# CSPC3003

# ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

## MODULE – 4

## LEARNING METHODS

**Course Objectives:**

- To learn the concepts of Artificial Intelligence
- To learn the methods of solving problems using Artificial Intelligence
- To introduce the concepts of Expert Systems and machine learning

## 1. What is Learning?

- Learning denotes a change in a system that enables the system to perform the same task more efficiently the next time.
- In Artificial Intelligence (AI), learning means using experience, data, examples, advice, or feedback to improve future performance.
- A learning system may change its stored knowledge, rules, model, or decision-making strategy.

### Simple example

A navigation system recommends a route. After receiving information about actual journey times, it may learn that a road is often congested at a particular time and recommend a better route in the future.

## ***Why is learning important in AI?***

- ***Improvement:*** the system can perform a task better after gaining experience.
- ***Adaptation:*** it can adjust when conditions or input data change.
- ***Reduced repeated work:*** it may remember useful results instead of calculating them again.
- ***Pattern discovery:*** it can identify patterns in examples and use them on new cases.
- ***Better decisions:*** feedback can help it avoid previous mistakes.

## ***2. What is Learning Agent?***

A Learning Agent is an intelligent agent that learns from its experience and feedback to improve its performance over time. It can modify its behavior to make better decisions in the future.

## ***Components of a Learning Agent***

A learning agent interacts with its environment through sensors and effectors. The main components are the performance element, learning element, critic, and problem generator.

### ❖ Performance Element (PE)

Chooses the action to take based on the percepts received from the environment and the knowledge available to the agent. It is the part that performs the task.

### ❖ *Learning Element (LE)*

Improves the performance element. It uses feedback and experience to decide what knowledge, rules, or behavior should be changed.

❖ ***Critic***

Compares the agent's performance with a performance standard and provides feedback to the learning element. It helps the agent identify how well it is doing.

❖ ***Problem Generator***

Suggests exploratory actions or experiments that may produce useful new experience. It helps the agent avoid relying only on familiar actions.

❖ ***Sensors***

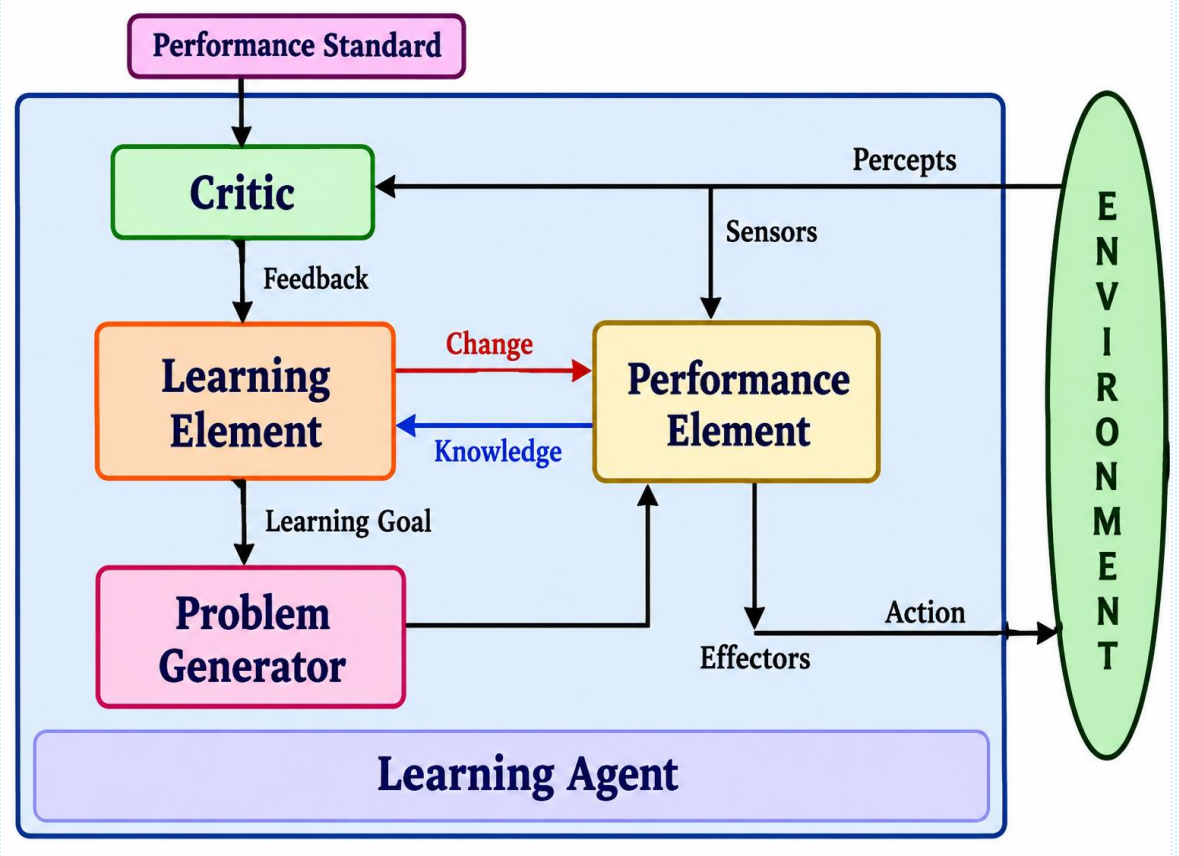
Collect information from the environment. Examples include cameras, microphones, GPS receivers, and temperature sensors.

❖ **Effectors**

Carry out the chosen actions. Examples include a robot's wheels, a car's steering and brakes, or a software agent's commands.

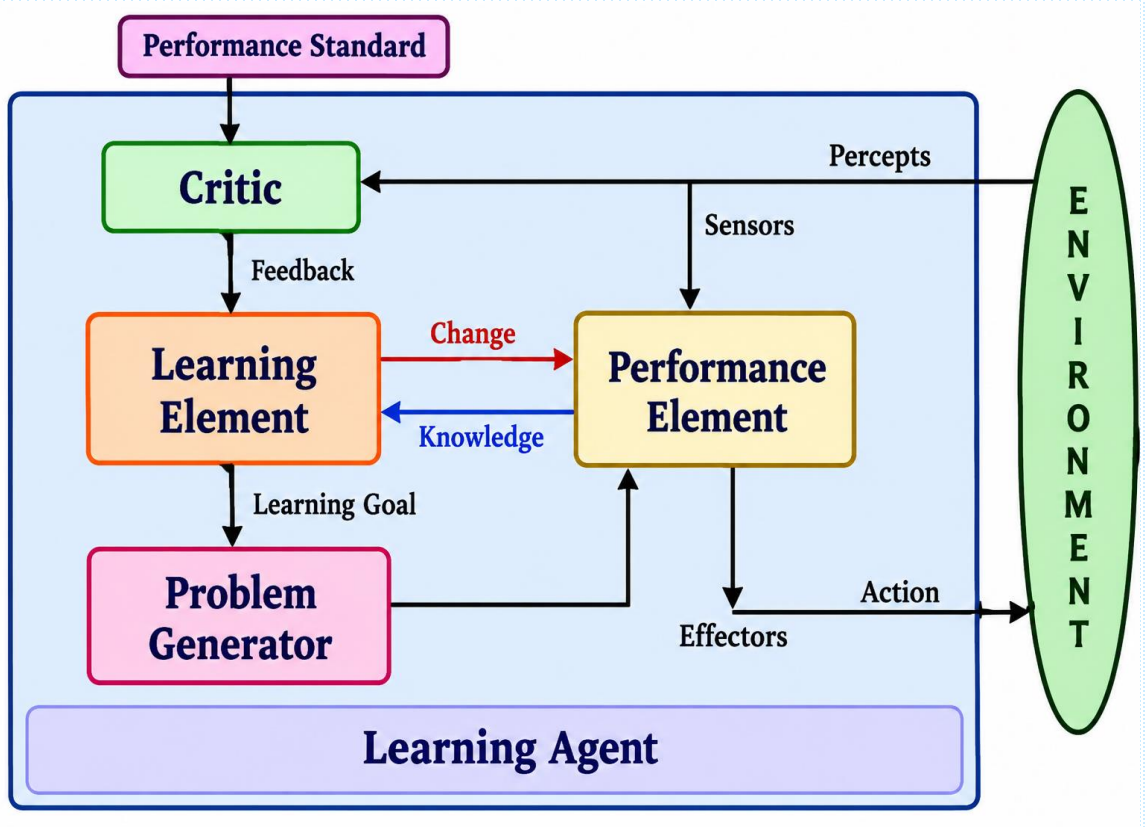
# How the components work together?

1) The sensors receive information from the environment.
2) The performance element selects an action.
3) The effectors carry out the action.
4) The critic compares the result with the performance standard.
5) The learning element uses feedback to improve the performance element.
6) The problem generator may suggest a new action or experiment to gain further experience.



# Learning-agent architecture

The environment supplies percepts through sensors. The performance element chooses actions, which are carried out through effectors. The critic evaluates performance and sends feedback to the learning element. The learning element may modify the performance element, while the problem generator suggests useful exploration.



## Example: a self-driving taxi

## Setup

- **Environment:** roads, traffic, pedestrians, weather, passengers.
- **Performance standard:** reach the destination safely, legally, quickly, and comfortably.
- **Starting knowledge:** basic driving rules, such as stop at red lights, keep a safe distance, and brake when something is ahead.

**Step 1: Sensing.** The cameras, radar, and GPS (the sensors) detect heavy rain, a wet road, and a car ahead. This percept goes to the performance element.

**Step 2: Acting with current knowledge.** The performance element follows its existing rule: brake when the gap to the car ahead is under 10 meters. The actuators (steering, brake, accelerator) carry this out.

**Step 3: The outcome.** The taxi brakes hard on the wet road and skids slightly. Passengers are jolted, and the taxi nearly hits the car ahead. The sensors record this too.

**Step 4: The critic evaluates.** It compares the outcome with the performance standard: safety and comfort were both violated. It sends the learning element feedback such as “braking at this distance on a wet road was too late and too harsh.”

**Step 5: The learning element improves the rules.** It works out the cause, which is that braking distance grows on wet surfaces. It changes the performance element’s rule from “brake at 10 m” to “brake earlier, and in proportion to road wetness and speed.”

**Step 6: The problem generator suggests an experiment.** To learn more, it proposes trying gentler braking from a longer distance, or testing how the taxi behaves on a different wet surface such as a painted road marking. This exploration keeps the taxi from settling on its first workable rule.

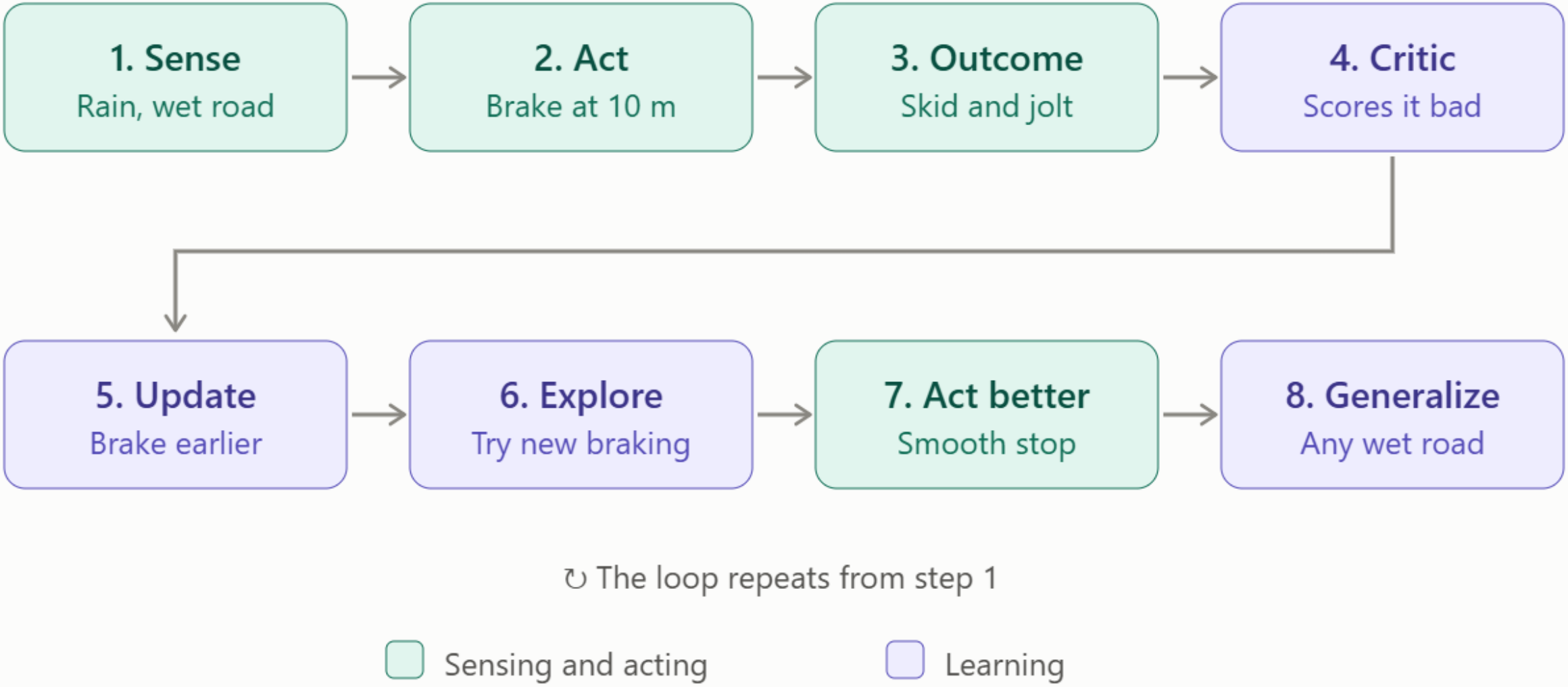
**Step 7: Acting again, now better.** The next rainy day, the performance element brakes earlier and more smoothly. The critic scores this as safe and comfortable, which confirms the new rule. If something is still off, the loop repeats.

**Step 8: Generalization.** The taxi doesn't just remember "brake early at that junction." It learns a general rule that applies to any wet road. This is what separates a learning agent from rote learning, which would only store that one incident.

***In short:***

sense → act → critic scores the result → learning element updates the rules → problem generator suggests new tries → act better next time.

# Example: a self-driving taxi

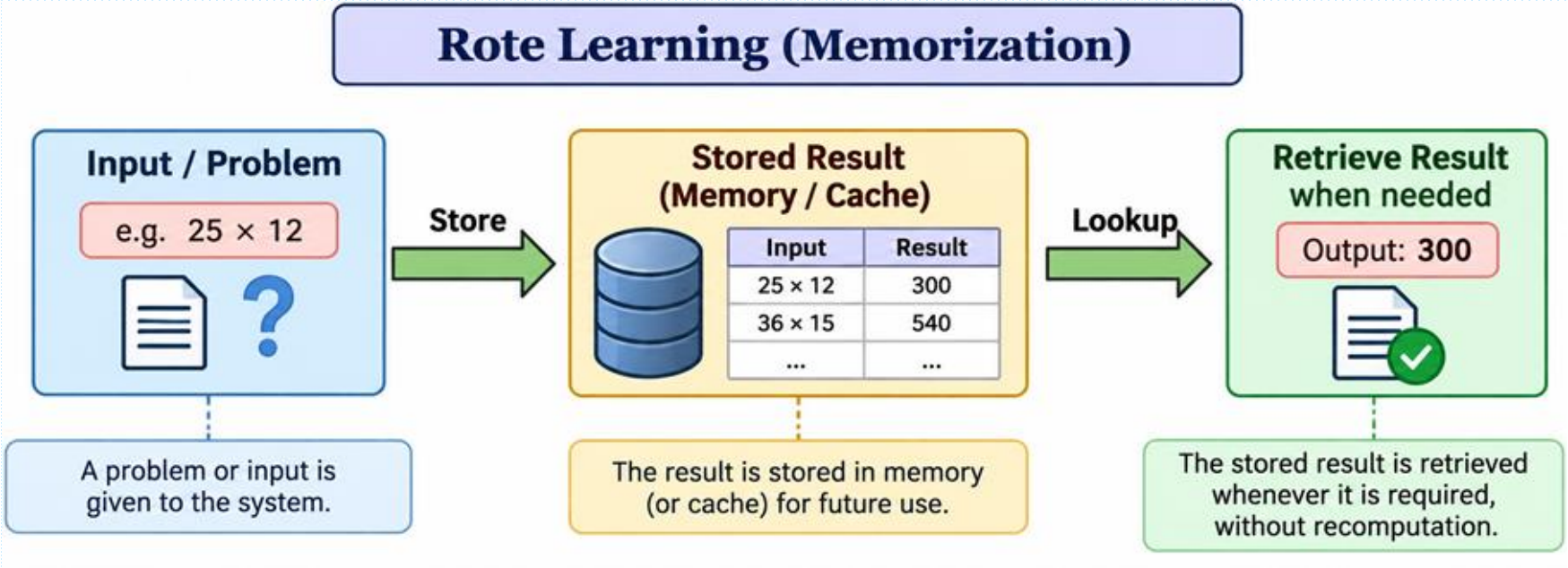


## 3. Rote Learning

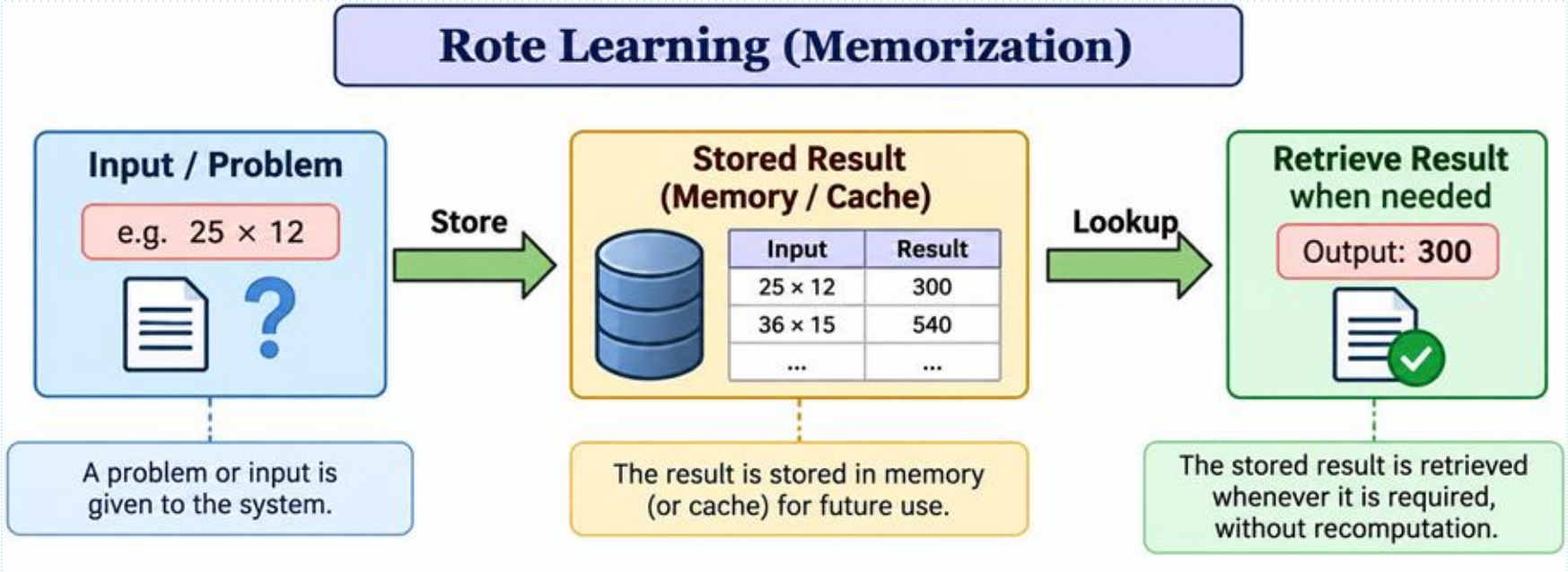
Rote learning is learning by memorization. The system stores a result, fact, or input-output mapping so that it can retrieve it later instead of repeating the same computation.

### Main characteristics

- ***One-to-one mapping:*** An input is associated with a stored output or result.



- ***Storage and retrieval:*** Information is saved in memory and retrieved when the same or a matching request occurs.
- ***Avoiding repeated computation:*** If the correct result is already stored, the system may not need to calculate it again.
- ***Limited generalization:*** Rote learning mainly remembers particular cases. By itself, it may not understand the deeper pattern behind them.



## ***Important considerations in rote learning***

- ***Organization:*** Stored knowledge should be organized so it can be found quickly. Sorting arranges items in order, indexing provides a quick route to information, and hashing helps locate a stored item using a key.
- ***Generalization:*** The number of stored items can become very large. Generalization can summarize common information or patterns and reduce the need to store every individual case.
- ***Stability of the environment:*** Rote learning works best when stored information remains valid. In a changing environment, stored results may become outdated, so the system must detect changes and update or discard old information.

## Everyday examples work the same way:

- ***A student memorizing a multiplication table.*** *They can answer 7 × 8 instantly but have not necessarily understood multiplication.*
- ***A web browser cache.*** *It saves a page it has already fetched and serves it again for the same address.*
- ***Memorization in programming.*** *A function stores fib(30) after computing it, so asking for fib(30) again returns the saved value.*

***Advantages:***

- fast retrieval,
- less repeated computation, and
- simple implementation for repeated tasks.

***Limitations:***

- memory requirements can grow;
- stored answers can become outdated; and
- memorizing examples does not necessarily allow the system to handle new situations.

## 4. Learning by Taking Advice

Learning by taking advice means improving a system by receiving guidance from a person or another source and translating that guidance into a form the system can use. The supplied notes identify expert systems as an example of this idea.

The notes associate the historical idea with John McCarthy in the 1950s. The central concept is that knowledge from an expert can be made useful to an AI system rather than requiring the system to discover every rule by itself.

Two basic approaches

- **High-level or abstract advice:** convert the advice into explicit rules that guide the performance element.
- **A sophisticated model:** use a model or tool, such as a knowledge-base debugging tool, to help translate expert knowledge into detailed rules.

## *Five steps in the advice-taking method*

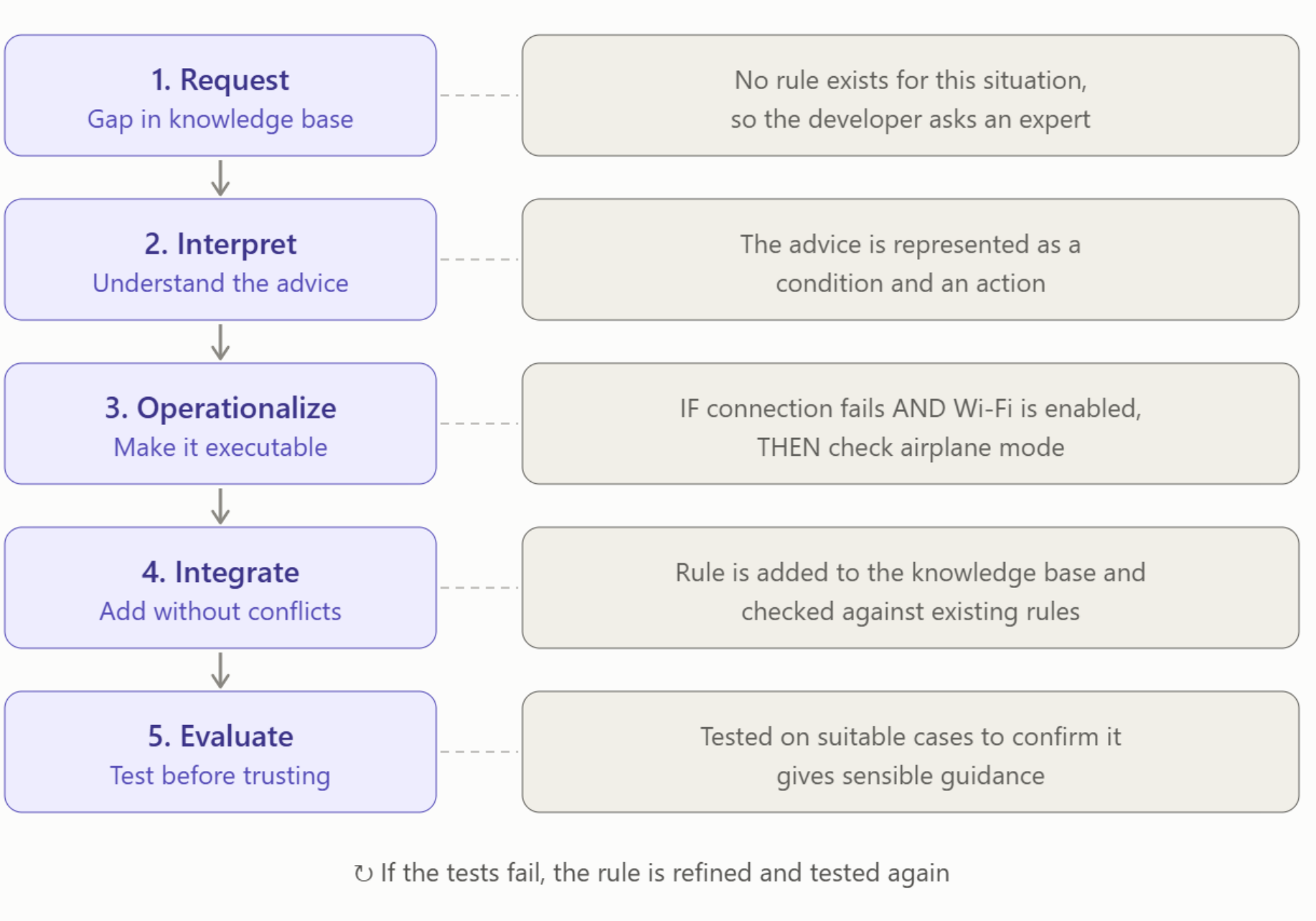
1. **Request:** Ask questions or make enquiries to identify a gap or shortcoming in the knowledge base and seek advice.
2. **Interpret:** Translate the advice into an internal representation that the system can understand.
3. **Operationalize:** Express the advice in a practical form that the performance element can use, such as a condition-action rule.
4. **Integrate:** Add the new knowledge to the knowledge base while checking for duplicate rules, conflicts, or contradictions.
5. **Evaluate:** Test or review the new knowledge to identify errors and ensure that it produces appropriate results.

## ***Worked example: advice for a help-desk system***

An IT expert gives the advice: “If a computer cannot connect to the network but Wi-Fi is enabled, first check whether airplane mode is on.”

- **Request:** The system’s knowledge base does not include a rule for this situation, so the developer asks an expert for advice.
- **Interpret:** The advice is represented as a condition and an action.
- **Operationalize:** Rule: IF network connection fails AND Wi-Fi is enabled, THEN check airplane mode.
- **Integrate:** The rule is added to the knowledge base and checked against existing troubleshooting rules.
- **Evaluate:** The rule is tested on suitable cases to confirm that it gives sensible guidance.

# Worked example: advice for a help-desk system

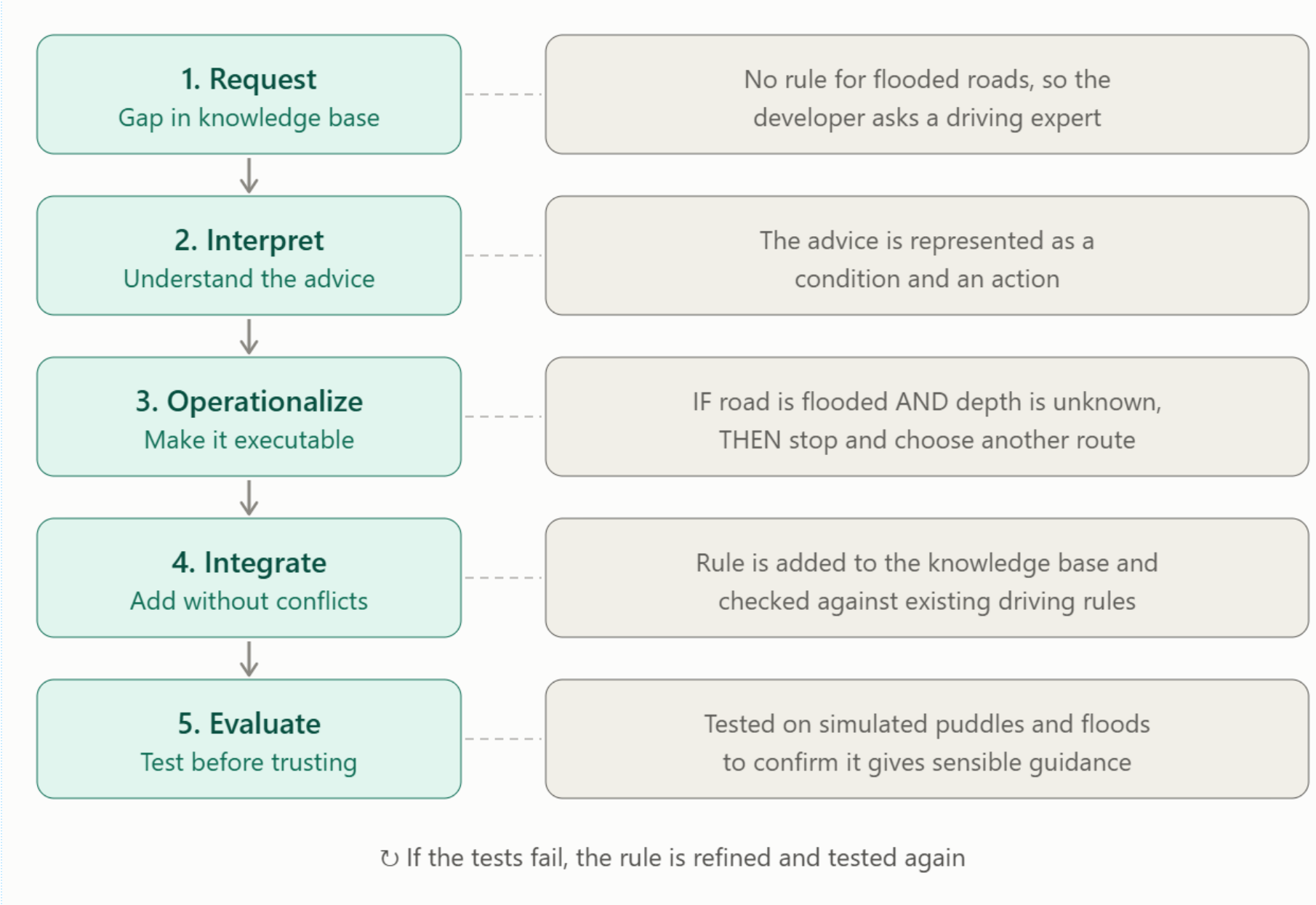


# ***Worked example: Self-driving car system***

Here is the self-driving car version in the same five steps.

1. **Request.** The car's knowledge base has no rule for a flooded road, so the developer asks a driving expert for advice.
2. **Interpret.** The advice is represented as a condition and an action.
3. **Operationalize.** It becomes a rule: IF road ahead is flooded AND water depth is unknown, THEN stop and choose another route.
4. **Integrate.** The rule is added to the knowledge base and checked against existing driving rules, so it doesn't conflict with rules like "keep moving in traffic."
5. **Evaluate.** The rule is tested on simulated cases, such as a shallow puddle and a deep flood, to confirm it gives sensible behavior.

# Worked example: Self-driving car system



This is an illustrative teaching example. An expert system should still handle exceptions and avoid treating one rule as a guarantee that the problem has been solved.

## ***Benefits and challenges***

### ***Benefits:***

- expert knowledge can be incorporated directly,
- rules can be easier to explain, and
- development may be faster than learning every rule from data.

### ***Challenges:***

- advice may be vague, incomplete, incorrect, or contradictory.
- It must be translated carefully and tested before use.

## 5. Learning by Parameter Adjustment

Many programs use an evaluation function to assign a score to a possible action, state, or solution. Learning by parameter adjustment improves this function by changing its weights based on accumulated experience or feedback.

**Weighted evaluation function**

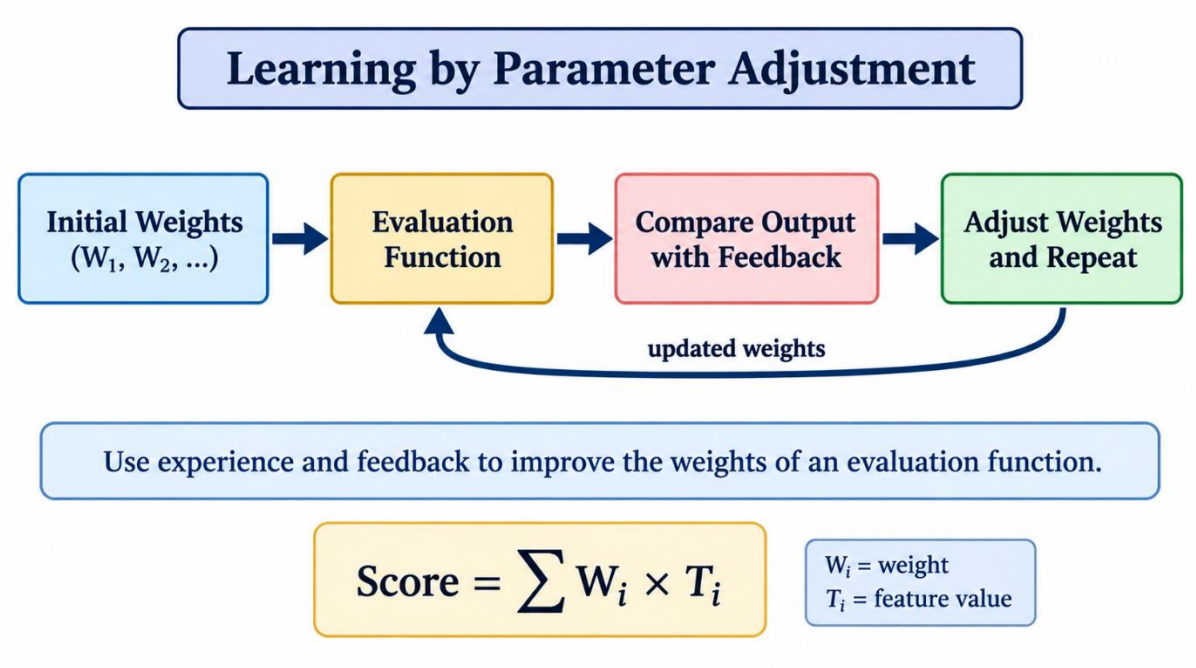
$$\text{Evaluation score} = \Sigma W_i \times T_i$$

- $W_i$ is the weight assigned to feature i.
- $T_i$ is the value of feature i.

A weight represents how strongly a feature contributes to the overall score.

## Steps

1. **Initialization:** begin with estimated values for the weights.
2. **Evaluation:** use the current weights to score a choice or prediction.
3. **Feedback:** compare the result with the desired outcome or performance standard.
4. **Modification:** increase or decrease selected weights based on the feedback.
5. **Repeat:** use the adjusted weights on later cases and continue improving.



## Worked example: route selection

Suppose a navigation agent scores routes using travel time and distance. For simplicity, let:

$$\text{Score} = (0.7 \times \text{travel-time score}) + (0.3 \times \text{distance score})$$

The values 0.7 and 0.3 are illustrative weights, not universal settings.

If experience shows that reaching a destination on time matters more than distance, the system may give travel time a larger weight. After changing the weights, it checks whether route recommendations improve.

## Good and poor predictors

A feature that consistently helps predict the correct outcome may deserve a larger weight. A feature that contributes little or misleads the system may receive a smaller weight. The exact update method depends on the algorithm.

## ***Example: a spam filter tuning its word weights***

A spam filter gives each email a score from a fixed formula, where each weight says how spammy a word is:

$$\textit{Spam score} = w_1 \times (\text{count of “free”}) + w_2 \times (\text{count of “winner”}) + w_3 \times (\text{count of “meeting”})$$

An email is flagged as spam if its score is 3 or more. All weights start at 1.0.

The structure of the formula never changes. Only the weights are learned (illustrative numbers).

## *Example: a spam filter tuning its word weights*

- **Predict.** A real email from your boss says: “Free to talk? Meeting at 3, meeting notes attached.” Score = $1.0 \times 1 + 1.0 \times 0 + 1.0 \times 2 = 3$, so it is wrongly flagged as spam.
- **Compare with the outcome.** You click “Not spam.” The filter now knows its prediction was a mistake.
- **Adjust the weights.** The word that contributed most to the mistake, “meeting,” drops from 1.0 to 0.2, and “free” drops slightly from 1.0 to 0.8.
- **Repeat.** The same email now scores $0.8 \times 1 + 0.2 \times 2 = 1.2$, so it is not spam. A real spam email like “Free! Free! Winner! Winner!” still scores $0.8 \times 2 + 1.0 \times 2 = 3.6$, so it is still caught.

After thousands of emails marked by users, the weights settle on values that separate spam from normal mail well.

# ***Advantages and limitations***

## Advantages:

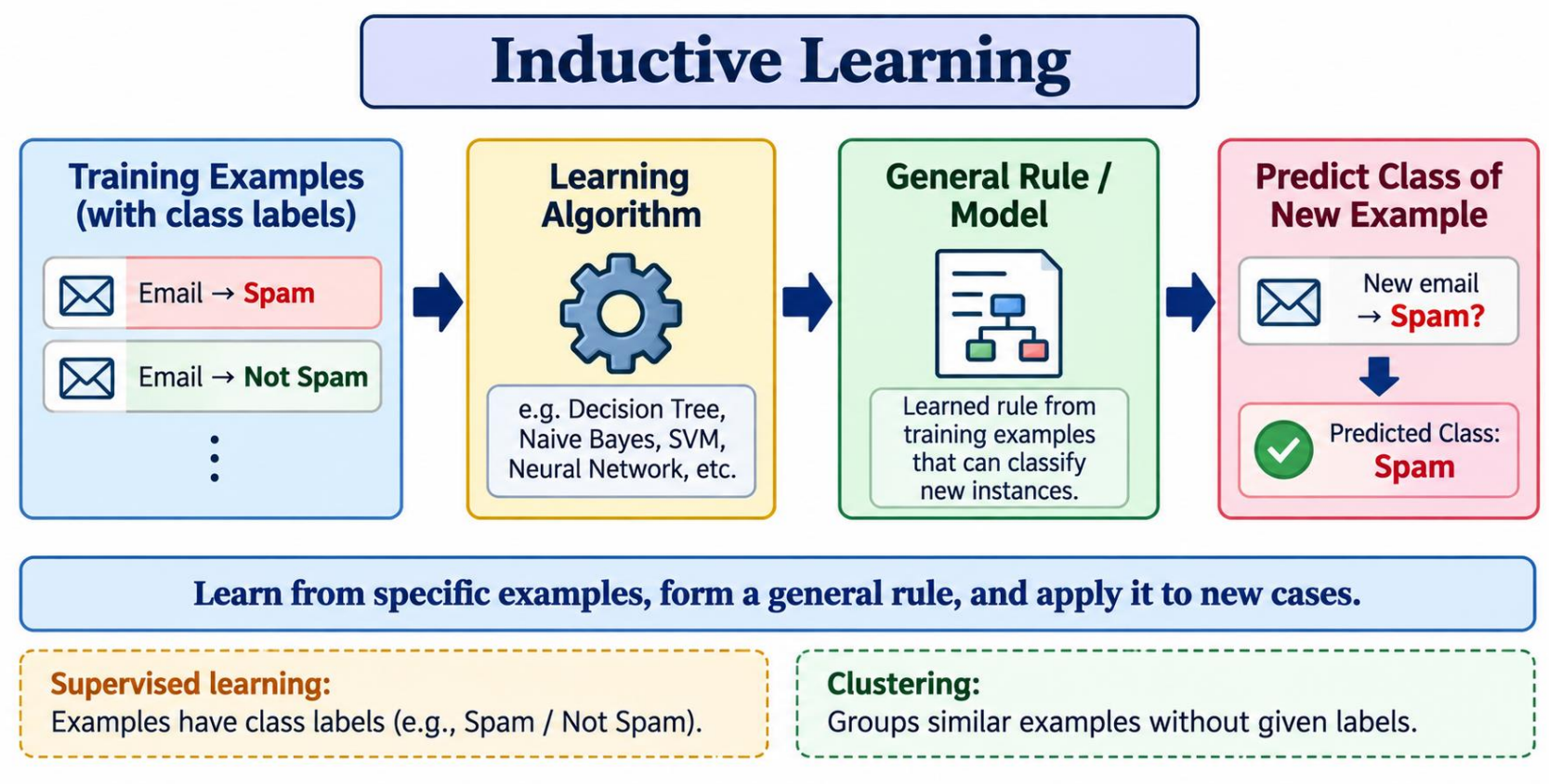
- the system can improve without redesigning the entire evaluation function;
- weights are often interpretable; and
- the method can be used in search, planning, and prediction.

## Limitations:

- choosing initial weights can be difficult;
- poor feedback can lead to poor updates; and
- adjusting weights does not automatically guarantee a globally optimal solution.

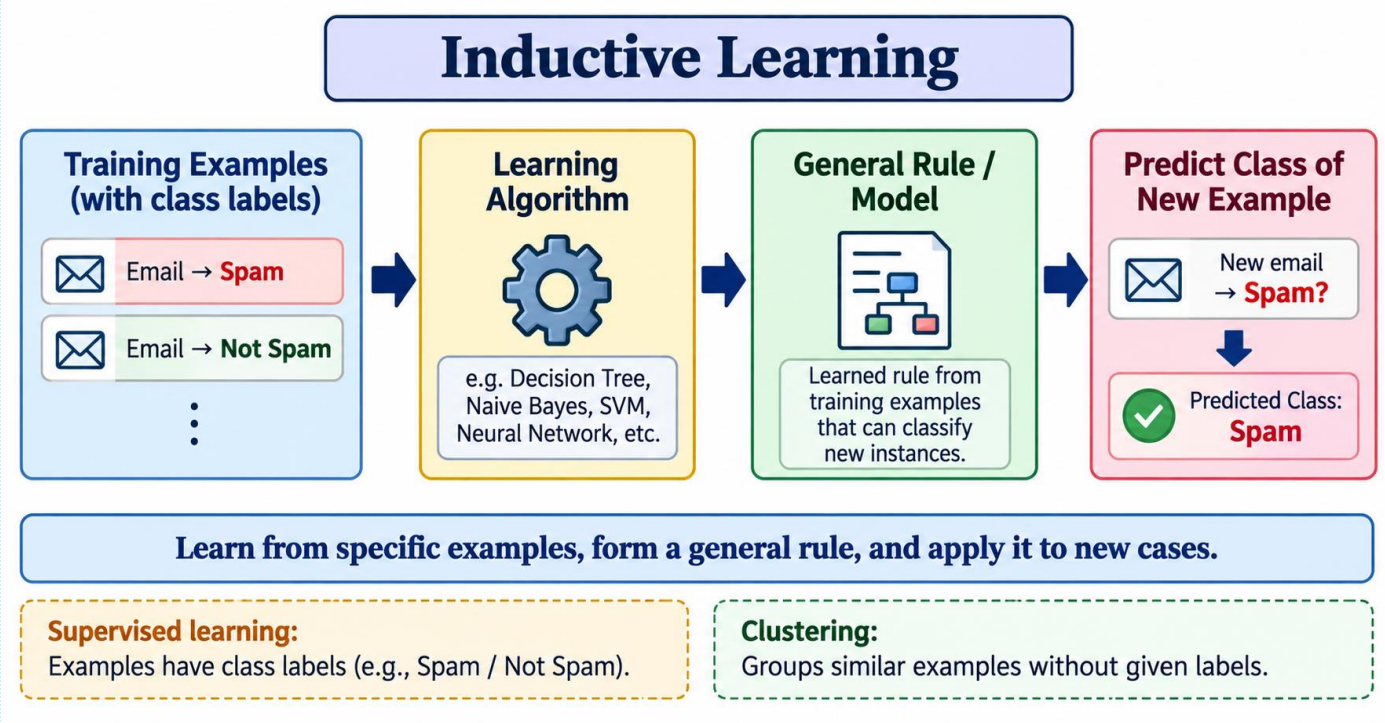
# 6. Inductive Learning

Inductive learning is the process by which a system learns from specific examples and uses them to derive general rules or class definitions. It moves from observed instances towards a general conclusion.



# *How inductive learning works?*

1. Collect examples or observations.
2. Identify patterns or similarities in the examples.
3. Construct a general rule, concept, or model.
4. Use the learned model to classify or predict new examples.
5. Review errors and improve the model when additional examples become available.



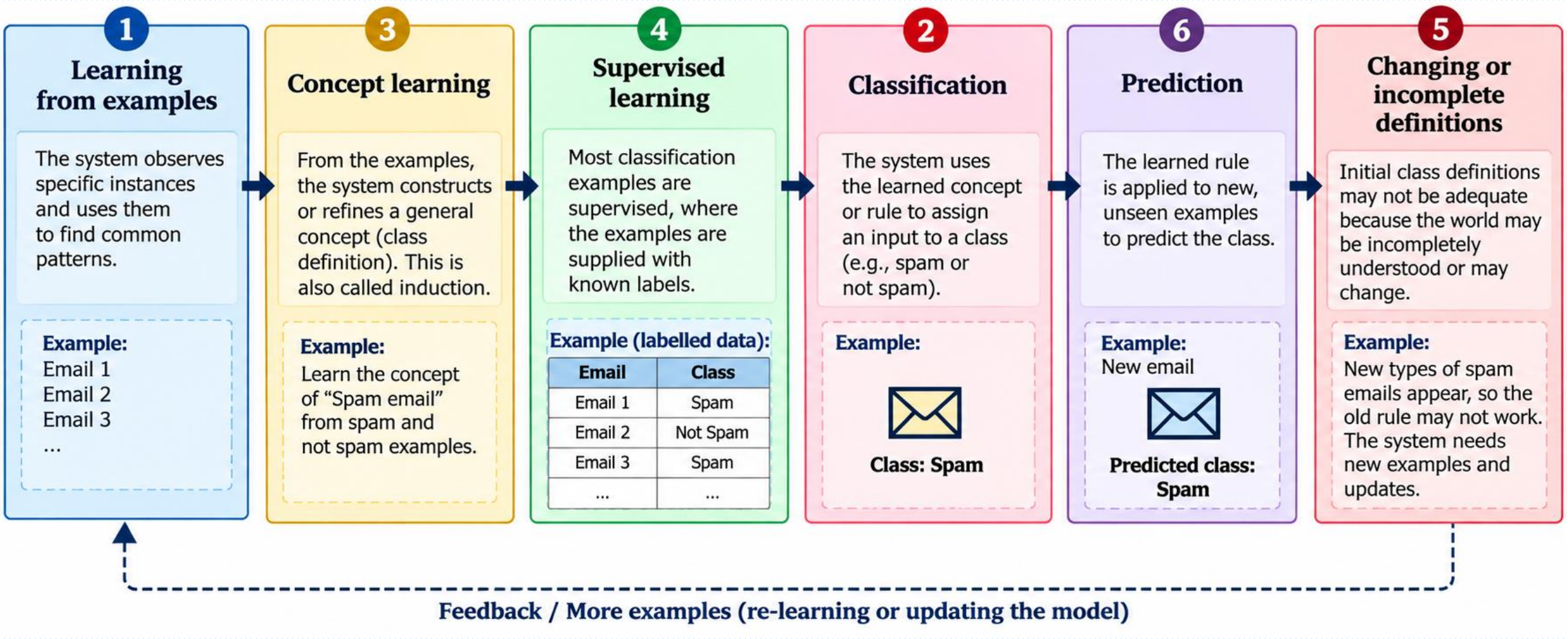
## *Worked example: email classification*

- A system is given examples of emails labelled “spam” and “not spam.”
- It observes patterns such as repeated promotional phrases, suspicious links, or the sender’s history.
- It builds a model from the labelled examples and uses that model to classify a new email.
- The new email may still be misclassified, so the system’s performance should be evaluated.

These six points explain different aspects of inductive learning:

- ***Learning from examples:*** How the system begins learning.
- ***Concept learning:*** How it develops general rules.
- ***Supervised learning:*** How labelled examples guide learning.
- ***Classification:*** How it assigns inputs to categories.
- ***Prediction:*** How it applies learned rules to new cases.
- ***Changing or incomplete definitions:*** Why learned rules may need improvement.

# This examples explain different aspects of inductive learning:



## ***I) Learning from Examples (LBE)***

The system observes several specific examples and identifies common patterns to develop a general rule.

***Example:***

- Example 1: An email containing suspicious links is labelled Spam.
- Example 2: An email containing repeated promotional messages is labelled Spam.
- Example 3: An email from a trusted sender is labelled Not Spam.

By studying these examples, the system learns patterns that may help it identify spam emails in the future.

## ***II) Classification***

Classification means assigning an input to one of the predefined classes based on the patterns learned from examples.

***Example:*** An email classification system has two classes:

- Spam
- Not Spam

When a new email arrives, the system analyses its features and predicts whether it belongs to the Spam or Not Spam class.

## ***III) Concept Learning***

Concept learning means learning a general concept or rule from examples belonging to different classes.

**Example:** Suppose an AI system is given examples of fruits.

- Apple → Fruit
- Mango → Fruit
- Banana → Fruit
- Stone → Not a fruit

By studying these examples, the system develops a concept of what belongs to the class Fruit. The learned concept can then help it classify a new example.

## ***IV) Supervised Learning***

Supervised learning is a type of machine learning in which the system learns from examples that already have the correct answers or labels.

**Example:**

| Training example | Correct label |
|---|---|
| Email 1 | Spam |
| Email 2 | Not Spam |
| Email 3 | Spam |
| Email 4 | Not Spam |

The system learns from these labelled examples. Later, it uses the learned patterns to predict the class of a new email.

## ***V) Changing or Incomplete Definitions***

The rules learned from examples may not always be complete or correct. The system may not have seen every possible situation, and conditions in the real world may change.

***Example:*** An email system may initially identify spam emails mainly by suspicious words. Later, spammers may use different words or methods. The old rules may no longer identify all spam emails correctly.

The system may need new examples and updated rules to improve its performance.

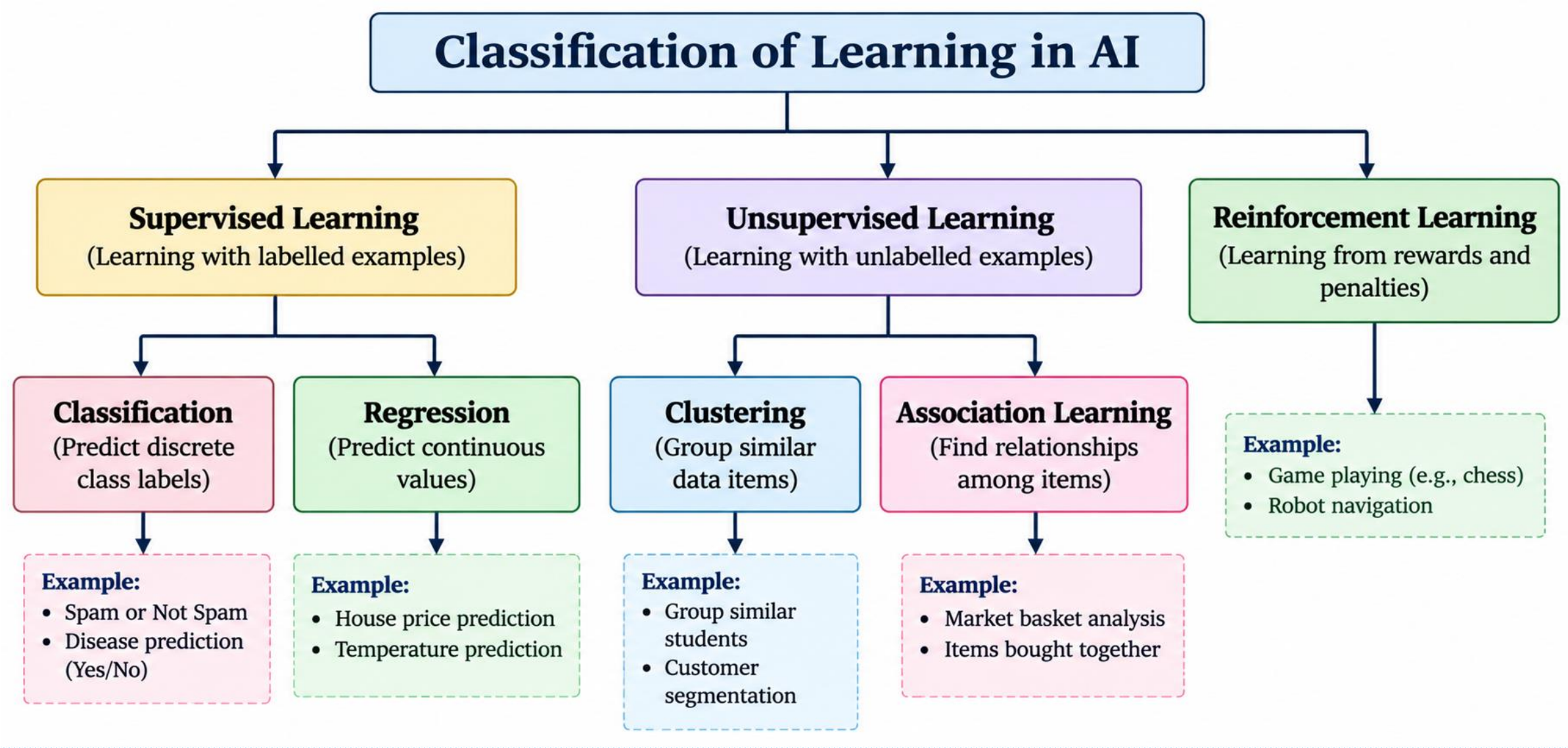
## ***VI) Prediction***

Prediction means using a learned rule or model to determine the likely class or outcome of a new example.

***Example:*** After learning from previous emails, the system receives a new email that contains suspicious links and promotional content. It predicts that the email is Spam.

However, the prediction may be incorrect because the learned model is not always perfect.

# Learning Methods in AI



# Suggestive Questions for examination

## Section A – Short Answer Questions (2 marks)

1. Define Rote Learning.
2. What is a Learning Agent?
3. Define Inductive Learning.
4. What is the role of the Critic in a Learning Agent?
5. What is Learning by Taking Advice?

## Section B – Descriptive Questions (5 marks)

6. Explain the architecture of a Learning Agent with a neat diagram.
7. Explain Rote Learning with a suitable example. Discuss its advantages and limitations.
8. Explain the five steps involved in Learning by Taking Advice with an example.
9. Explain Learning by Parameter Adjustment with a suitable mathematical example.

## Section C – Long Answer Questions (15 marks)

10. (a) Explain Inductive Learning and its working with a suitable diagram. (7)
    (b) Explain Classification, Concept Learning and Supervised Learning with examples. (8)

11. (a) Explain the architecture and working of a Learning Agent. (8)
    (b) Explain Rote Learning and Learning by Taking Advice with suitable examples. (7)