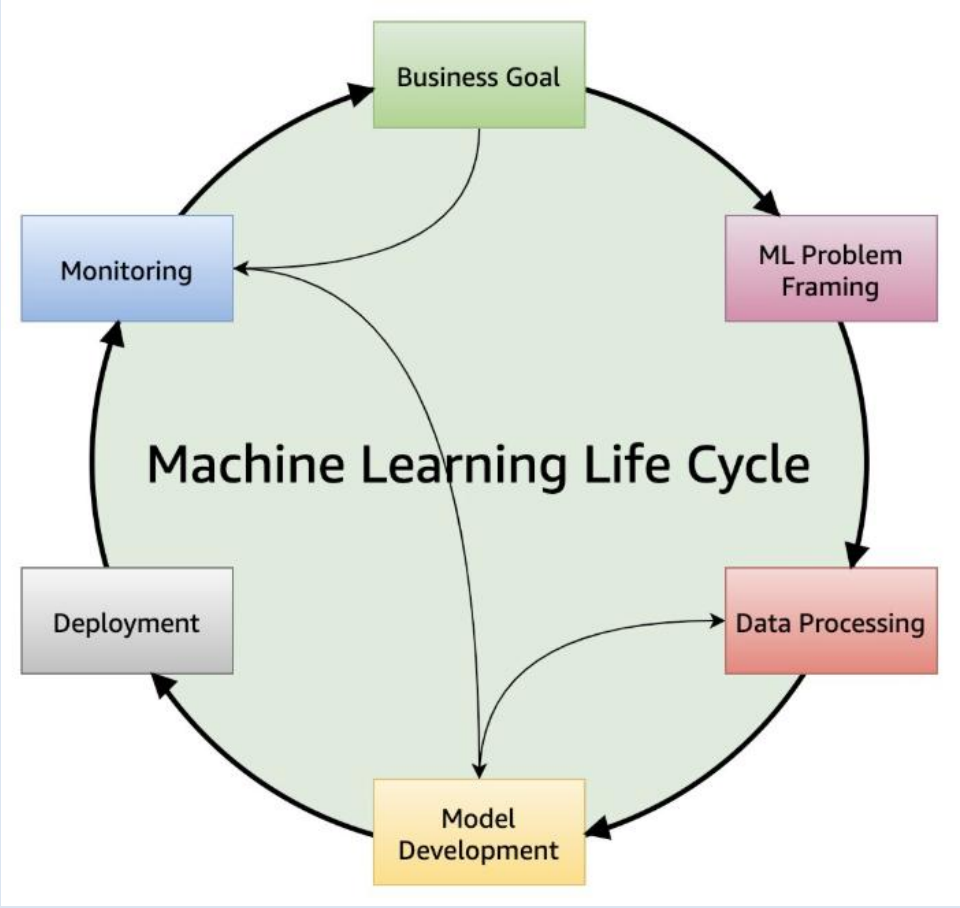
INTRODUCTION TO MACHINE LEARNING

"Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed."

— Arthur Samuel

With suitable illustration describe the end-to-end process of machine learning lifecycle?





1. Problem Definition

- Clearly define the **objective** (e.g., prediction, classification, clustering).
- Identify:
  - Inputs (features)
  - Outputs (target variable)
- Example: Predict student performance based on attendance and marks.

2. Data Collection

- Gather data from:
  - Databases, sensors, APIs, surveys, logs.
- Ensure:
  - Sufficient volume
  - Relevant features

3. Data Preprocessing (Data Cleaning)

- Handle:
  - Missing values
  - Noise and outliers
  - Duplicate data
- Convert raw data into usable format:
  - Encoding categorical variables
  - Normalization/scaling

4. Exploratory Data Analysis (EDA)

- Analyze data patterns using:
  - Graphs (histograms, scatter plots)
  - Statistical summaries
- Helps in:
  - Understanding relationships
  - Selecting important features

5. Feature Engineering

- Create new meaningful features
- Select important variables
- Reduce dimensionality (e.g., PCA)

6. Model Selection

- Choose appropriate algorithm:
  - Regression → Linear Regression
  - Classification → Decision Tree, SVM
  - Clustering → K-means
- Based on:
  - Problem type
  - Data size and complexity

7. Model Training

- Train model using training data
- Model learns patterns:
  - Adjusts weights/parameters

8. Model Evaluation

- Test using unseen data
- Metrics:
  - Accuracy, Precision, Recall, F1-score
  - RMSE (for regression)
- Detect:
  - Overfitting / Underfitting

9. Model Tuning (Optimization)

- Improve performance by:
  - Hyperparameter tuning (Grid Search, Random Search)
  - Cross-validation

10. Deployment

- Deploy model into real-world system:
  - Web apps, mobile apps, APIs
- Example:
  - Recommendation system in e-commerce

11. Monitoring & Maintenance

- Track performance over time
- Handle:
  - Data drift
  - Model degradation
- Retrain model when needed

Discuss the applications of Machine Learning across various domains such as healthcare, finance, manufacturing, and education.

Machine Learning (ML) has become a transformative technology across multiple domains by enabling systems to learn from data and make intelligent decisions. Below is a structured discussion of its applications in **healthcare, finance, manufacturing, and education**.

1. Healthcare Applications

Key Uses:

- **Disease Diagnosis**
  - ML models analyze medical images (X-rays, MRI) to detect diseases like cancer.
- **Predictive Analytics**
  - Predict patient outcomes, disease progression, and hospital readmissions.
- **Personalized Medicine**
  - Tailors treatment plans based on patient data and genetics.
- **Drug Discovery**
  - Speeds up identification of new drugs and reduces research cost.
- **Remote Monitoring**
  - Wearable devices + ML track patient health in real time.

Impact:

- Improved accuracy in diagnosis
- Reduced human error & Faster treatment decisions

2. Finance Applications

Key Uses:

- **Fraud Detection**
  - Identifies unusual transaction patterns in real time.
- **Algorithmic Trading**
  - Predicts stock prices and automates trading decisions.
- **Credit Scoring**
  - Evaluates loan eligibility using customer data.
- **Risk Management**
  - Assesses financial risks and market fluctuations.
- **Chatbots & Virtual Assistants**
  - Automate customer support in banking.

Impact:

- Enhanced security
- Faster financial decision-making
- Improved customer experience

3. Manufacturing Applications

Key Uses:

- **Predictive Maintenance**
  - Predicts machine failures before they occur.
- **Quality Control**
  - Detects defects using computer vision.
- **Process Optimization**
  - Improves efficiency and reduces waste.
- **Robotics Automation**
  - Smart robots perform repetitive tasks.
- **Supply Chain Optimization**
  - Forecasts demand and manages inventory.

Impact:

- Reduced downtime
- Increased productivity
- Cost savings

4. Education Applications

Key Uses:

- **Personalized Learning**
  - Adapts content based on student performance.
- **Student Performance Prediction**
  - Identifies at-risk students early.
- **Intelligent Tutoring Systems**
  - Provides customized guidance and feedback.
- **Automated Grading**
  - Evaluates assignments and exams efficiently.
- **Virtual Classrooms**
  - Enhances online learning with AI tools.

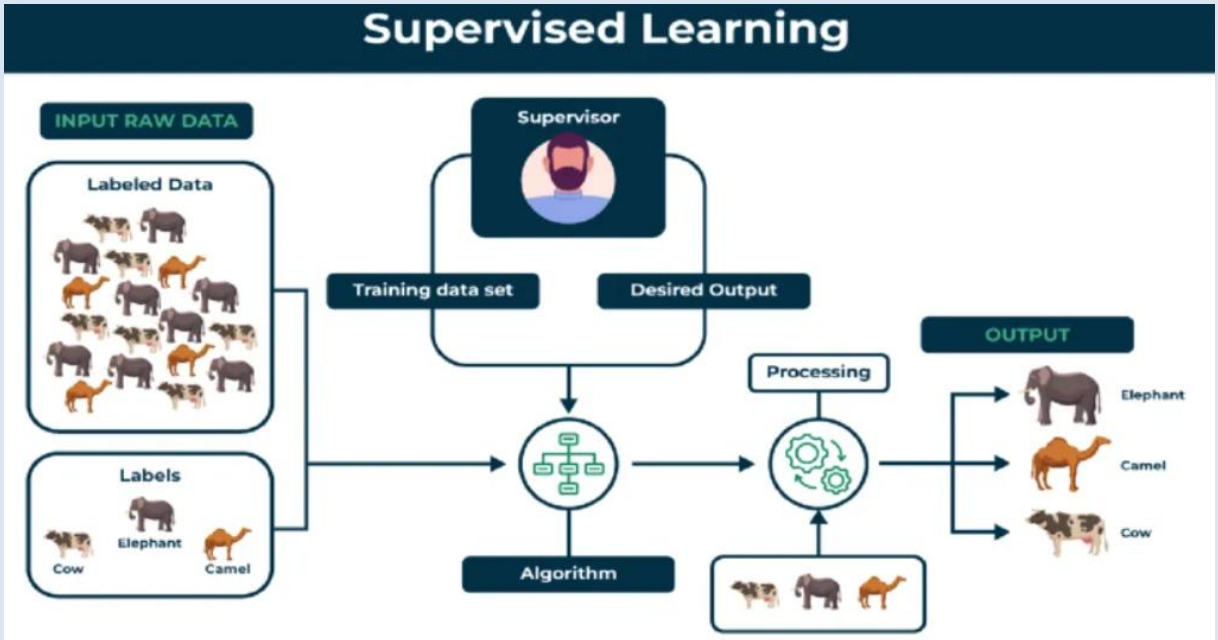
Impact:

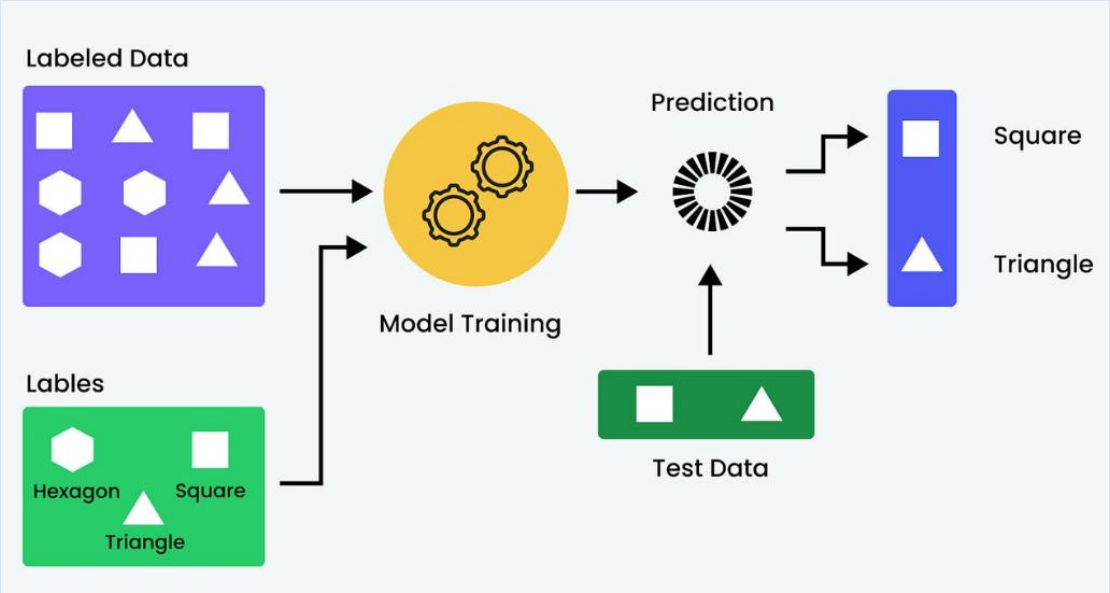
- Improved learning outcomes
- Reduced teacher workload
- Better student engagement

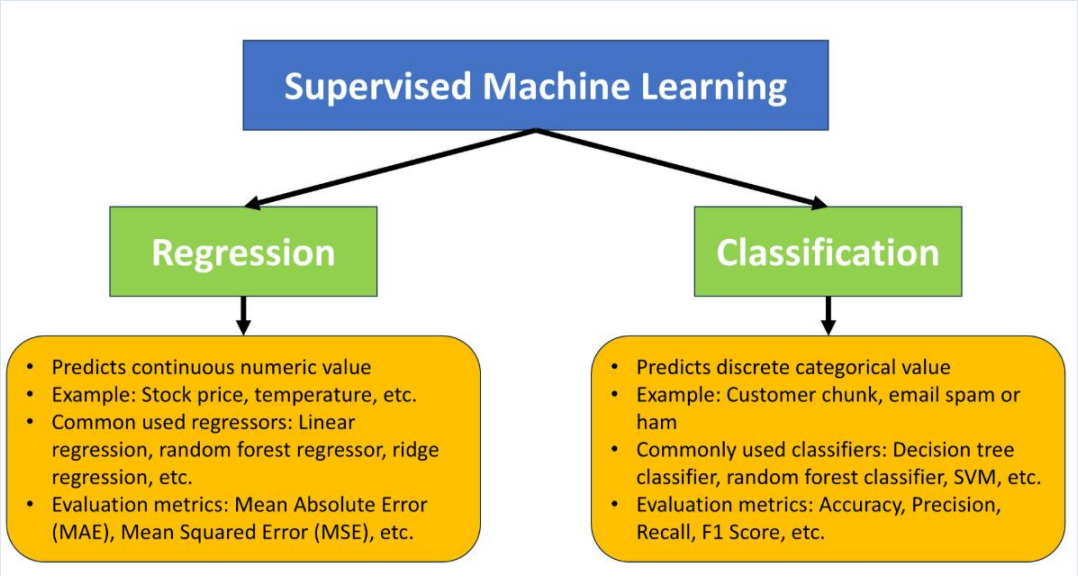
Explain supervised learning. How is it used to solve classification and regression problems?

Supervised Learning is a type of machine learning where a model is trained using a **labeled dataset**. Each training example consists of:

- **Input features (X)**
- **Correct output (Y)**







Steps:

1. Collect labeled data
2. Split into training and testing sets
3. Train the model using training data
4. Evaluate using test data
5. Predict outcomes for new inputs

📌 Definition:

Classification is used when the output variable is **categorical (discrete classes)**.

🎯 Goal:

Assign input data into predefined categories.

📊 Examples:

- Email → Spam / Not Spam
- Disease → Positive / Negative
- Student Result → Pass / Fail

Common Algorithms:

- Logistic Regression
- Decision Trees
- Support Vector Machines (SVM)
- K-Nearest Neighbors (KNN)
- Naïve Bayes

Evaluation Metrics:

- Accuracy
- Precision
- Recall
- F1-score

Definition:

Regression is used when the output variable is **continuous (numerical values)**.

Goal:

Predict a numeric value.

Examples:

- House price prediction
- Temperature forecasting
- Salary estimation

Common Algorithms:

- Linear Regression
- Polynomial Regression
- Ridge/Lasso Regression
- Decision Tree Regression

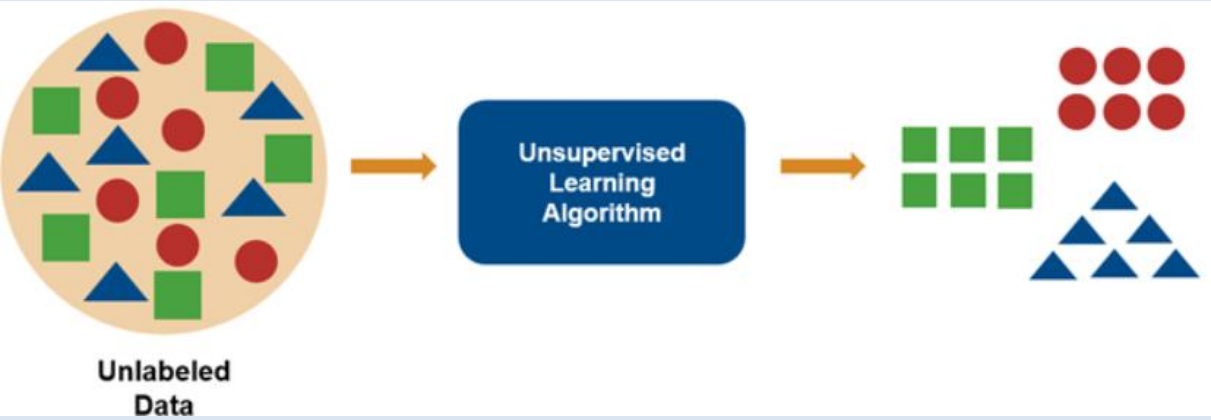
Evaluation Metrics:

- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- R^2 Score

Discuss unsupervised learning. How does clustering differ from classification?

Unsupervised Learning – Explanation

Unsupervised Learning is a type of machine learning where the model is trained on **unlabeled data**.
Unlike supervised learning, there is **no predefined output (target variable)**. The system tries to **discover hidden patterns, structures, or relationships** within the data.



Key Steps:

1. Input raw, unlabeled data
2. Identify similarities or patterns
3. Group or transform data
4. Output structured insights (clusters, features, associations)

🎯 Types of Unsupervised Learning

1. Clustering

- Groups similar data points together
- Example: Customer segmentation

2. Dimensionality Reduction

- Reduces number of features while preserving information
- Example: Principal Component Analysis (PCA)

3. Association Rule Learning

- Finds relationships between variables
- Example: Market basket analysis (items bought together)

Demonstrate clustering techniques for grouping data in an unsupervised learning scenario with an example.

1. K-Means Clustering

📌 Concept:

- Divides data into **K clusters**
- Each cluster has a **centroid (center point)**

🔄 Steps:

1. Choose number of clusters (K)
2. Initialize centroids randomly
3. Assign each data point to nearest centroid
4. Recompute centroids
5. Repeat until convergence

📊 Example:

- Group customers based on **income & spending score**
  - Cluster 1 → High income, high spending
  - Cluster 2 → Low income, low spending
  - Cluster 3 → Mixed behavior

2. Hierarchical Clustering

📌 Concept:

- Builds a **tree-like structure (dendrogram)**

Types:

- **Agglomerative (bottom-up)** → merge clusters
- **Divisive (top-down)** → split clusters

📊 Example:

- Group documents based on similarity of words

3. DBSCAN (Density-Based Clustering)

📌 Concept:

- Groups points based on **density (closely packed points)**

Features:

- Detects **arbitrary-shaped clusters**
- Identifies **noise/outliers**

📊 Example:

- Detect unusual transactions in fraud detection

🎯 Problem: Customer Segmentation

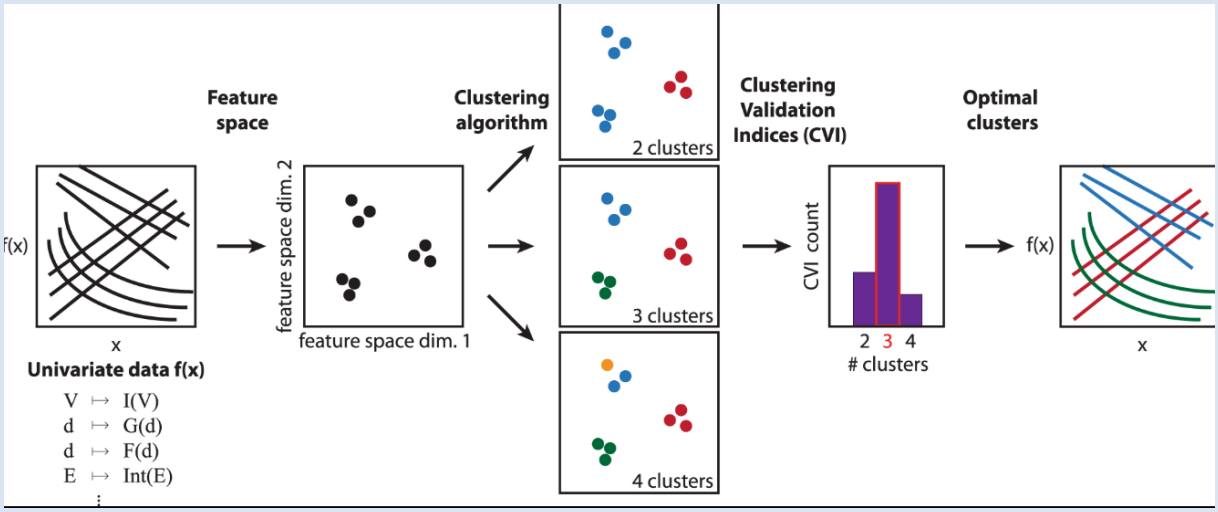
Dataset:

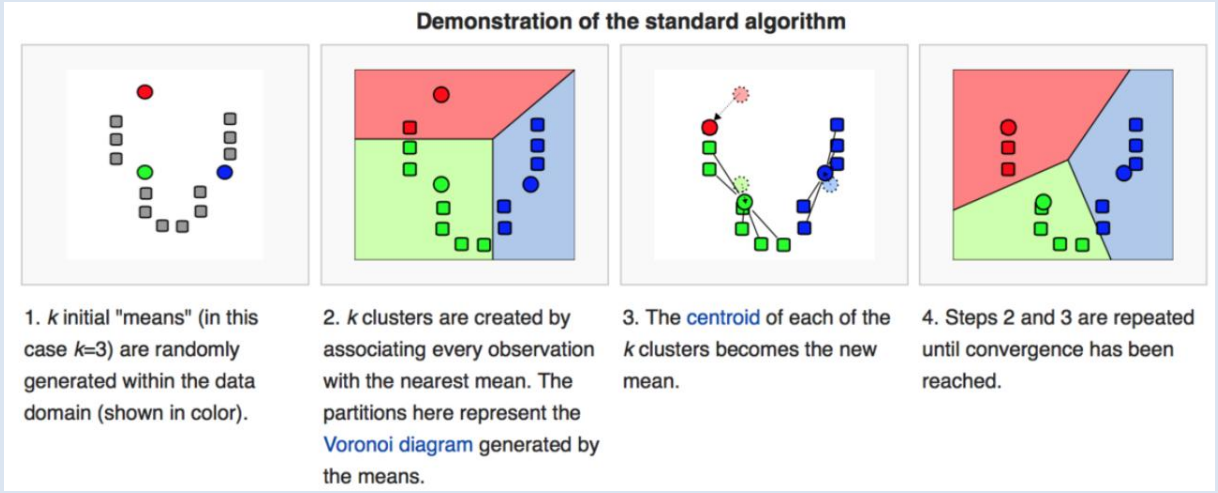
Customers described by:

- Age
- Annual Income
- Spending Score

Step-by-Step:

1. Collect customer data
2. Apply **K-Means (K=3)**
3. Algorithm groups customers into clusters





📈 Another Simple Example

Data Points:

Marks of students in 2 subjects:

- Math
- Science

Clustering Result:

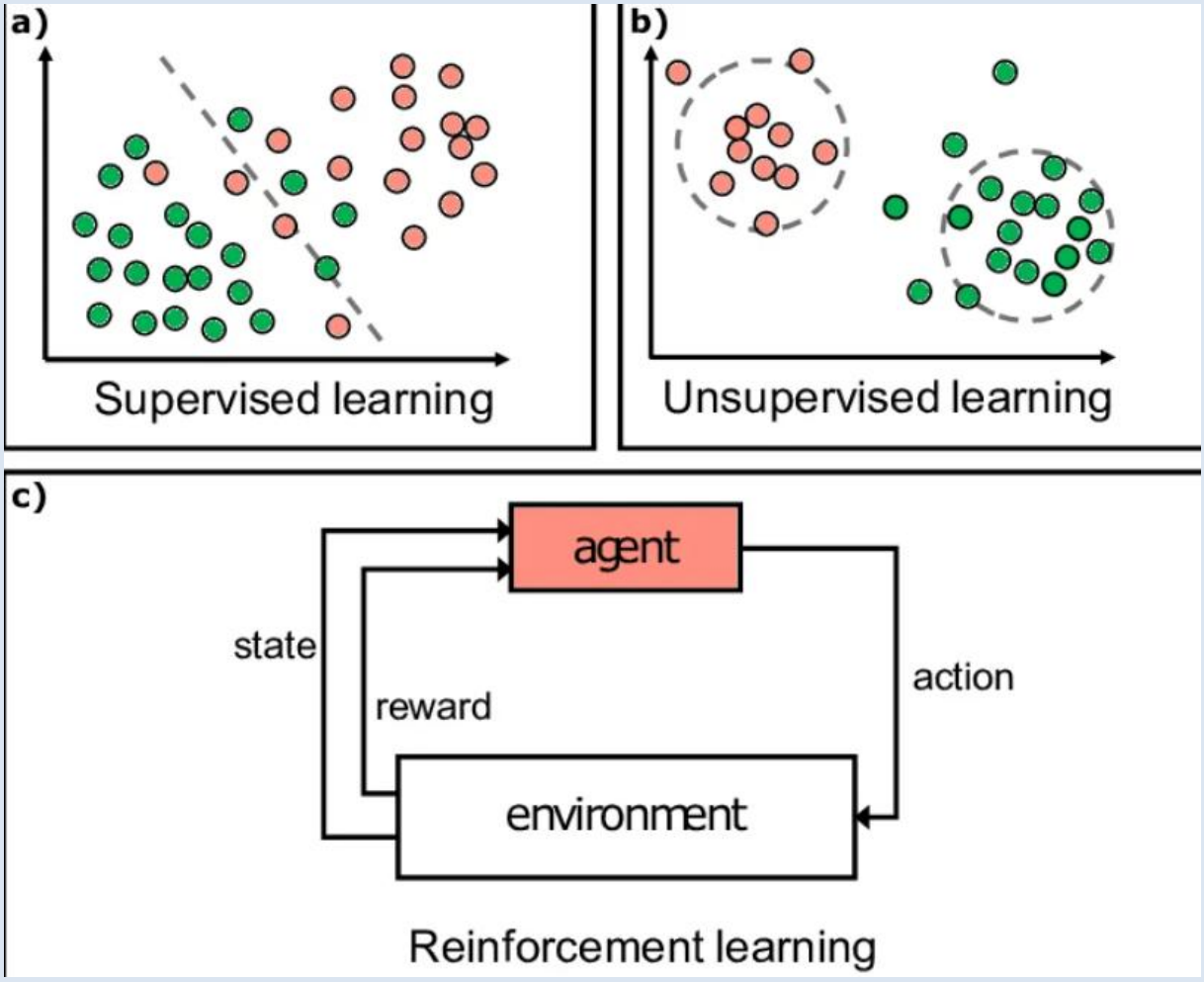
- Cluster 1 → High performers
- Cluster 2 → Average students
- Cluster 3 → Low performers

(No labels were given initially — clusters emerged automatically)

Comparison of Learning Paradigms in Machine Learning

Machine Learning is broadly categorized into **Supervised Learning, Unsupervised Learning, and Reinforcement Learning**. Each differs in how data is used, how learning occurs, and where it is applied.

Feature	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Data Labeling	Requires labeled data	No labels required	Uses rewards and penalties
Objective	Predict outputs	Identify patterns	Optimize decision-making
Common Algorithms	Regression, SVMs, Neural Networks	Clustering, PCA, Autoencoders	Q-Learning, DQN, Policy Gradient
Applications	Healthcare, Finance, Marketing	Cybersecurity, Retail, NLP	Robotics, Gaming, Finance



Feature	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Data Type	Labeled data	Unlabeled data	No explicit dataset (interaction)
Goal	Predict output	Find hidden patterns	Maximize reward
Feedback	Direct (correct answers given)	No feedback	Reward/penalty feedback
Output	Classes or values	Clusters/associations	Optimal actions/policy
Example	Spam detection	Customer segmentation	Game playing
Learning Style	Supervised	Self-organized	Trial-and-error
Complexity	Moderate	Moderate	High

🎯 Key Differences Explained

🔺 Supervised vs Unsupervised

- Supervised uses **labeled data**, unsupervised does not
- Supervised predicts outcomes, unsupervised discovers patterns

🔺 Supervised vs Reinforcement

- Supervised learns from **given answers**
- Reinforcement learns from **experience (rewards/penalties)**

🔺 Unsupervised vs Reinforcement

- Unsupervised → pattern discovery
- Reinforcement → decision-making over time

1. Supervised Learning

📌 Definition:

Learning from **labeled data (input-output pairs)**.

🎯 Objective:

Predict outputs for new inputs.

📊 Examples:

- Email spam detection (Spam / Not Spam)
- House price prediction
- Medical diagnosis

🧠 Algorithms:

- Linear Regression
- Decision Trees
- Support Vector Machines (SVM)

2. Unsupervised Learning

📌 Definition:

Learning from **unlabeled data** to find hidden patterns.

🎯 Objective:

Discover structure in data.

📊 Examples:

- Customer segmentation
- Market basket analysis
- Document clustering

🧠 Algorithms:

- K-Means Clustering
- Hierarchical Clustering
- PCA (Dimensionality Reduction)

3. Reinforcement Learning

📌 Definition:

Learning through **interaction with an environment** using rewards and penalties.

🎯 Objective:

Maximize cumulative reward over time.

🔁 Process:

- Agent takes action → receives reward → improves strategy

📊 Examples:

- Game playing (Chess, AlphaGo)
- Self-driving cars
- Robot navigation

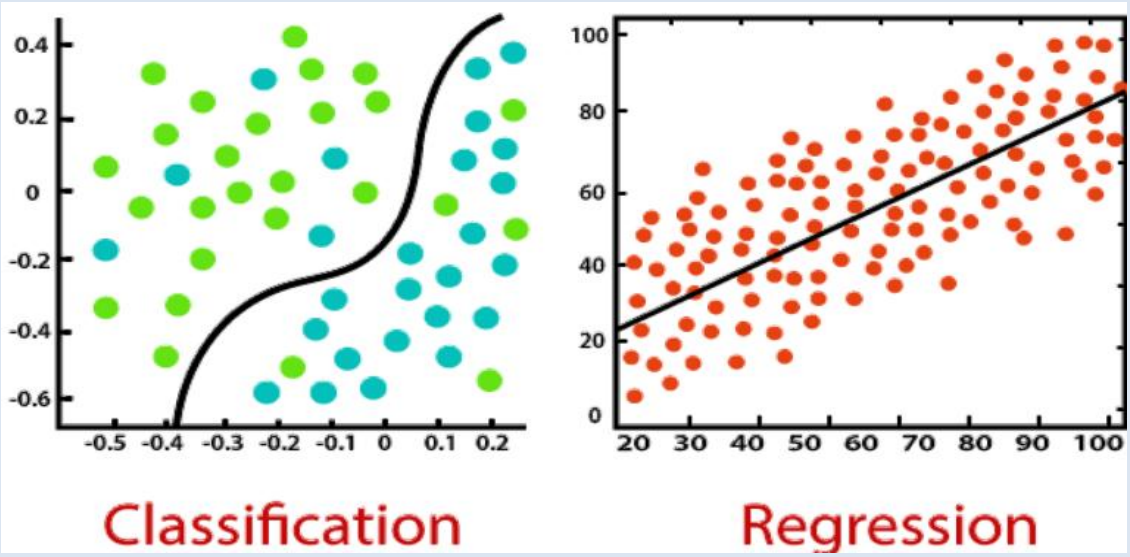
🧠 Algorithms:

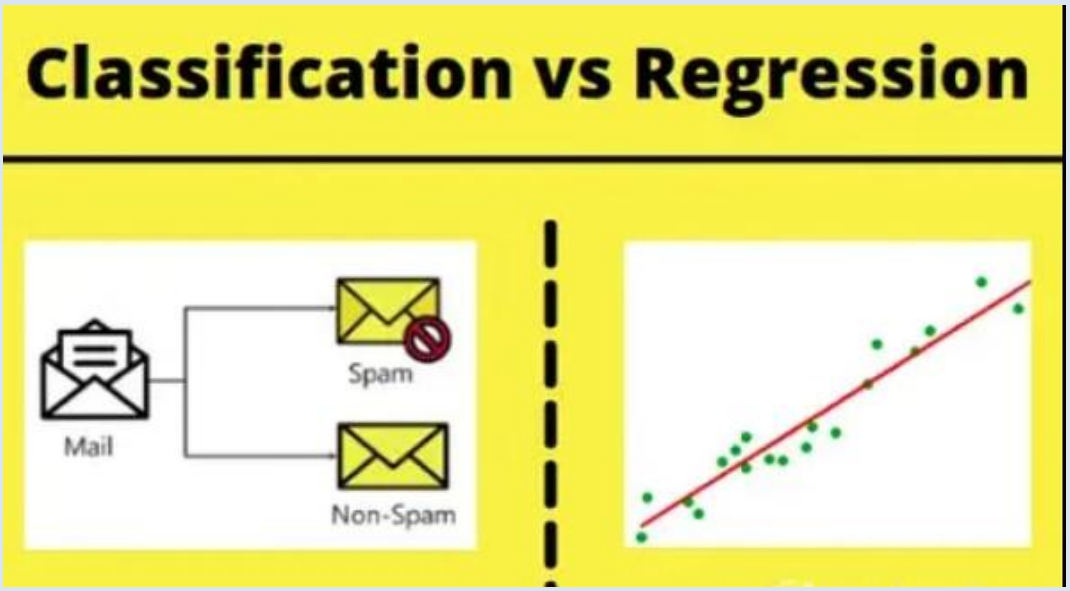
- Q-Learning
- Deep Q Networks (DQN)

Analyze the differences between classification and regression problems. Provide suitable examples.

Classification vs Regression – Detailed Analysis

Both **classification** and **regression** are types of **supervised learning**, but they differ mainly in the **type of output they predict** and how results are interpreted.





1. Classification Problems

📌 Definition:

Classification predicts **discrete (categorical) labels**.

🎯 Objective:

Assign input data to one of the **predefined classes**.

📊 Examples:

- Email → Spam / Not Spam
- Student → Pass / Fail
- Disease → Positive / Negative

🔍 Characteristics:

- Output is **finite categories**
- Decision boundary separates classes
- Often probabilistic (e.g., 0.9 spam probability)

🧠 Algorithms:

- Logistic Regression
- Decision Tree
- Support Vector Machine (SVM)
- K-Nearest Neighbors (KNN)

2. Regression Problems

📌 Definition:

Regression predicts **continuous numerical values**.

🎯 Objective:

Estimate a **real-valued output**.

📊 Examples:

- House price prediction
- Temperature forecasting
- Salary estimation

🔍 Characteristics:

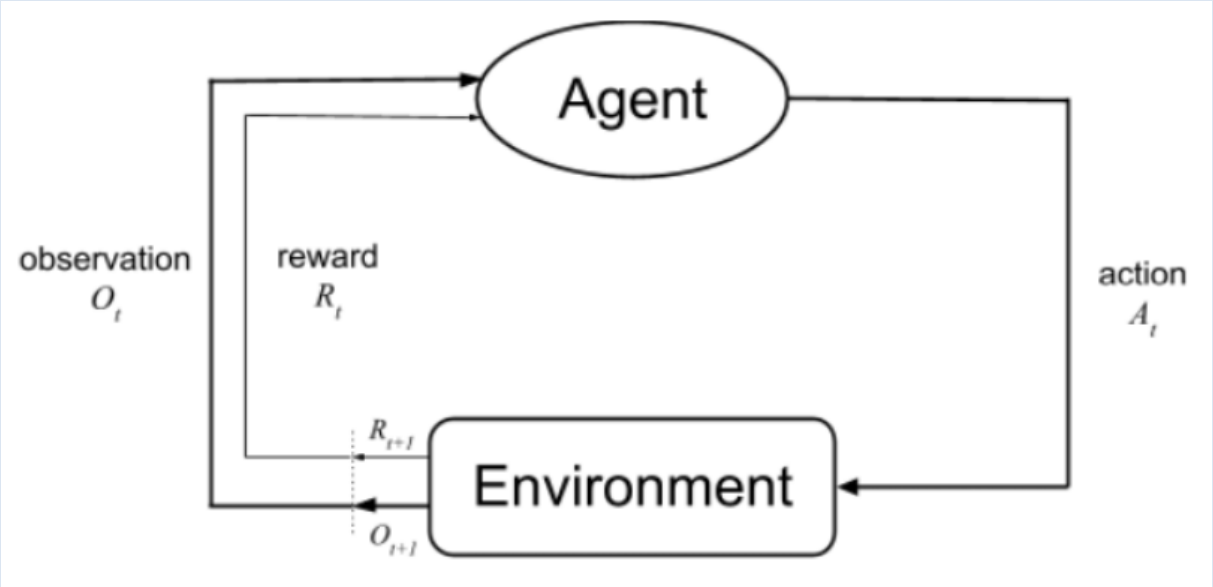
- Output is **continuous range**
- Fits a curve/line to data
- Sensitive to outliers

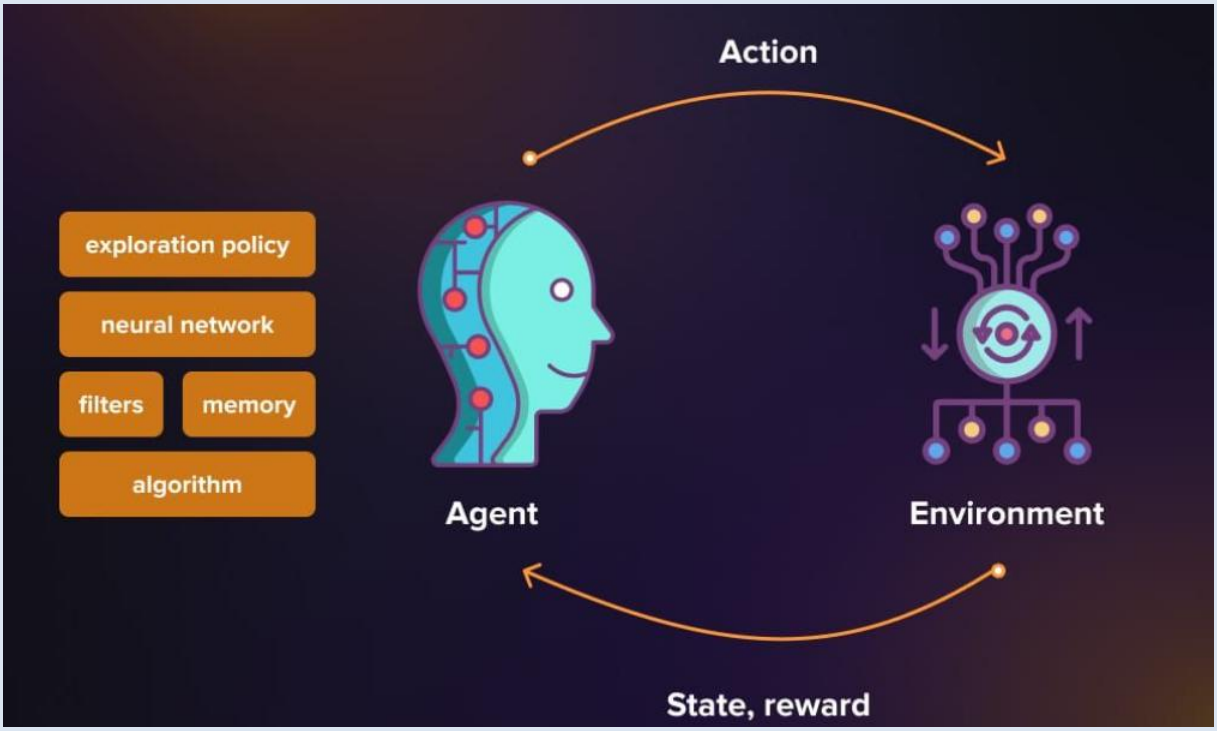
🧠 Algorithms:

- Linear Regression
- Polynomial Regression
- Ridge/Lasso Regression

Aspect	Classification	Regression
Output Type	Discrete (categories)	Continuous (numeric values)
Example	Spam detection	House price prediction
Goal	Assign class labels	Predict numeric value
Model Output	Class probabilities / labels	Real numbers
Evaluation Metrics	Accuracy, Precision, Recall, F1-score	MSE, RMSE, R^2
Decision Boundary	Yes	Not applicable
Error Interpretation	Misclassification	Prediction error magnitude

Evaluate the effectiveness of reinforcement learning in decision making problems. Discuss real-world applications.





Core Elements:

- **Agent** – decision-maker
- **Environment** – system it interacts with
- **State (S)** – current situation
- **Action (A)** – choices available
- **Reward (R)** – feedback signal
- **Policy (π)** – strategy learned

🎯 Effectiveness in Decision-Making

✅ Strengths

1. Sequential Decision Optimization

- RL considers **long-term consequences**, not just immediate outcomes
- Ideal for multi-step problems (e.g., navigation, games)

2. Learning Without Labeled Data

- No need for predefined outputs
- Learns directly from **experience**

3. Adaptability

- Continuously improves with new interactions
- Handles **dynamic and uncertain environments**

4. Handles Complex State Spaces

- With Deep RL, can manage high-dimensional problems (images, sensor data)

⚠️ Limitations

1. High Data Requirement

- Needs large number of interactions (episodes)
- Training can be time-consuming

2. Exploration vs Exploitation Trade-off

- Balancing trying new actions vs using known best actions is challenging

3. Reward Design Complexity

- Poorly designed rewards → undesired behavior

4. Computational Cost

- Requires significant computational resources (especially Deep RL)

🌍 Real-World Applications

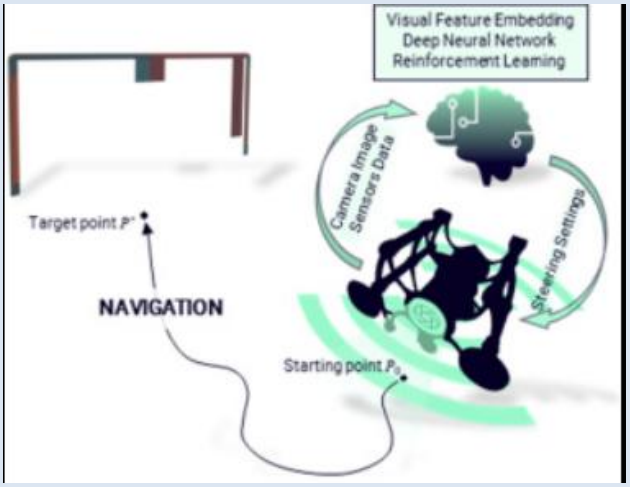
🚗 1. Autonomous Vehicles

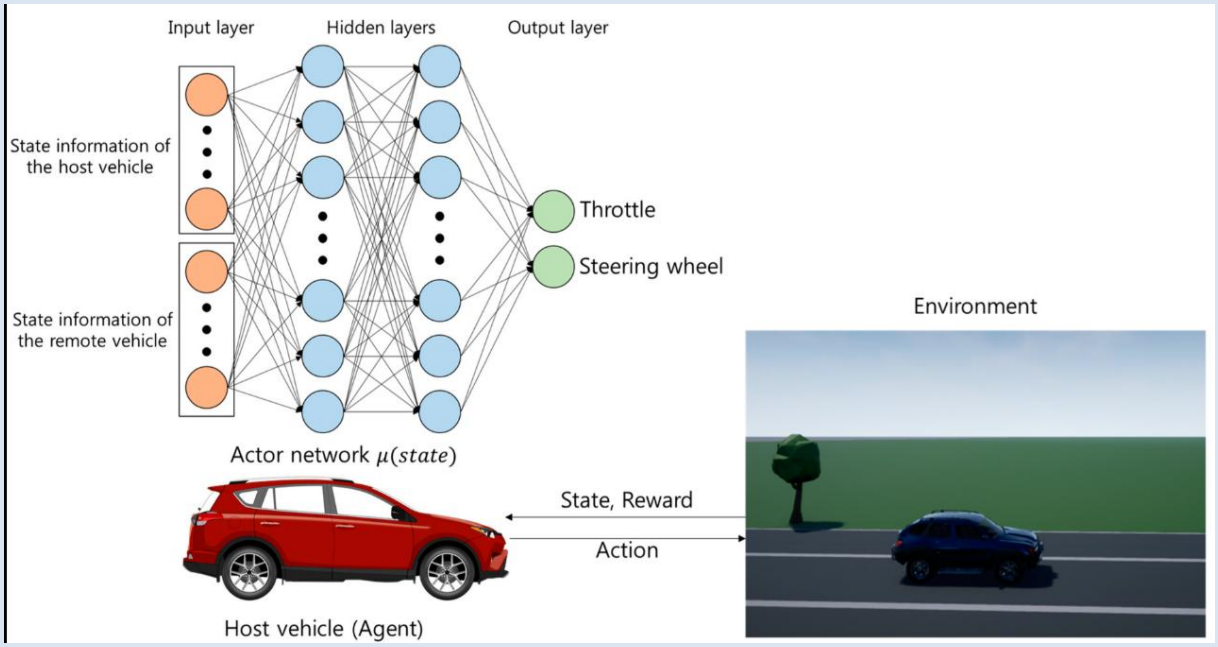
🎮 2. Game Playing

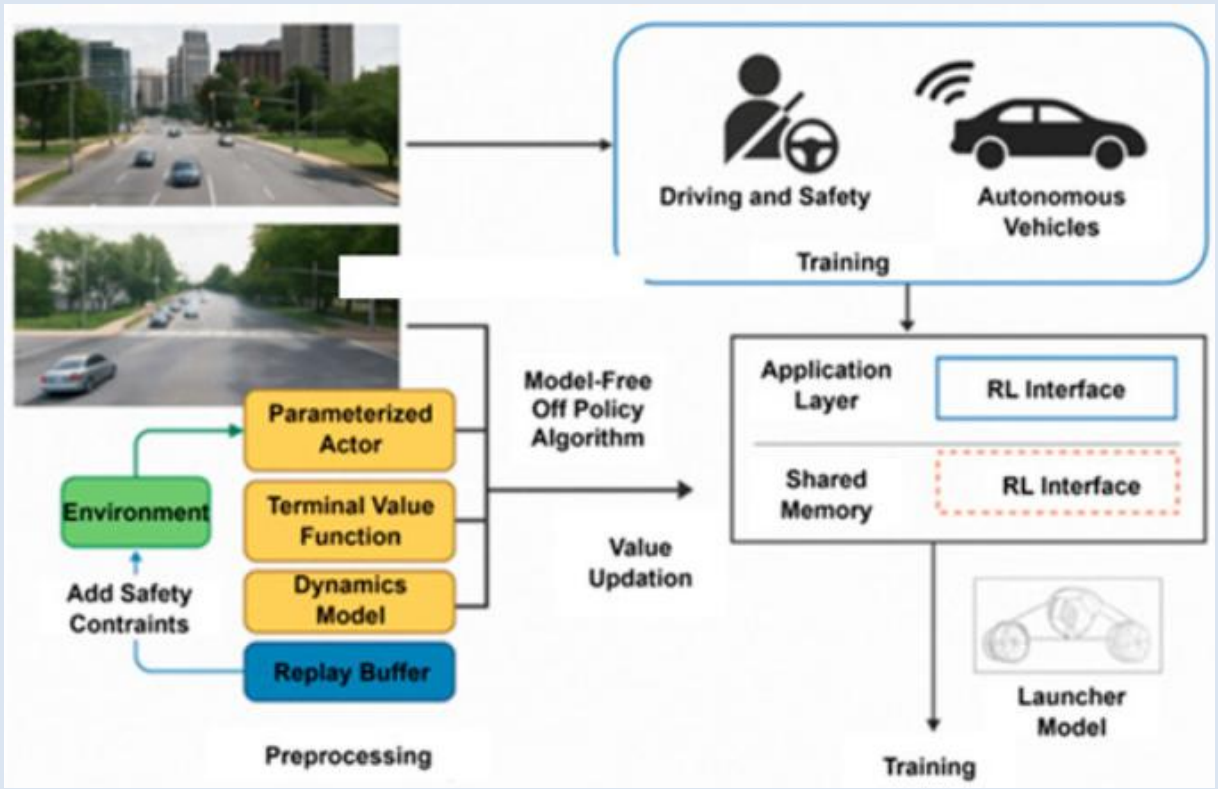
🏭 3. Robotics & Automation

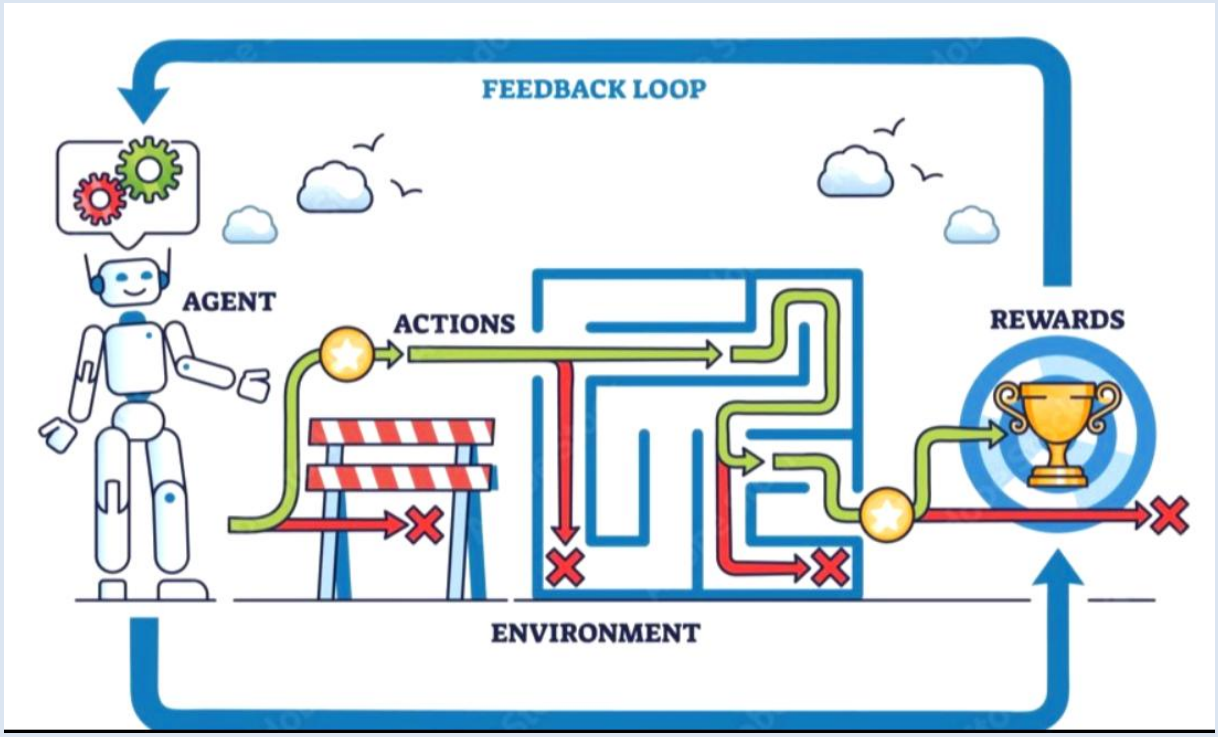
💰 4. Finance & Trading

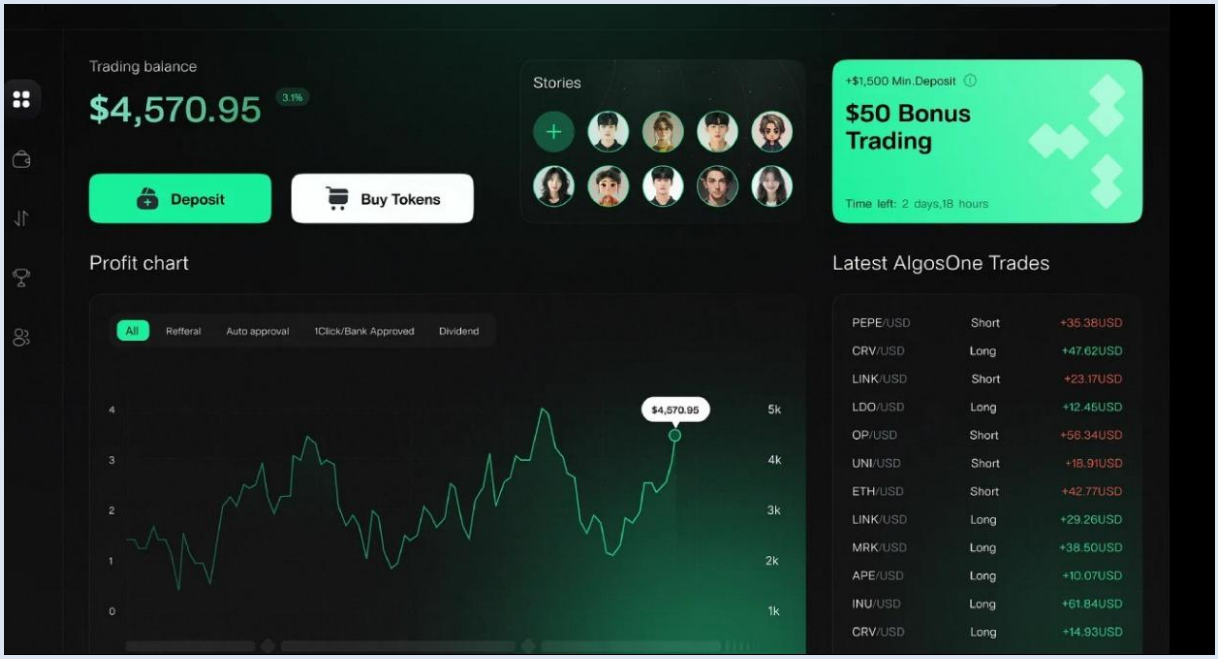
🛡️ 5. Healthcare

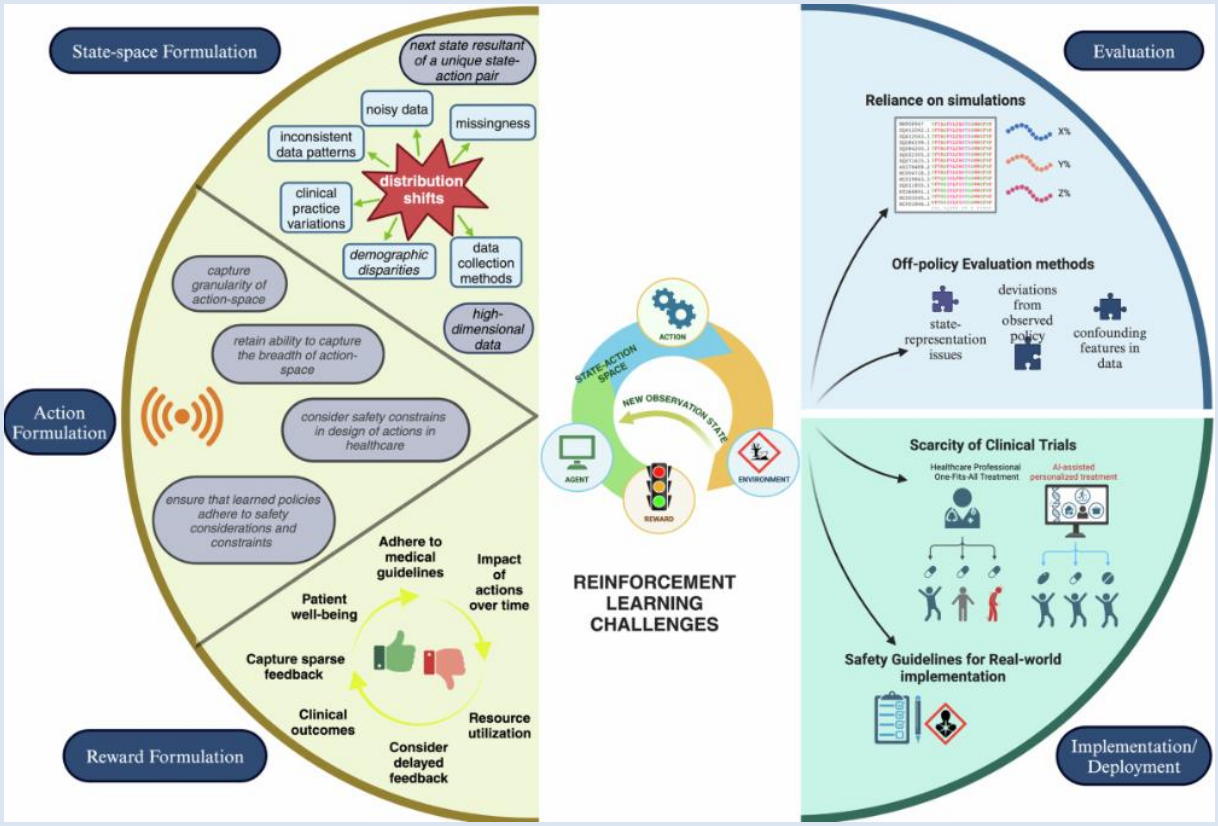

Visual Feature Embedding
Deep Neural Network
Reinforcement Learning
Camera Image
Sensors Data
Steering Settings
Target point P'
NAVIGATION
Starting point P0

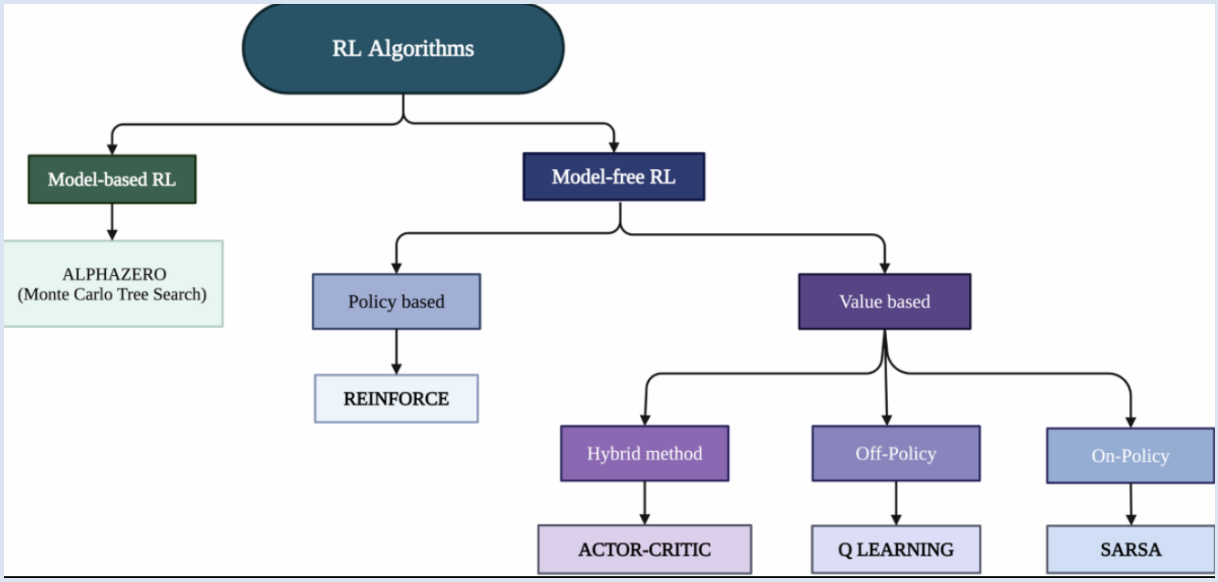

Input layer
Hidden layers
Output layer
State information of the host vehicle
State information of the remote vehicle
Throttle
Steering wheel
Actor network μ(state)
Environment
State, Reward
Action
Host vehicle (Agent)


Driving and Safety
Autonomous Vehicles
Training
Application Layer
RL Interface
Shared Memory
RL Interface
Model-Free Off Policy Algorithm
Parameterized Actor
Terminal Value Function
Dynamics Model
Replay Buffer
Environment
Add Safety Contraints
Value Updation
Launcher Model
Preprocessing
Training


FEEDBACK LOOP
AGENT
ACTIONS
REWARDS
ENVIRONMENT

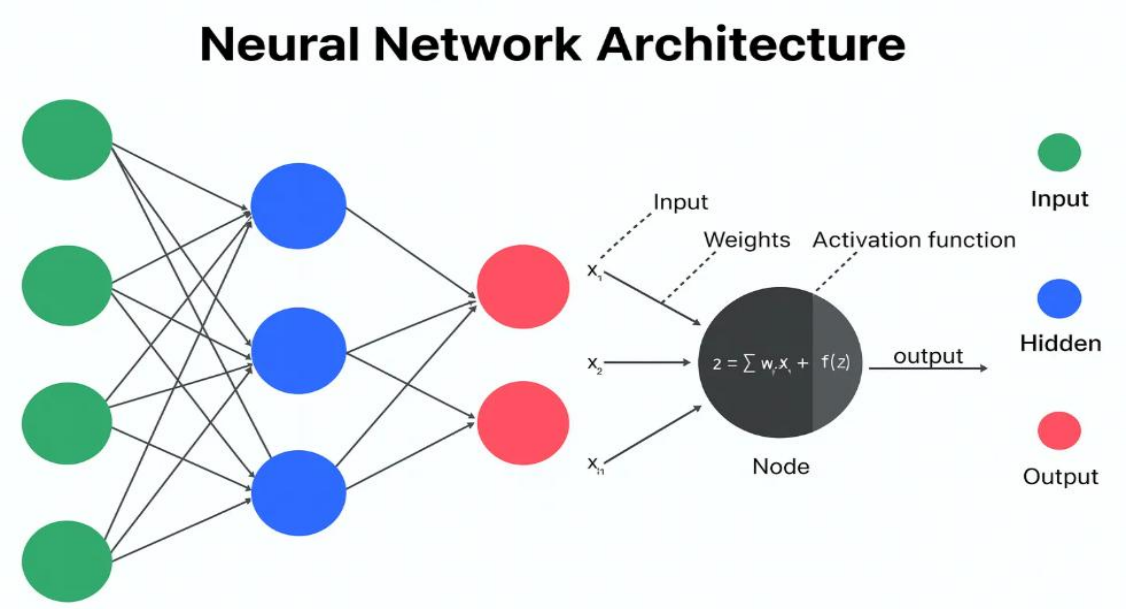

Trading balance
$4,570.95
3.1%
Deposit
Buy Tokens
Stories
+$1,500 Min.Deposit
$50 Bonus Trading
Time left: 2 days,18 hours
Profit chart
All
Refferal
Auto approval
1Click/Bank Approved
Dividend
$4,570.95
Latest AlgosOne Trades
PEPE/USD Short +35.38USD
CRV/USD Long +47.62USD
LINK/USD Short +23.17USD
LDO/USD Long +12.45USD
OP/USD Short +56.34USD
UNI/USD Short +18.91USD
ETH/USD Short +42.77USD
LINK/USD Long +29.26USD
MRK/USD Long +38.50USD
APE/USD Long +10.07USD
INU/USD Long +61.84USD
CRV/USD Long +14.93USD

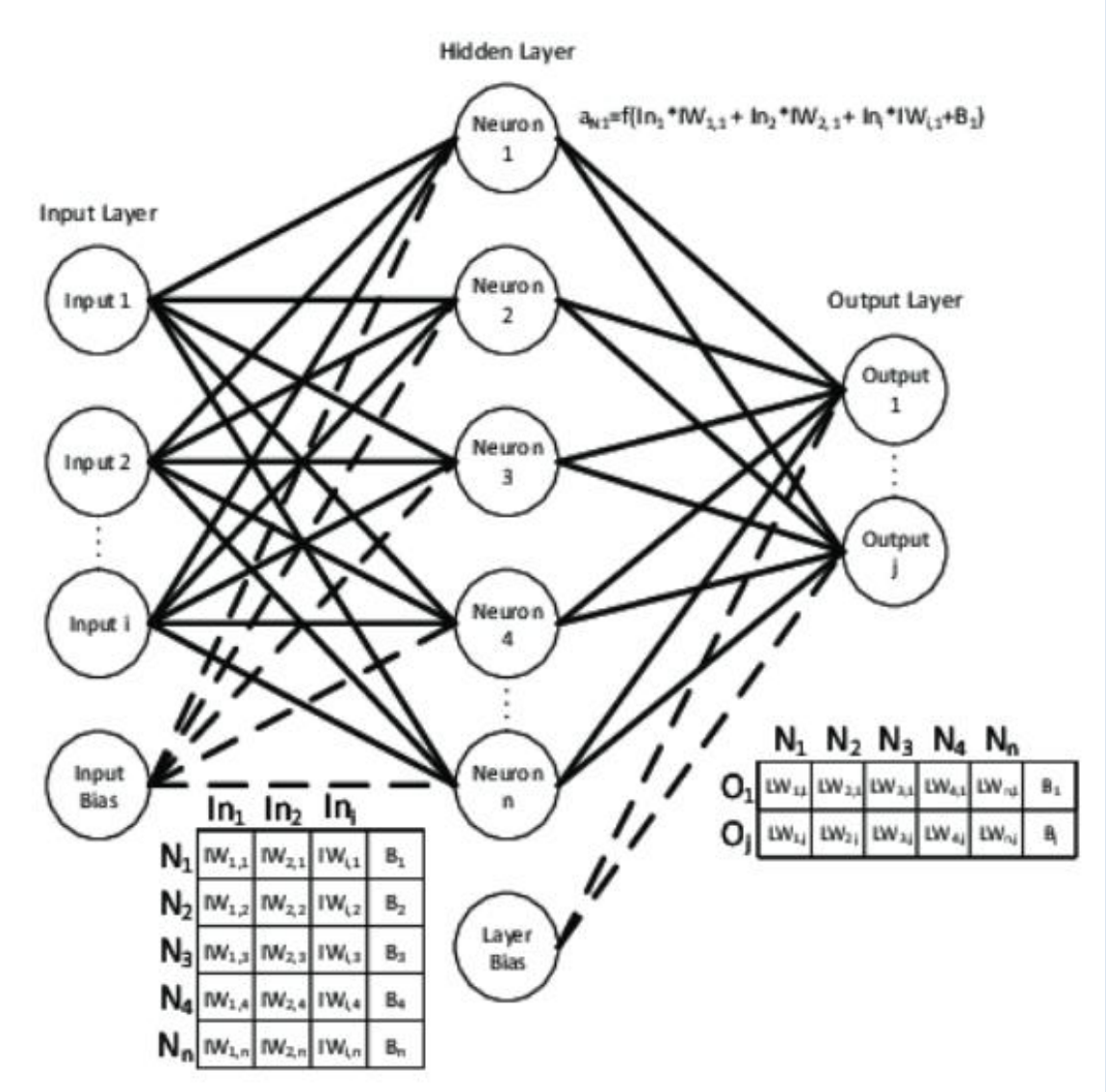

State-space Formulation
next state resultant of a unique state-action pair
noisy data
missingness
inconsistent data patterns
distribution shifts
clinical practice variations
demographic disparities
data collection methods
high-dimensional data
capture granularity of action-space
retain ability to capture the breadth of action-space
Action Formulation
consider safety constrains in design of actions in healthcare
ensure that learned policies adhere to safety considerations and constraints
Adhere to medical guidelines
Impact of actions over time
Patient well-being
Capture sparse feedback
Clinical outcomes
Resource utilization
Consider delayed feedback
Reward Formulation
ACTION
STATE-ACTION SPACE
NEW OBSERVATION STATE
AGENT
ENVIRONMENT
REWARD
REINFORCEMENT LEARNING CHALLENGES
Evaluation
Reliance on simulations
Off-policy Evaluation methods
state-representation issues
deviations from observed policy
confounding features in data
Scarcity of Clinical Trials
Healthcare Professional One-Fits-All Treatment
AI-assisted personalized treatment
Safety Guidelines for Real-world implementation
Implementation/ Deployment

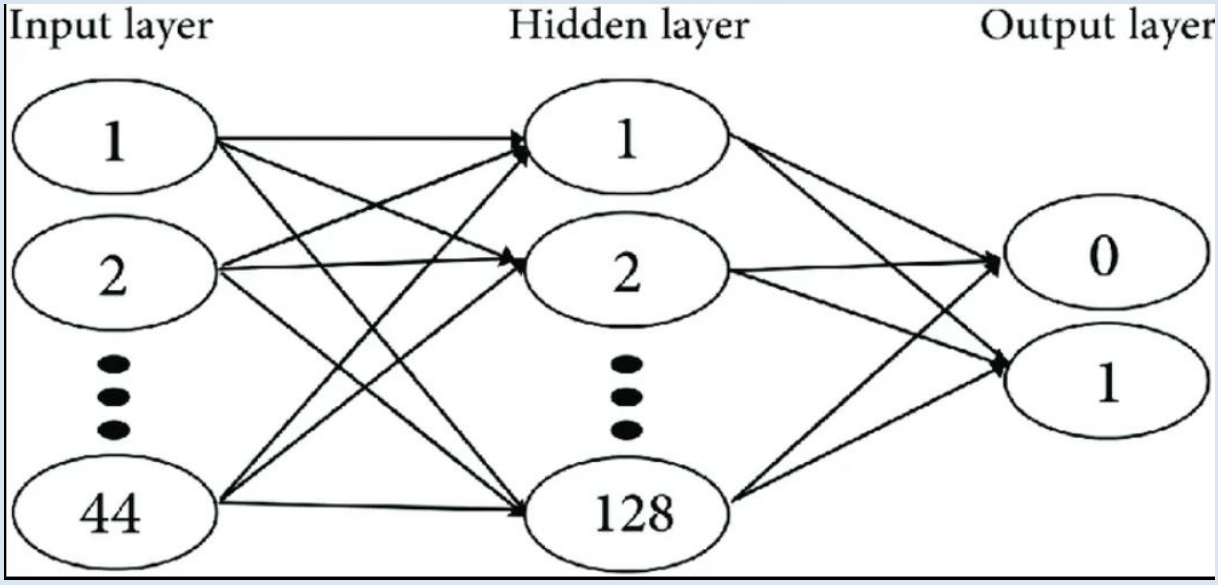

RL Algorithms
Model-based RL
Model-free RL
ALPHAZERO
(Monte Carlo Tree Search)
Policy based
Value based
REINFORCE
Hybrid method
Off-Policy
On-Policy
ACTOR-CRITIC
Q LEARNING
SARSA

Develop a basic neural network model for solving a classification problem. Explain its architecture and how learning enhances performance.

A **Neural Network (NN)** is a computational model inspired by the human brain, used to learn patterns from data. For **classification**, it predicts **class labels** (e.g., spam/not spam, disease positive/negative).







Main Components:

1. **Input Layer**
   - Receives features (e.g., age, income, marks)
2. **Hidden Layer(s)**
   - Performs transformations using weights and activation functions
   - Extracts complex patterns
3. **Output Layer**
   - Produces final prediction
   - Uses:
     - **Sigmoid** (binary classification)
     - **Softmax** (multi-class classification)

Example: Binary Classification Model

Problem:

Classify whether a student will **Pass (1)** or **Fail (0)**

Model Design:

- Input: 2 features (study hours, attendance)
- Hidden Layer: 3 neurons (ReLU activation)
- Output Layer: 1 neuron (Sigmoid)

Training Process (Learning)

Steps:

1. **Forward Propagation**
   - Input → Hidden → Output → Prediction
2. **Loss Calculation**
   - Compare predicted vs actual output
   - Example: Binary Cross-Entropy Loss
3. **Backpropagation**
   - Compute gradients (error signals)
4. **Weight Update**
   - Adjust weights using **Gradient Descent**

📈 How Learning Enhances Performance

🔹 1. Error Minimization

- Repeated training reduces prediction error
- Loss decreases over epochs

🔹 2. Feature Learning

- Hidden layers automatically learn important patterns
- No need for manual feature engineering

🔹 3. Generalization

- Learns to predict unseen data accurately
- Avoids memorization (overfitting) with techniques like regularization

🔹 4. Adaptive Improvement

- Model improves with:
  - More data
  - Better hyperparameters
  - More training iterations

A basic neural network for classification consists of **input, hidden, and output layers**. Through **forward propagation, loss computation, and backpropagation**, the network learns optimal weights. This learning process progressively enhances performance, enabling accurate classification of new data.

CASE STUDIES

AIML for Mechanical Engineering

"The brain has about 100 trillion connections, making it the most complex machine we know."

—Geoffrey Hinton

1. Healthcare Case Study – Disease Prediction

📌 Problem:

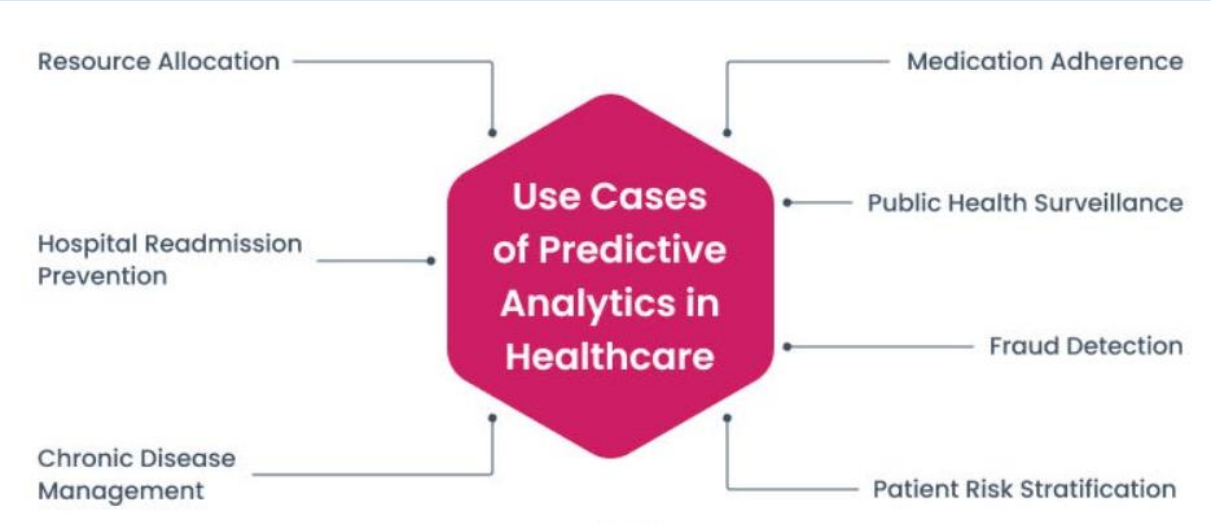
Early detection of diseases like **heart disease or diabetes**

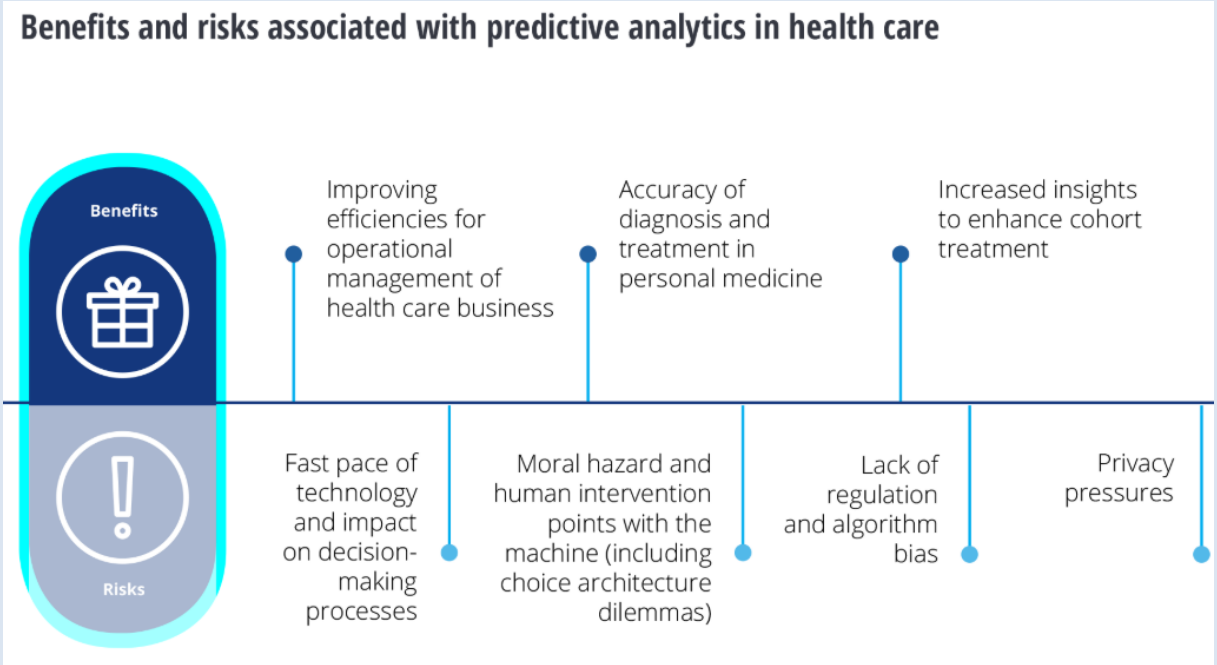
🤖 ML Solution:

- Algorithms: Logistic Regression, Decision Trees
- Input: Patient data (age, BP, cholesterol, lifestyle)
- Output: Disease risk (Yes/No)

✅ Outcome:

- Early diagnosis
- Reduced mortality rate
- Better treatment planning





💰 2. Finance Case Study – Fraud Detection

📌 Problem:

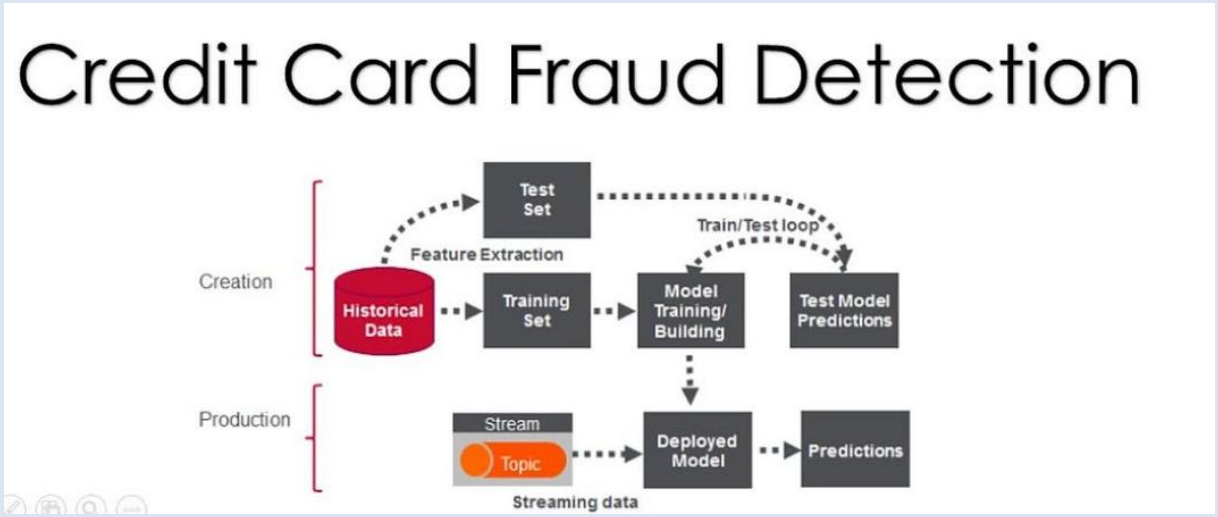
Detect fraudulent credit card transactions

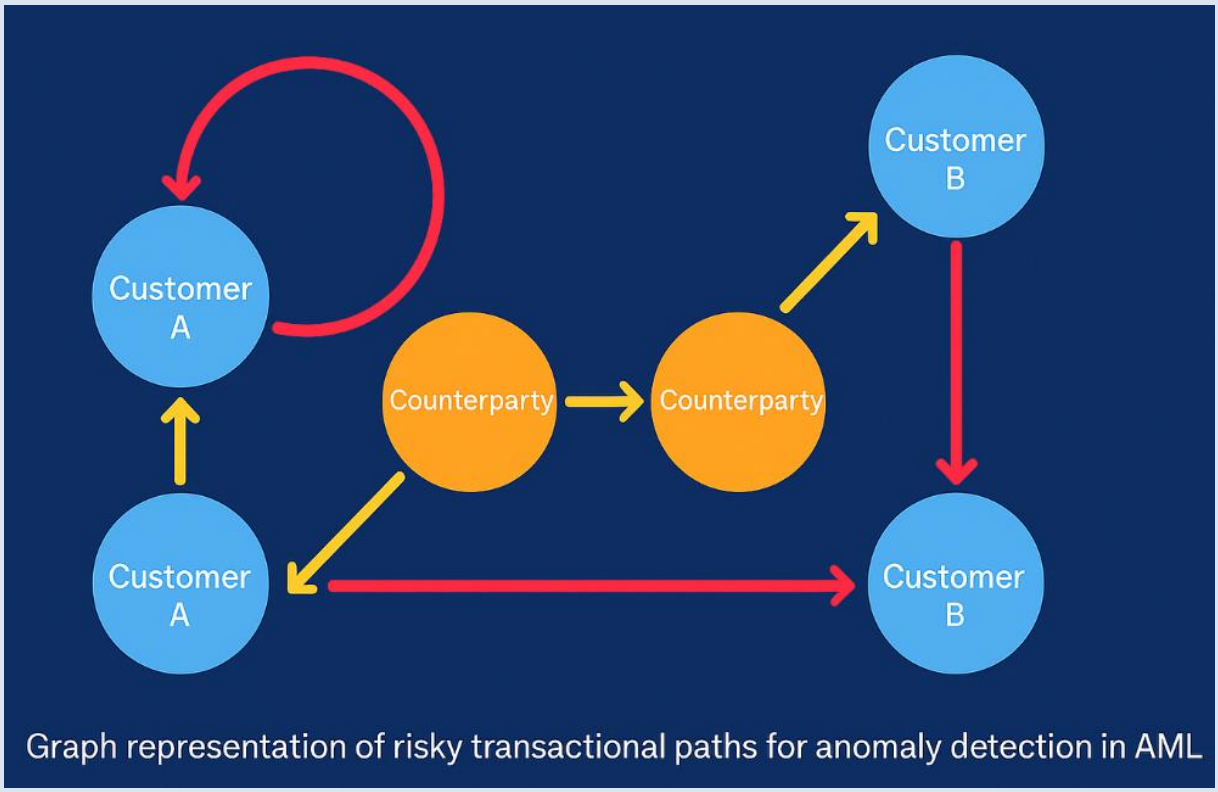
🤖 ML Solution:

- Algorithms: Random Forest, Neural Networks, Isolation Forest
- Input: Transaction patterns (amount, location, frequency)
- Output: Fraud / Genuine

✅ Outcome:

- Real-time fraud alerts
- Reduced financial losses
- Improved customer trust





Instagram Fraud Detection

The dataset contains information on 696 Instagram accounts monitored in the United States from February 1st, 2020 to February 10th, 2020, with a focus on detecting and monitoring fake accounts for fraud management purpose. After data cleaning, the dataset was free of null values and outliers.

Number of Fake Accounts

2020-02-10

347

347 fake account detected

Total Number of Suspicious Account

695

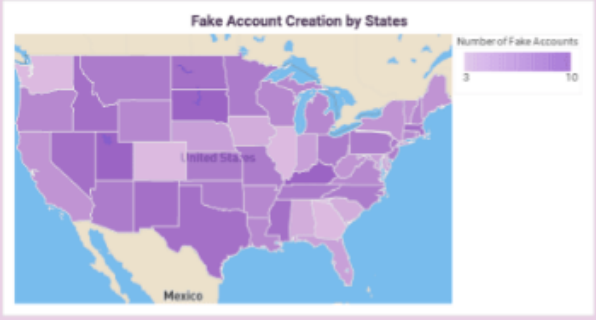
suspicious accounts are scanned

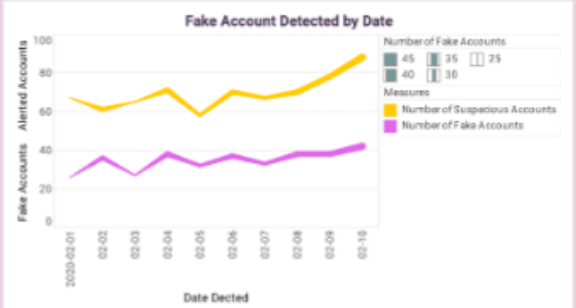
Drill Down to Detection Date

2020-02-01..2020-02-10

Drill Down to IP Location

- Alabama
- Alaska
- Arizona
- Arkansas
- California
- Colorado
- Connecticut
- Delaware





🏭 3. Manufacturing Case Study – Predictive Maintenance

📌 Problem:

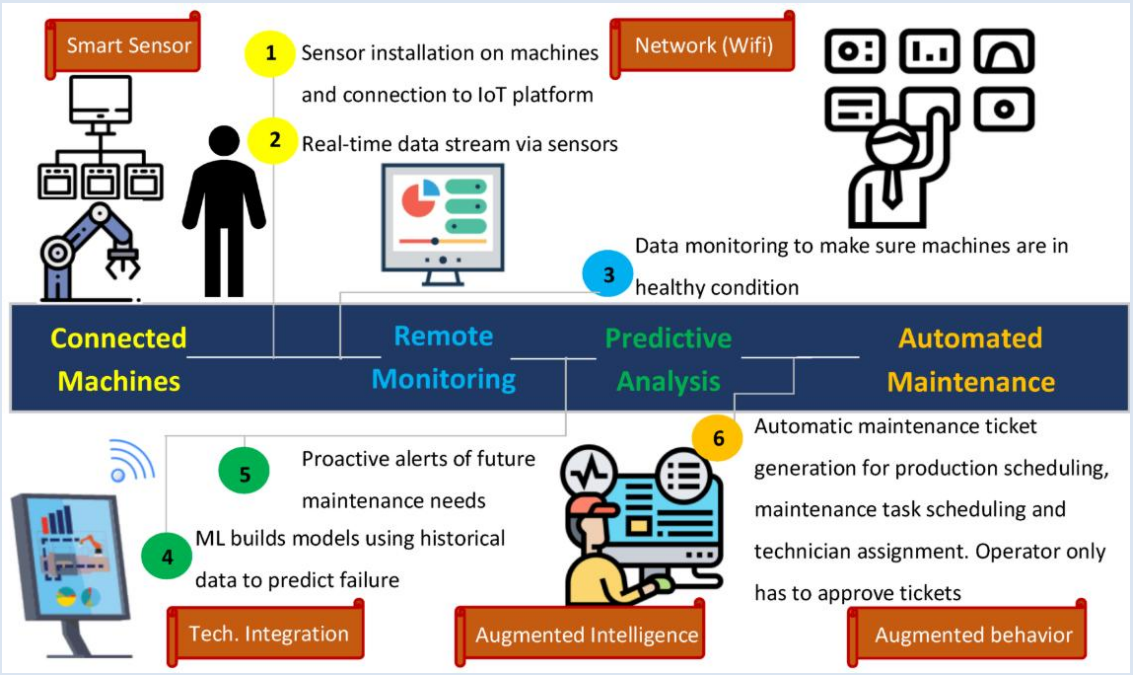
Unexpected machine failures causing downtime

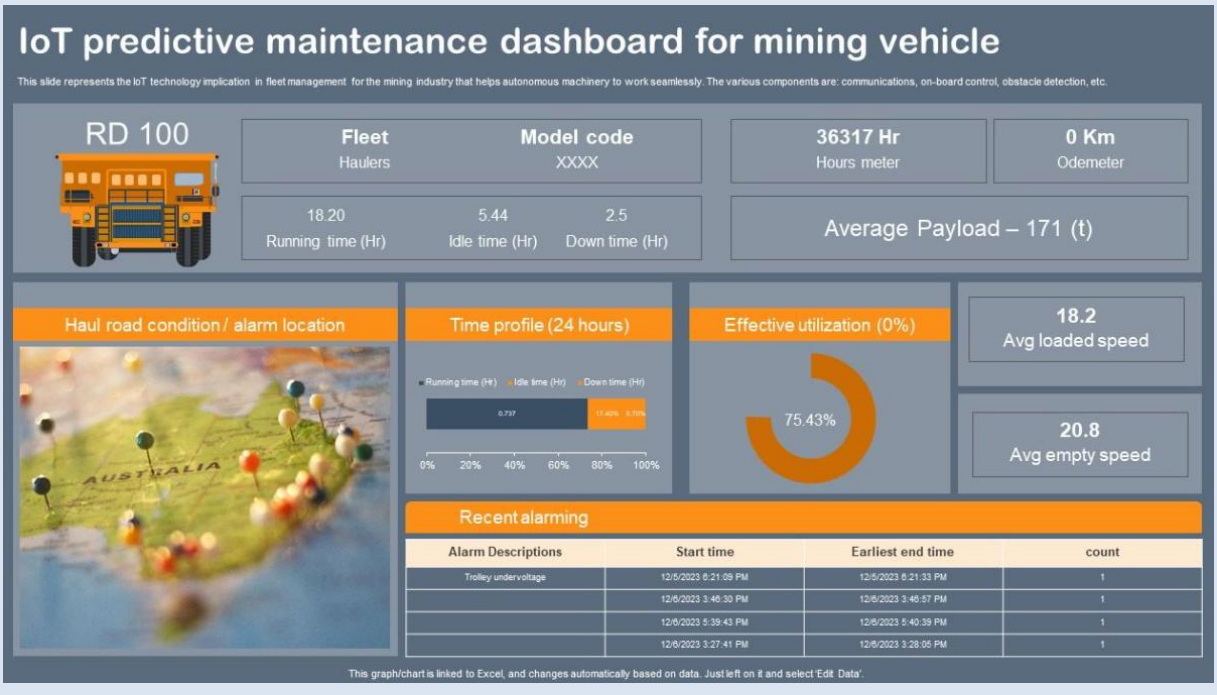
🤖 ML Solution:

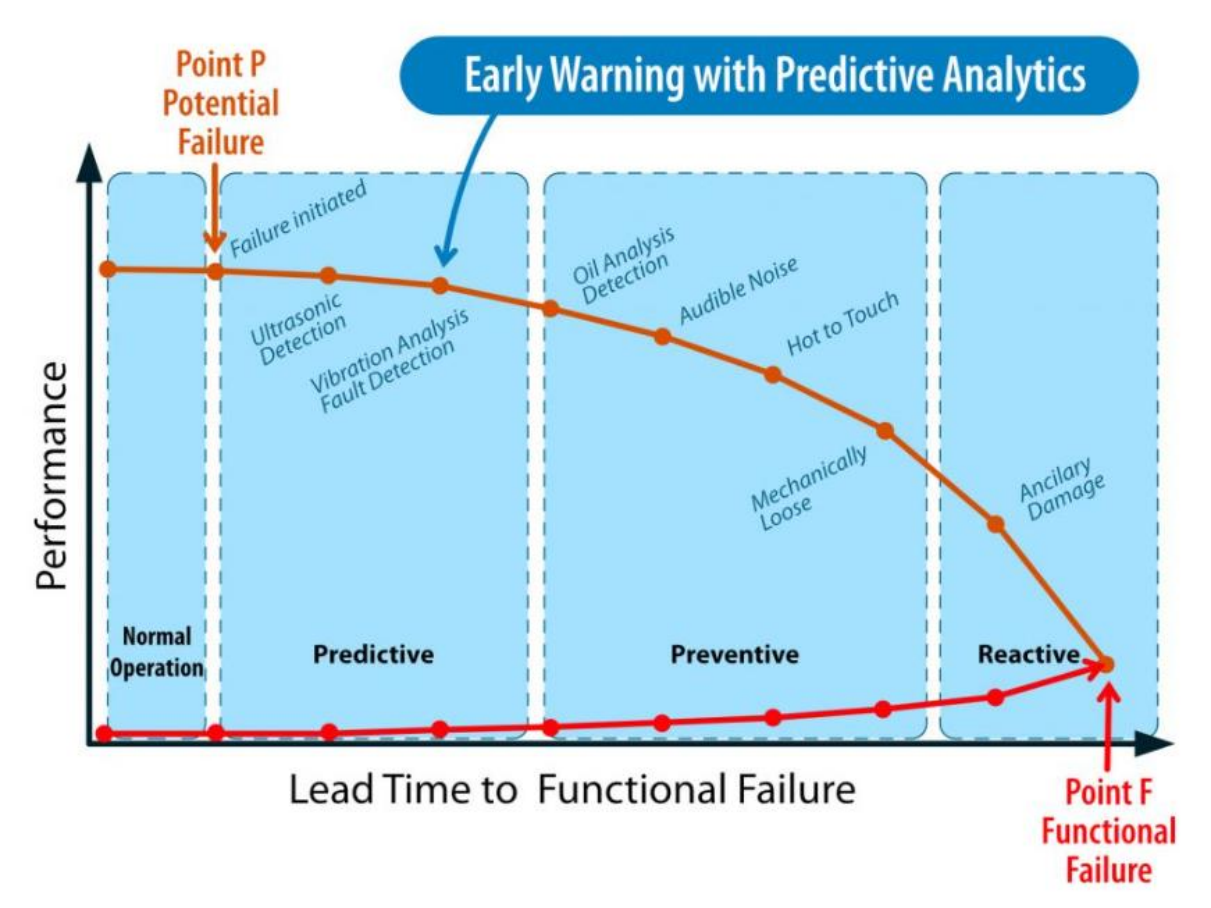
- Algorithms: Regression, Time Series Models
- Input: Sensor data (temperature, vibration, pressure)
- Output: Failure prediction

✅ Outcome:

- Reduced downtime
- Cost savings
- Increased productivity






Point P
Potential
Failure
Early Warning with Predictive Analytics
Failure initiated
Ultrasonic
Detection
Vibration Analysis
Fault Detection
Oil Analysis
Detection
Audible Noise
Hot to Touch
Mechanically
Loose
Ancilary
Damage
Performance
Normal
Operation
Predictive
Preventive
Reactive
Lead Time to Functional Failure
Point F
Functional
Failure

🎓 4. Education Case Study – Student Performance Prediction

📌 Problem:

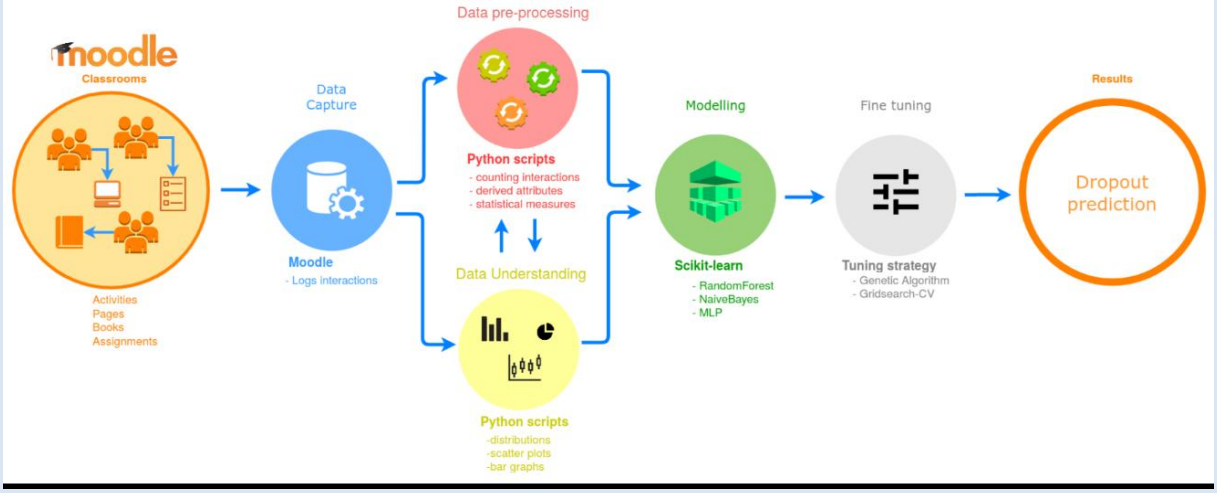
Identify students at risk of failure

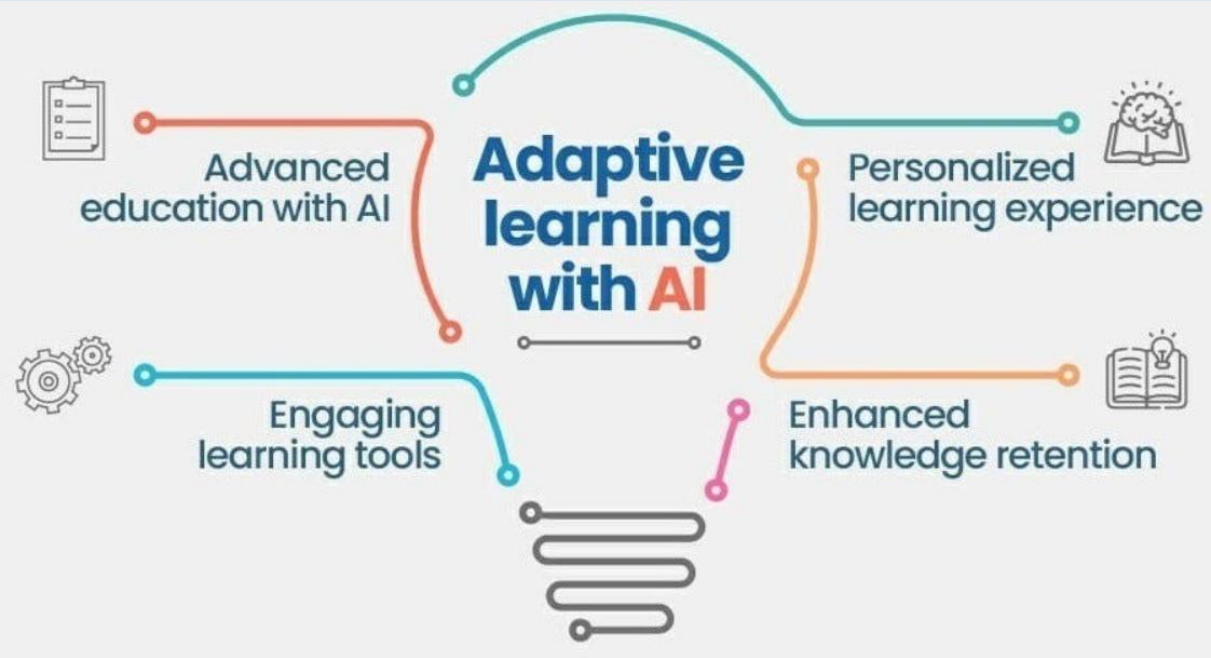
🤖 ML Solution:

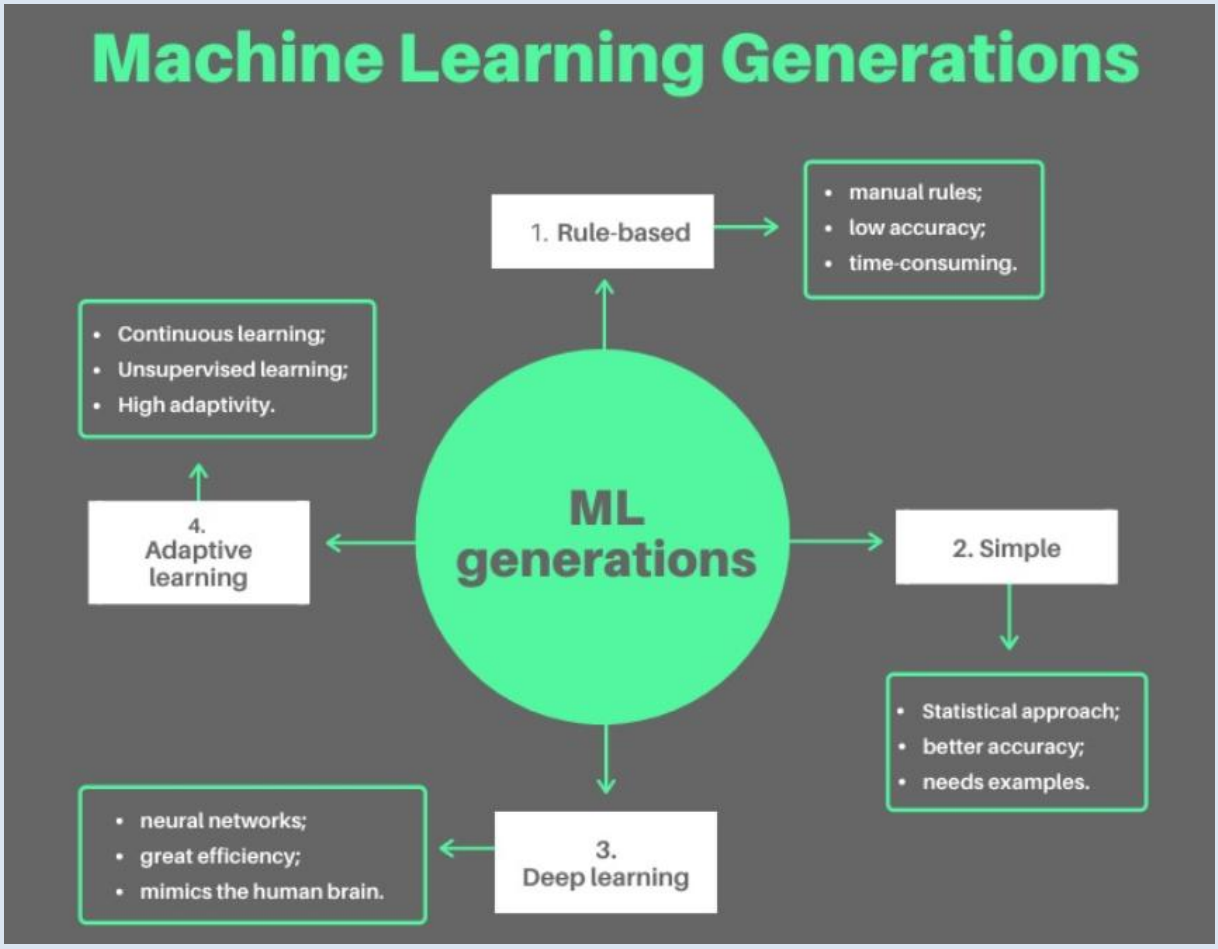
- Algorithms: Classification models (SVM, Decision Trees)
- Input: Attendance, marks, participation
- Output: Pass/Fail prediction

✅ Outcome:

- Early intervention
- Improved academic results
- Personalized support






Machine Learning Generations
ML generations
1. Rule-based
manual rules;
low accuracy;
time-consuming.
2. Simple
Statistical approach;
better accuracy;
needs examples.
3. Deep learning
neural networks;
great efficiency;
mimics the human brain.
4. Adaptive learning
Continuous learning;
Unsupervised learning;
High adaptivity.

🛒 5. E-Commerce Case Study – Recommendation System

📌 Problem:

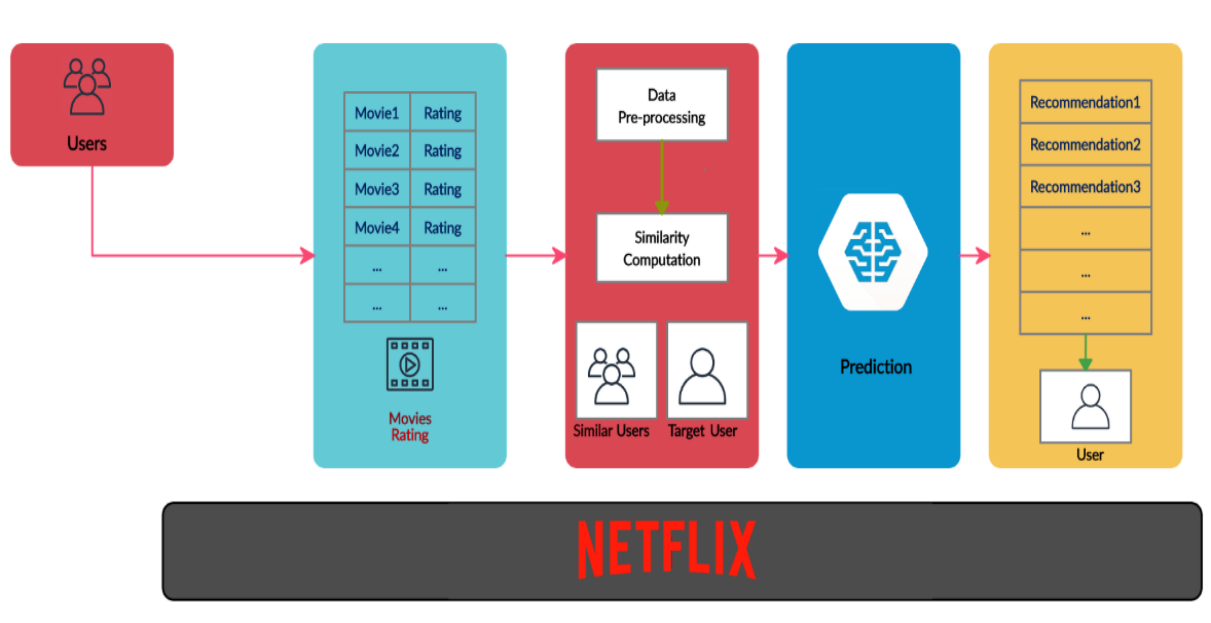
Suggest relevant products to users

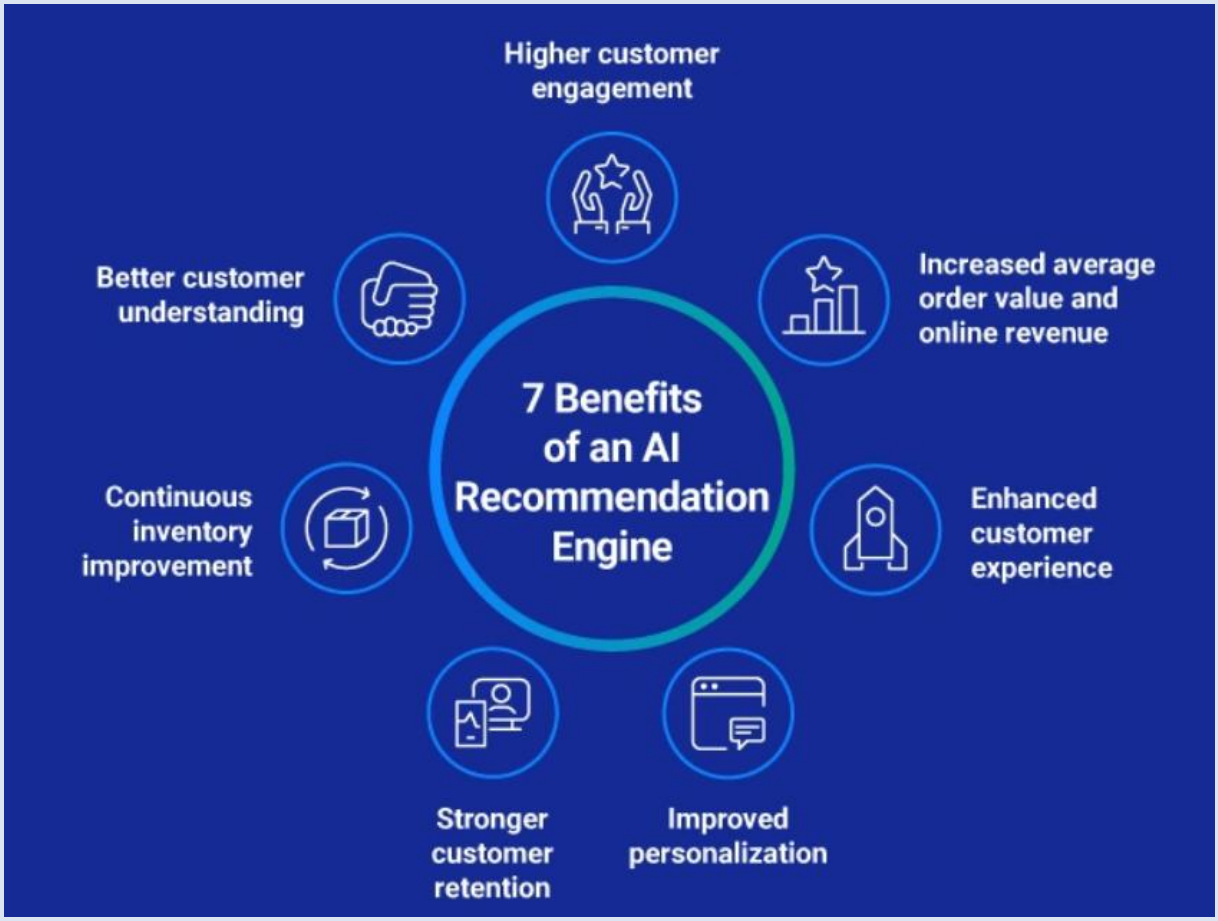
🤖 ML Solution:

- Algorithms: Collaborative Filtering, Matrix Factorization
- Input: User behavior, purchase history
- Output: Personalized recommendations

✅ Outcome:

- Increased sales
- Better user experience
- Customer retention




Higher customer engagement
Better customer understanding
Increased average order value and online revenue
7 Benefits of an AI Recommendation Engine
Continuous inventory improvement
Enhanced customer experience
Stronger customer retention
Improved personalization

🚗 6. Transportation Case Study – Traffic Prediction

📌 Problem:

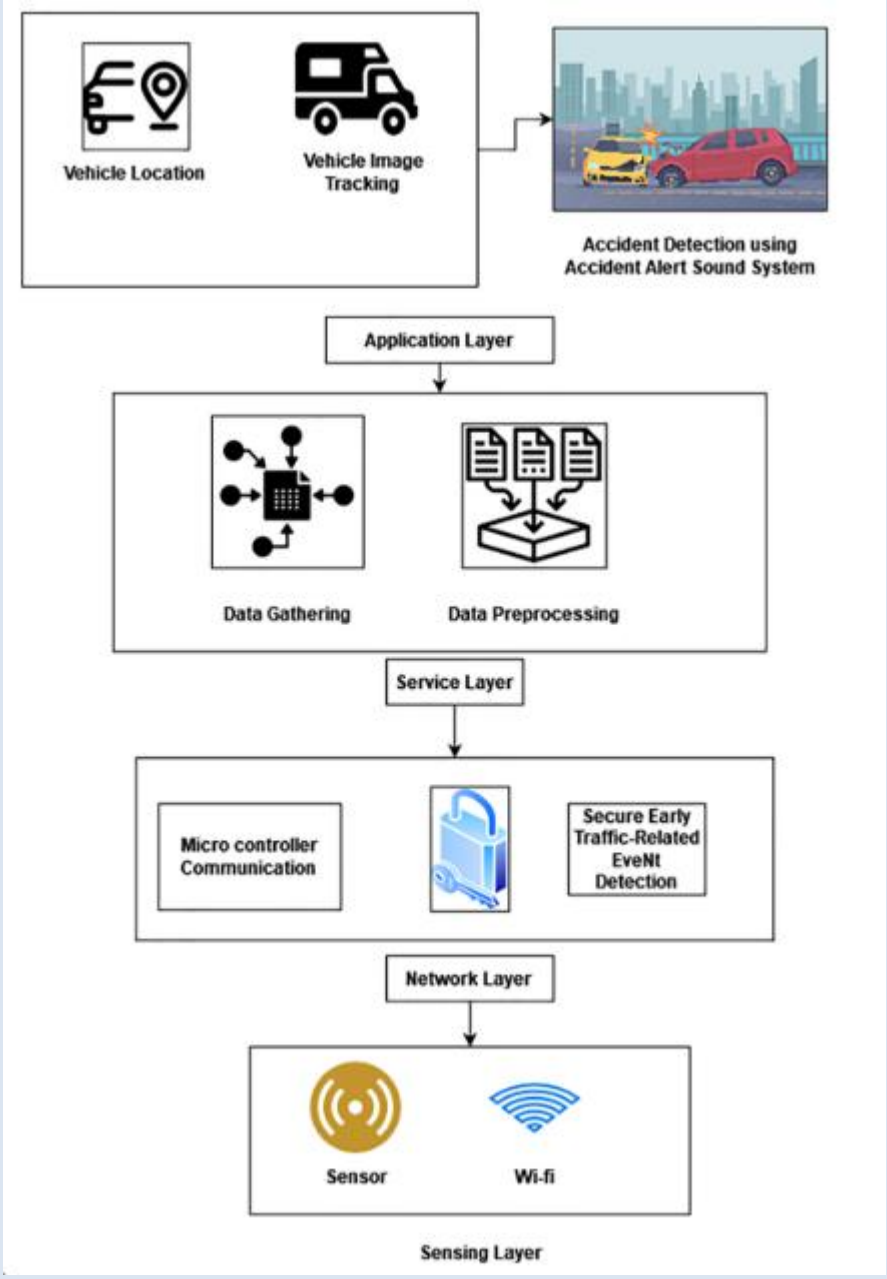
Predict traffic congestion and travel time

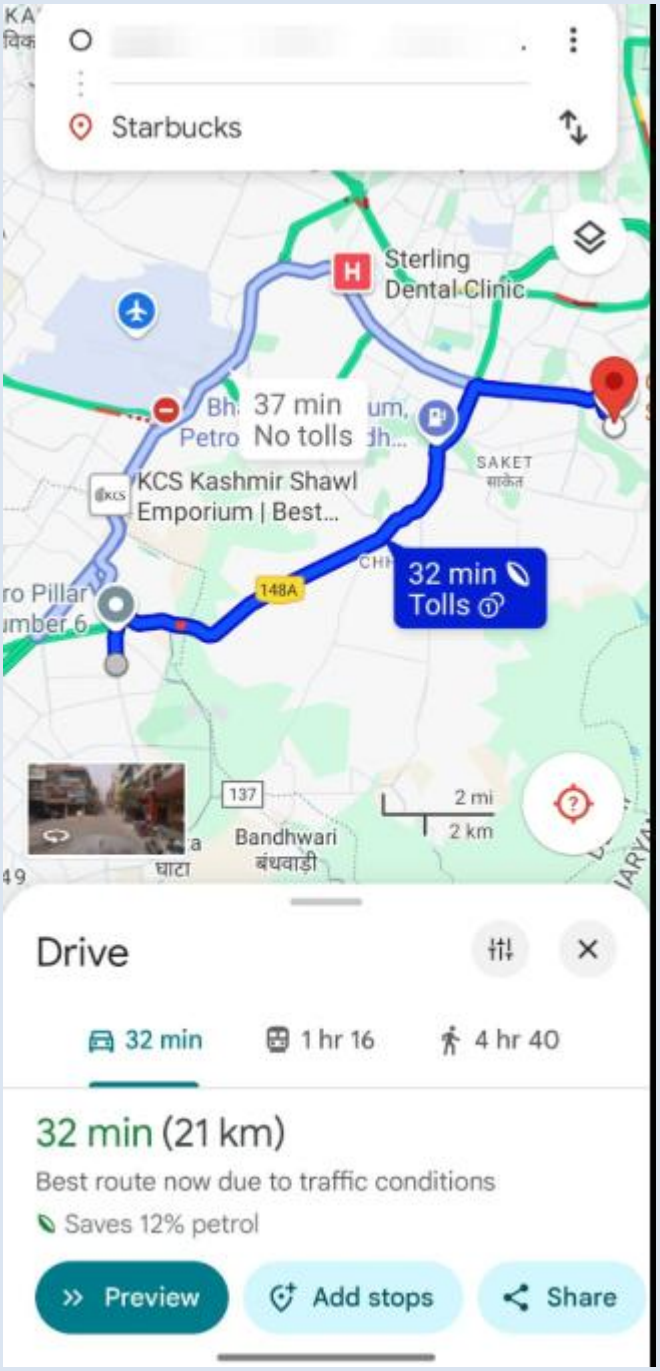
❓ ML Solution:

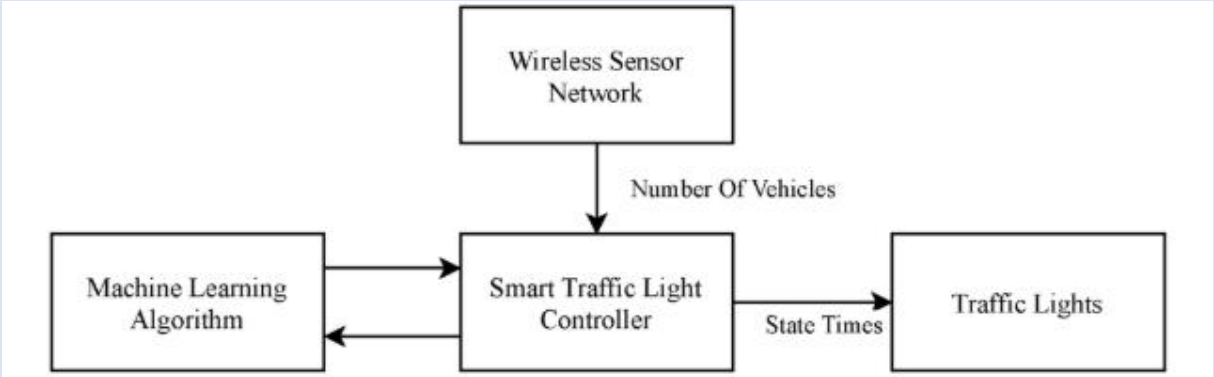
- Algorithms: Time Series, Deep Learning
- Input: GPS data, traffic sensors, historical data
- Output: Traffic forecasts

✅ Outcome:

- Optimized routes
- Reduced travel time
- Smart city planning

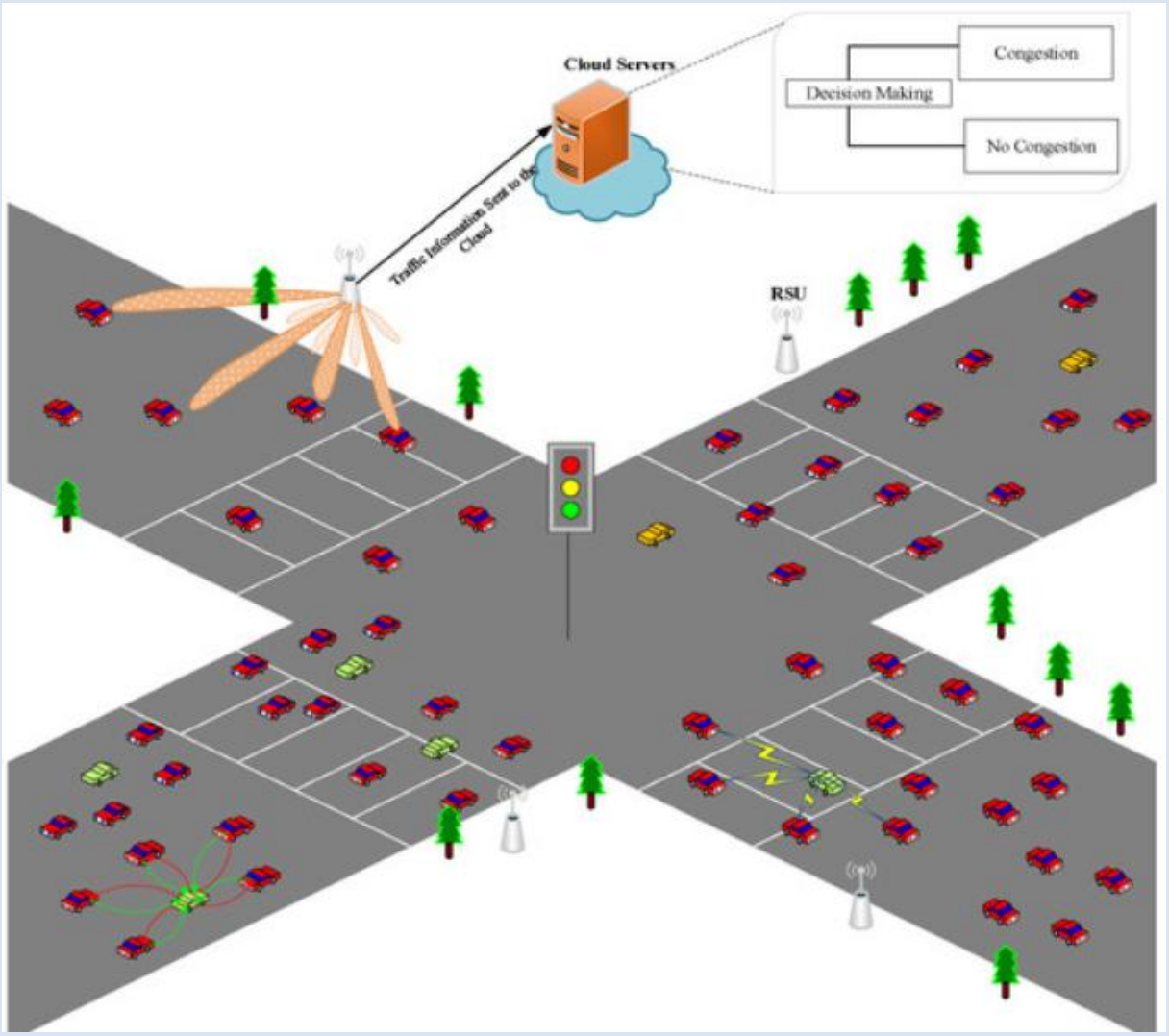



Starbucks
Sterling
Dental Clinic
37 min
No tolls
Petro
KCS Kashmir Shawl
Emporium | Best...
SAKET
32 min
Tolls
148A
137
Bandhwari
2 mi
2 km
Drive
32 min
1 hr 16
4 hr 40
32 min (21 km)
Best route now due to traffic conditions
Saves 12% petrol
Preview
Add stops
Share


Wireless Sensor Network
Number Of Vehicles
Machine Learning Algorithm
Smart Traffic Light Controller
State Times
Traffic Lights

(a)


Sensor
Wireless communication


Cloud Servers
Congestion
Decision Making
No Congestion
Traffic Information Sent to the Cloud
RSU

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)

QUESTION BANK

Year / Semester: III B.Tech VI Semester **Regulation: R23**

Subject and Code: 23MEC361T ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR MECHANICAL ENGINEERING

SYLLABUS

UNIT –1: INTRODUCTION TO ARTIFICIAL INTELLIGENCE AND PROBLEM-SOLVING AGENT:

Problems of AI, AI technique, Tic–Tac–Toe problem. Intelligent Agents, Agents &environment, nature of environment, structure of agents, goal-based agents, utility-based agents, learning agents. Defining the problem as state space search, production system, problem characteristics, and issues in the design of search programs.

UNIT –2: SEARCH TECHNIQUES:

Problem solving agents, searching for solutions; uniform search strategies: breadth first search, depth first search, depth limited search, bidirectional search, comparing uniform search strategies. Heuristic search strategies Greedy best-first search, A*search, AO* search, memory bounded heuristic search: local search algorithms & optimization problems: Hill climbing search, simulated annealing search, local beam search.

UNIT –3: CONSTRAINT SATISFACTION PROBLEMS AND GAME THEORY:

Local search for constraint satisfaction problems. Adversarial search, Games, optimal decisions & strategies in games, the minimal search procedure, alpha-beta pruning, additional refinements, iterative deepening

UNIT –4: KNOWLEDGE &REASONING: STATISTICAL REASONING:

Probability an Bays‘Theorem, Certainty Factors and Rule-Base Systems, Bayesian Networks, Dumpster-Shafer Theory, Fuzzy Logic. AI for knowledge representation, rule-based knowledge representation, procedural and declarative knowledge, Logic programming, forward and backward reasoning.

UNIT –5: INTRODUCTION TO MACHINE LEARNING:

Exploring sub-discipline of AI: Machine Learning, Supervised learning, Unsupervised learning, Reinforcement learning, Classification problems, Regression problems, cluster problems, Introduction to neural networks and deep learning.

Prepared by Dr.S.Rajesh

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)

QUESTION BANK

Year / Semester: III B.Tech VI Semester **Regulation: R23**

Subject and Code: 23MEC361T ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR MECHANICAL ENGINEERING

Max Marks: 10

S.No.	CO	Questions	BT
Unit I: (INTRODUCTION TO ARTIFICIAL INTELLIGENCE AND PROBLEM- SOLVING AGENT)			
1	1	Define Artificial Intelligence and explain the major problems addressed in AI.	L1
2	1	Describe the concept of an intelligent agent and list different types of agents in AI	L1
3	1	Explain the Tic–Tac–Toe problem as an AI problem, highlighting how it can be represented and solved.	L2
4	1	Discuss the nature of environments in AI, including different types such as deterministic, stochastic, static, and dynamic environments.	L2
5	1	Illustrate how a goal-based agent operates in a given environment with a suitable real-world example.	L3
6	1	Apply the concept of state space search to formulate a water jug problem	L3
7	1	Analyze the structure of intelligent agents, explaining how perception, decision-making, and action are interconnected.	L4
8	1	Compare and analyze goal-based agents and utility-based agents, highlighting their advantages and limitations.	L4
9	1	Evaluate the effectiveness of AI techniques in solving complex real-world problems compared to traditional programming methods.	L5
10	1	Critically examine the issues involved in the design of search programs, including completeness, optimality, time, and space complexity.	L5

Prepared by Dr.S.Rajesh

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)

QUESTION BANK

Year / Semester: III B.Tech VI Semester **Regulation: R23**

Subject and Code: 23MEC361T ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR MECHANICAL ENGINEERING

S.No.	CO	Questions	BT
Unit II: (SEARCH TECHNIQUES)			
1	2	Define problem-solving agents and explain the process of searching for solutions in AI.	L1
2	2	List and describe the uniform search strategies, including Breadth First Search, Depth First Search, Depth Limited Search, and Bidirectional Search.	L1
3	2	Explain the working of Breadth First Search (BFS) and Depth First Search (DFS) with suitable examples.	L2
4	2	Discuss the concept of heuristic search strategies and explain how they differ from uninformed search methods.	L2
5	2	Apply A* search algorithm to solve a path-finding problem and demonstrate how optimal solutions are obtained.	L3
6	2	Illustrate the use of Greedy Best-First Search in solving a real-world problem such as route finding.	L3
7	2	Analyze the differences between Greedy Best-First Search and A* Search, highlighting their advantages and limitations.	L4
8	2	Compare and analyze uniform search strategies: BFS, DFS, DLS, and Bidirectional Search in terms of completeness, optimality, time, and space complexity.	L4
9	2	Evaluate the performance of memory-bounded heuristic search algorithms, discussing their suitability for large-scale problems.	L5
10	2	Critically examine local search algorithms in solving optimization problems.	L5
11	2	Develop a hybrid search approach combining heuristic and local search techniques to solve complex optimization problems, explaining its advantages.	L6

Prepared by Dr.S.Rajesh

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)

QUESTION BANK

Year / Semester: III B.Tech VI Semester **Regulation: R23**

Subject and Code: 23MEC361T ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR MECHANICAL ENGINEERING

S.No.	CO	Questions	BT
Unit III: (CONSTRAINT SATISFACTION PROBLEMS AND GAME THEORY)			
1	3	Define Constraint Satisfaction Problems. Explain their key components with examples.	L1
2	3	What is adversarial search? Describe its role in game-playing AI systems.	L1
3	3	Explain local search techniques for Constraint Satisfaction Problems. How do they differ from systematic search methods?	L2
4	3	Describe the Minimax algorithm. How does it determine optimal decisions in games?	L2
5	3	Demonstrate the use of alpha-beta pruning on a game tree and show how it reduces the number of nodes evaluated.	L3
6	3	Apply the Minimax algorithm to a simple two-player game tree and determine the optimal move.	L3
7	3	Examine the role of iterative deepening in adversarial search. How does it improve decision-making under time constraints?	L4
8	3	Analyze the performance of Minimax and Alpha-Beta pruning algorithms. Compare their efficiency in terms of time complexity.	L4
9	3	Evaluate different strategies used in game-playing AI. Which strategy is most effective for complex games and why?	L5
10	3	Critically evaluate local search methods for CSPs. Discuss their advantages and limitations in real-world applications	L5
11	3	Develop a CSP model for a real-world problem and propose a suitable local search strategy and justify your choice.	L6

Prepared by Dr.S.Rajesh

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(AUTONOMOUS)

QUESTION BANK

Year / Semester: III B.Tech VI Semester **Regulation: R23**

Subject and Code: 23MEC361T ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR MECHANICAL ENGINEERING

S.No.	CO	Questions	BT
Unit IV: (KNOWLEDGE &REASONING: STATISTICAL REASONING)			
1	4	Define probability and Bayes' Theorem. Explain their significance in statistical reasoning	L1
2	4	What are certainty factors? Describe their role in rule-based systems.	L1
3	4	Explain Bayesian Networks. How do they represent uncertain knowledge in AI systems?	L2
4	4	Discuss Fuzzy Logic. How does it differ from classical (crisp) logic?	L2
5	4	Illustrate forward and backward reasoning using a rule-based system with suitable examples.	L3
6	4	Apply Bayes' Theorem to a real-world problem involving uncertainty and interpret the results	L3
7	4	Analyze the differences between procedural and declarative knowledge. How are they represented in AI?	L4
8	4	Examine the Dempster–Shafer Theory. How does it handle uncertainty differently compared to probability theory?	L4
9	4	Critically evaluate certainty factors and fuzzy logic. Discuss their advantages and limitations in real-world applications	L5
10	4	Evaluate following knowledge representation techniques: Rule-based, logic programming and Bayesian networks. Which is most effective under uncertainty and why?	L5

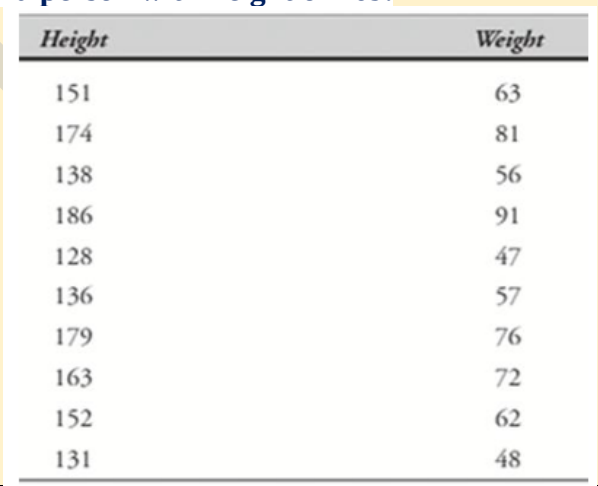
Prepared by Dr.S.Rajesh

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES (AUTONOMOUS)

QUESTION BANK

Year / Semester: III B.Tech VI Semester **Regulation: R23**

Subject and Code: 23MEC361T ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR MECHANICAL ENGINEERING

S.No.	CO	Questions	BT
		Unit V: (INTRODUCTION TO MACHINE LEARNING)	
1	5	**With suitable illustration describe the end-to-end process of machine learning lifecycle?**	L1
2	5	**Discuss the applications of Machine Learning across various domains such as healthcare, finance, manufacturing, and education.**	L1
3	5	**Explain supervised learning. How is it used to solve classification and regression problems?**	L2
4	5	**Discuss unsupervised learning. How does clustering differ from classification?**	L2
5	5	**Demonstrate clustering techniques for grouping data in an unsupervised learning scenario with an example.**	L3
6	5	**Compare and contrast Supervised, Unsupervised, and Reinforcement Learning with suitable examples.**	L4
7	5	**Analyze the differences between classification and regression problems. Provide suitable examples.**	L4
8	5	**Evaluate the effectiveness of reinforcement learning in decision-making problems. Discuss real-world applications.**	L5
9	5	**Develop a basic neural network model for solving a classification problem. Explain its architecture and how learning enhances performance.**	L6
10	5	**Compute the correlation coefficient for the given dataset; and find the relation between height and weight of ten students. Also predict the weight of a person with height of 169.** 	L3

Height	*Weight*
151	63
174	81
138	56
186	91
128	47
136	57
179	76
163	72
152	62
131	48

Note: L1-Remembering, L2-Understanding, L3-Applying, L4-Analyzing, L5-Evaluating, and L6-Creating

Prepared by Dr.S.Rajesh

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES (AUTONOMOUS)

QUESTION BANK

Year / Semester: III B.Tech VI Semester **Regulation: R23**

Subject and Code: 23MEC361T ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR MECHANICAL ENGINEERING

TEXT BOOKS:

1. **Artificial Intelligence: A Modern Approach**, Stuart Russell and Peter Norvig, Prentice Hall, 3rd Edition, 2015.
2. **Artificial Intelligence: A New Synthesis,** Nils J. Nilsson, Morgan Kaufmann Publishers, 1st Edition, 1998.

REFERENCE BOOKS:

1. **Artificial Intelligence**, Elaine Rich, Kevin Knight, and Shiva Shankar B. Nair, McGraw Hill, 3rd Edition, 2017.
2. **Introduction to Artificial Intelligence and Expert Systems,** D. W. Patterson, Pearson Education, 1st Edition, 2015.
3. **Logic and Prolog Programming,** Saroj Kaushik, New Age International Publishers, 1st Edition, 2002.
4. **Expert Systems: Principles and Programming,** Joseph C. Giarratano and Gary D. Riley, 4th Edition, 2007.

Prepared by Dr.S.Rajesh