Module 5: Planning and Learning

Module Outline

5.1 Planning

- The planning problem
- Planning Vs Searching
- STRIPS and ADL representation languages
- Planning with state space search (Progression, Regression)
- Partial order planning
- Hierarchical planning
- Contingent Planning
- *(Self-learning: Multiagent planning)*

5.2 Learning

- Forms of Learning
- Inductive Learning
- Decision Trees
- Applications of learning
- *(Self-learning: Practical machine learning)*

5.3 Applications of AI

- **Natural Language Processing (NLP)**
    - Language models
    - Text classification
    - Information retrieval
    - Information extraction
- **Expert Systems (ES)**
    - Components of expert systems
    - ES vs Traditional Systems
    - Characteristics of expert systems
    - Roles in ES implementation
    - ES implementation process
    - Applications, advantages, and limitations of ES

Planning in AI

What is Planning in AI?

- **Planning** in Artificial Intelligence refers to the **decision-making tasks** performed by intelligent agents, such as **robots** or **computer programs**, to determine a sequence of actions needed to achieve a **specific goal**.
- The core task during the execution of planning involves **choosing a sequence of actions** that has a high probability of successfully completing the intended task.

Simplified Summary

Planning is like figuring out the steps needed to get something done. For an AI, this means deciding which actions to take, in what order, to reach a desired outcome (e.g., a robot planning how to move blocks to build a tower).

Real-world Application

- **Logistics:** Planning optimal delivery routes for trucks.
- **Robotics:** A warehouse robot planning how to fetch items and navigate obstacles.
- **Game AI:** An opponent in a strategy game planning its moves to defeat the player.
- **Autonomous Driving:** A self-driving car planning its path through traffic, considering speed limits, obstacles, and destination.

Planning for Real-World Problems

- **Planning** is fundamentally the task of generating a **sequence of actions** that will lead to the achievement of a desired goal.
- Planning problems can be addressed by different types of agents:
  - **Search-based problem-solving agents:** These agents explore possible sequences of actions using search algorithms (like Breadth-First Search or A*). This connects back to search algorithms covered previously.
  - **Logical planning agents:** These agents use logical inference to deduce a valid plan.
- Planning is particularly crucial for **complex and large-scale problems** where simple reactive behavior is insufficient.

Classical Planning Environments

- For initial understanding, we often focus on environments with specific simplifying assumptions:
    1. **Fully Observable:** The agent knows the complete state of the world at all times.
    2. **Deterministic:** The outcome of any action is perfectly predictable.
    3. **Finite:** There are a limited number of states, actions, and objects.
    4. **Static:** The environment only changes as a result of the agent's actions; there are no external events.
    5. **Discrete:** Time, actions, objects, and effects occur in distinct, separate steps or categories.
- Environments satisfying these conditions are known as **classical planning environments**.

Contrast with Non-Classical Planning

- **Non-classical planning** addresses more complex, realistic scenarios involving:
    - **Partially Observable Environments:** The agent doesn't know everything about the current state.
    - **Stochastic Environments:** Actions can have uncertain outcomes (probabilistic effects).
    - **Dynamic Environments:** The world can change independently of the agent's actions.
- These require different, often more complex, algorithms and agent designs (e.g., planning under uncertainty, using sensors).

Assumptions and Representation Languages

- **Assumption of Near Decomposability:** Many planning approaches assume that real-world problems are **nearly decomposable**.
    - This means a planner can often work on **sub-goals independently** and then combine the resulting sub-plans, perhaps with some additional steps to resolve interactions.
    - *Example:* Planning a trip involves independent sub-goals like booking flights, booking hotels, and packing, which are then combined.
- **Finding the Right Language:** A key challenge is finding a planning **language** that is:
    - **Expressive enough** to describe a wide variety of problems accurately.
    - **Restrictive enough** to allow efficient planning algorithms to operate effectively. Too much expressiveness can lead to computational intractability.
- **STRIPS Language:** The foundational representation language for many classical planners is the **STRIPS** language.

- **STRIPS** stands for **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver.

STRIPS (STanford Research Institute Problem Solver)

STRIPS provides a formal way to define planning problems. It consists of:

1. **A set of States:**
   - Each state is represented as a **conjunction** (AND) of **positive ground literals** (facts that are true). Literals are atomic statements or their negations. Ground means no variables. Positive means no negations.
   - *Example:* `At(Robot, RoomA) ∧ Holding(Robot, Block1) ∧ BlockOnTable(Block2)`
2. **A set of Goals:**
   - A goal is a **partially specified state**, represented as a **conjunction of positive literals**. It describes the desired condition to be achieved.
   - *Example:* `At(Robot, RoomB) ∧ Holding(Robot, Block2)`
3. **A set of Actions:**
   - Each action defines how the state can change. An action typically includes:
     - **Precondition:** A conjunction of positive literals that must be true in the current state for the action to be applicable.
     - **Effect:** A conjunction of literals describing how the state changes after the action is performed. Effects typically include an **ADD list** (literals that become true) and a **DELETE list** (literals that become false).

Deeper Dive into STRIPS Components:

- STRIPS offers a somewhat **restrictive** way to express states, actions, and goals, but this restriction enhances **efficiency**.
- **States:** Defined by **conjunctions of ground, function-free, positive literals**.
  - *Example:* `At(Home) ∧ Have(Banana)`
  - It uses the **Closed-World Assumption (CWA)**: any literal not explicitly mentioned in the state description is assumed to be false.
- **Goals:** Defined by **conjunctions of literals**. They can contain **variables** (implicitly existentially quantified), meaning a single goal description might represent multiple concrete goal states.
  - *Example (Goal with Negation - Note: Goals can have negations, states usually don't in basic STRIPS):* `At(Home) ∧ ¬Have(Bananas)`
  - *Example (Goal with Variables):* `At(x) ∧ Sells(x, Bananas)` (Find some place `x` that sells bananas and be at `x`).
- **Actions:** Defined by:

- **Preconditions:** Conjunction of literals that must be true *before* executing the action.
- **Effects:** Conjunction of literals describing the changes *after* execution (ADD list and DELETE list).

STRIPS Action Schema

- An **action schema** provides a template for actions, often involving variables.
- It includes:
  1. **Action Name & Parameter List:** Identifies the action and its variables.
  2. **Precondition:** A conjunction of literals (possibly with variables from the parameter list) that must hold.
  3. **Effect:** A conjunction of literals (possibly with variables) describing the outcome. Effects include literals to be added (ADD list) and literals to be removed (DELETE list).
- **Example Schema:**
  - **Action:** `Buy(x)` (Buy object `x`)
  - **Parameters:** `x` (the object to buy)
  - **Precondition:** `At(p) ∧ Sells(p, x)` (Agent must be at a place `p` that sells `x`)
  - **Effect:** `Have(x)` (Agent now has `x`. Implicitly, `¬Have(x)` might be deleted if it was present)
    - *Note:* In pure STRIPS, effects are often represented by ADD and DELETE lists explicitly. E.g., ADD: `Have(x)`, DELETE: `¬Have(x)`.

Challenges in AI Planning

1. **Closed World Assumption (CWA):**
   - Assumes the agent's model of the world contains *everything* relevant. There are no surprises or unknown factors.
   - *Limitation:* Real-world environments often violate this; unexpected events can occur.
2. **Frame Problem:**
   - How to efficiently represent and reason about what *stays the same* after an action is performed. Explicitly listing everything unchanged is computationally intractable for complex worlds.
   - *STRIPS Solution:* STRIPS implicitly handles this by assuming anything not mentioned in an action's effects remains unchanged. This is efficient but can be too simplistic for some scenarios.

Planning with State-Space Search

- One of the most fundamental approaches to solving planning problems is using **state-space search**.
- This involves viewing the planning problem as searching through a graph where nodes are states and edges are actions.
- Two main directions:
  1. **Forward State-Space Search (Progression):**
     - Starts from the initial state.
     - Applies applicable actions to generate successor states.
     - Searches forward until a state satisfying the goal is reached.
     - Uses standard search algorithms (BFS, DFS, A*, etc.).
  2. **Backward State-Space Search (Regression):**
     - Starts from the goal state(s).
     - Applies actions in reverse, finding states from which the goal could have been reached.
     - Searches backward until the initial state is reached.
     - Often more efficient when the goal is specific and the initial state is complex, or when there are many irrelevant actions.

Problem Formulation for Progression Planning

When using forward search (progression), the planning problem maps to a standard search problem formulation:

- **Initial State:** The initial state defined in the planning problem description.
  - *Example:* `At(Home) ∧ ¬Have(Banana) ∧ Have(Money)`
- **Actions:** Actions applicable in a given state `s` are those whose preconditions are satisfied by `s`. Applying an action leads to a successor state determined by the action's effects.
- **Goal Test:** Check if the current state `s` satisfies the goal description of the planning problem.
  - *Example:* Check if `At(Home) ∧ Have(Banana)` is true in the current state.
- **Step Cost:** Typically, each action has a cost of 1 (finding the shortest plan in terms of number of actions). Costs can be generalized.

Total Order vs. Partial Order Planning

Planning Approaches

- **TOP – Total Order Planning:**

    - These planners search for a **strictly ordered sequence** of actions from the initial state to the goal.
    - Examples include basic forward or backward state-space search algorithms.
- **POP – Partial Order Planning:**
    - These planners search in the space of *plans* rather than states.
    - They work by adding actions to achieve goals and imposing ordering constraints only when necessary, maintaining flexibility.

Total-Order Planning (TOP)

- Explores only **linear sequences** of actions (plan is a simple list).
- They **cannot easily take advantage of problem decomposition**. They commit to a specific ordering early on, even if sub-problems could be solved independently.
    - *Limitation Example:* If the goal is `GoalA ∧ GoalB`, a TOP planner might try to achieve `GoalA` fully, then `GoalB`, even if some steps for `GoalB` could have been interleaved efficiently while working on `GoalA`.

Partial-Order Planning (POP)

- Embraces **problem decomposition**.
- Divides the main problem into sub-goals and finds **sub-plans** to achieve them, often independently.
- Combines these sub-plans and **reorders actions based on dependencies** (preconditions and effects).
- The **ordering of actions is partial**: the final plan specifies only necessary ordering constraints (e.g., Action A *must* come before Action B), leaving other actions unordered relative to each other if their order doesn't matter.
- Does *not* specify upfront which of two independent actions should come first; this commitment is delayed. This often reduces the search space compared to TOP.

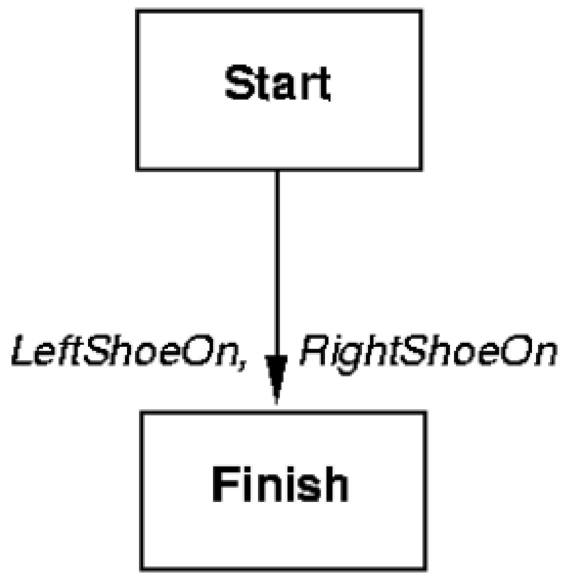
POP Example: Putting on Shoes

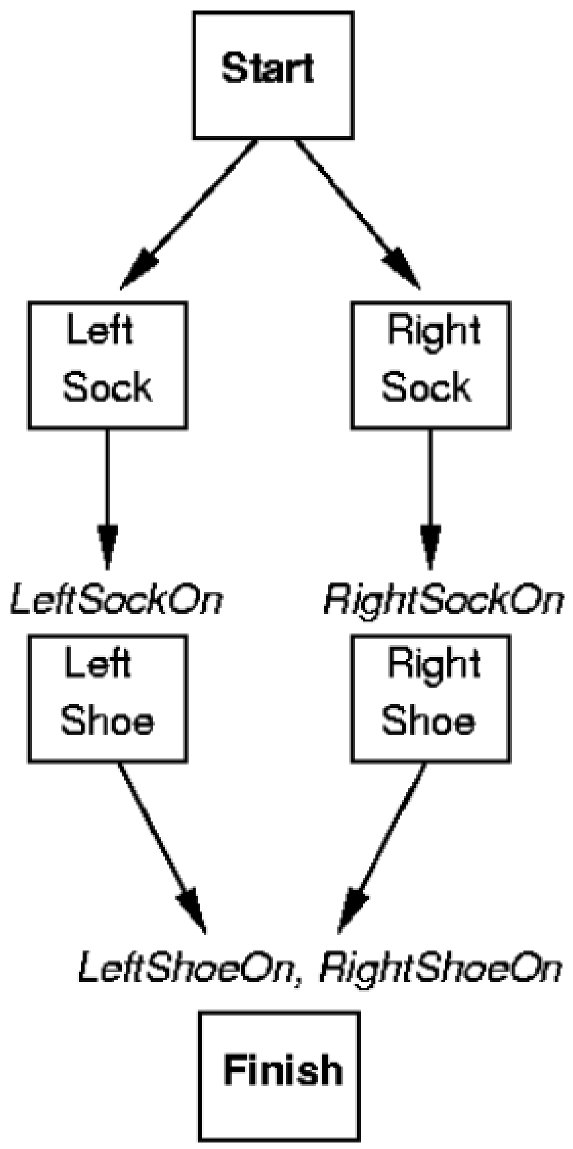
Let's formalize the "putting on shoes" task for a POP planner:

- **Goal:** `RightShoeOn ∧ LeftShoeOn`
- **Initial State (Init):** `Barefoot` (Implicitly means `¬RightSockOn`, `¬LeftSockOn`, `¬RightShoeOn`, `¬LeftShoeOn`)
- **Actions:**
    1. **Action:** `RightShoe`
        - **Precondition:** `RightSockOn`

- **Effect:** `RightShoeOn` (Add: `RightShoeOn`, Delete: `¬RightShoeOn`)

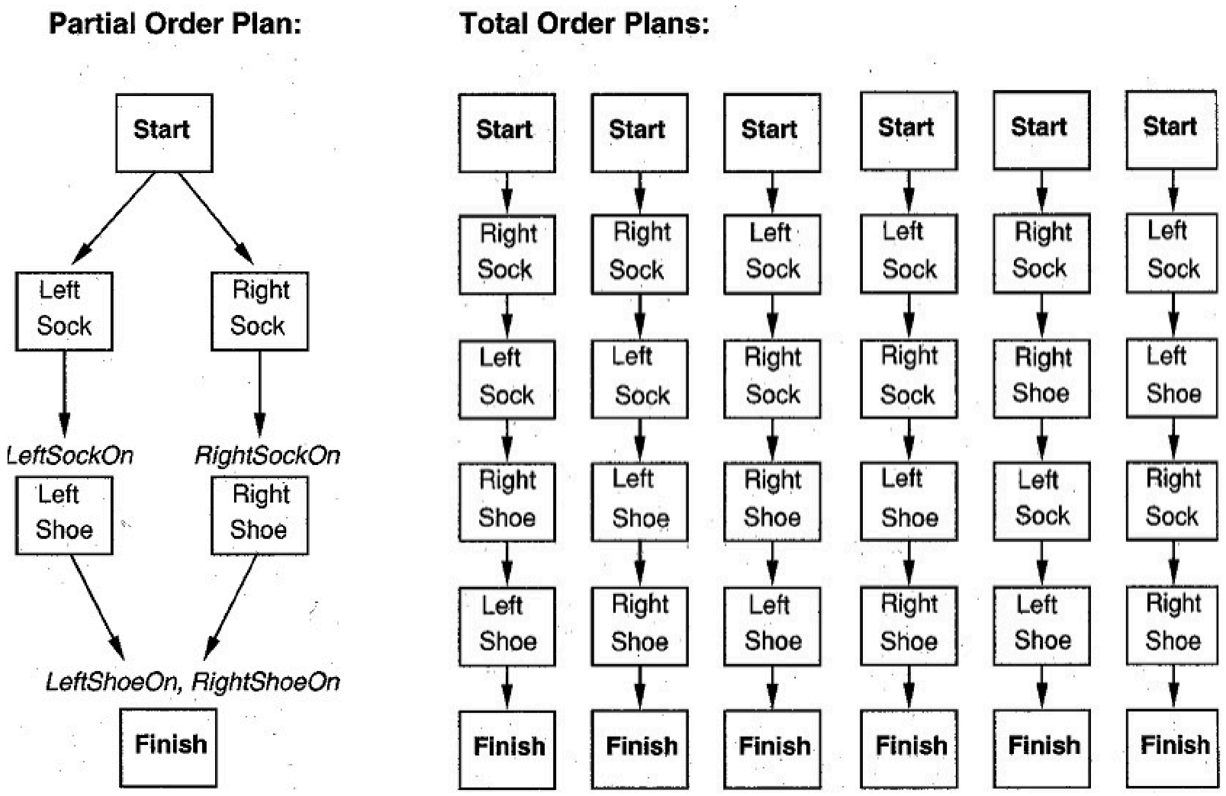
2. **Action:** `LeftShoe`
   - **Precondition:** `LeftSockOn`
   - **Effect:** `LeftShoeOn` (Add: `LeftShoeOn`, Delete: `¬LeftShoeOn`)
3. **Action:** `RightSock`
   - **Precondition:** `Barefoot` (or more generally, `¬RightSockOn`)
   - **Effect:** `RightSockOn` (Add: `RightSockOn`, Delete: `¬RightSockOn`, Delete: `Barefoot` if applicable)
4. **Action:** `LeftSock`
   - **Precondition:** `Barefoot` (or more generally, `¬LeftSockOn`)
   - **Effect:** `LeftSockOn` (Add: `LeftSockOn`, Delete: `¬LeftSockOn`, Delete: `Barefoot` if applicable)





- **Page 18 Diagram (Left):** Shows the initial plan structure. It contains a `Start` action (with effects representing the initial state, e.g., `Barefoot`) and a `Finish` action (with preconditions representing the goal `RightShoeOn ∧ LeftShoeOn`). Initially, there are no other actions or ordering constraints.
- **Page 18 Diagram (Right):** Shows a possible partially ordered plan.
  - The planner adds actions to achieve the goal preconditions (`RightShoeOn`, `LeftShoeOn`).
  - To achieve `RightShoeOn` via `RightShoe`, the precondition `RightSockOn` must hold. This is achieved by adding the `RightSock` action. An ordering constraint `RightSock → RightShoe` is added.
  - Similarly, `LeftSock → LeftShoe` is added.
  - The `Start` action achieves the preconditions for `RightSock` and `LeftSock` (e.g., `Barefoot`). Ordering constraints `Start → RightSock` and `Start → LeftSock` are added.

- The effects of `RightShoe` and `LeftShoe` achieve the `Finish` node's preconditions. Ordering constraints `RightShoe → Finish` and `LeftShoe → Finish` are added.
- Crucially, there is **no ordering constraint** between the { `LeftSock` , `LeftShoe` } sequence and the { `RightSock` , `RightShoe` } sequence. They can happen in parallel or in either order. This is the **partial order**.



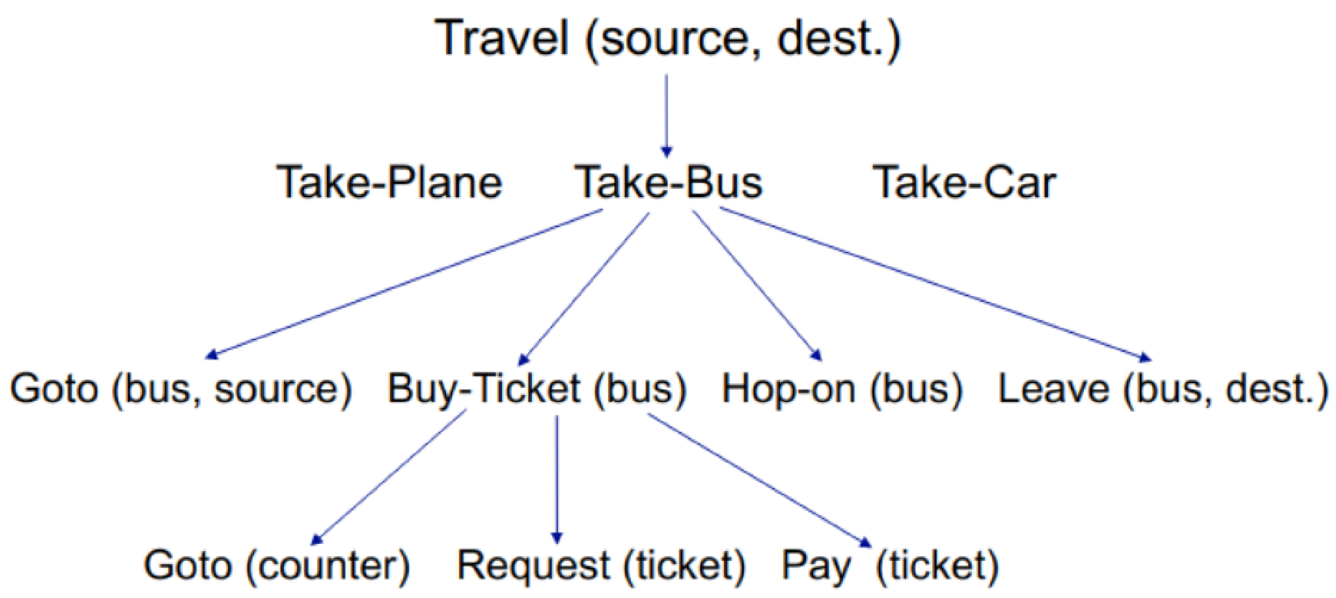
- **Page 19 Diagram:**
  - **Left:** Repeats the partial order plan structure.
  - **Right:** Shows the **six possible total order plans** (linear sequences) that are consistent with the single partial order plan. A TOP planner might explore many branches leading to these, while a POP planner works with the more flexible partial plan structure.

Hierarchical Planning

- Another approach to handle complexity, often combined with other methods like POP.
- Plans are organized in a **hierarchical format**, working on **plan decomposition**.
- **Complex actions** (high-level actions) are decomposed into sequences of **simpler or primitive actions**.

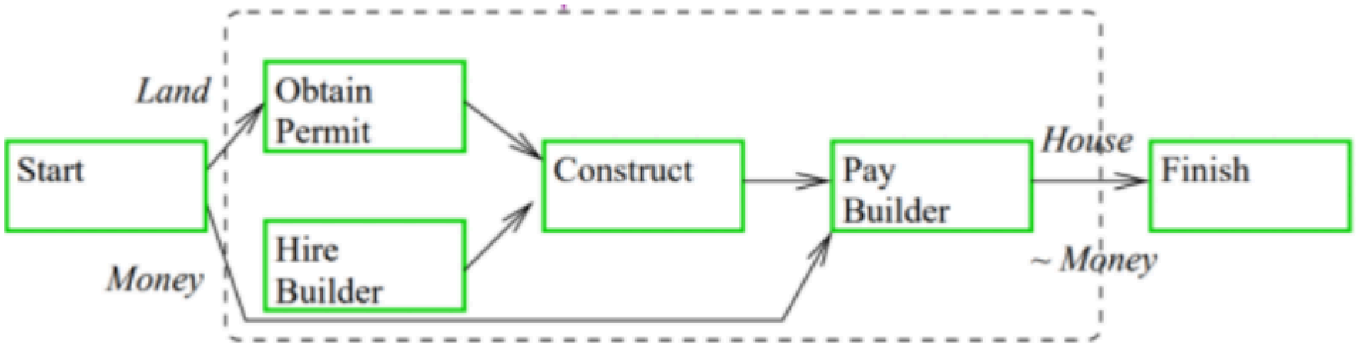
- This decomposition can be represented using links between states or actions at different levels of abstraction.

Hierarchical Planning Example: Travel



- **Explanation:** The high-level action `Travel(source, dest)` can be decomposed into mid-level actions like `Take-Plane`, `Take-Bus`, or `Take-Car`.
- The `Take-Bus` action can be further decomposed into lower-level actions like `Goto(bus, source)`, `Buy-Ticket(bus)`, `Hop-on(bus)`, `Leave(bus, dest)`.
- `Buy-Ticket(bus)` can be decomposed even further into primitive actions like `Goto(counter)`, `Request(ticket)`, `Pay(ticket)`.
- This allows planning at different levels of detail. The planner might first find a high-level plan (e.g., `Take-Bus`) and then refine it by replacing the high-level action with its lower-level decomposition.

Hierarchical Planning Example: Building a House



- **Explanation:** This diagram shows a high-level plan for building a house.
    - The `Start` state requires `Land` and `Money`.
    - A complex action "Construct" (inside the dashed box) might represent the entire building process.
    - This complex action could be decomposed into `Obtain Permit` and `Hire Builder`, followed by the actual construction, and finally `Pay Builder`.
    - Preconditions (`Land`, `Money`) and effects (`House`, `~Money`) link these actions. Hierarchical planning would allow refining the "Construct" step into many smaller sub-steps (foundation, framing, roofing, etc.).

Planning Under Uncertainty

Classical planning assumes deterministic actions and full observability. Non-classical planning handles uncertainty.

Conditional Planning (or Contingency Planning - Older Terminology)

- Deals with **non-deterministic environments** (actions have uncertain outcomes) but assumes **full observability**.
- The plan must work **regardless of the outcome** of uncertain actions.
- It generates plans with **branching** based on expected outcomes. It involves inspecting the environment state at predetermined points to decide which branch to follow.
- The plan must specify what to do for **every possible contingency** (outcome).
- *Example:* A plan to retrieve an object might include: "Try picking up the object. If successful, proceed. If failed (e.g., dropped), try picking up again."

Contingent Planning (Modern Usage)

- Deals with **partial observability** (agent doesn't know the full state) and possibly non-deterministic actions.
- Requires the use of **sensing actions** (or sensor actions) where the agent actively gathers information to reduce uncertainty.
- The agent uses sensing to discover **meaningful facts about the world** to make informed decisions.
- Plans often include loops like: "Sense the environment. If condition X holds, do Action A. If condition Y holds, do Action B. Repeat."
- *Example:* A robot in an unknown maze might use a sensor action "CheckNorthWall" before deciding whether to move North.

Comparing Representation Languages: STRIPS vs. ADL

While STRIPS is foundational, more expressive languages exist. **ADL (Action Description Language)** is one such extension.

Key Differences (STRIPS vs. ADL - based on table):

Feature	STRIPS	ADL (Action Description Language)	Explanation
Stands for	Stanford Research Inst. Problem Solver	Action Description Language	Naming convention.
Literals in States	Only Positive	Positive and Negative allowed	ADL states can explicitly represent what is false (`~Ugly`).
Unmentioned Literals	CWA (Closed World Assumption - assumed False)	OWA (Open World Assumption - assumed Unknown) possible	ADL can handle incomplete knowledge more naturally.
Equality	No built-in support	Built-in equality predicate (`x=y`)	ADL allows reasoning about object identity.
Types	No built-in support	Variables can have types (e.g., `p:Person`)	ADL allows stronger typing for variables, improving modelling.
Effects	Simple conjunctions (Add/Delete lists)	Conditional Effects (`when P: E`)	ADL actions can have effects that depend on the state (`when Hungry: Eat`).

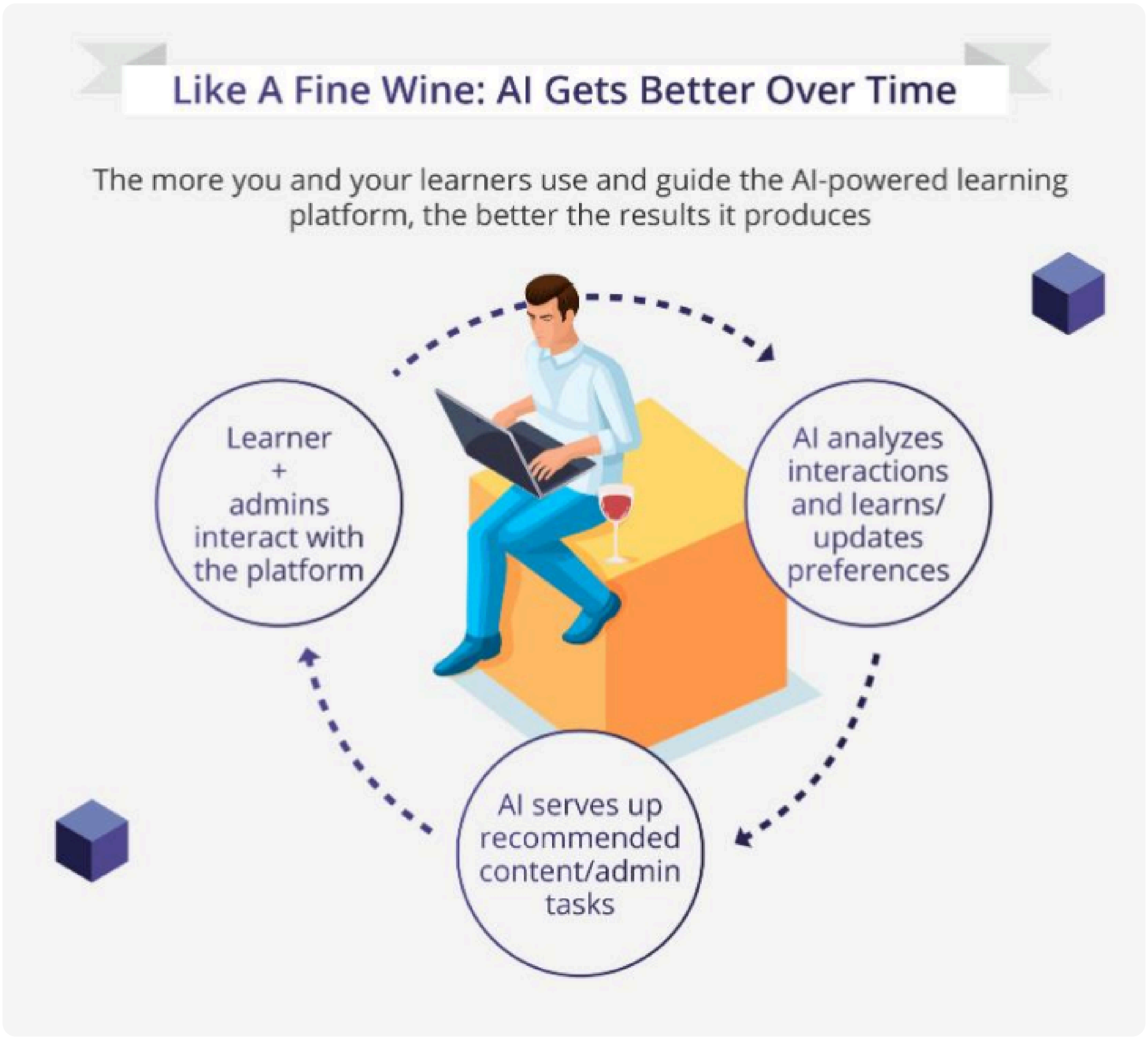
Feature	STRIPS	ADL (Action Description Language)	Explanation
Effect `P ∧ ¬Q`	Add P, Delete Q	Add `P ∧ ¬Q`, Delete `¬P ∧ Q`	ADL handles effects involving negations more consistently.
Goals	Conjunctions of literals	Allows conjunctions & disjunctions (`Famous ∨ Smart`)	ADL goals can be more complex (e.g., achieve one of several conditions).
Goals (Variables)	Only ground literals (or simple vars)	Allows Quantified Variables (e.g., `∀x`, `∃x`)	ADL goals can express requirements over sets of objects.
Schema Includes	Action name, Params, Precond, Effect	Action name, Params (opt), Labeled clauses (Precond, Add, Delete, Update - opt)	ADL offers a more structured and flexible schema definition.
Real-world Modelling	Often Not Suitable (too restrictive)	More Suitable	ADL's increased expressiveness better handles real-world complexities, which drove its development.

Key Points Summary: Planning

- **Planning** is the AI task of finding a sequence of actions to achieve a goal.
- **Classical Planning** makes simplifying assumptions (fully observable, deterministic, static, finite, discrete).
- **STRIPS** is a foundational language using states (positive literals), goals (conjunctions), and actions (preconditions, effects).
- **State-Space Search** (Forward/Progression, Backward/Regression) is a common planning method.
- **Partial Order Planning (POP)** offers flexibility by delaying ordering commitments and decomposing problems, contrasting with **Total Order Planning (TOP)**.
- **Hierarchical Planning** manages complexity by refining high-level actions into lower-level ones.

- **Conditional/Contingent Planning** handle uncertainty (non-determinism, partial observability), often requiring sensing actions.
- **ADL** is a more expressive language than STRIPS, better suited for complex, real-world modelling.

Learning in AI


Like A Fine Wine: AI Gets Better Over Time
The more you and your learners use and guide the AI-powered learning platform, the better the results it produces
Learner + admins interact with the platform
AI analyzes interactions and learns/ updates preferences
AI serves up recommended content/admin tasks

- **Concept:** AI systems, particularly those involving learning, improve their performance over time as they process more data and receive more feedback from interactions. The cycle shows: Learner interacts → AI analyzes & learns → AI serves recommendations/takes actions → Learner interacts...

What is Learning?

- An **AGENT** is considered to be **learning** if it **improves its performance on future tasks** after making observations about the world.
- Learning can range from simple memorization (like **jotting down a phone number**) to profound discovery (like **Albert Einstein** inferring new physical theories).

Why Do We Want Agents to Learn?

If an agent's design *can* be improved, why not just program the better design from the start? There are three main reasons why learning is essential:

1. **Designers cannot anticipate all possible situations:** The environment might be unknown or too complex to pre-program for every eventuality.
    - *Example:* A robot navigating **mazes** must learn the layout of each new maze it encounters.
    - ***Image Suggestion:*** Insert image from page 36 (Traffic) - predicting commute time depends on unforeseen conditions.
2. **Designers cannot anticipate all changes over time:** The environment or the task might change in ways that require adaptation.
    - *Example:* A program predicting **stock market prices** must learn to adapt as market conditions change.
3. **Designers may not know how to program a solution:** Some tasks are easy for humans but incredibly hard to program directly. Learning algorithms can discover solutions.
    - *Example:* Recognizing **faces** is easy for humans but difficult to code explicitly; learning algorithms are used instead.
    - ***Image Suggestion:*** recognizing subtle differences between animals or specific individuals often relies on learning.

Forms of Learning

The way an agent learns depends on several factors. Any component of an agent can potentially be improved through learning from data. The specific techniques used depend on four major factors:

1. **Which component is to be improved?** (e.g., action selection, world model, utility function)
2. **What prior knowledge does the agent already possess?** (Learning can be faster/more effective if it builds on existing knowledge).

3. **What representation is used for the data and the component?** (How information is structured affects learning).
4. **What feedback is available to learn from?** (This determines the main type of learning).

Components to be Learned

An agent's architecture includes several components, each of which could be learned:

1. **Direct mapping:** From conditions on the current state directly to actions (e.g., stimulus-response rules).
2. **Inferring properties:** Means to infer relevant, possibly hidden, properties of the world from the sequence of percepts (e.g., learning object permanence).
3. **World model:** Information about how the **world evolves** and the results of possible actions (learning physics or action outcomes).
4. **Utility information:** Indicating the desirability of world states (learning what constitutes a "good" outcome).
5. **Action-value information:** Indicating the desirability of taking specific actions in specific states (often used in reinforcement learning).
6. **Goals:** Descriptions of classes of states whose achievement maximizes the agent's utility (learning what objectives to pursue).

Example: Learning Components (Taxi Driver)

Consider an agent learning to be a taxi driver:

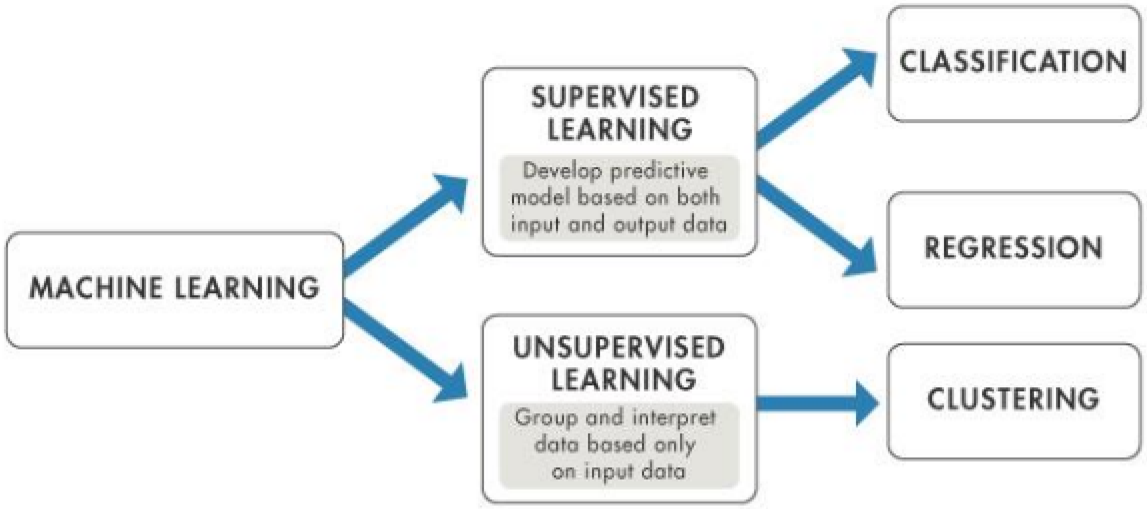
- Instructor shouts "Brake!" → Agent learns a **condition-action rule** for braking (**Component 1**).
- Instructor *doesn't* shout → Agent learns when braking is *not* needed (**Component 1**).
- Seeing many camera images labeled as "bus" → Agent learns to **recognize buses** (**Component 2**).
- Trying actions (braking hard on wet road) and observing results → Agent learns the **effects of its actions** (**Component 3**).
- Receiving no tip after a shaky ride → Agent learns a component of its **utility function** (passenger comfort is desirable) (**Component 4**).

Feedback to Learn From: Main Types of Learning

The type of feedback available is the primary determinant of the learning paradigm:

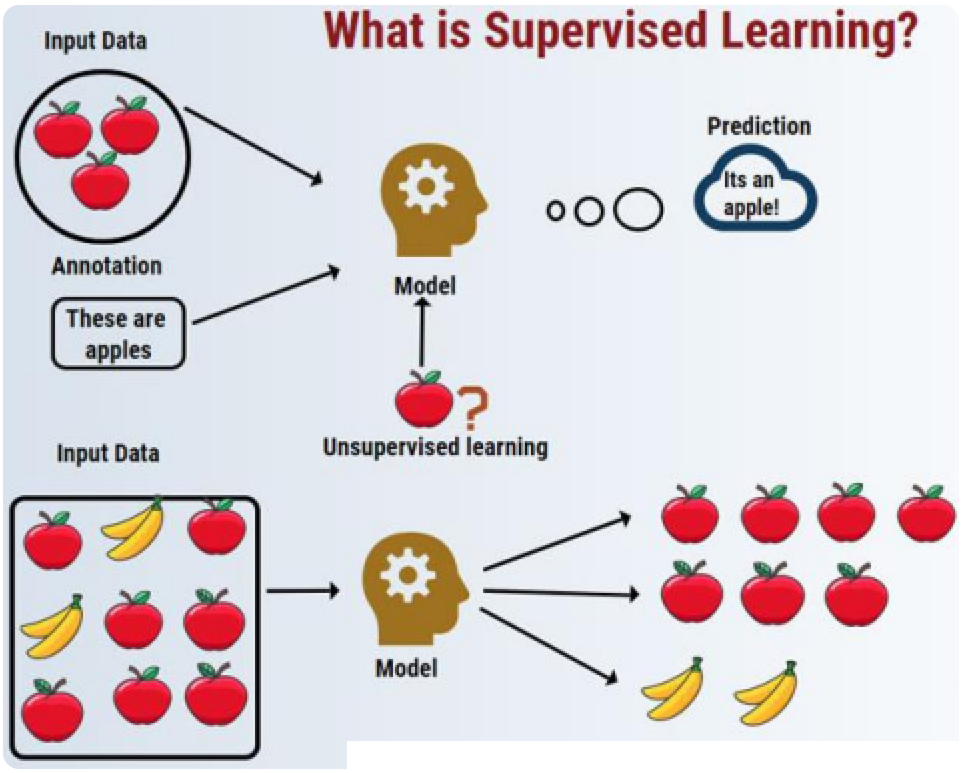
1. **Supervised Learning:** Learns from labeled examples (input-output pairs).

2. **Unsupervised Learning:** Learns patterns from unlabeled data.
3. **Reinforcement Learning:** Learns from rewards or penalties received through interaction with an environment.



MACHINE LEARNING
SUPERVISED LEARNING
Develop predictive model based on both input and output data
UNSUPERVISED LEARNING
Group and interpret data based only on input data
CLASSIFICATION
REGRESSION
CLUSTERING

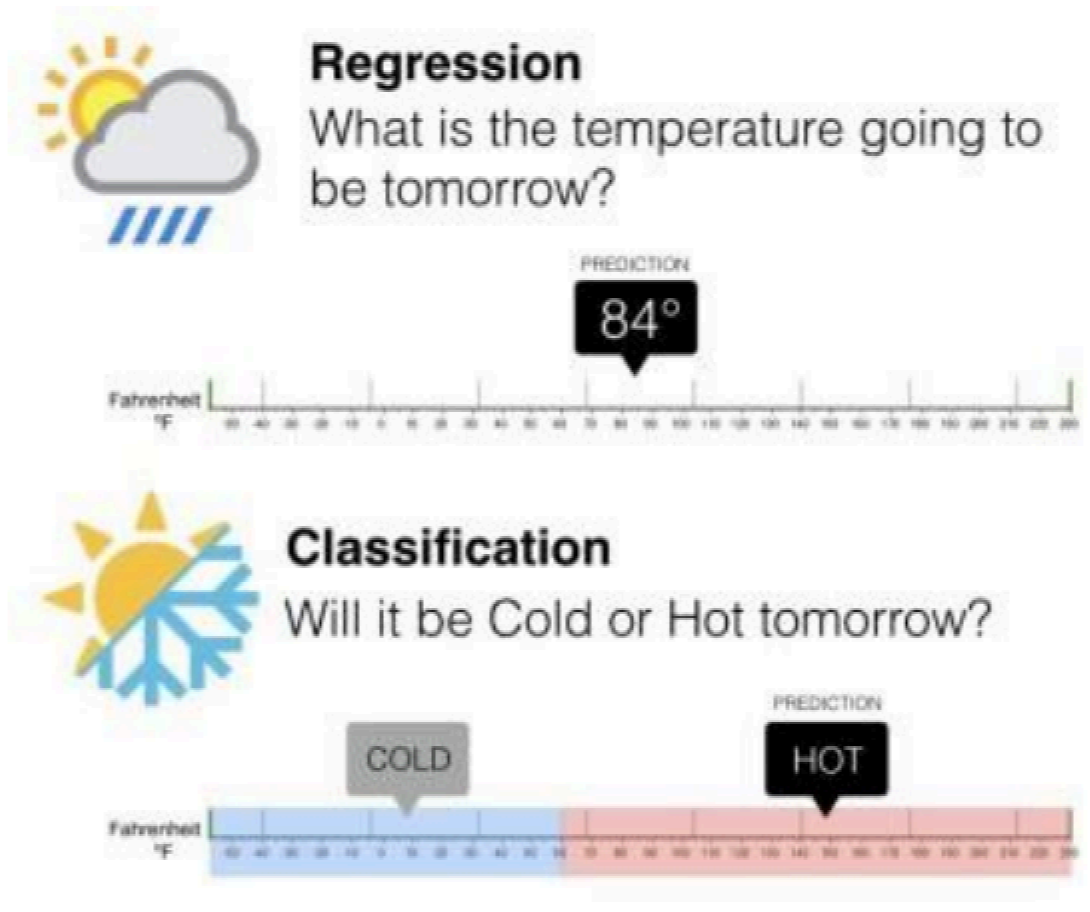
1. Supervised Learning



- **Definition:** In **supervised learning**, the agent learns a function that maps inputs to outputs based on a **dataset of "right answers"** (labeled examples) provided during training. The goal is to generalize from these examples to make accurate predictions on new, unseen inputs.
- The diagram shows labeled input data (apples) used to train a model that can then predict the label (apple) for new input. It also contrasts this with unsupervised

learning (no labels) and shows the typical outputs: classification or regression.

- Requires access to examples of **correct input-output pairs**.
- *Classic Example:* **Handwriting recognition**. The agent is shown many images of digits (inputs) along with their correct labels (outputs, e.g., "7"). It learns the patterns connecting images to labels.



Classification of Supervised Learning Tasks:

1. **Regression:**
   - Problem of **estimating or predicting** a **continuous quantity**.
   - *Examples:* Predicting tomorrow's temperature (page 40 diagram), the value of the S&P 500 next month, the height a child will reach as an adult, the number of customers likely to churn.
   - Requires identifying input **features** believed to be predictive of the continuous outcome.
2. **Classification:**
   - Problem of assigning observations into **discrete categories** (classes).

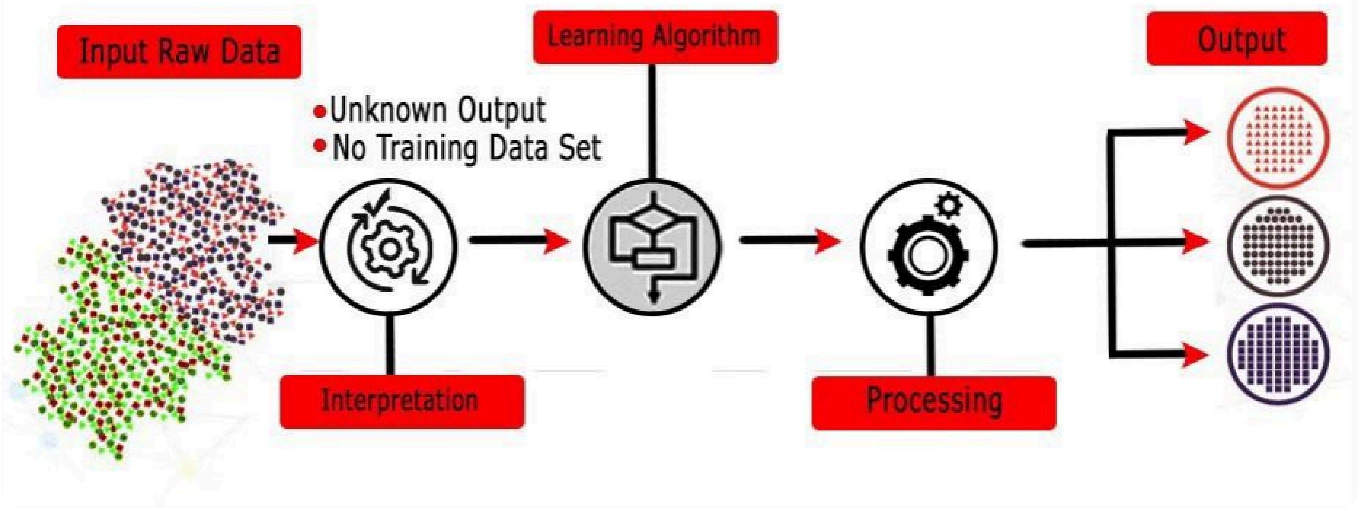
- *Examples:* Predicting if tomorrow will be "Cold" or "Hot" (page 40 diagram), if a customer will churn ("Yes" or "No"), if a patient has cancer ("Positive" or "Negative"), if an image contains a hot dog ("Yes" or "No").
- **Binary classification:** Problems with only two categories.
- *Multi-class Classification:* Problems with more than two categories (e.g., handwriting digits 0-9). Can sometimes be solved by training multiple binary classifiers (e.g., a 0-detector, 1-detector, etc.) and choosing the class whose detector is most confident.

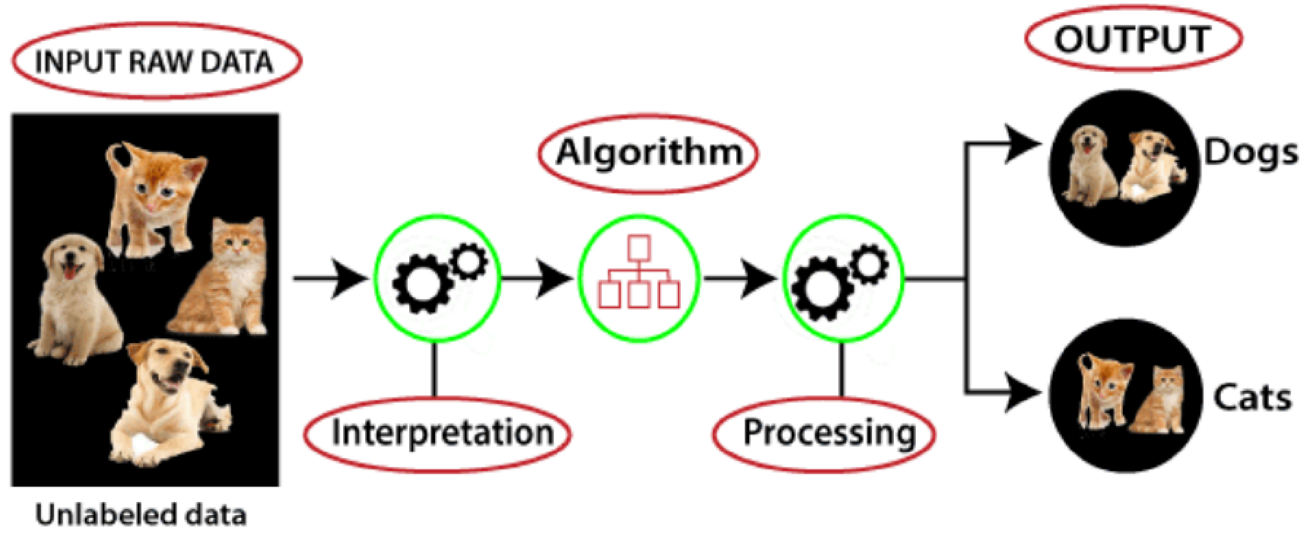
Common Supervised Learning Algorithms:

- Linear Regression
- Logistic Regression
- Neural Networks (can also be used in other paradigms)
- Linear Discriminant Analysis (LDA)
- Decision Trees
- Similarity Learning (e.g., k-Nearest Neighbors - KNN)
- Bayesian Logic (e.g., Naive Bayes)
- Support Vector Machines (SVM)
- Random Forests (Ensemble of Decision Trees)

2. Unsupervised Learning

Unsupervised Learning





Clustering

- **Definition:** In **unsupervised learning**, the agent **learns patterns in the input** data **without explicit feedback** or labeled examples. The "right answers" are not provided.
- *Goal:* To explore the data and discover inherent structure, groupings, or patterns within it.
- The diagrams (pages 45, 46) illustrate this: Raw, unlabeled data goes into an algorithm that interprets and processes it to find structure (e.g., grouping similar images like cats vs. dogs, or potatoes by type - page 48).
- Used with data that has **no historical labels**.
- The system is **not given** a predetermined set of correct outputs or correlations.
- The algorithm must discover patterns by itself, with no reference points.

Clustering: A Key Unsupervised Task

- **Clustering** is a major subclass of unsupervised learning.
- Refers to **grouping observations** together such that members within a group are similar to each other, and different from members of other groups.
- *Common Application:* **Marketing**, identifying customer segments with similar preferences or buying habits.
- *Major Challenge:* It's often difficult or impossible to know beforehand how many clusters should exist or what characteristics define a cluster.

Common Unsupervised Learning Algorithms:

- **Clustering:**
  - K-means clustering
  - Hierarchical clustering
  - DBSCAN
- **Dimensionality Reduction:**
  - Principal Component Analysis (PCA)
  - Singular Value Decomposition (SVD)
  - Independent Component Analysis (ICA)
- **Association Rule Learning:**
  - Apriori algorithm
- **Other:**
  - Anomaly detection
  - Neural Networks (e.g., Autoencoders)
  - KNN (can be used for density estimation)

3. Reinforcement Learning (RL)

- **Definition:** In **reinforcement learning**, the agent learns to behave optimally by **interacting with its environment** and receiving feedback in the form of **rewards** (for performing correctly/well) or **penalties** (for performing incorrectly/poorly).
- The agent learns **without direct instruction** from a human teacher; instead, it learns by **seeking the greatest cumulative reward and minimizing penalties** over time through trial and error.
- **Context Matters:** An action that leads to a reward in one situation might lead to a penalty in another. Learning is tied to the **context** (state) of the environment.
- The agent learns from a series of **reinforcements**.
  - *Example (Taxi):* Lack of a tip indicates something went wrong.
  - *Example (Chess):* Two points for a win indicate something went right.
- **Credit Assignment Problem:** A key challenge in RL is determining which of the *prior* actions were most responsible for a delayed reward or penalty.

RL Applications and Framework:

- Tends to be used for **gaming** (e.g., AlphaGo), **robotics** (learning motor skills), and **navigation**.
- The algorithm discovers sequences of actions (policies) that lead to maximum rewards through **trial and error**.
- **Markov Decision Process (MDP):** The standard mathematical framework for modelling reinforcement learning problems where the environment's response

depends only on the current state and the action taken (the Markov property).

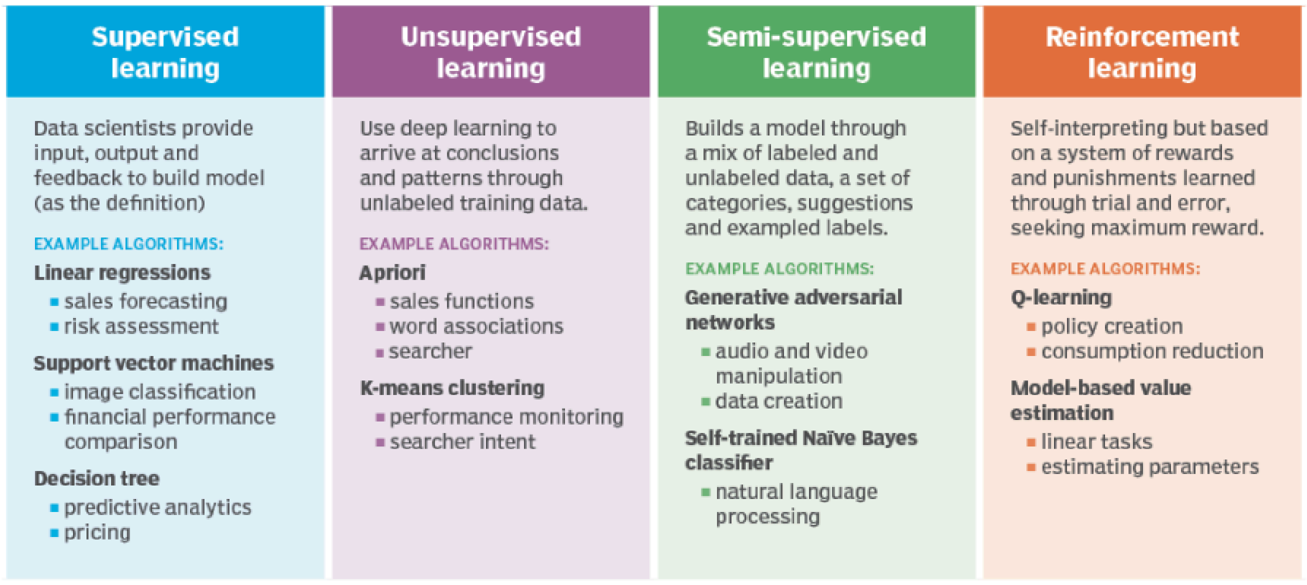
RL Example (Facebook News Feed - *Nuance*)

- Facebook's News Feed personalization uses machine learning. Frequent interaction (reading/liking) with a friend's posts leads to seeing more of their activity.
- *Connection to RL:* User engagement (likes, reads) can be seen as an implicit **reward signal**. The algorithm learns a policy (which posts to show) to maximize this reward.
- *Nuance:* While it involves learning from interaction and feedback, complex systems like news feeds often use a mix of supervised learning (predicting click-through rates), recommendation systems, and sometimes online learning or bandit algorithms, which are related to but distinct from classic RL focused on sequential decision-making in an MDP.

4. Semi-supervised Learning

- Falls **between supervised and unsupervised learning**.
- Uses a **combination of labeled and unlabeled data**. Typically, a small amount of labeled data and a large amount of unlabeled data.
- Useful when acquiring labels is expensive or time-consuming, but unlabeled data is plentiful.
- Aims to leverage the structure found in the unlabeled data to improve upon the model learned from the labeled data alone.
- Many real-world AI problems benefit from this balanced approach.

Machine Learning Models Cheat Sheet



Supervised learning	Unsupervised learning	Semi-supervised learning	Reinforcement learning
Data scientists provide input, output and feedback to build model (as the definition)	Use deep learning to arrive at conclusions and patterns through unlabeled training data.	Builds a model through a mix of labeled and unlabeled data, a set of categories, suggestions and exampled labels.	Self-interpreting but based on a system of rewards and punishments learned through trial and error, seeking maximum reward.
EXAMPLE ALGORITHMS:	EXAMPLE ALGORITHMS:	EXAMPLE ALGORITHMS:	EXAMPLE ALGORITHMS:
Linear regressions ▪ sales forecasting ▪ risk assessment	**Apriori** ▪ sales functions ▪ word associations ▪ searcher	**Generative adversarial networks** ▪ audio and video manipulation ▪ data creation	**Q-learning** ▪ policy creation ▪ consumption reduction
Support vector machines ▪ image classification ▪ financial performance comparison	**K-means clustering** ▪ performance monitoring ▪ searcher intent	**Self-trained Naïve Bayes classifier** ▪ natural language processing	**Model-based value estimation** ▪ linear tasks ▪ estimating parameters
Decision tree ▪ predictive analytics ▪ pricing			

- **Summary:** This table provides a concise overview of the four learning paradigms (Supervised, Unsupervised, Semi-supervised, Reinforcement), their definitions, example algorithms, and typical use cases (e.g., Linear Regression for forecasting, K-means for clustering, Generative Adversarial Networks in semi-supervised, Q-learning in RL).

Inductive Learning & Decision Trees

Inductive Learning

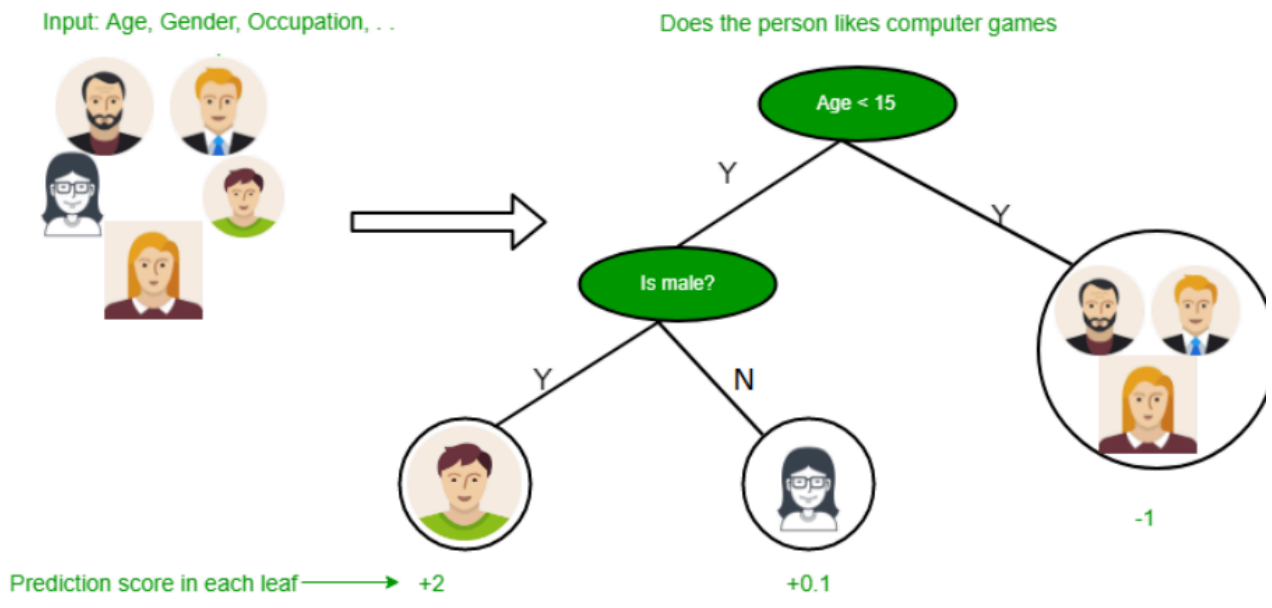
- **Inductive Learning** is the process of **learning by example**.
- The system tries to **induce a general rule** from a **set of observed instances**.
- Often involves **classification**: assigning an input to the correct class/category.
- Classification is fundamental to many problem-solving tasks.
- A key capability for learning systems is evolving their own class descriptions, as initial definitions might be inadequate or the world might change.
- The task of constructing class definitions from data is called **induction** or **concept learning**.

Learning Decision Trees

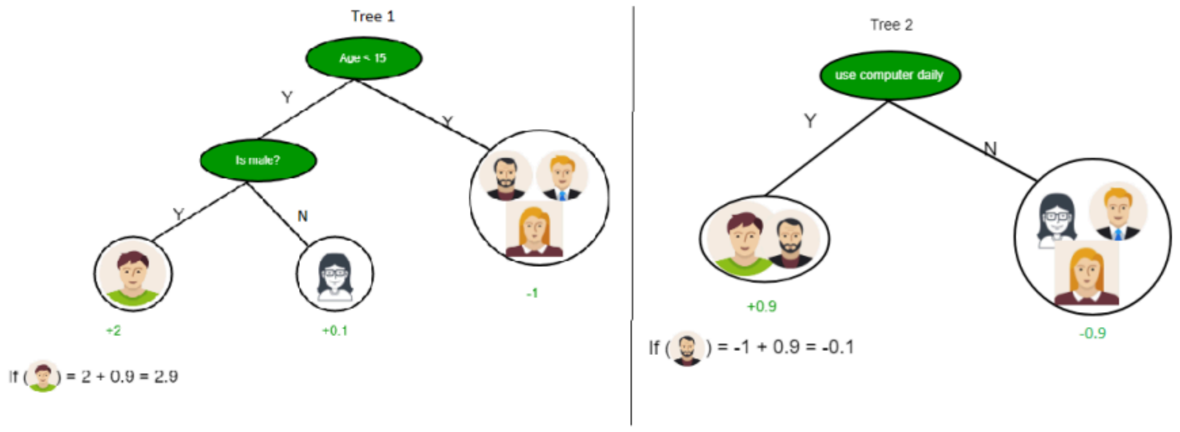
- A **decision tree** provides a simple, hierarchical representation for classifying examples. It's essentially a flowchart of tests.
- Decision tree learning is one of the most successful and widely used techniques for **supervised classification learning**.
- **Assumptions:** Typically assumes features have finite discrete domains (though extensions exist for continuous features) and there's a single target feature called the **classification**. Each possible value of this target feature is a **class**.

Decision Tree Structure and Example:

- **Structure:** Trees consist of nodes representing tests on input features (e.g., `Age < 15`, `Is male?`), branches corresponding to the outcomes of the tests, and leaf nodes predicting the classification (or a score/probability).
- **Example (Page 59 - Computer Games):** Input features (Age, Gender, etc.) are tested. Following the branches leads to a prediction score (e.g., +2 if Age < 15 and Is male).


Input: Age, Gender, Occupation, . .
Does the person likes computer games
Age < 15
Y
Y
Is male?
Y
N
-1
Prediction score in each leaf
+2
+0.1

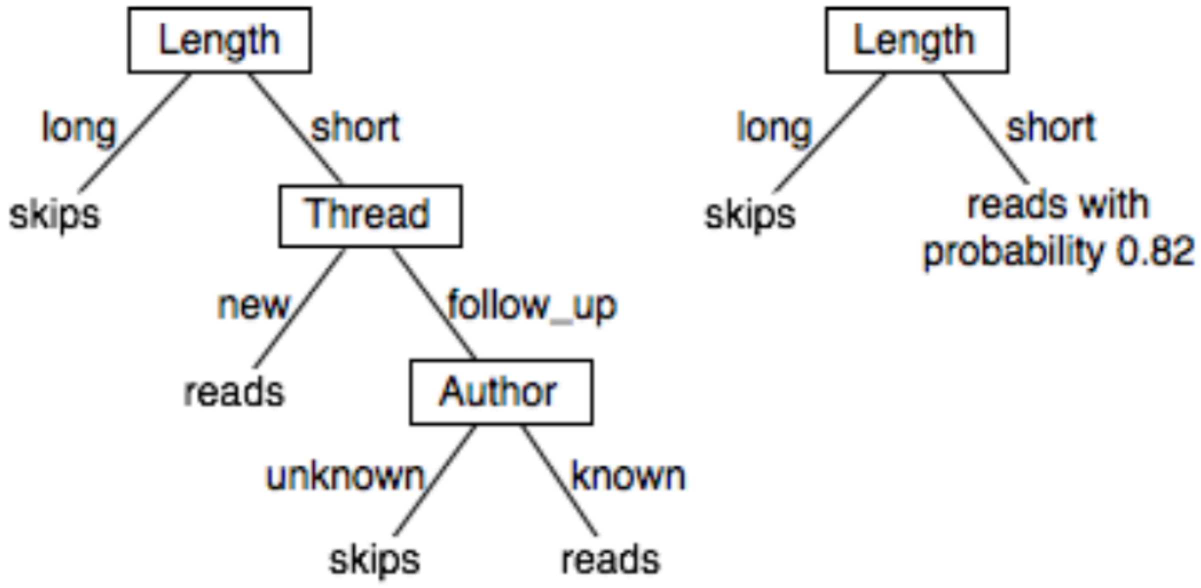
-
- **Example (Page 60 - Multiple Trees):** Shows two different trees potentially learned for the same task. Predictions from multiple trees can be combined (e.g., averaging scores, voting) – this is the basis of ensemble methods like Random Forests.



-

Decision Tree Example:

- **Goal:** Classify emails based on user actions (e.g., predict if the user will 'read' or 'skip').



-
- **Tree 1 (Left Diagram):**
  1. Check `Length`. If `long`, predict `skips`.
  2. If `short`, check `Thread`. If `new`, predict `reads`.
  3. If `follow_up`, check `Author`. If `known`, predict `reads`.
  4. If `unknown`, predict `skips`.
  - This tree makes deterministic predictions.
- **Tree 2 (Right Diagram):**
  1. Check `Length`. If `long`, predict `skips`.
  2. If `short`, make a **probabilistic prediction**: predict `reads` with probability 0.82 and `skips` with probability 0.18.
  - This tree handles uncertainty or provides confidence scores.

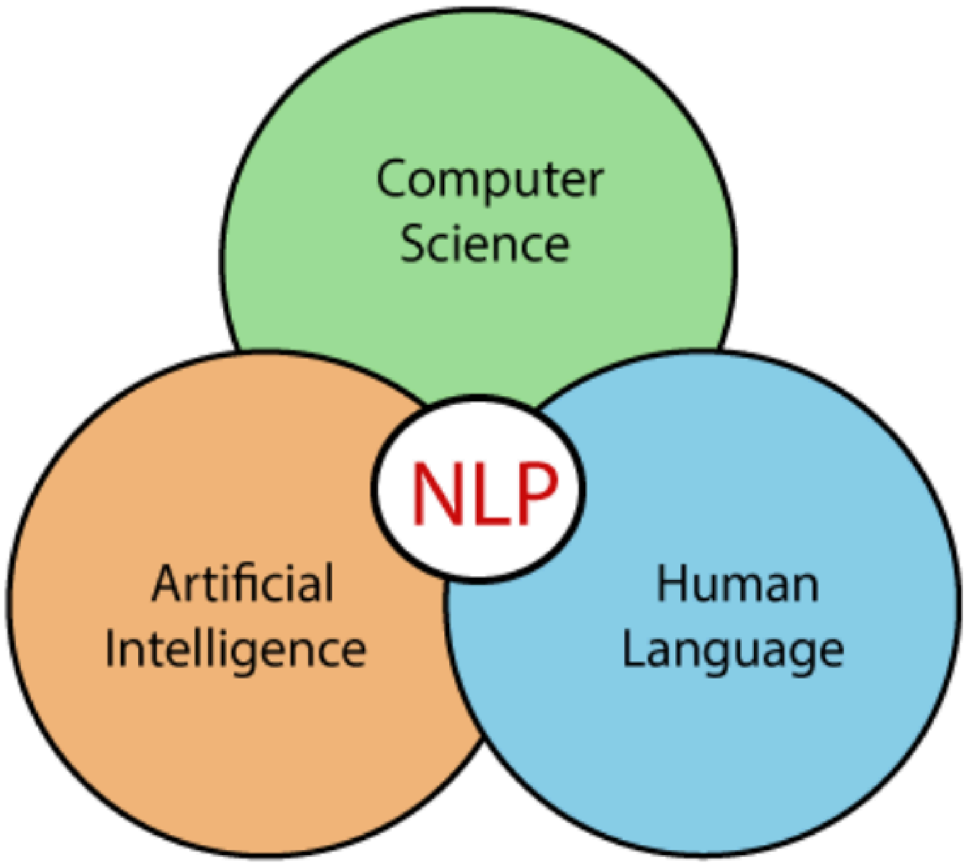
Key Points Summary: Learning

- **Learning** enables agents to improve performance over time by observing the world.
- **Reasons for Learning:** Unpredictable situations, changes over time, difficulty in direct programming.
- **Learning Factors:** Component to improve, prior knowledge, representation, feedback type.
- **Types of Learning:**

    - **Supervised:** Learns from labeled data (input-output pairs); includes **Regression** (continuous output) and **Classification** (discrete categories).
    - **Unsupervised:** Learns patterns from unlabeled data; key task is **Clustering**.
    - **Reinforcement:** Learns from rewards/penalties via environmental interaction (trial and error); formalized by **MDPs**.
    - **Semi-supervised:** Uses a mix of labeled and unlabeled data.
- **Inductive Learning:** Generalizing rules from specific examples (concept learning).
- **Decision Trees:** A successful supervised classification technique using a tree structure of tests to classify examples; can be deterministic or probabilistic.

Applications of AI: NLP and Expert Systems

Natural Language Processing (NLP)



- **Concept:** NLP sits at the intersection of Artificial Intelligence, Computer Science, and Human Language (Linguistics).
- **Definition: Natural Language Processing (NLP)** refers to AI methods enabling computers to process, understand, and generate human language (like English) for communication and performing useful tasks.
- **Usage:** Used in intelligent systems (e.g., robots understanding instructions, dialogue-based expert systems providing decisions).
- **Goal:** Making computers perform useful tasks involving the natural languages humans use.
- **Input/Output:** Can be **Speech** or **Written Text**.

Components of NLP

1. **Natural Language Understanding (NLU):**
   - Focuses on enabling machines to **understand and analyze** human language.
   - Involves extracting **metadata** like **concepts, entities, keywords, emotion, relations, and semantic roles** from content.
   - *Application:* Business applications using NLU to understand customer problems from spoken or written language.
   - *Tasks:*
     - Mapping input language into a useful internal representation.
     - Analyzing various aspects of the language (syntax, semantics, pragmatics).
2. **Natural Language Generation (NLG):**
   - Acts as a **translator** that **converts computerized data (internal representations) into natural language** representation (text or speech).
   - *Tasks:*
     - **Text planning:** Deciding what information to convey.
     - **Sentence planning:** Structuring the information into sentences (lexical choice, grammar).
     - **Text Realization:** Converting the sentence plan into actual text using correct grammar and morphology.

Steps in a Typical NLP Pipeline

Building an NLP system often involves several sequential steps:

1. **Sentence Segmentation:** Breaking a paragraph into individual sentences. (Paragraph → Sentences)
2. **Word Tokenization:** Breaking sentences into individual words or tokens. (Sentences → Words/Tokens)

3. **Stemming:** Reducing words to their root form by chopping off endings (e.g., "intelligence", "intelligent", "intelligently" → "intelligen"). *Crude but fast.*
4. **Lemmatization:** Reducing words to their dictionary form (lemma), considering context (e.g., "am", "is", "are" → "be"). *More sophisticated than stemming.* Grouping related forms together.
5. **Identifying Stop Words:** Removing common words (like "is", "and", "a", "the") that often don't carry significant meaning for the task.
6. **Dependency Parsing:** Analyzing the grammatical structure and relationships between words in a sentence.
7. **Part-of-Speech (POS) Tagging:** Assigning grammatical categories (Noun, Verb, Adverb, Adjective) to each word.
8. **Named Entity Recognition (NER):** Identifying and categorizing named entities in text (e.g., Person names: "Steve Jobs"; Organization names; Locations; Movie titles: "I phones" - *example notes typo, likely meant iPhone*).
9. **Chunking (Shallow Parsing):** Grouping individual pieces of information (like adjacent nouns and adjectives) into small phrases or "chunks" (e.g., Noun Phrases).

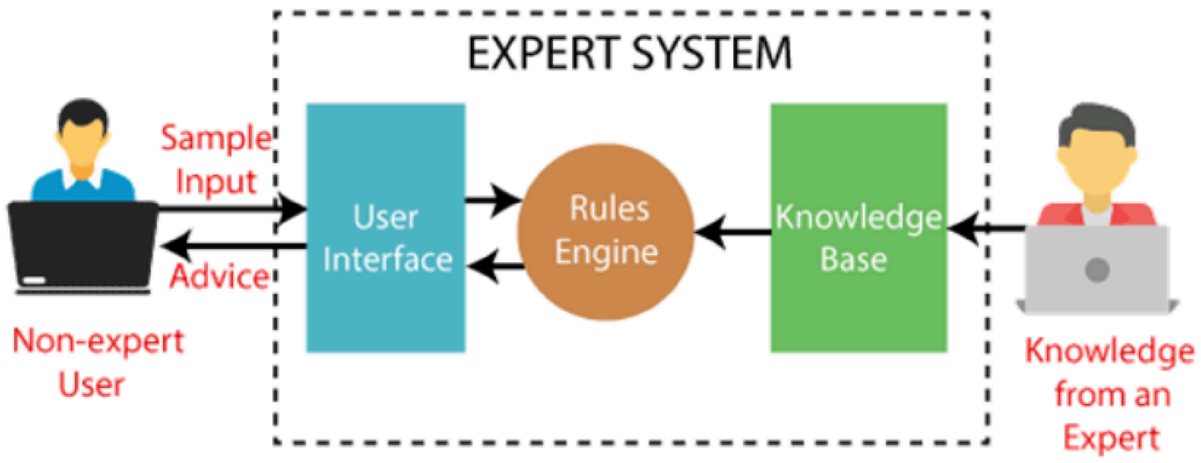
Applications of NLP

- **Question Answering:** Systems that automatically answer questions posed by humans in natural language.
- **Spam Detection:** Detecting unwanted emails based on their content.
- **Sentiment Analysis (Opinion Mining):** Analyzing text to determine the attitude, opinion, behaviour, or emotional state of the sender (e.g., positive/negative reviews).
- **Machine Translation:** (e.g., Google Translator)
- **Spelling Correction**
- **Speech Recognition**
- **Chatbots & Virtual Assistants**
- **Information Extraction & Retrieval**
- **Text Summarization**

Expert Systems (ES)

- One of the prominent and early research domains within AI.
- Introduced by researchers at Stanford University's Computer Science Department.

What are Expert Systems?



- **Definition: Expert Systems (ES)** are computer applications developed to solve **complex problems** within a **particular, narrow domain**, performing at a level comparable to an **extraordinary human expert** in that field.
- **Diagram Explanation (Page 70):** Shows the core components. A non-expert user interacts via the **User Interface**, providing input (e.g., symptoms). The **Inference Engine** uses rules (from the **Rules Engine**) and facts (from the **Knowledge Base**) to reason about the problem. The Knowledge Base is typically populated with knowledge acquired from a human **Expert**. The system provides output (e.g., diagnosis, advice) back to the user.

Characteristics of Expert Systems

- **High Performance:** Solve complex problems with high efficiency and accuracy within their specific domain.
- **Understandable:** Can often explain their reasoning process in a way users can understand (explanation facility). Can interact using human-like language (though often restricted).
- **Reliable:** Generate efficient and accurate output for problems within their domain.
- **Highly Responsive:** Provide results for complex queries relatively quickly.

Components of an Expert System

An ES mainly consists of three core components (as shown in the diagram):

1. **User Interface:** Allows interaction between the user and the system. Handles input and output presentation.

2. **Inference Engine:** The "brain" of the ES. Applies logical rules (from the knowledge base) to the input facts to deduce new facts or conclusions. Uses reasoning methods like:
    - **Forward Chaining:** Starts with known facts and applies rules to derive conclusions (data-driven).
    - **Backward Chaining:** Starts with a hypothesis (goal) and works backward to find facts supporting it (goal-driven).
3. **Knowledge Base:** Contains the domain-specific knowledge required to solve the problem. This includes:
    - **Facts:** Assertions about the domain.
    - **Rules:** Typically in IF-THEN format, representing heuristic knowledge or causal relationships. (e.g., IF `symptom = fever` AND `symptom = cough` THEN `suggest_disease = flu`).

Benefits of Expert Systems

- **Availability:** Easily replicated and distributed once developed (mass production of software). Available 24/7.
- **Less Production Cost (Potentially):** While development can be expensive, the cost per deployed unit can be low compared to hiring many human experts. Can make expertise more affordable.
- **Speed:** Offer high speed in processing information and reaching conclusions. Can reduce workload for individuals.
- **Less Error Rate:** Less prone to fatigue, emotion, or random errors compared to humans (within their specific domain). Consistency.
- **Reducing Risk:** Can operate in environments dangerous or inaccessible to humans.
- **Steady Response:** Work consistently without being affected by emotion, tension, or fatigue.

Limitations of Expert Systems

Expert systems are powerful but have significant limitations:

- **No "Magic Bullet":** No technology offers easy and complete solutions for all problems. ES are typically narrow.
- **Development Effort:** Large systems are costly, require significant development time, and substantial computer resources.
- **Limitations of the Technology:**
    - **Narrow Domain:** Expertise is limited to the specific domain they were built for; they lack general common sense.

    - **Brittle:** Performance can degrade rapidly when encountering problems outside their specific knowledge base.
- **Difficult Knowledge Acquisition:** Extracting, structuring, and encoding expert knowledge into the knowledge base (knowledge engineering) is often the bottleneck – it's difficult and time-consuming.
- **Maintenance:** ES can be difficult and costly to update and maintain as domain knowledge evolves.
- **High Development Costs:** Initial investment in building an ES can be very high.

Glossary of New Terms

- **ADL (Action Description Language):** An expressive language for defining planning actions, supporting features like conditional effects, negation, and variable types, extending STRIPS.
- **Action Schema:** A template defining a type of action in planning, including name, parameters, preconditions, and effects.
- **Classical Planning:** Planning under simplifying assumptions: fully observable, deterministic, static, finite, discrete environments.
- **Classification:** A supervised learning task of assigning inputs to predefined discrete categories or classes.
- **Clustering:** An unsupervised learning task of grouping similar data points together based on inherent patterns, without predefined labels.
- **Closed-World Assumption (CWA):** The assumption that any fact not explicitly stated as true in the knowledge base is false. Used in STRIPS states.
- **Conditional Planning:** Planning for non-deterministic environments (uncertain action outcomes) assuming full observability, often resulting in branched plans.
- **Contingent Planning:** Planning for partially observable environments, often requiring sensing actions to gather information.
- **Decision Tree:** A supervised learning model using a tree-like structure of tests on features to classify data points.
- **Expert System (ES):** An AI system designed to solve complex problems in a specific domain at the level of a human expert, using a knowledge base and inference engine.
- **Forward Chaining:** An inference strategy that starts with known facts and applies rules to derive conclusions.
- **Frame Problem:** The challenge in AI of efficiently representing and reasoning about what *remains unchanged* after an action occurs.

- **Hierarchical Planning:** A planning approach that decomposes complex, high-level actions into simpler, lower-level actions, organizing the plan hierarchically.
- **Inductive Learning:** Learning general rules or concepts from specific examples (learning by example).
- **Inference Engine:** The component of an expert system that applies rules from the knowledge base to facts to derive conclusions.
- **Knowledge Base:** The component of an expert system containing domain-specific facts and rules.
- **Lemmatization:** Reducing words to their base or dictionary form (lemma).
- **Markov Decision Process (MDP):** A mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker, commonly used in Reinforcement Learning.
- **Natural Language Generation (NLG):** The process by which AI systems produce natural language output (text or speech) from internal data representations.
- **Natural Language Processing (NLP):** The field of AI focused on enabling computers to understand, interpret, and generate human language.
- **Natural Language Understanding (NLU):** The subfield of NLP focused on machine reading comprehension and understanding the meaning of language.
- **Partial Order Planning (POP):** A planning technique that builds a plan by adding actions and ordering constraints only when necessary, maintaining flexibility and often decomposing the problem.
- **Planning:** The AI task of finding a sequence of actions to achieve a specific goal.
- **Progression Planning:** Forward state-space search planning, starting from the initial state and searching towards the goal.
- **Regression (in Learning):** A supervised learning task of predicting a continuous output value.
- **Regression Planning:** Backward state-space search planning, starting from the goal and searching towards the initial state.
- **Reinforcement Learning (RL):** A type of machine learning where an agent learns to make decisions by taking actions in an environment and receiving rewards or penalties.
- **Semi-supervised Learning:** Learning from a dataset containing both labeled and unlabeled examples.
- **State-Space Search:** A search method where the problem is represented as states and actions are transitions between states; used in planning.
- **Stemming:** Reducing words to their root form by crudely chopping off endings.
- **STRIPS (Stanford Research Institute Problem Solver):** A classic, restricted language for representing planning problems (states, goals, actions).

- **Supervised Learning:** Learning from a dataset where inputs are paired with correct outputs (labels).
- **Total Order Planning (TOP):** Planning that searches for a strictly linear sequence of actions.
- **Unsupervised Learning:** Learning to find patterns or structure in data without predefined labels or correct outputs.

Suggested Further Reading/Learning

- **Planning:** *Artificial Intelligence: A Modern Approach* (Russell & Norvig) - Chapters on Planning. Explore online resources on PDDL (Planning Domain Definition Language).
- **Machine Learning:** *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow* (Géron) for practical implementation. Coursera's Machine Learning course by Andrew Ng.
- **NLP:** *Speech and Language Processing* (Jurafsky & Martin). Online courses from Stanford or platforms like fast.ai. Explore libraries like NLTK, spaCy (Python).
- **Expert Systems:** *Artificial Intelligence: A Modern Approach* (Russell & Norvig) - Chapters on Knowledge Representation and Reasoning. Research historical systems like MYCIN, DENDRAL.

Concept Map Suggestion

Consider creating a concept map with "AI" at the center. Branch out to major areas like "Planning" and "Learning".

- Under "Planning":
  - Connect to "Goal", "Actions", "States".
  - Branch into "Classical Planning" (mention assumptions, STRIPS, ADL) and "Non-Classical Planning" (mention uncertainty, sensing, Conditional/Contingent).
  - Branch into techniques: "State-Space Search" (Progression/Regression), "POP", "Hierarchical Planning".
- Under "Learning":
  - Connect to "Improve Performance", "Data", "Feedback".
  - Branch into major types: "Supervised" (link to Regression, Classification, labeled data, Decision Trees, SVM etc.), "Unsupervised" (link to Clustering,

    dimensionality reduction, unlabeled data, K-means etc.), "Reinforcement" (link to Rewards/Penalties, Environment Interaction, MDPs, Q-Learning), "Semi-supervised".
  - Connect "Inductive Learning" primarily to "Supervised Learning".
- Under "AI Applications":
  - Branch to "NLP" (link to NLU, NLG, pipeline steps, applications like translation, sentiment analysis).
  - Branch to "Expert Systems" (link to components like Knowledge Base, Inference Engine, characteristics, benefits/limitations).

This visual map can help connect the diverse topics covered in this module.