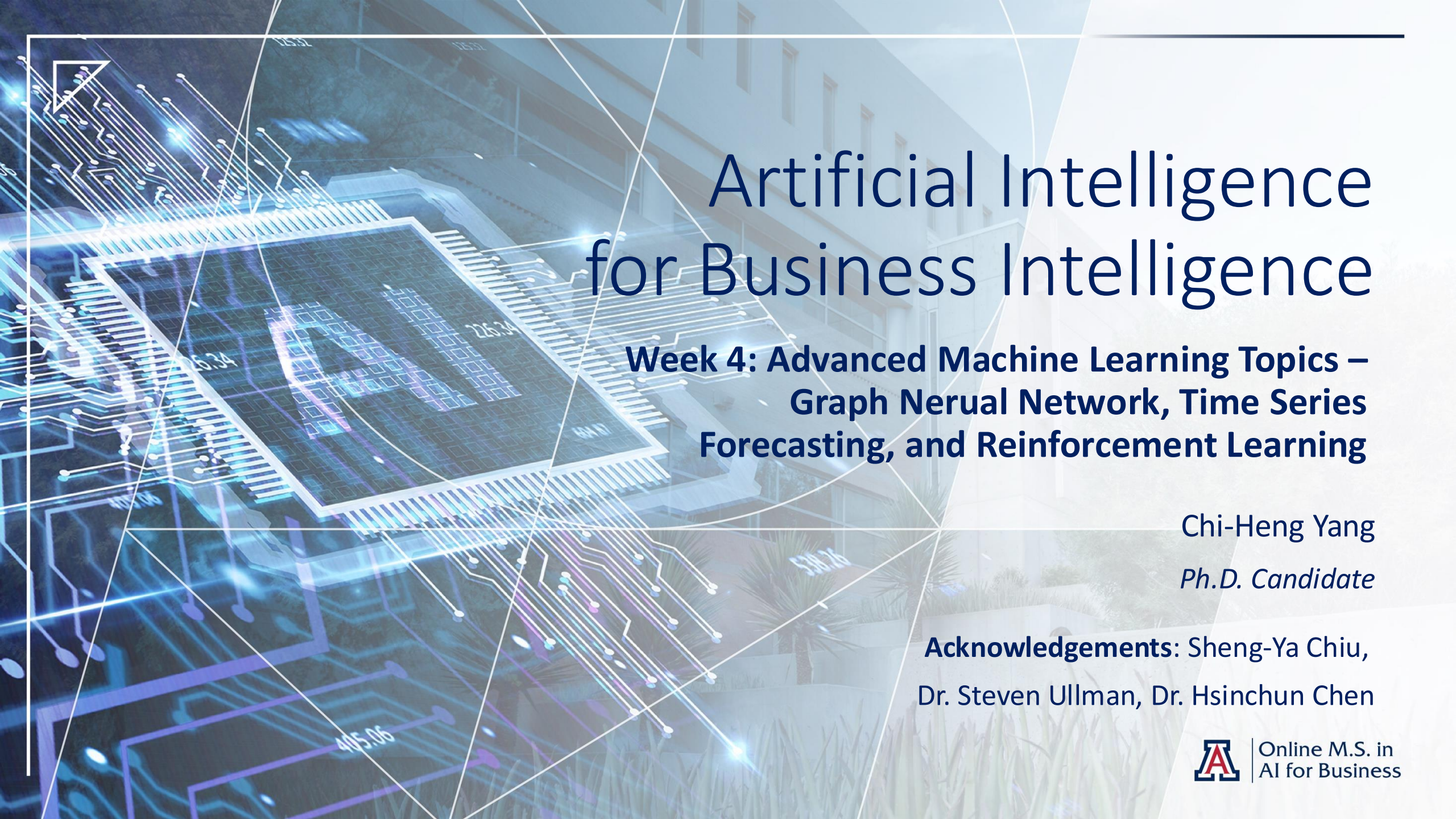


Artificial Intelligence for Business Intelligence

Week 4: Advanced Machine Learning Topics – Graph Nerual Network, Time Series Forecasting, and Reinforcement Learning

Chi-Heng Yang

Ph.D. Candidate

Acknowledgements: Sheng-Ya Chiu, Dr. Steven Ullman, Dr. Hsinchun Chen

Online M.S. in AI for Business

Today's Learning Objectives

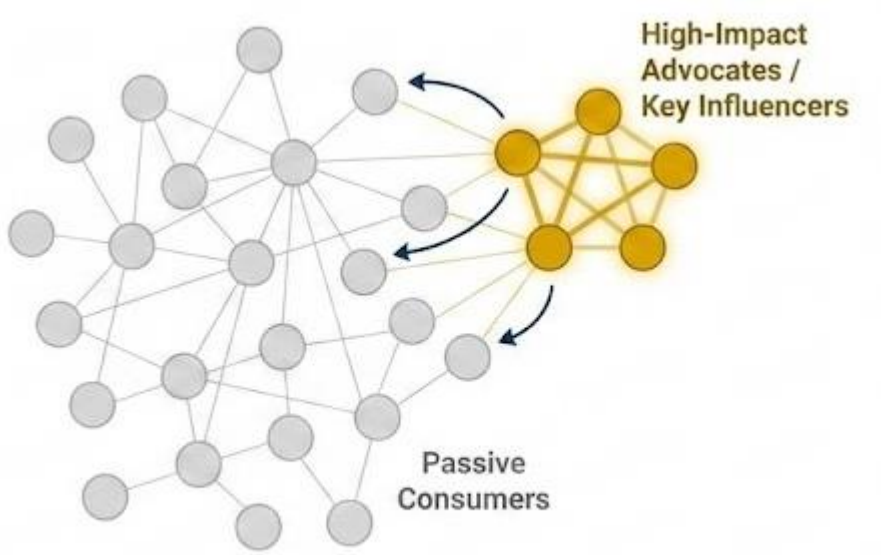
- **Module 1**: **Graph Neural Network**
  - Understand how GNN captures the connections in graph data.
  - Explain how to adopt GNN and graph transformers on node and edge-level tasks.
- **Module 2: Time Series Forecasting**
  - Compare the pros and cons of each time series forecasting model.
  - Understand how TSFM captures various types of temporal patterns.
- **Module 3: Reinforcement Learning**
  - Understand the problem characteristics that require reinforcement learning.
  - Explain the difference between policy and value-based methods.

Module 1: Graph Neural Network

Motivation

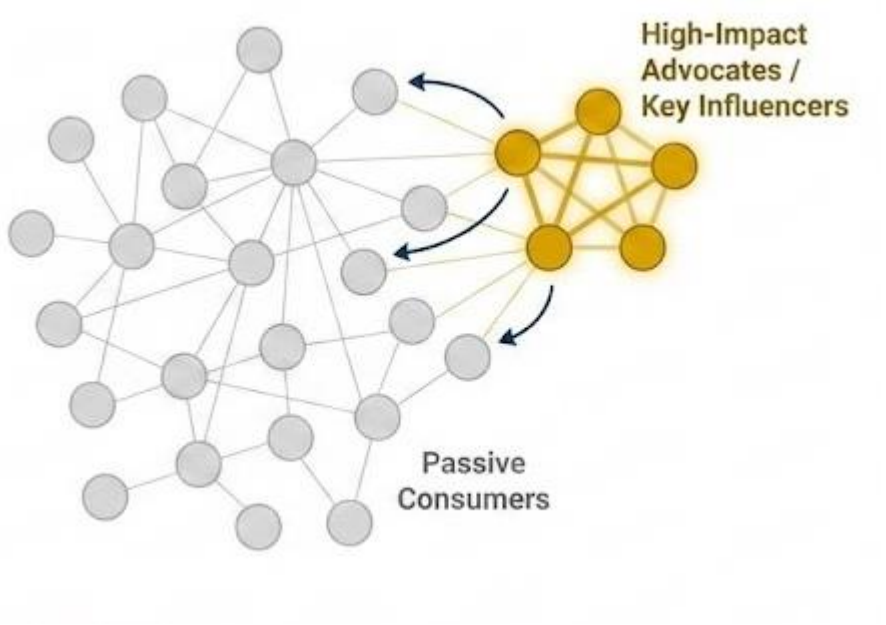
- Companies often want to find *Advocates* who aren‘t just posting a lot on social media, but are truly central to the social conversation.
  - Companies can offer exclusive early access to products or affiliate partnerships to them

- We want to know whether each consumer is:
  - **Passive Consumer** (Reads but doesn't lead).
  - **High-Impact Advocate** (The "Influencer" we want to hire).



Motivation

- Traditional ML can look at follower count, like count, and post frequency.
  - E.g, Use SVM and MLP for classification
- But whether users are influential depends on their connections.
- We need to capture this connected structure ➔ **Graph**

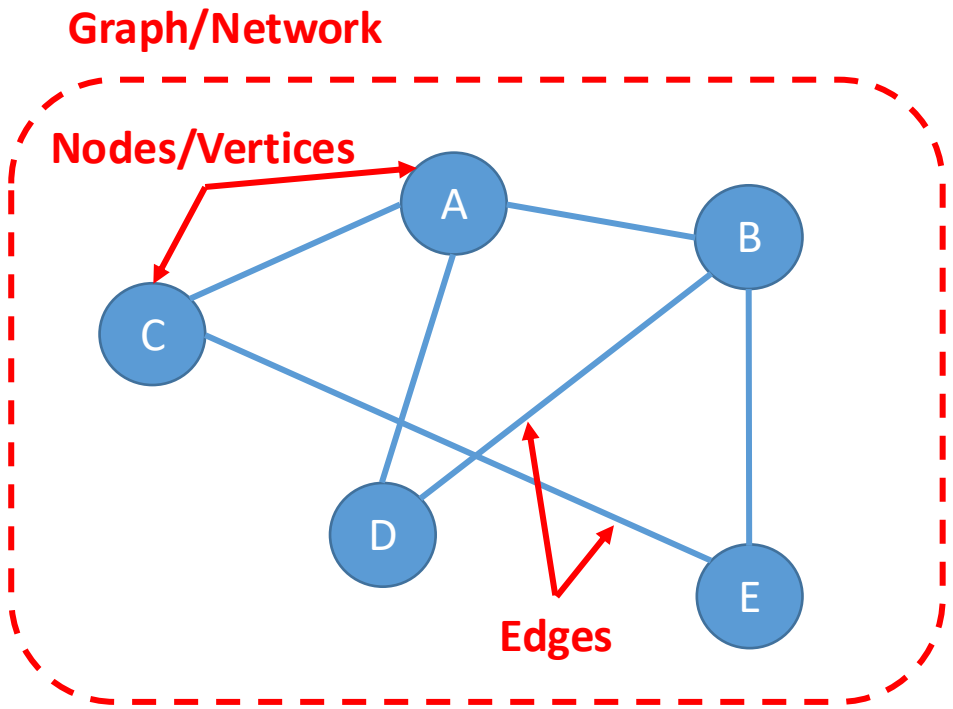


Graph

- A graph G is made up of two key features:
  - Nodes/Vertices (V)
  - Edges (E)

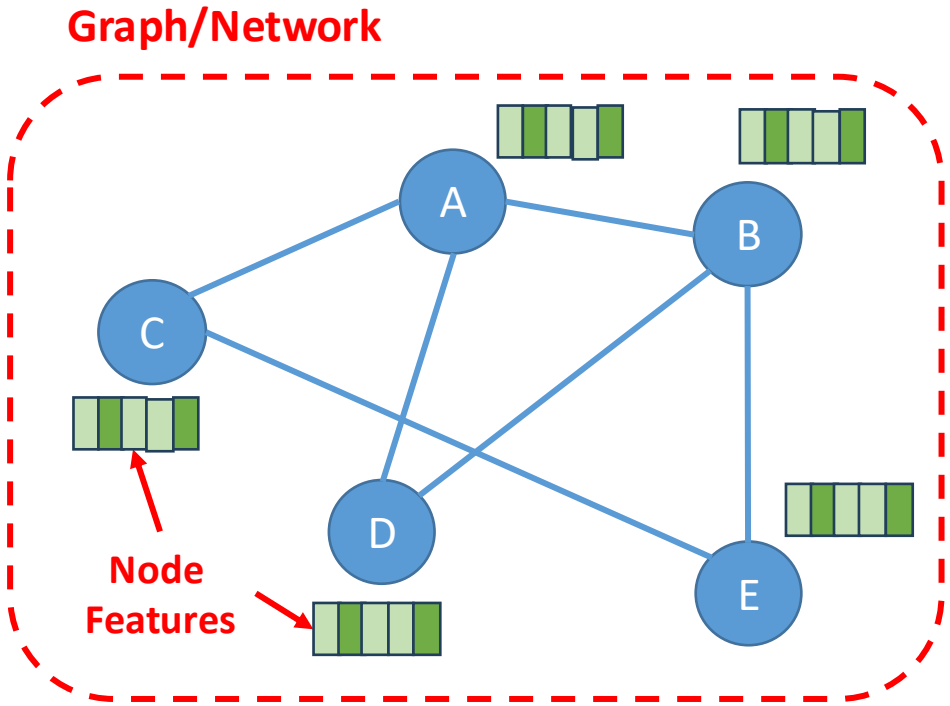
- **Nodes** (Vertices) represent entities (e.g., social media users).

- **Edges** represent the connections or relationships between the objects in a graph (e.g., social media following/connections).



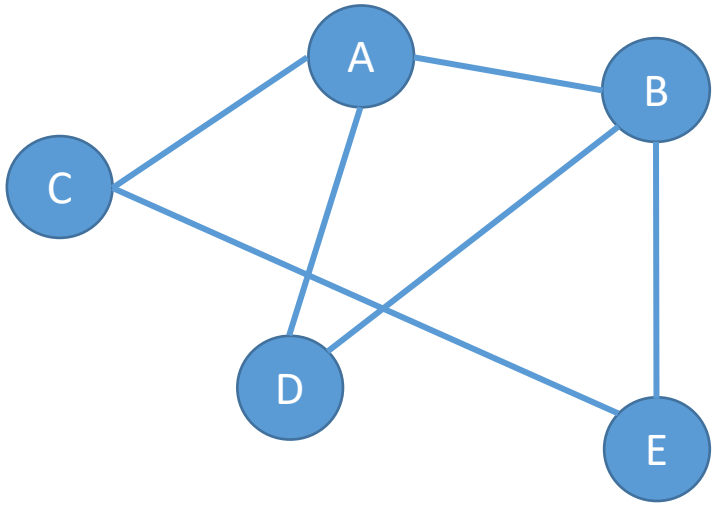
Graph

- Nodes in a graph can have features (denoted as X).
- Features are often used to represent attributes of each entity (e.g., user post frequency)
- The formal notation is then $G = (V, E, X)$.
  - $V_i \in \{V_1, V_2, \ldots, V_n\}$ is the node set.
  - $E_i \in \{E_1, E_2, \ldots, E_m\}$ is the edge set.
  - $X_i \in \{X_1, X_2, \ldots, X_n\}$ is the node feature set



Graph Data Representations

- A graph can also be represented numerically as an *adjacency matrix* (A).
  - It is often the technical input format to deep learning models.
- An $n \times n$ matrix of all nodes $v_1, \ldots, v_n$ represents the adjacency matrix $A_{i,j}$
  - 1 if there is a connection between v_i and v_j
  - 0 if there is no connection

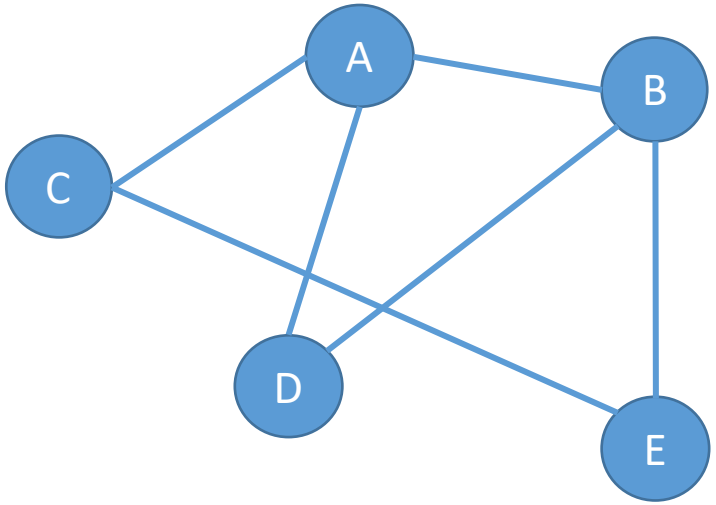


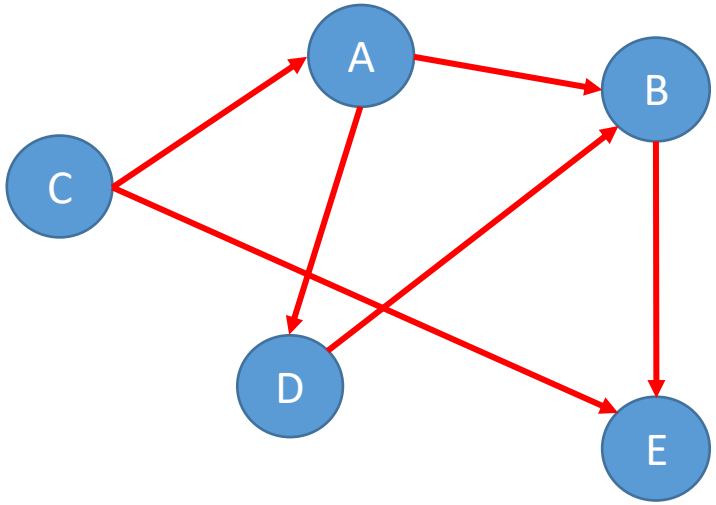
$$A_{i,j} = \begin{Bmatrix} 1 & if\ (v_i, v_j) \in E \\ 0 & otherwise \end{Bmatrix}$$

$$A_{ij} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Graph Data Representations

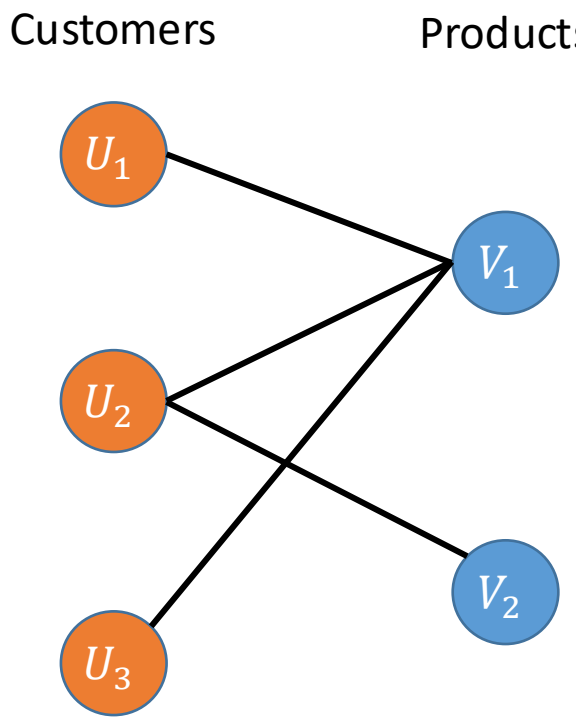
- Graphs can be undirected or directed:
  - **Undirected Graph:** Connections between vertices are bidirectional (e.g., transportation network).
  - **Directed Graph:** Connections between vertices are directional (e.g., social media following)





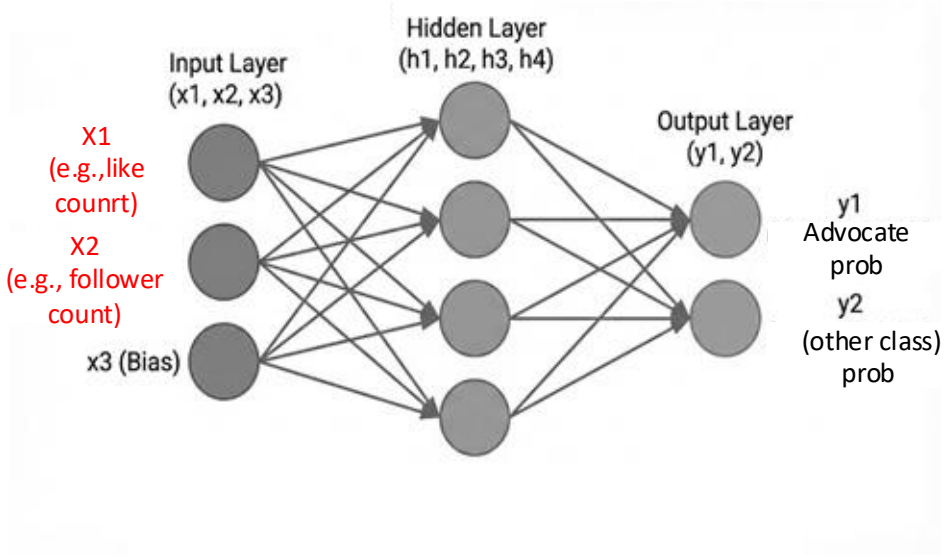
Graph Data Representations

- A Graph can have multiple types of nodes.
- For example, to capture the product that each customer bought for product recommendation:
  - **Nodes**: Customers, Products
  - **Edges**: Whether a customer buys a product
- If there are no connections between the same type of nodes, the graph is called a **bipartite graph**
  - E.g., no customer-customer or product-product connections.



Graph Neural Networks

- How do we build a neural network on graph data for classification?
- Applying a standard neural network (e.g., MLP) only captures features of each user (like count, follower count).
  - But the connections (edges) are not included.
- **Graph neural networks (GNN)** is designed to capture the connections and features at the same time.

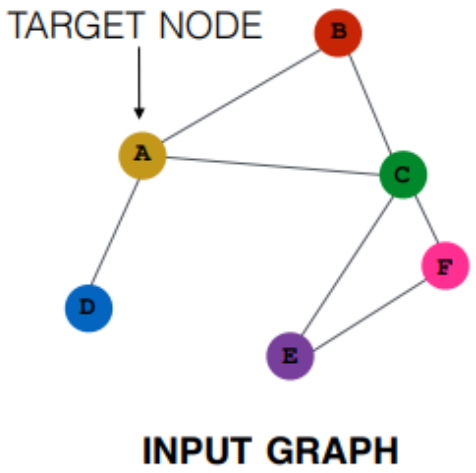


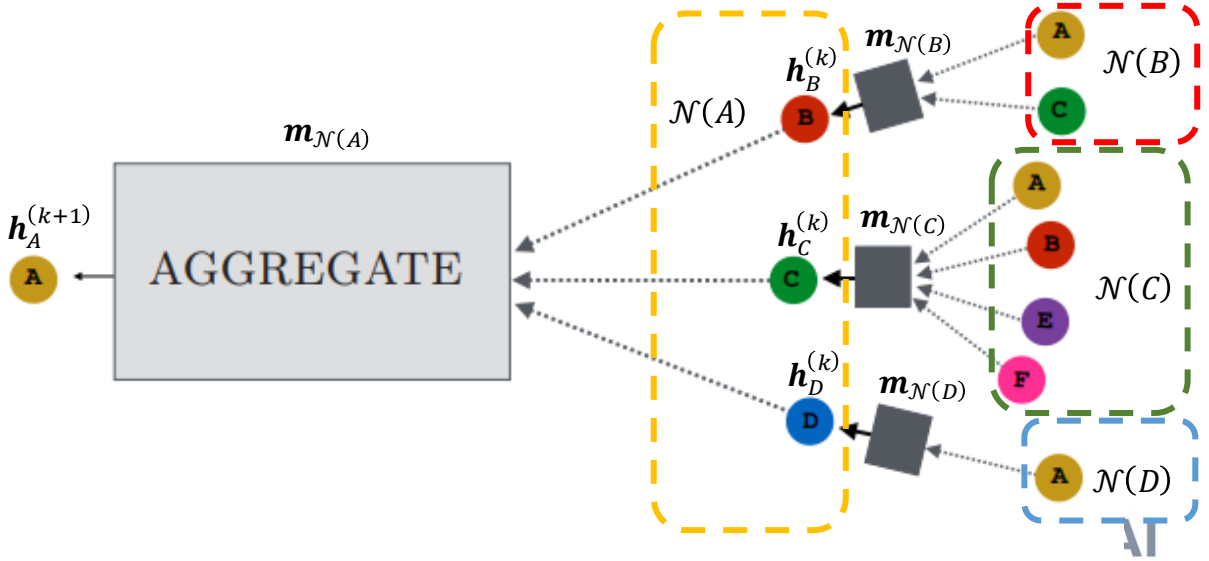
Graph Neural Networks

- The vanilla GNN is the **message passing framework.**
- Each message $\boldsymbol{m}_{\mathcal{N}(u)}$ represents the information gathered from each node neighborhood.
  - Messages include graph structures (number of connections) and node features.
- The $\mathrm{UPDATE}^{(k)}$ function then combines the aggregated messages $\boldsymbol{m}_{\mathcal{N}(u)}^{(k)}$ with the hidden node embeddings $\boldsymbol{h}_u^{(k)}$ to create the next node embedding $\boldsymbol{h}_u^{(k+1)}$.

Message Passing Function:

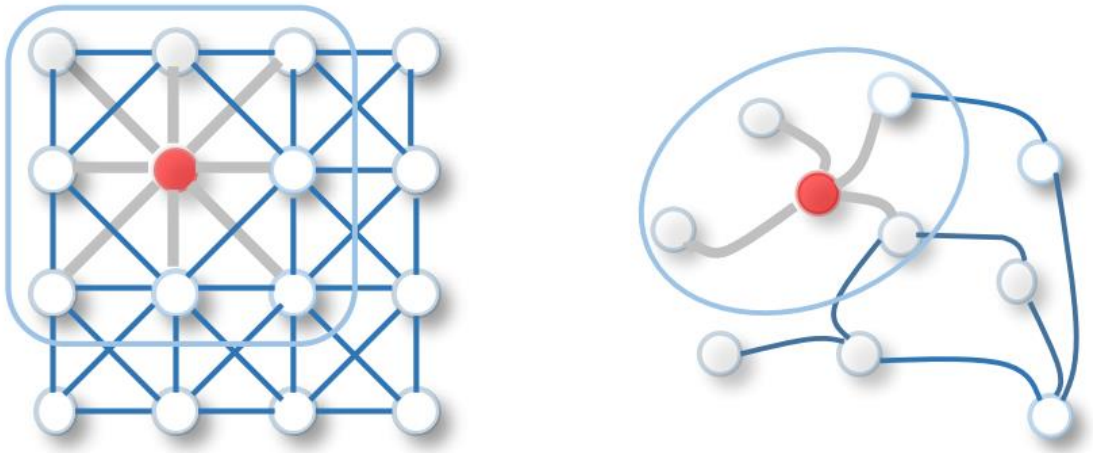
$$\boldsymbol{h}_u^{(k+1)} = \mathrm{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(k)}\right)$$





Graph Neural Networks – GCN

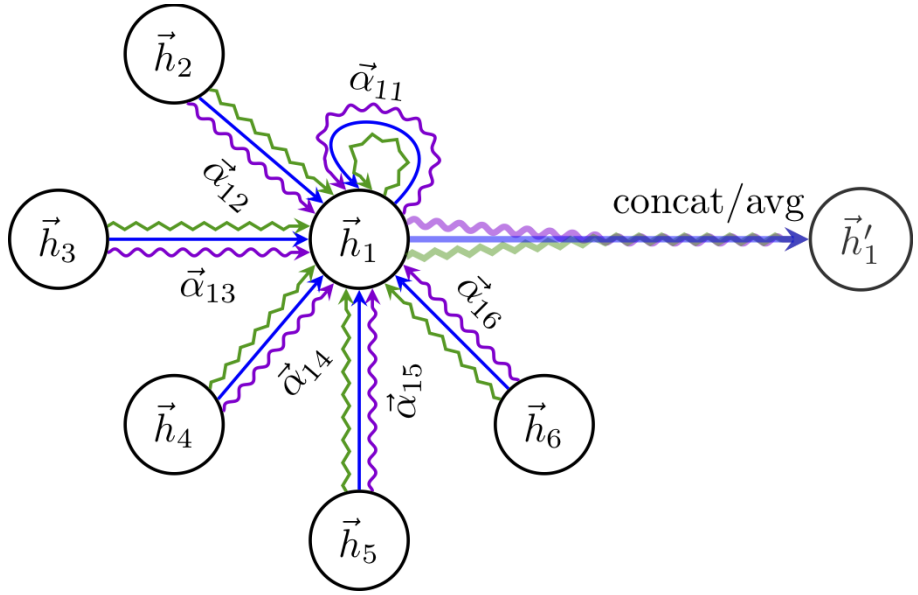
- Inspired by the success of CNNs, researchers adapted convolution operations into GNNs.
- This extension of the GNN is referred to as a **Graph Convolutional Network (GCN)**.
- GCNs extend the GNN aggregation operation to account for Neighborhood normalization and Self-loops (connections between the node and itself).



Convolution for Image (left) and Graph (right)

Graph Neural Networks – GAT

- Sometimes connections are not equally important.
  - E.g., Your actual purchasing habits are likely only influenced by 3-4 close friends, even if you have 500 friends on a social media platform.
- **Graph Attention Network** (**GAT**) incorporates an attention for aggregations to determine a "weight" for each connection.
  - Attention weights α_{ij}^{k} are computed based on k-attention layers.
- The attention scores $\vec{\alpha}_{1i}$ and neighbor embeddings $\vec{h}_i$ are then aggregated to create the next node embedding $\vec{h}_1'$.

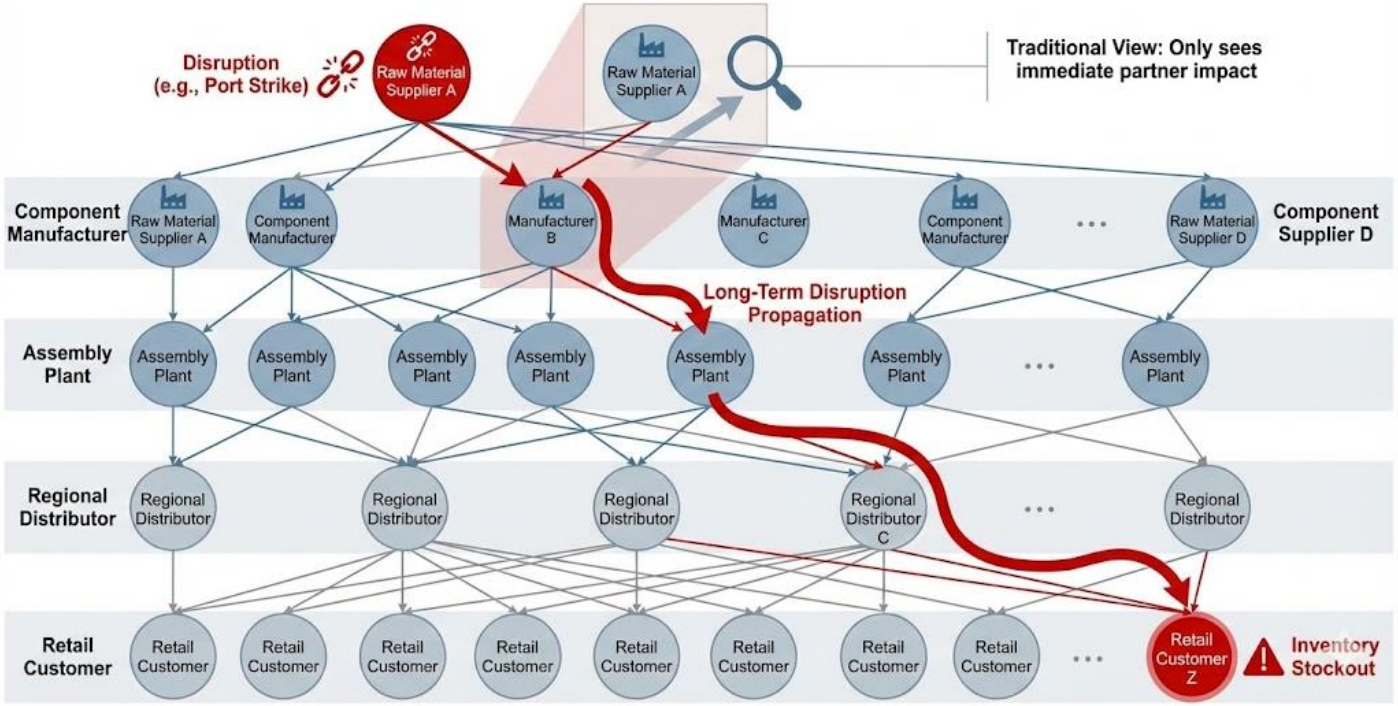


Attention Aggregation for *k* Layers:

$$\vec{h}_j' = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{k} \boldsymbol{W}^{k} \vec{h}_j\right)$$

Graph Neural Networks – Graph Transformers

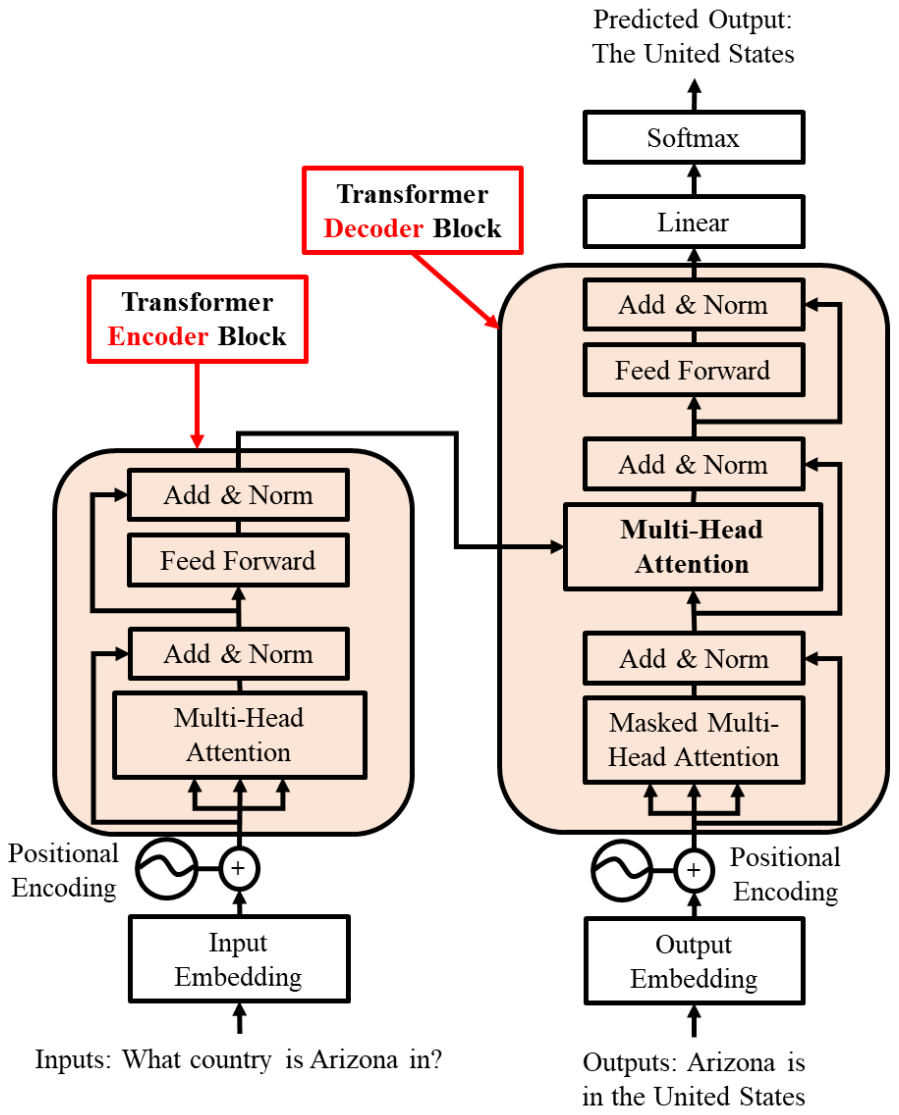
- Standard GNNs like GCN and GAT primarily aggregate information from immediate neighbors.
- In large networks, as signals travel across multiple "hops," the features of distant nodes tend to be diluted or lost before reaching the target.
  - E.g., supply chain network with 5-6 tiers of suppliers + customers



Recall: Transformers

- Transformers can handle long-range dependencies across long text sequences through **attention**.

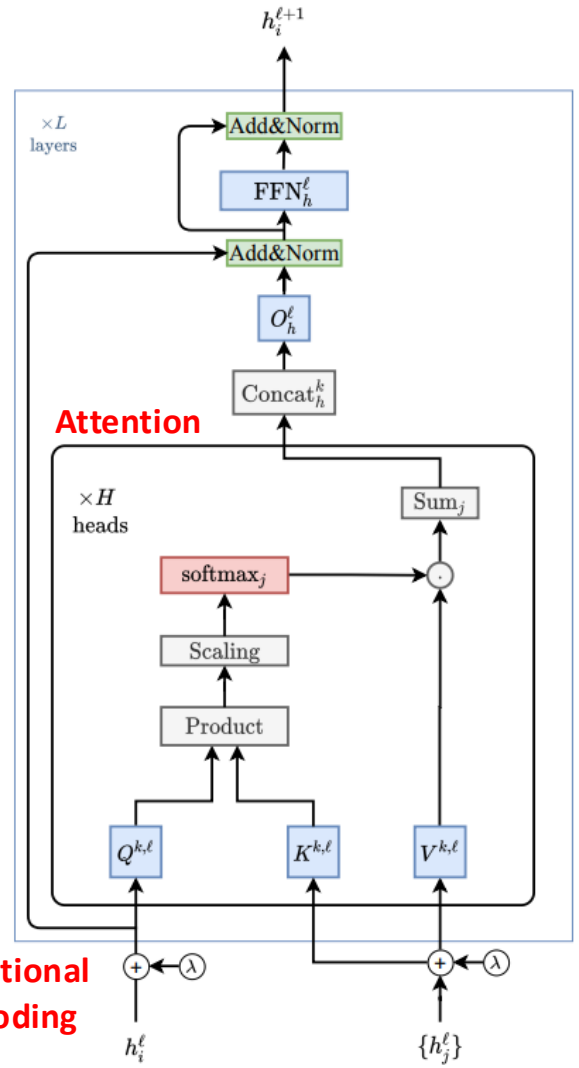
- Key mechanisms:
  1. **Embedding and encoding**: transform textual input into a numerical format
  2. **Attention:** capture the context by determining the importance of each word.



Graph Transformers

- **Graph transformer** is inspired by the transformer architecture.
- Instead of focusing on aggregation between neighbors, it captures information from all nodes in a graph through **attention**.
  - Good at capturing long-range dependencies (e.g., multi-hop propagations).
- **Key mechanisms**:
  - **Token choice:** use graph components (e.g., nodes) as tokens
  - **Positional encoding**: identify the role of each token in a graph (e.g., hub, leaves)
  - **Attention mechanism**: identify the **context** (e.g., other nodes) and their importance by looking at other tokens (not just neighbors)

$$\hat{h}_i^{\ell+1} = O_h^{\ell} \Big\|_{k=1}^{H} \Big(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^{\ell} \Big)$$

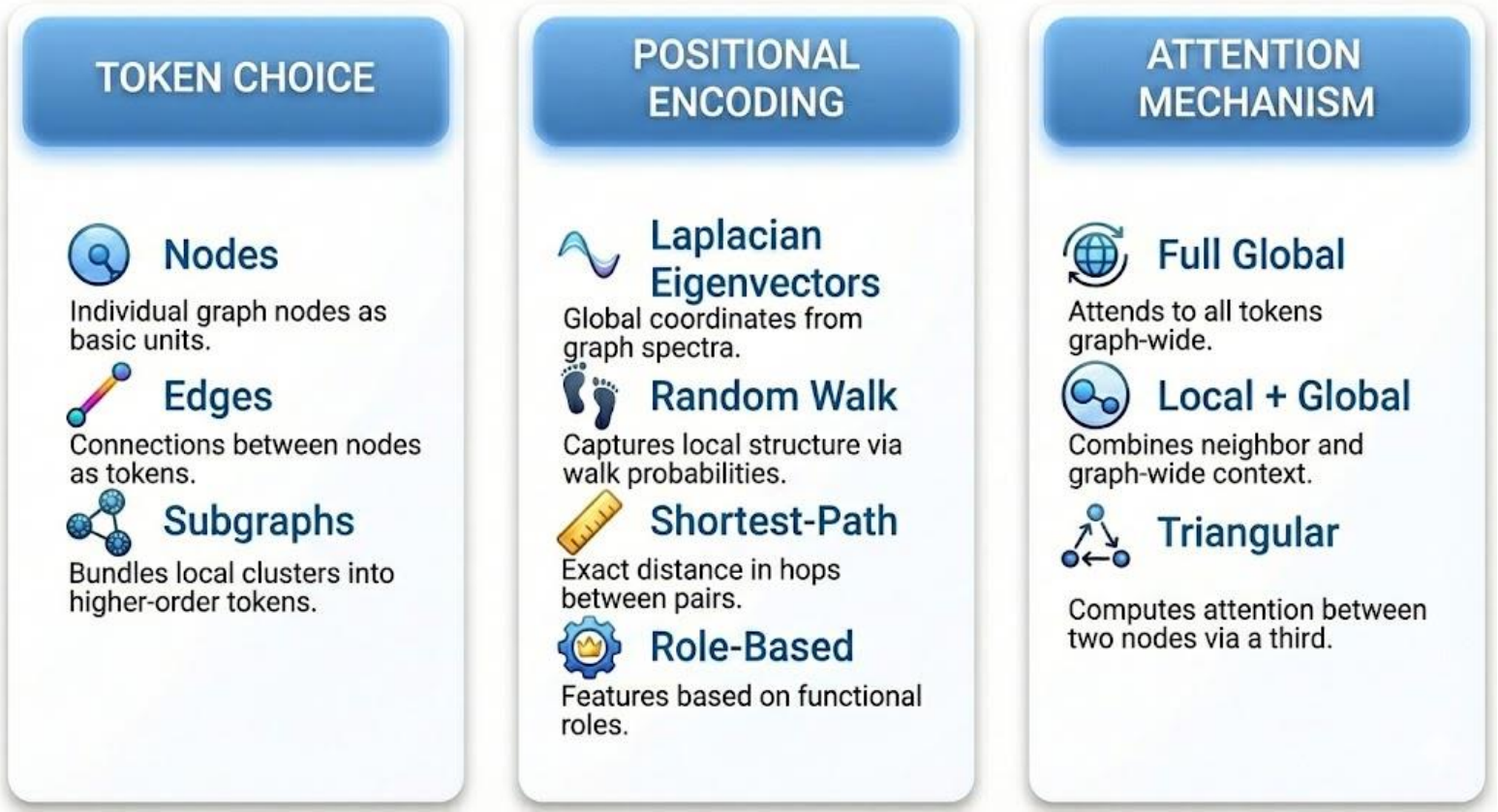


Graph transformer
(Adapted from Dwivedi and Bresson 2021)

Reference: Dwivedi, V. P., & Bresson, X. (2021, February). A generalization of transformer networks to graphs [Paper presentation]. AAAI Workshop on Deep Learning on Graphs: Methods and Applications (DLG-AAAI'21)

Graph Transformers – Variants

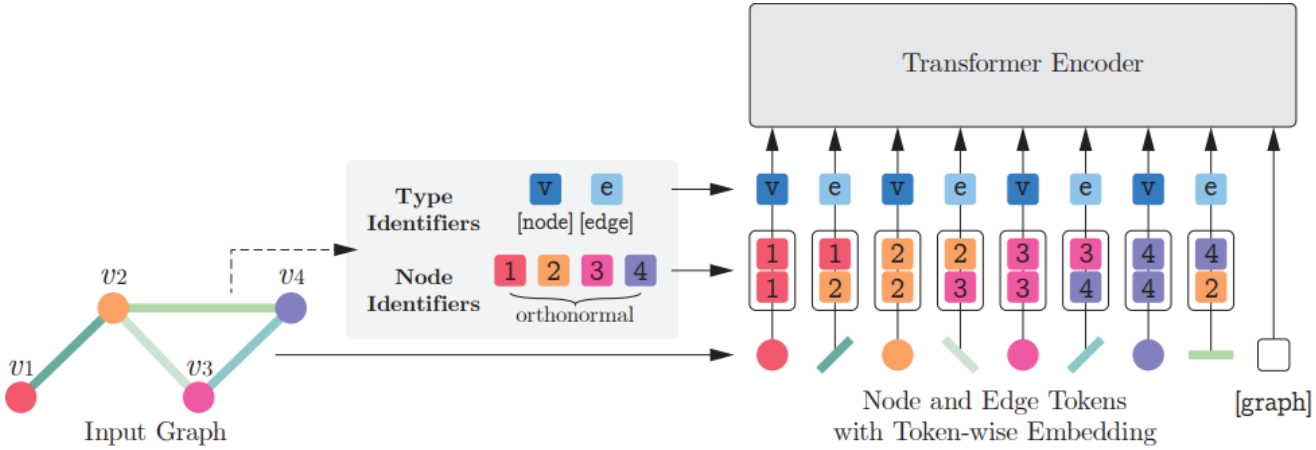
- Recent Graph Transformer development primarily focuses on three key mechanisms.



Graph Transformers – TokenGT

- **Tokenized Graph Transformer** (TokenGT) treats all nodes and edges as independent tokens, similar to words in a sentence (Kim et al. 2022).

- **Positional encoding via token augmentation:** extend the node or edge tokens with embeddings to help recognize the graph's connectivity structure.
  - **Type identifiers:** trainable parameters indicating whether a token is a node or edge
  - **Node identifiers:** one vector for each node, indicating the nodes that each token is associated with.

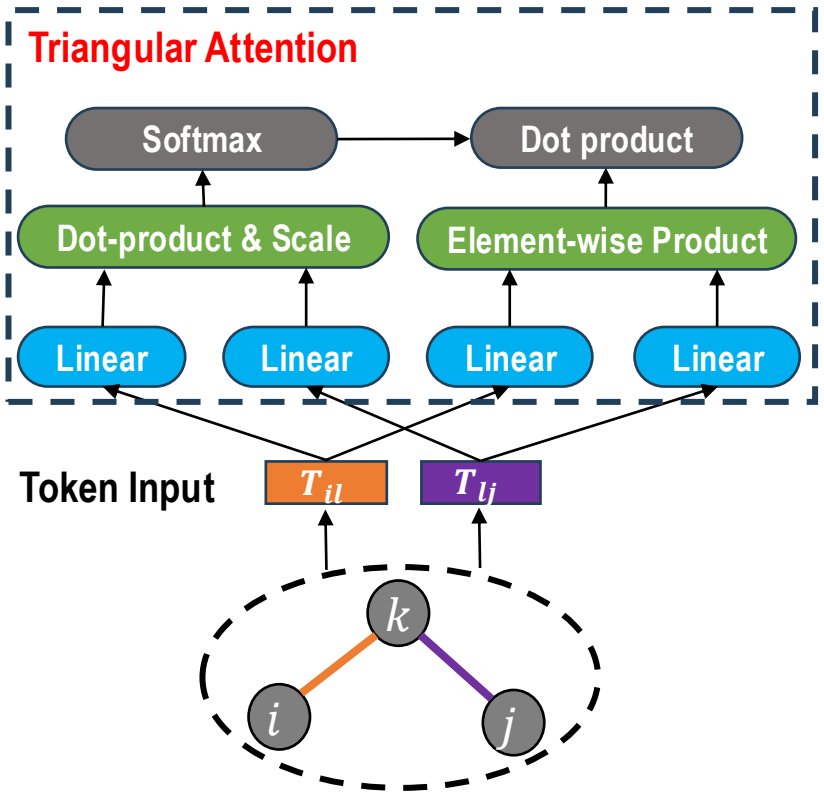


Reference: Kim, J., Nguyen, D., Min, S., Cho, S., Lee, M., Lee, H., & Hong, S. (2022). Pure transformers are powerful graph learners. Advances in Neural Information Processing Systems, 35, 14582-14595.

Graph Transformers – Edge Transformer

- **Edge Transformer** uses **node pairs** as tokens and enhances the standard attention to a **triangular attention**.

- **Triangular attention**: determine the weight between pairs of nodes(i, j) through a third node (k).
  - As the triangular attention already captures structural relationships between nodes, there is **no positional encoding**.

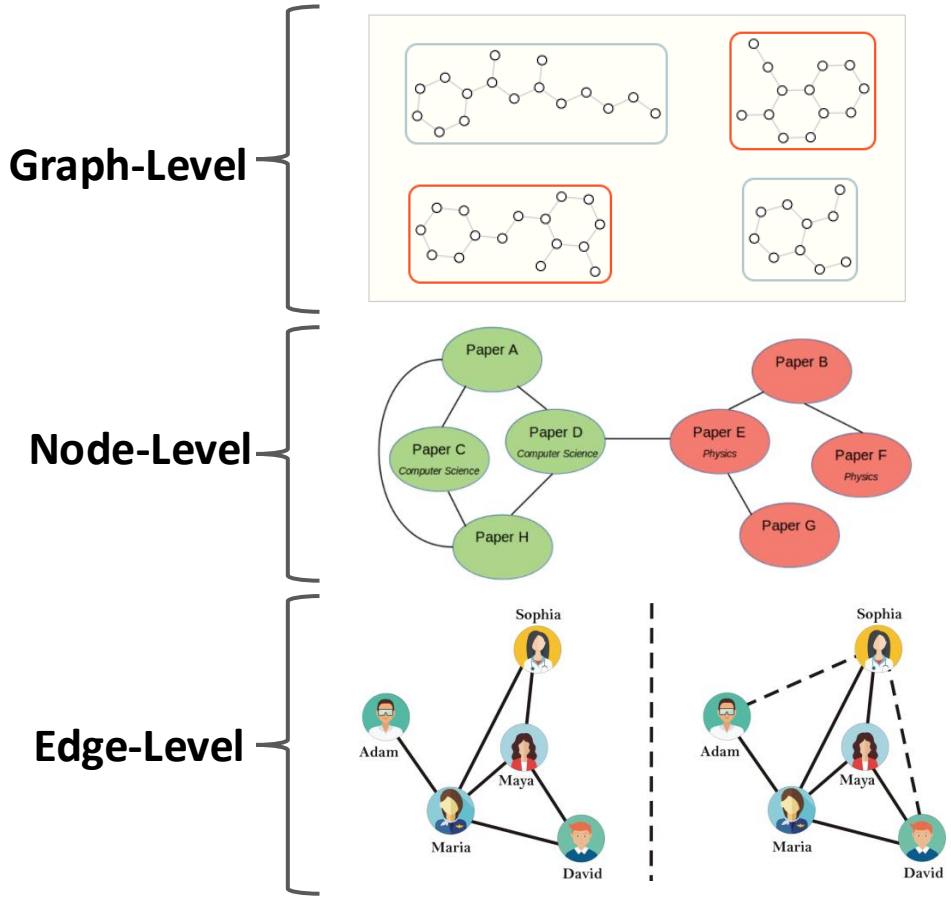


Edge Transformer operations (adapted from Muller et al. 2024)

Reference: Muller, L., Kusuma, D., Bonet, B., & Morris, C. (2024). Towards principled graph transformers. Advances in Neural Information Processing Systems, 37, 126767-126801.

Graph Neural Networks

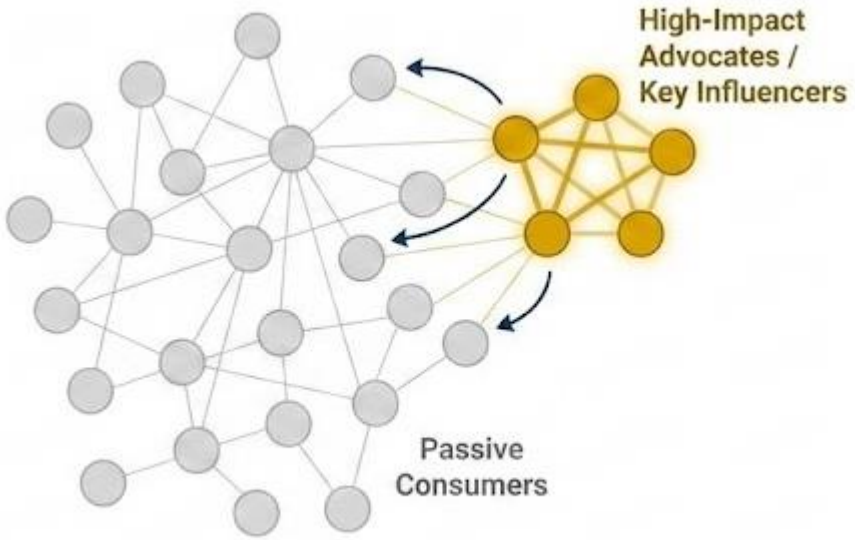
- GNN and graph transformers can be adopted for tasks at different levels (supervised):
  - **Graph-Level:**
    - Whole-graph classification → Classify proteins based on molecular structure.
  - **Node-Level:**
    - Node classification → Predict topic of paper based on citations.
  - **Edge-Level:**
    - Edge/link prediction → Predict if two friends are likely to have a connection/friendship.
- They can also learn representations for various downstream tasks (unsupervised).



GNN – Node Classification

- **Objective**: predict the label (category) of a node based on its own features **and** its position within the network.
  - **Input**: A graph $G = (V, E)$, node features X, and a subset of nodes with known labels Y_L.
  - **Output**: Predicted labels for the remaining unlabeled nodes Y_U.

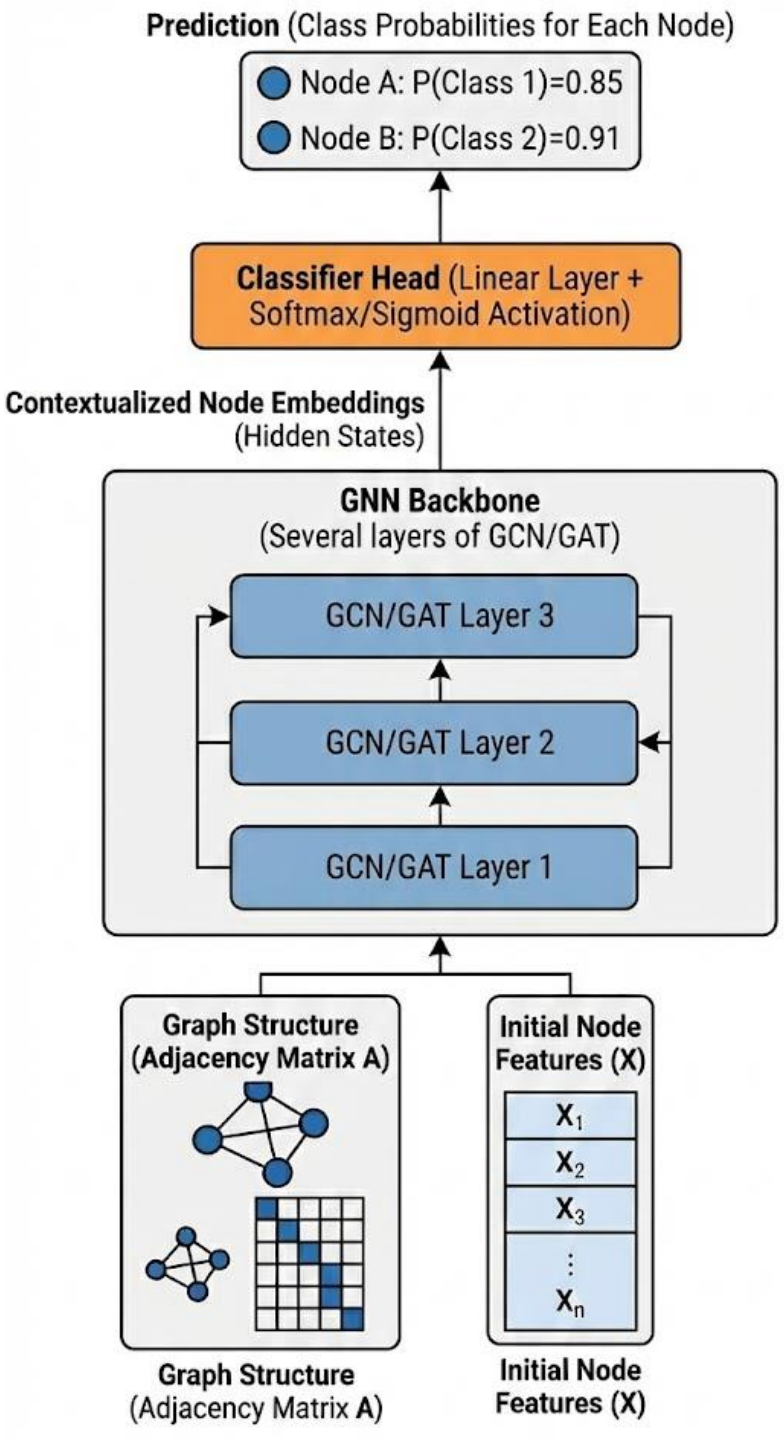
- **Applications:**
  - Social media user classification
  - Fraudulent account detection
  - Product categorization

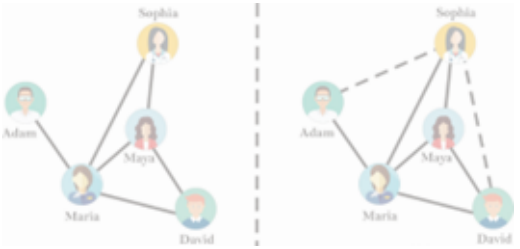


GNN – Node Classification

- **GNN architecture**:
  - **GNN backbone**: several layers of GCN/GAT or a single graph transformer.
  - **Classifier head**: a linear layer and a sigmoid/softmax activation function to produce class probabilities.

- **Learning**: Backpropagation using cross-entropy loss

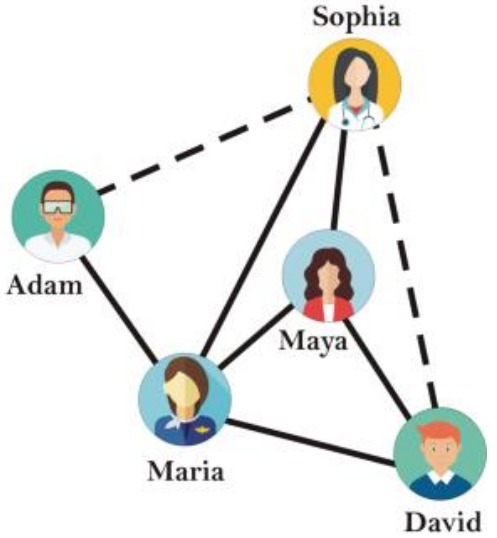




GNN – Link Prediction

- **Objective**: Predict the likelihood of a relationship (edge) existing between two nodes (identifies missing info or future connections).
  - **Input**: Graph $G = (V, E)$, node features X, and sets of *Positive (real)* and *Negative (fake)* edges.
  - **Output**: A probability score $S_{uv} \in [0, 1]$ for any pair of nodes (u, v).

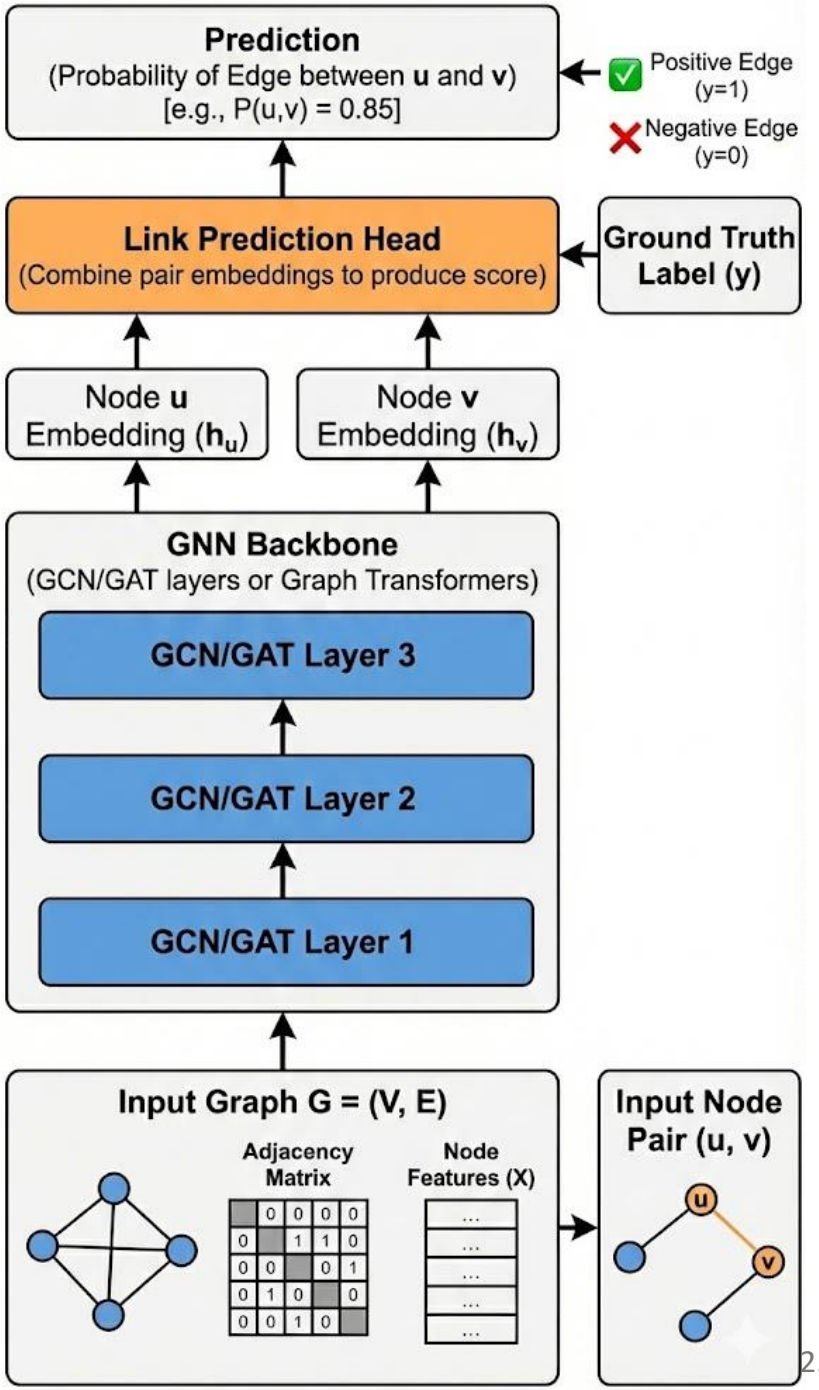
- **Applications**:
  - Product recommendation
  - Social media friend suggestion (“people you may know”)
  - Supply chain logistic route identification



GNN – Link Prediction

- **GNN architecture**
  - **GNN Backbone**: Standard GNN layers (GCN/GAT) or graph transformers to generate a latent embedding h for every node.
  - **Link head**: combine each pair of embeddings to produce a score for each node pair
    - Simple: dot product
    - Complex: linear layer + sigmoid

- **Learning** (equivalent to Edge Classification):
  - **Negative Sampling:** Randomly pick pairs of nodes that **do not** have an edge to teach the model what "no connection" looks like.
  - **Loss:** backpropagation using Binary Cross-Entropy (BCE) loss on each node pair.



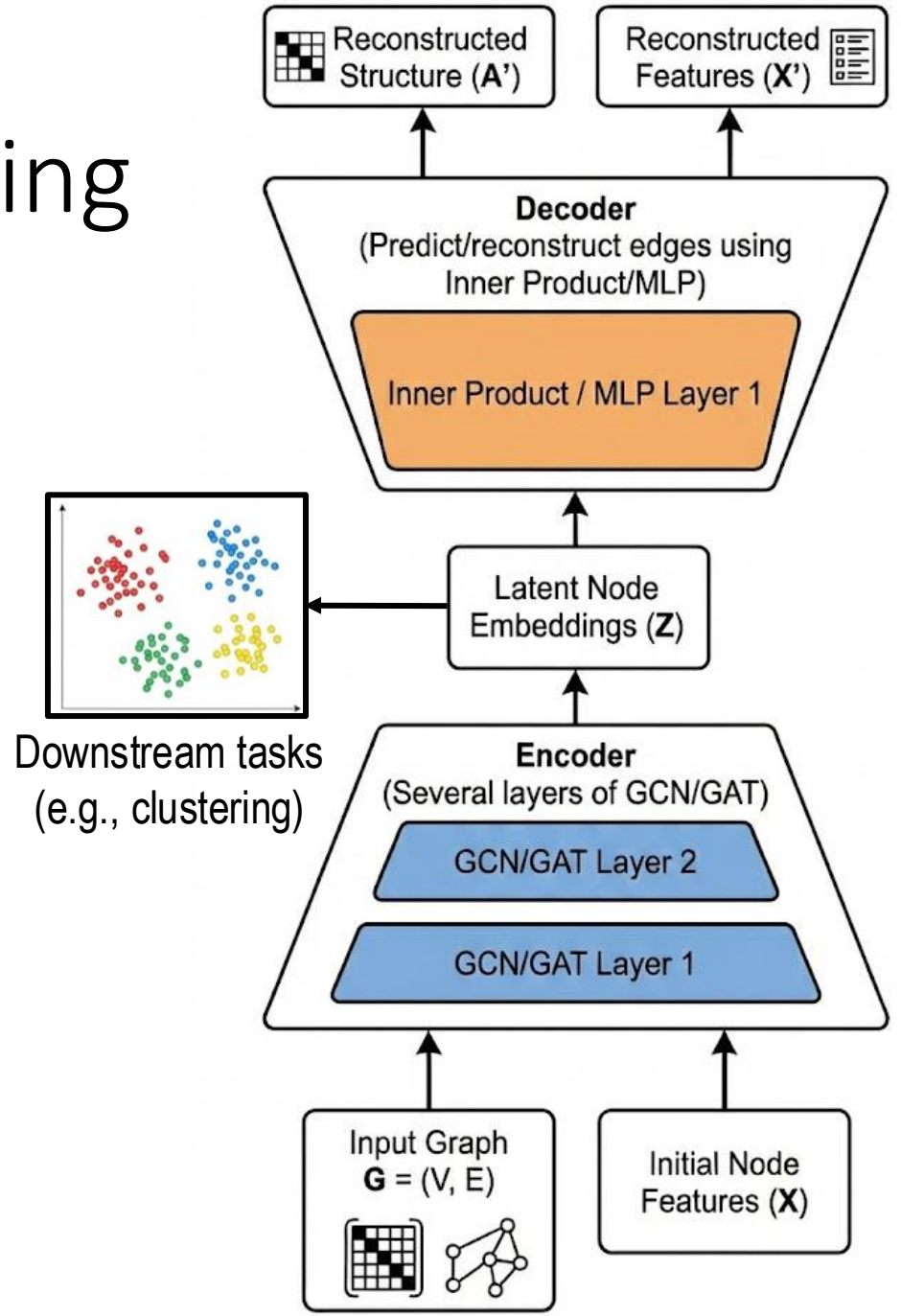
GNN – Representation Learning

- **Objective**: learn low-dimensional vectors (**embeddings**) that capture the "essence" of a node's features and its position in the network without any manual label.
  - **Input**: Graph $G = (V, E)$, node features X
  - **Output**: an embedding (vector) of each node.

- **Applications** (downstream tasks)
  - **Customer segmentation**: group similar customers for targeted marketing
  - **Anomaly detection**: identifying nodes that breaks the learned structural rules

GNN – Representation Learning

- **GNN architecture**: Graph autoencoder (GAE)
  - **Encoder**: Several layers (GCN/GAT) that compress nodes into a latent representations Z.
  - **Decoder**: predict/reconstruct the edges in the original graph based on the latent representation Z (inner product, MLP)
- **Optimization** (unsupervised): binary cross-entropy loss for each edge reconstruction.
- After optimization is complete, latent representation Z are taken out for downstream tasks.



GNN – Python Example

- **Python example**: production recommendation using graph neural networks
  - Refer to supplemental materials
- **Data**: past user-item interaction (movielens)
- **Key Packages**: PyTorch, Torch Geometric (PyG)

Module 2: Time Series Forecasting

Motivation

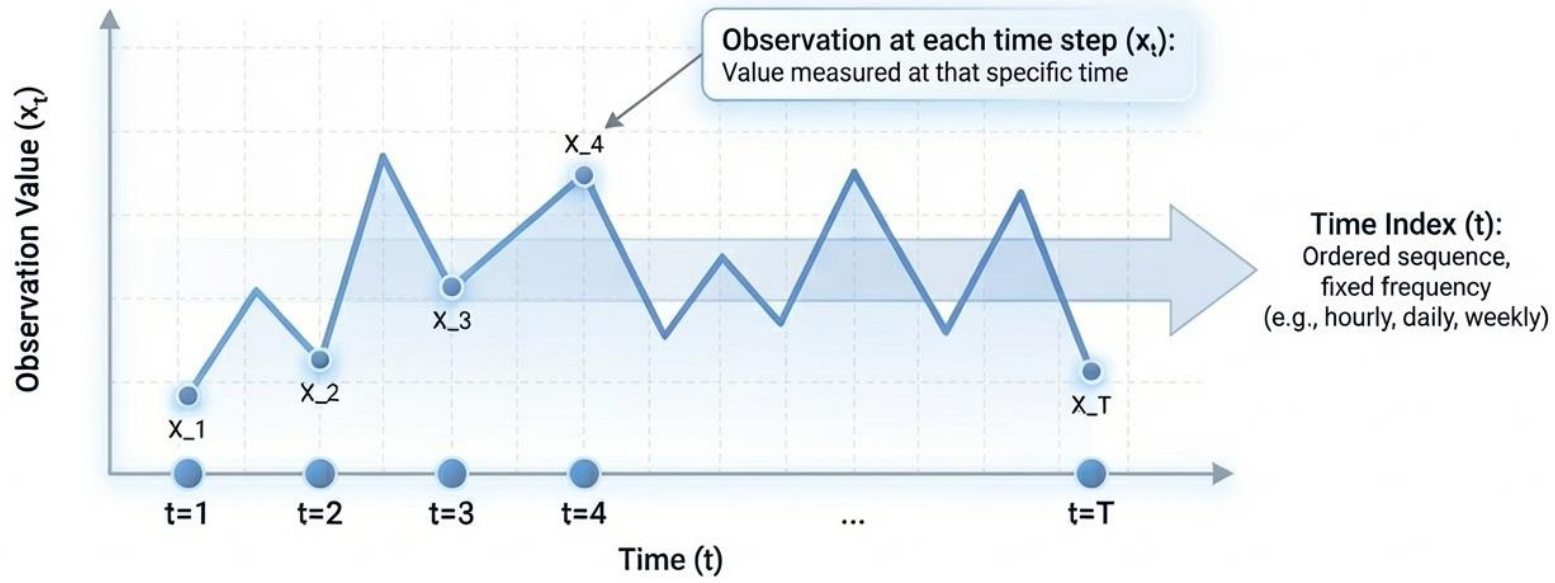
- Companies want to better anticipate future demand of their products.
  - Avoid overstock (money tied up in inventory) and stockouts (lost sales)

- Inventories can have distinct demand patterns:
  - **Smooth demand**: Consistent, high-volume items (e.g., milk or detergent)
  - **Intermittent**: Low-volume or high-price items with lumpy sales patterns
  - **Trend-driven**: Seasonal fashion or viral products with a narrow window of relevance

- We need ways to capture temporal patterns and dynamics → **Time Series**.

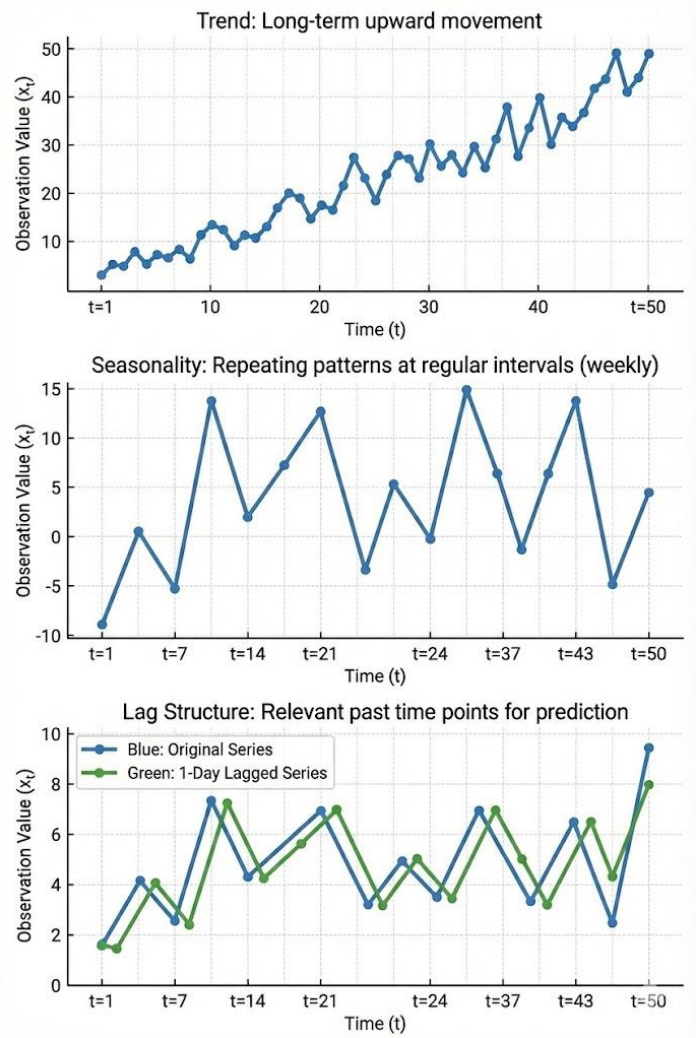
Time Series

- A **time series** is an ordered sequence of observations indexed by time $x_1, x_2, \ldots, x_T$, where:
  - **Time index** t is the time in fixed frequency e.g., hourly, daily, or weekly
  - **Observation** x_t is Value measured at time t



Time Series

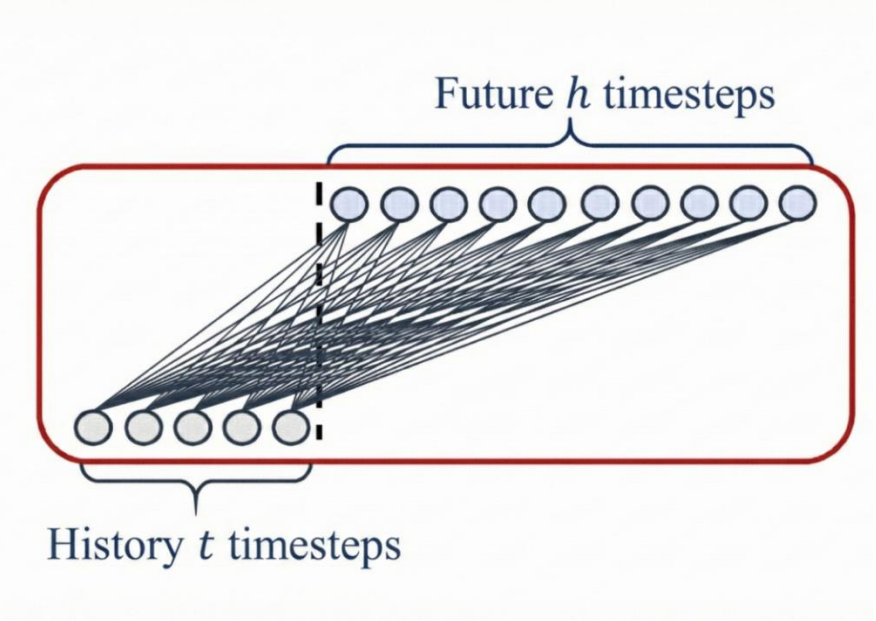
- Properties of time series:
  - **Temporal ordering**: The order of observations cannot be shuffled
  - **Temporal dependency**: Past values influence future values
  - **Trend**: Long-term upward or downward movement over time
  - **Seasonality**: Repeating patterns at regular intervals (e.g., weekly, yearly)
  - **Lag structure**: Which past time points are relevant for prediction



Time Series Forecasting – Problem Description

- Forecasting setup
  - **Input window**: Historical observations up to time t
  - **Forecast horizon** (h): Predict values at $t + 1, \ldots, t + h$

- Time series forecasting can be:
  - **Univariate**: One variable over time (e.g., product sales)
  - **Multivariate**: Multiple variables evolving together (e.g., price, promotion, weather)



(Adapted from Zeng et al. 2023)

Reference: Zeng, A., Chen, M., Zhang, L. and Xu, Q., 2023, June. Are transformers effective for time series forecasting?. In Proceedings ofthe AAAI conference on artificial intelligence (Vol. 37, No. 9, pp. 11121-11128).

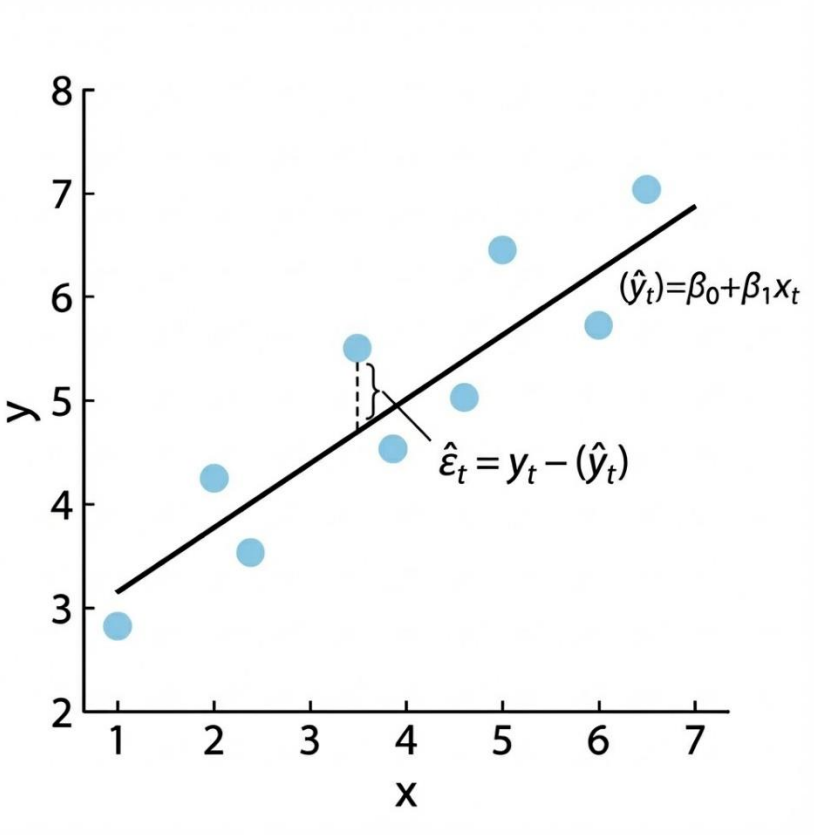
Regression

- **Linear regression** is a traditional time series forecasting method.
  - It assumes a straight-line relationship between predictor and target.

- **Model**: (x_t): $\widehat{y_t} = \beta_0 + \beta_1 x_t$
  - Predicts a future value (y_t) using one predictor
  - Choose coefficients β_0, β_1 so that predictions are as close as possible to historical data

- It learns through **Least Squares Estimation**, minimizing the sum of the squared errors:

$$\sum_{t=1}^{T} (y_t - \widehat{y_t})^2$$



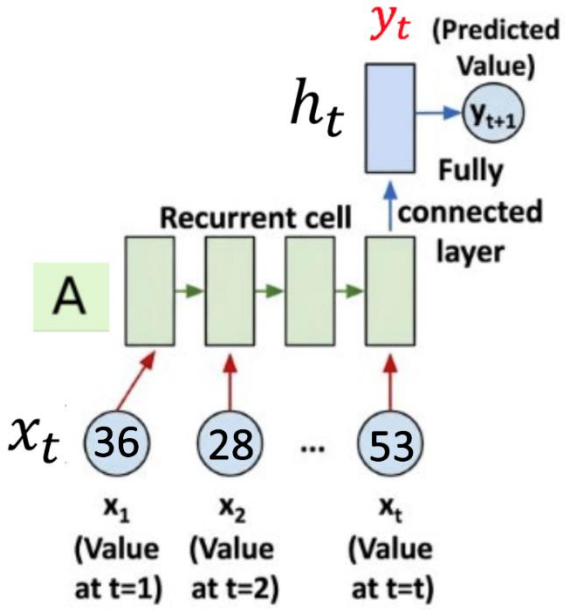
Regression

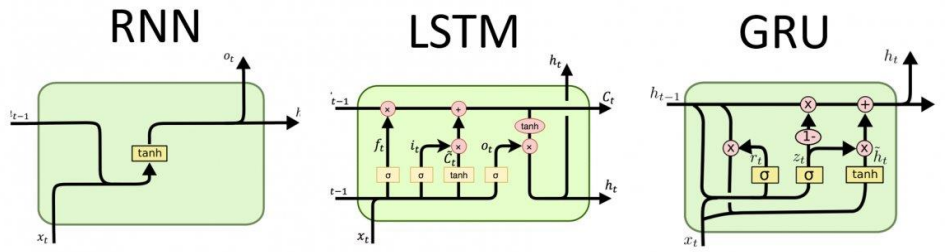
- Regression is suitable when predictors are numerical, and interpretability is important.
- When temporal relationships are complex, regression can be adapted to include:
  - Polynomial terms (e.g., x^2, x^3) for non-linear effect
  - Interaction terms (e.g., $x_1 x_2$) for cross-variable effect
- However, regression relies on manually engineered features.
  - Nonlinear and cross-variable relationships are hard to specify explicitly.
- RNN/LSTM/GRU are methods designed to learn complex temporal patterns automatically

RNN/LSTM/GRU

- Recall: **RNN/LSTM/GRU** are deep learning models for sequential data.
  - Shared recurrent cells are adopted across time steps
  - The hidden state allows past information to influence future predictions

- Applying RNNs to time series forecasting
  - **Input**: Each observation x_t is fed into the recurrent cell sequentially
  - **Output**
    - The final hidden state is passed to a fully connected layer
    - The model outputs the predicted value $\widehat{y_{t+1}}$





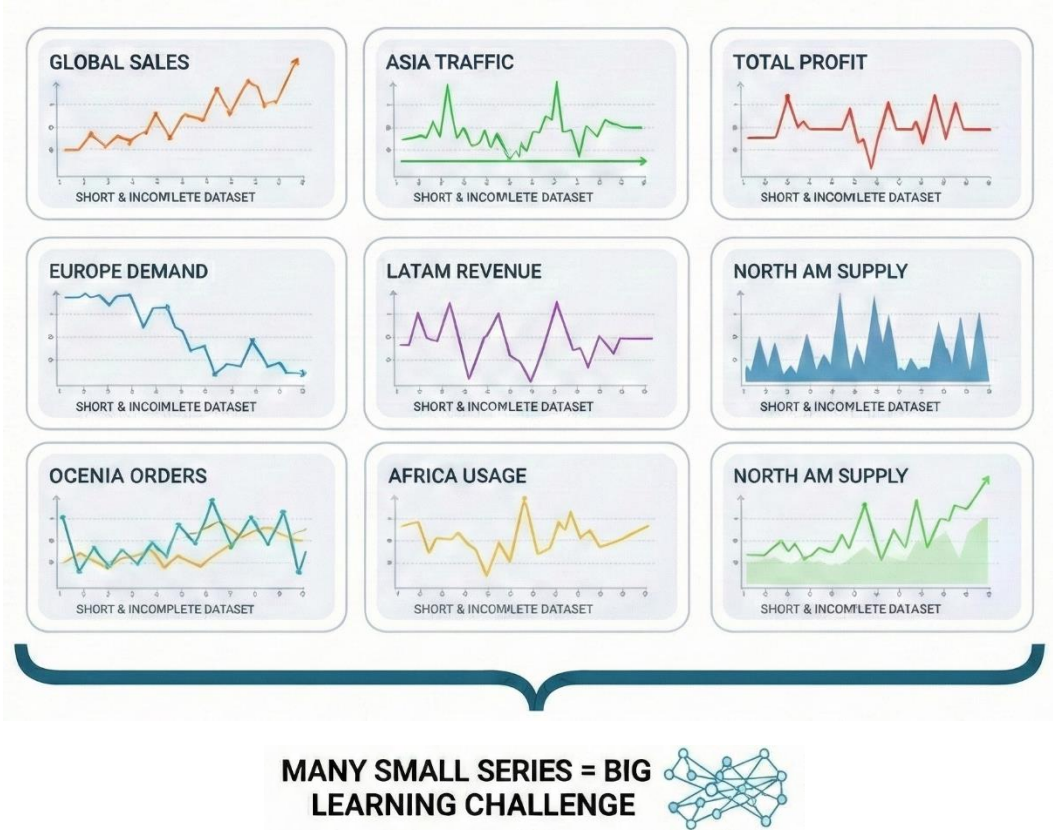
RNN/LSTM/GRU

- RNN/GRU/LSTM are:
  - Designed specifically for sequential data
  - Effective in modeling temporal dependence
  - Typically trained per task, per dataset

- What if a company has hundreds of products, each with only a few years of sales data?
  - Training a separate RNN for each case becomes inefficient and brittle

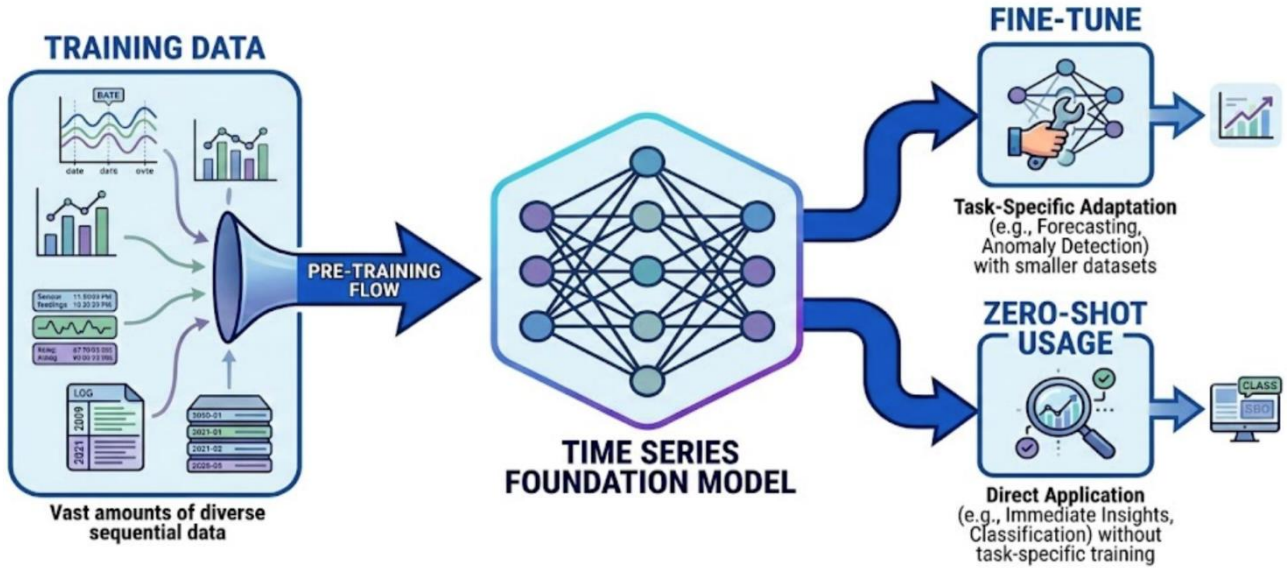
TSFM

- Regression and RNN-based methods are typically trained per task, per dataset.
- However, many time series share similar temporal patterns (e.g., seasonality, trend, dependencies), even if their scales and contexts differ.
- Instead of learning from scratch for each task and dataset, we can learn a common reusable temporal pattern for easy adaption ➔ **Time Series Foundation Model (TSFM)**.



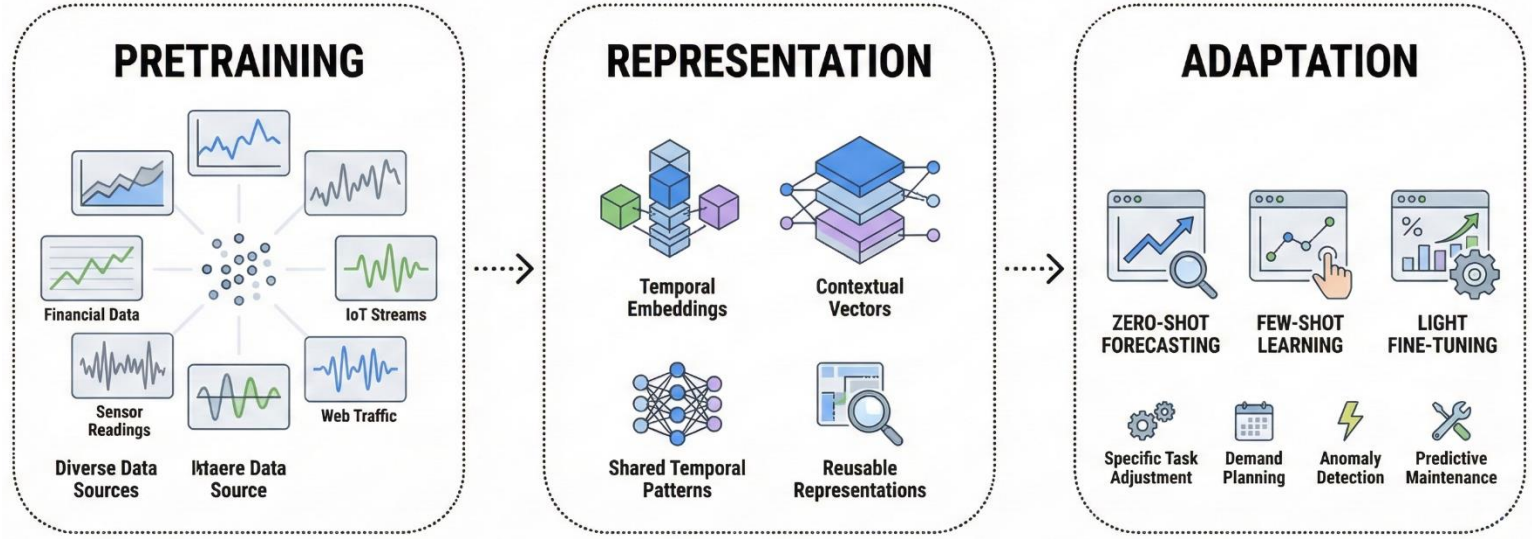
TSFM

- Inspired by pretrained language models, Time Series Foundation Model (TSFM) is a **large pre-trained model** for time series
  - Trained on many time series across domains (e.g., financial markets, weather, web traffic)
  - Provide transferable temporal knowledge, not task-specific curve fitting
  - Can be adapted (fine-tune) or reused (zero-shot, few-shot) for multiple downstream tasks



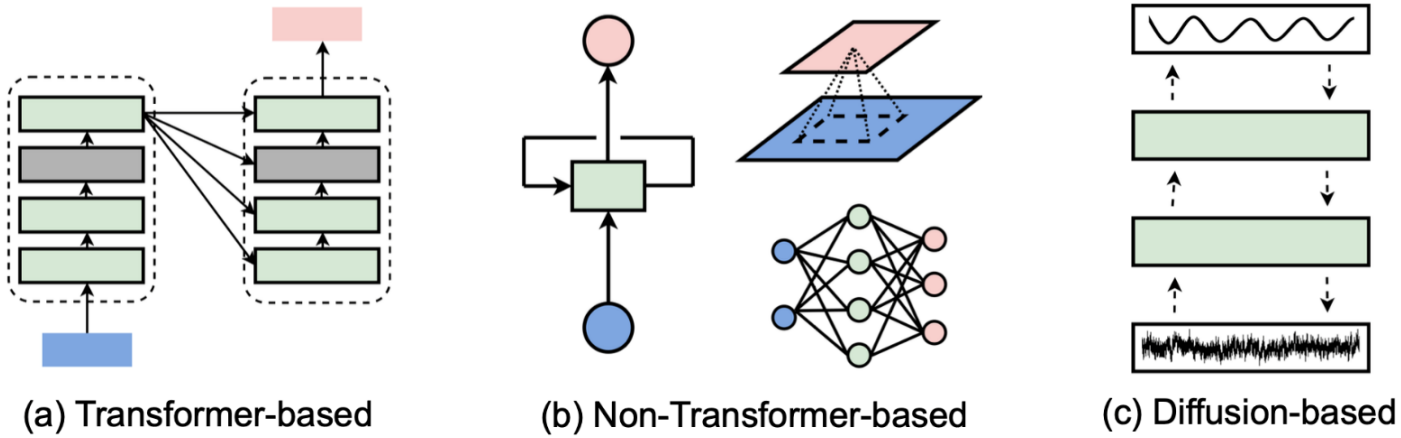
TSFM

- **Pretraining**
  - **Input**: massive, heterogeneous time series (different scales, domains, frequencies, lengths)
  - **Pretraining objective**: learn to reconstruct (unsupervised), predict (supervised), or generate temporal patterns (generative)
- **Representation**: Output the reusable temporal embeddings
- **Adaptation**: Downstream use includes zero-shot, few-shot, or fine-tuned forecasting.



TSFM

- There are three types of TSFM based on the architecture (Liang et al. 2024):
  - **Transformer-based** (dominant): Learn global temporal dependencies via attention and parallel sequence modeling
  - **Non-Transformer-based:** Use alternative architectures (MLP, CNN, or RNN) to capture temporal patterns with better efficiency or simplicity
  - **Diffusion-based**: Generative models that produce probabilistic forecasts via iterative sampling

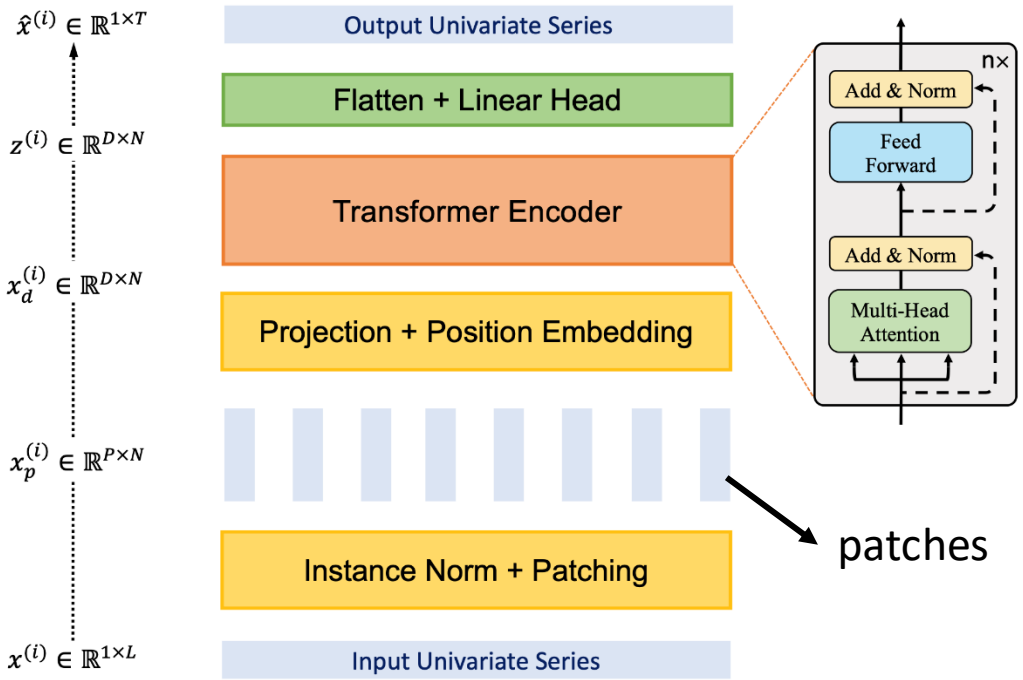


Reference: Liang, Y., Wen, H., Nie, Y., Jiang, Y., Jin, M., Song, D., ... & Wen, Q. (2024, August). Foundation models for time series analysis: A tutorial and survey. In Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining (pp. 6555-6565).

TSFM - PatchTST

- **Channel-independence Patch Time Series Transformer (PatchTST)** learns temporal representations via a transformer encoder (Nie et al. 2023).

- **Key mechanism**: Aggregating time steps into patches instead of individual time points
  - Reduces attention cost
  - Allows the model to attend to longer histories

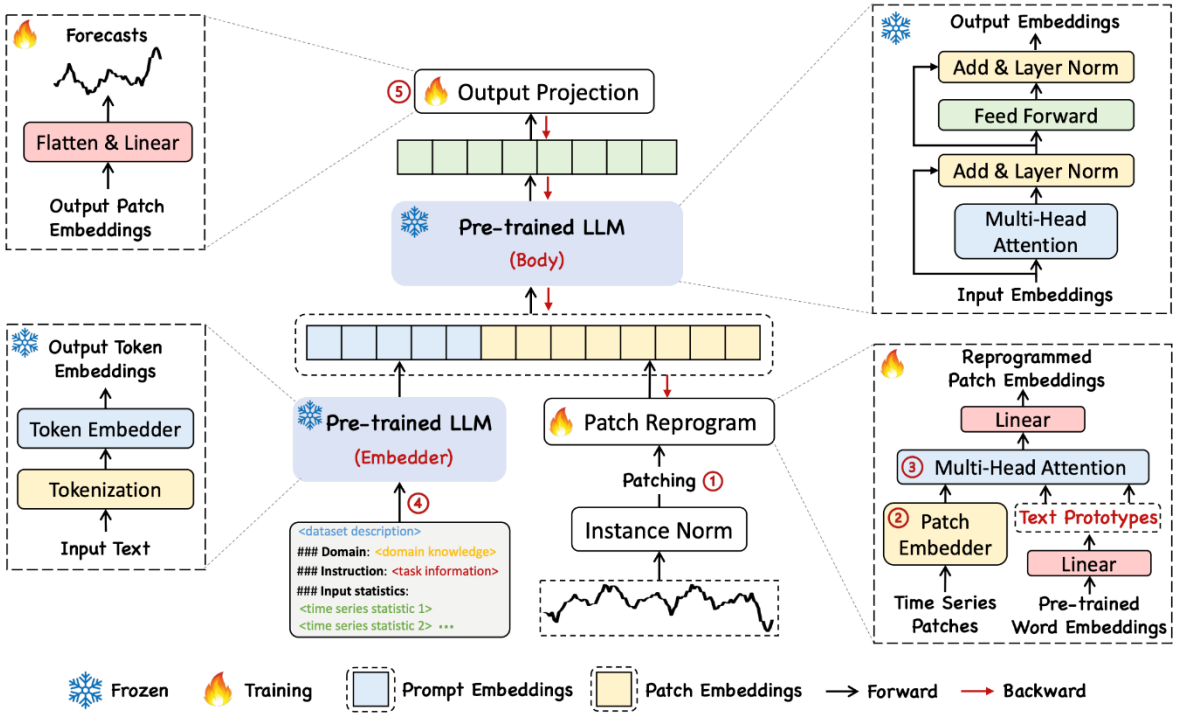


Transformer Backbone (Supervised)

Reference: Nie, Y., Nguyen, N. H., Sinthong, P., & Kalagnanam, J. (2023). A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. The Eleventh International Conference on Learning Representations.

TSFM – Time-LLM

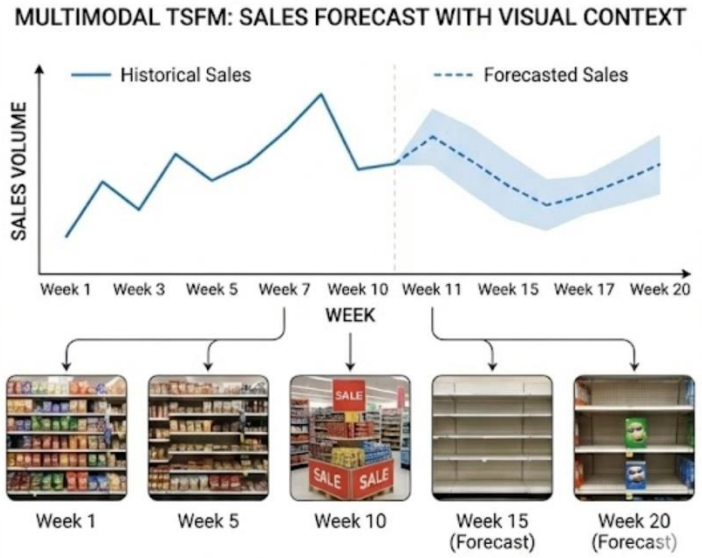
- **Time-LLM** leverages a frozen pre-trained LLM as the temporal reasoning backbone (Jin et al. 2024)
  - Pretrained LLM can encode rich sequence-level structure and long-range dependency from massive data.
- **Key mechanism**: Patch and dataset-aware prompts
  - Dataset descriptions (domain, task, statistics) are injected as prompts to guide how the LLM interprets temporal patterns.

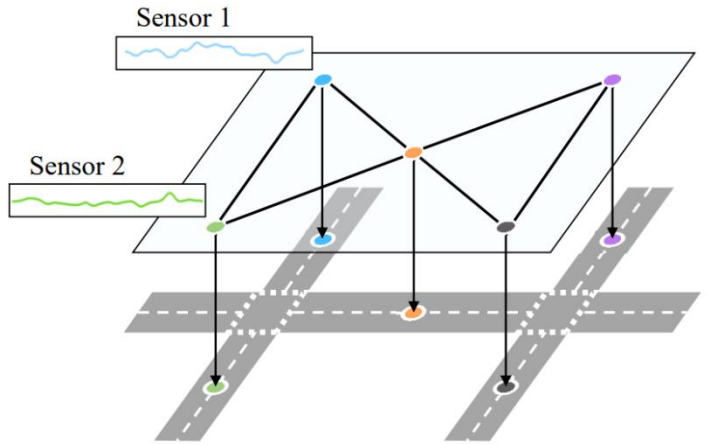


Reference: Jin, M., Wang, S., Ma, L., Chu, Z., Zhang, J. Y., Shi, X., Chen, PY., Liang, Y., Pan, S. Wen, Q. (2024). Time-LLM: Time Series Forecasting by Reprogramming Large Language Models. The Twelfth International Conference on Learning Representations.

TSFM – Variants

- Real-world forecasting problems often involve multiple related signals, not a single isolated time series.
  - **Multimodal TSFM**
    - Combine numerical time series with visual or textual signals
    - E.g., forecasting retail sales using historical demand plus in-store images (shelf availability)
  - **Spatio-Temporal TSFM**
    - Jointly model temporal dynamics and spatial dependencies
    - E.g., Traffic flow forecasting across connected road segments



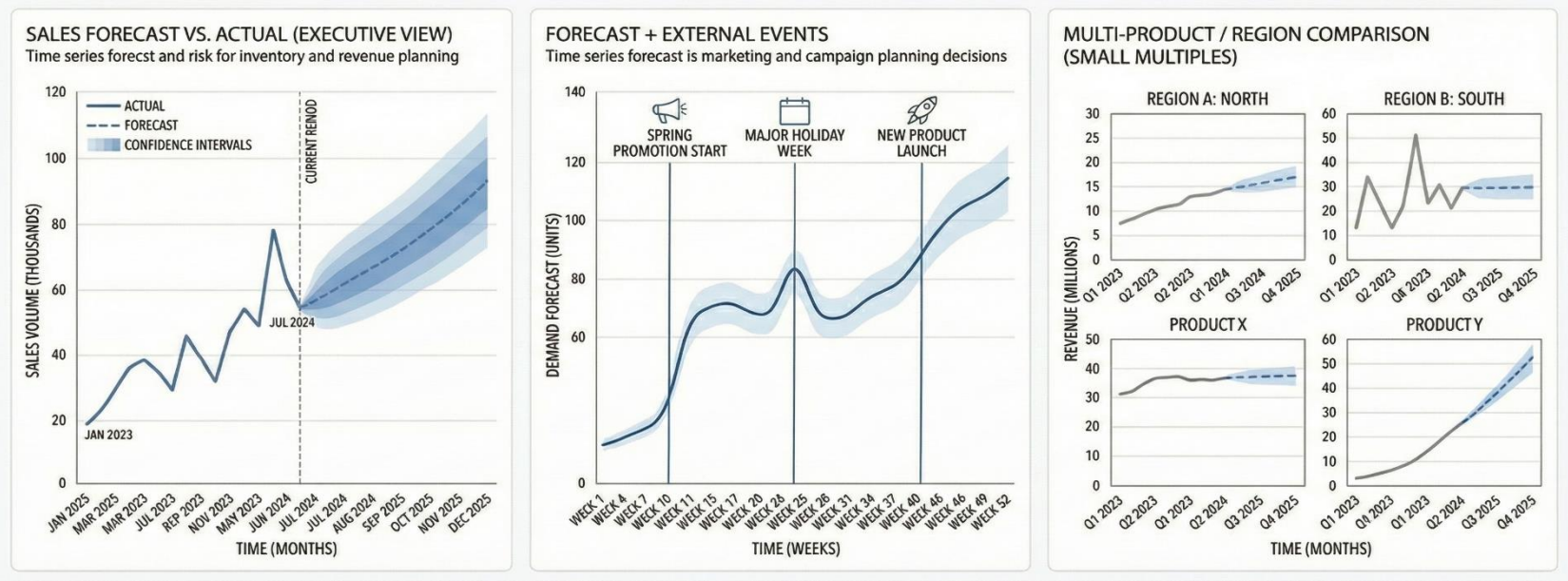


(From Liu et al. 2024)

Reference: Liu, C., Yang, S., Xu, Q., Li, Z., Long, C., Li, Z., & Zhao, R. (2024, June). Spatial-temporal large language model for traffic prediction. In 2024 25th IEEE International Conference on Mobile Data Management (MDM) (pp. 31-40). IEEE.

Visualization

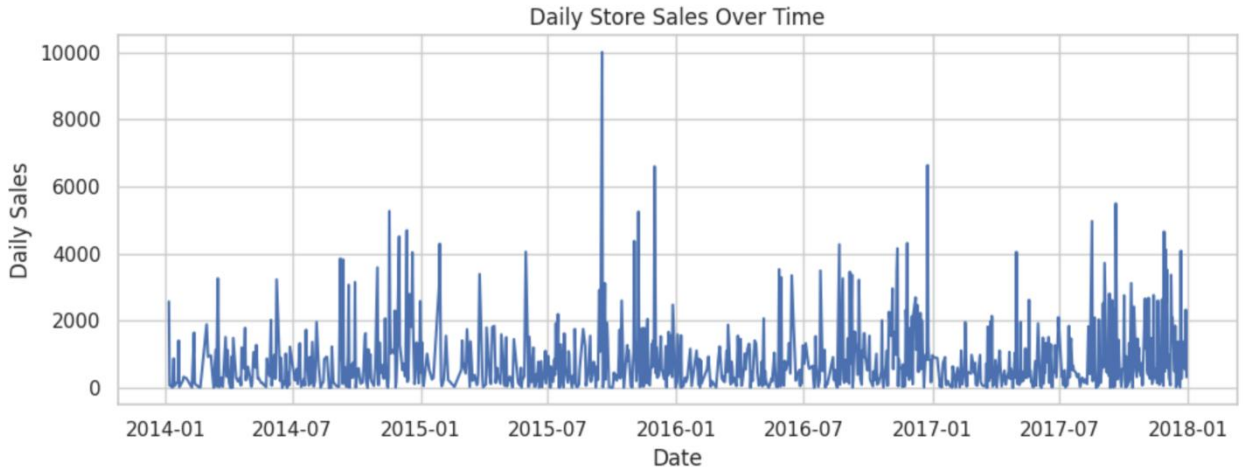
- Time series forecasting results are typically visualized as trajectories over time, examples include:
  - **Fan charts** to show forecast uncertainty using prediction intervals;
  - **Event-annotated forecasts** to align predictions with external factors such as promotions or holidays;
  - **Comparative views** to compare forecasts across products or stores.



nline M.S. in AI

Time Series Forecasting – Python Example

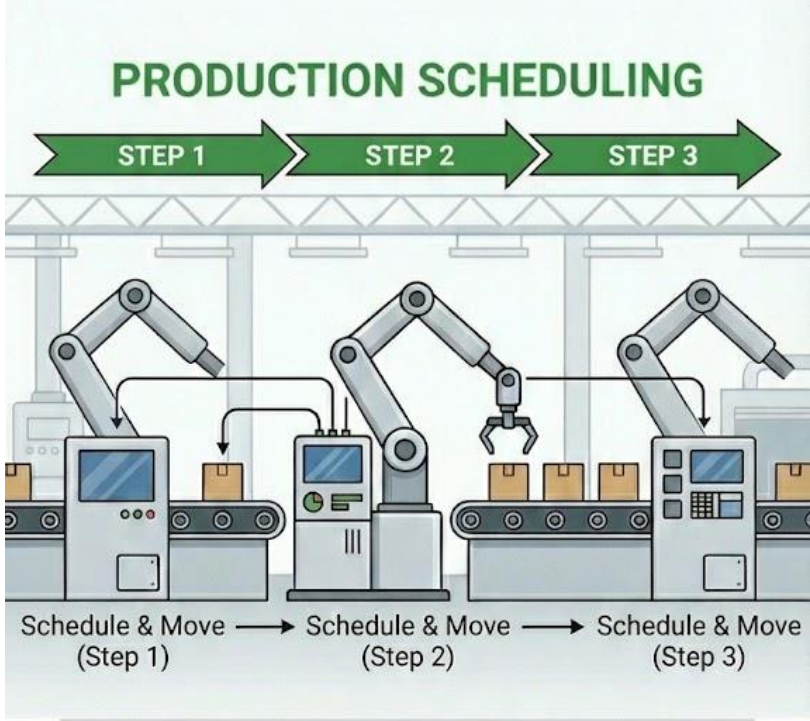
- **Python Example**: Store Sales Forecasting
- **Data**: Daily sales observations for a single store/product
- **Model**:
  - **Classical models**: Moving Average, ARIMA, Linear Regression
  - **Deep learning**: LSTM, GRU, and a TSFM (TimesFM)
- **Key packages**: statsmodels, scikit-learn

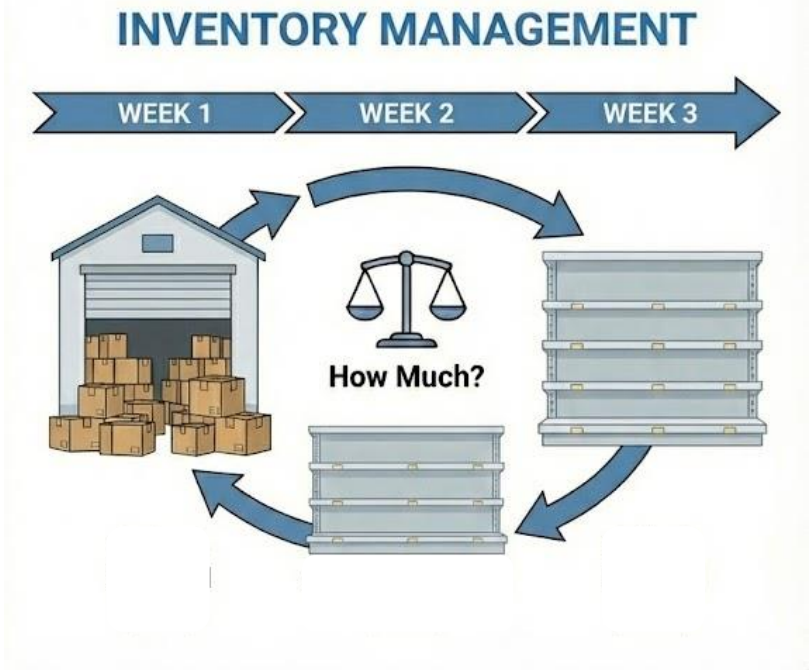


Module 3: Reinforcement Learning

Optimization Problem

- Some business problems aim to find what we should do instead of what happened/will happen.
- **Production scheduling**: at each processing step, when should we move the robot to pickup product, and where to move?
- **Inventory management**: how much inventory should we order each week at different customer demand
  - Ordering too much leads to high warehouse cost, while too little disappoints customers





Optimization Problem

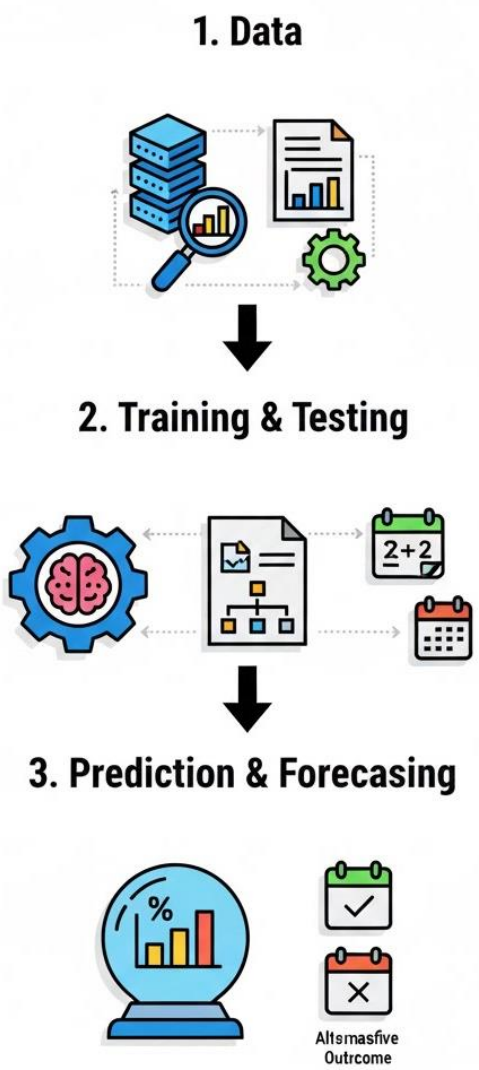
- Characteristics:
  - We try to optimize a certain **objective** (minimize cost, maximize production throughput) through **sequential decisions** (e.g., amount to order each week) under different **circumstances** (e.g., market situation).
  - The outcome of such a decision may not be clear
    - We don’t know the exact customer demand; our ordering amount may affect the situation next week (e.g., some remaining inventories get spoiled)
    - We don’t know whether picking up a product helps the production or not; the movement affects the machine’s status.

Optimization Problem

- Why not supervised learning?

- Supervised learning requires:
  - **Every decision is independent**: Classifying x_1 into class A does not affect the classification x_2
  - **A clear outcome**: classifying of x_1 into class A is either correct or incorrect, which we can know before classification based on the training data.

- **But**:
  - Inventory ordering amount at $t+1$ may be affected by t
  - It is unclear whether ordering n products is good or bad before really ordering them and observing the result, as there may be some unobservable seasonal effects.



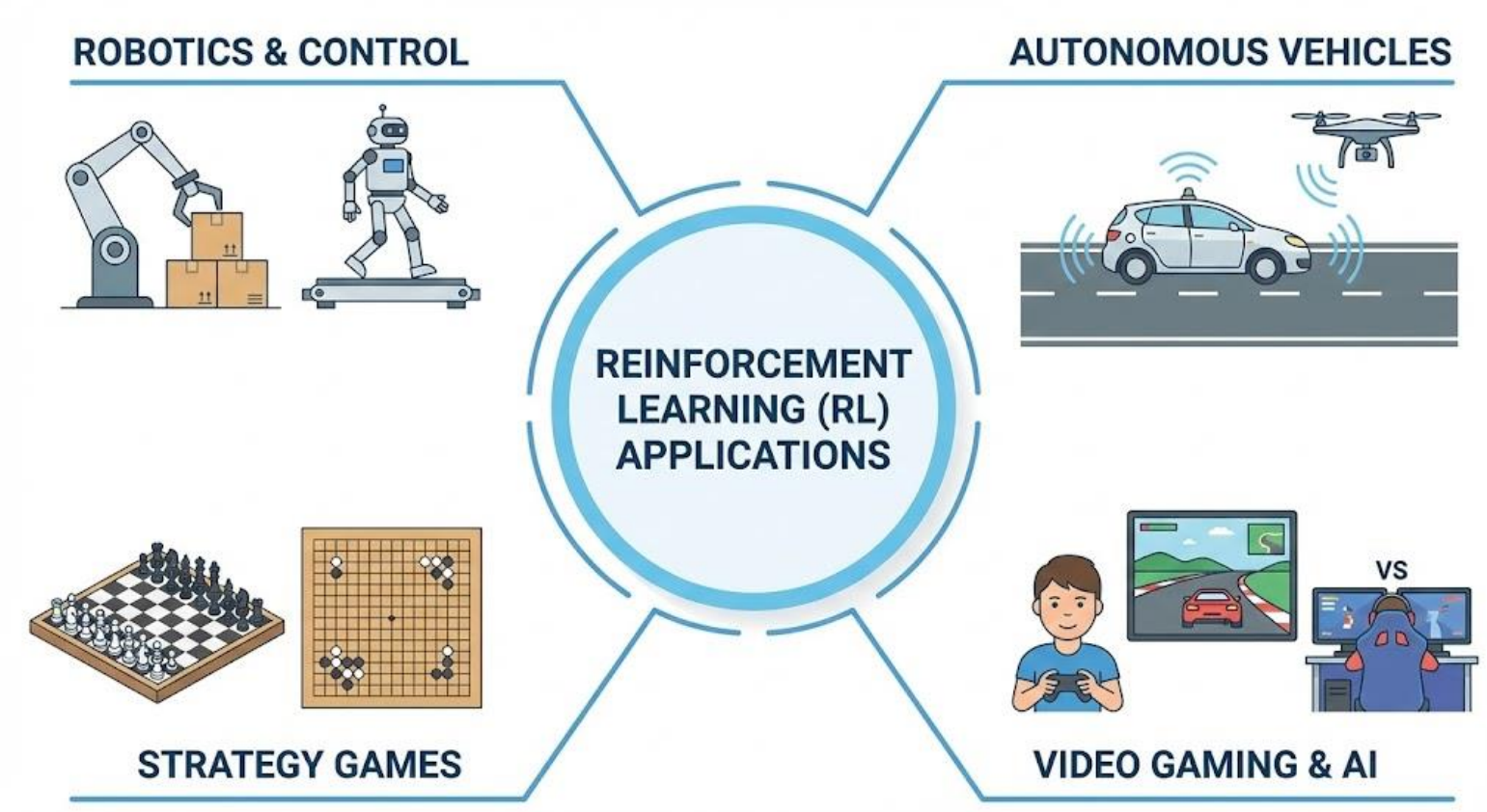
Optimization Problem

- This type of learning typically repeats the following:
  - Make a decision at certain circumstances
  - The circumstances change
  - Observe benefit or losses after making this decision

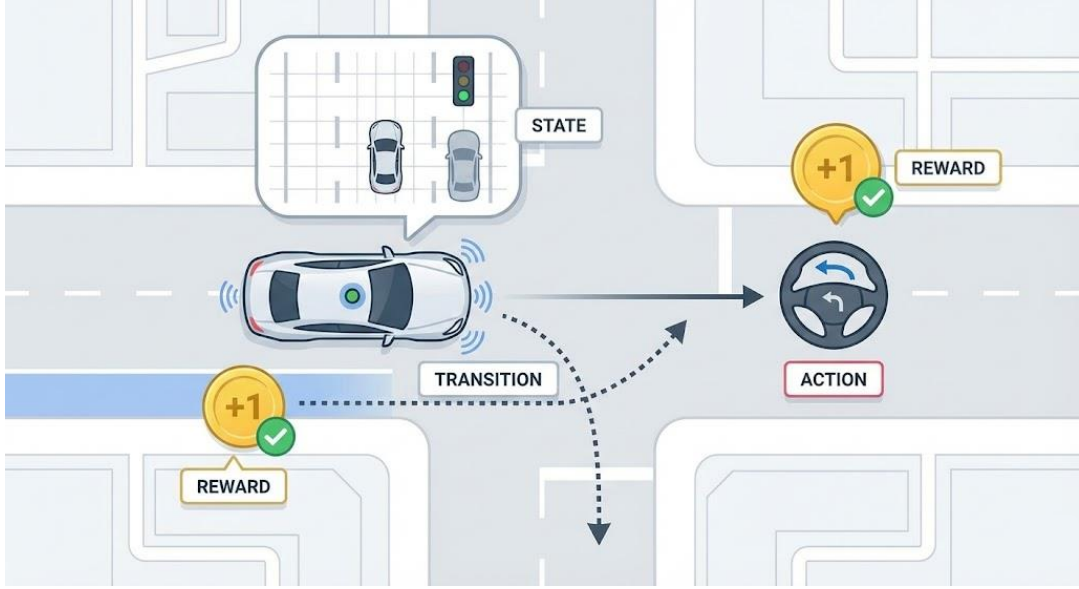
- Similar to how human learn from hands-on practices (e.g., learn to drive):
  - The driver presses the gas, hits the brakes, or steers left/right based on the speed of the car and the traffic
  - The speed or direction of the car changes
  - The car gets closer to the destination, or crashes into other cars

Reinforcement Learning

- **Reinforcement learning** is a learning paradigm that is inspired by this process.



Reinforcement Learning

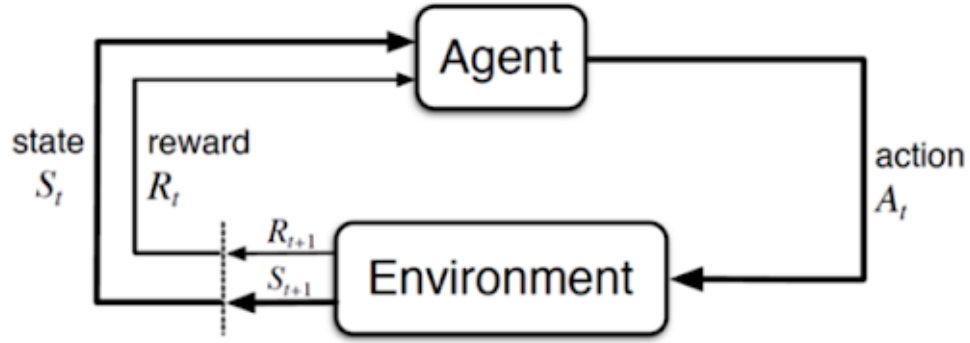


- **Markov Decision Process (MDP)**:
  - **State**: the current circumstances
  - **Action**: the decision to make
  - **Transition**: how the circumstances change
  - **Reward**: feedback indicating the outcome of the decision

- The decision maker is called the **agent** (e.g., driver)

- **Environment**: Processes the agent's action, changes to a new state, and calculates a reward (e.g., a road simulator).

Reinforcement Learning



- We learn through continuous interaction with the environment to know **what we should do (action) under which circumstances (state).**

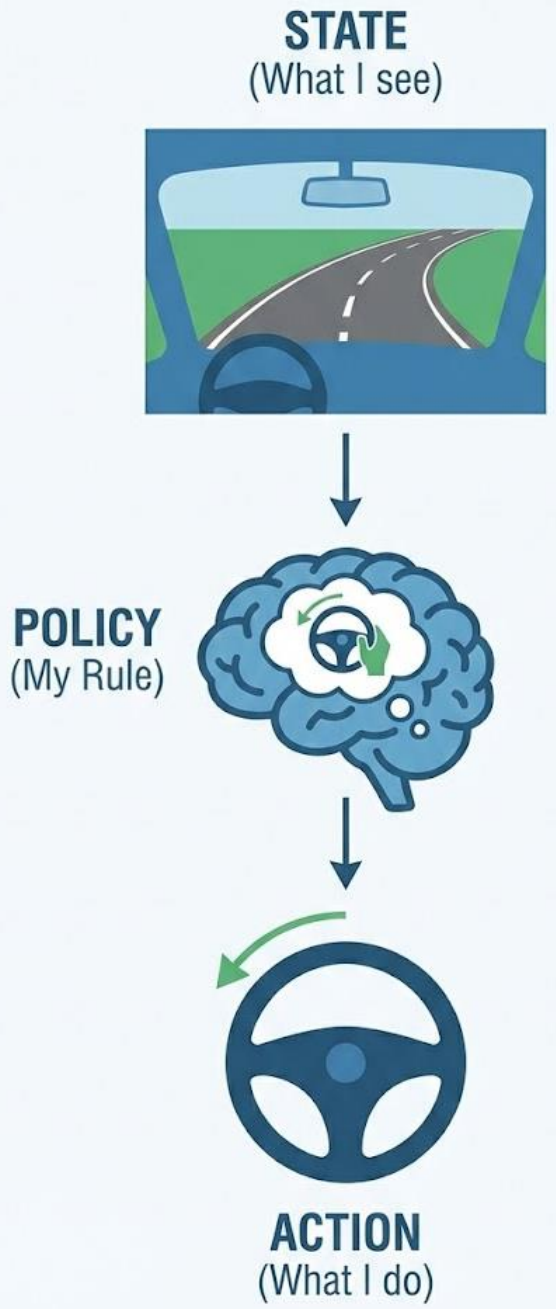
Repeat these three steps

Reinforcement Learning Steps	**Description**	**Example 1 (Scheduling)**	**Example 2 (Autonomous driving)**
1. Take an **action** under a **state**	Performing a decision at certain circumstances	Determine whether to pick up a product from a machine at a certain processing step	Determine whether to turn at an intersection given the vehicle condition and the traffic
2. State **transition**	The circumstances changes	A product is moved to another machine	The vehicle speed or direction changes
3. Receive a **reward**	Observe whether you gain some benefit by performing this decision	A product becomes closer to being finished	The vehicle is not crashing and closer to the destination

Policy-based Methods

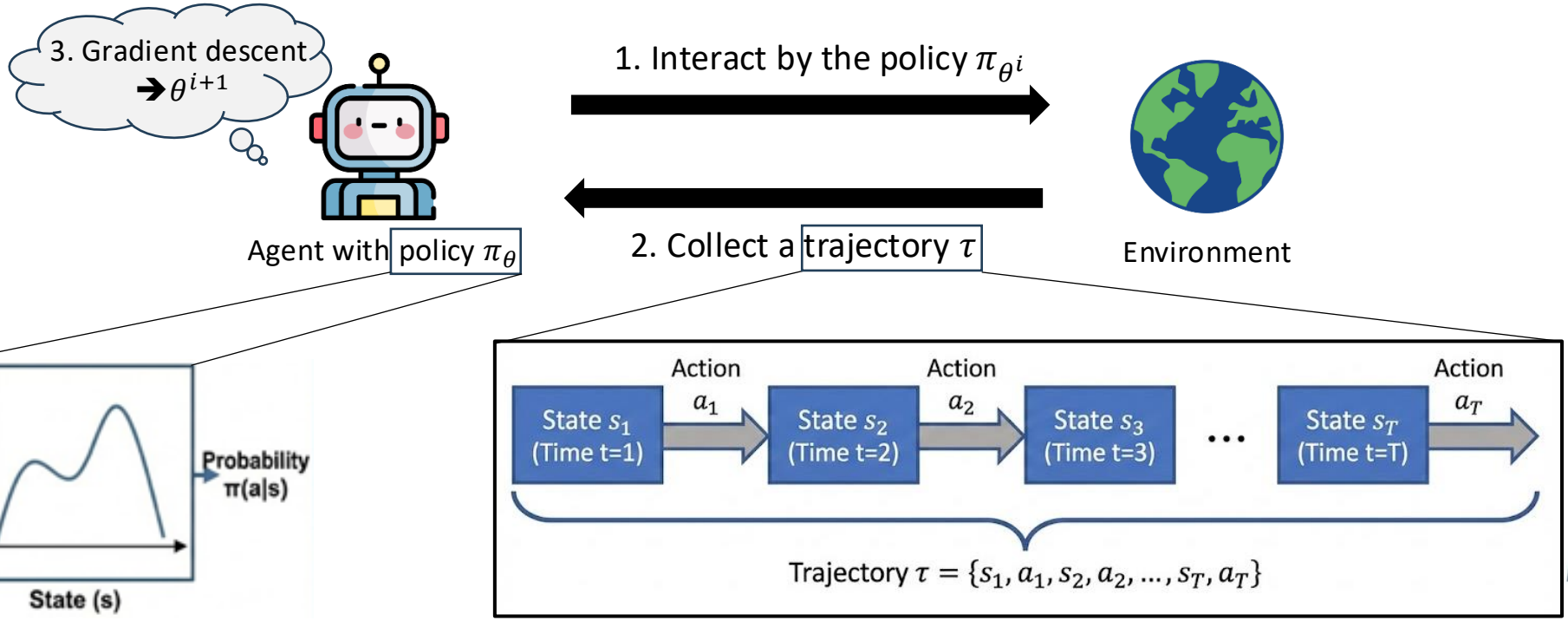
- **Key idea**: Focus on our own **behavior** under different circumstances (a "policy").
  - E.g., when the car drifts right, I steer left
- Through practice, we refine the *probability* of how much you turn the wheel (*action* a) when the car drift (*state* s).
  - Learn the **best move** for that visual state.
- **Technically**: learn a policy π, the probability distribution over actions given a state

$$\pi(a|s) = P(A_t = a|S_t = s)$$



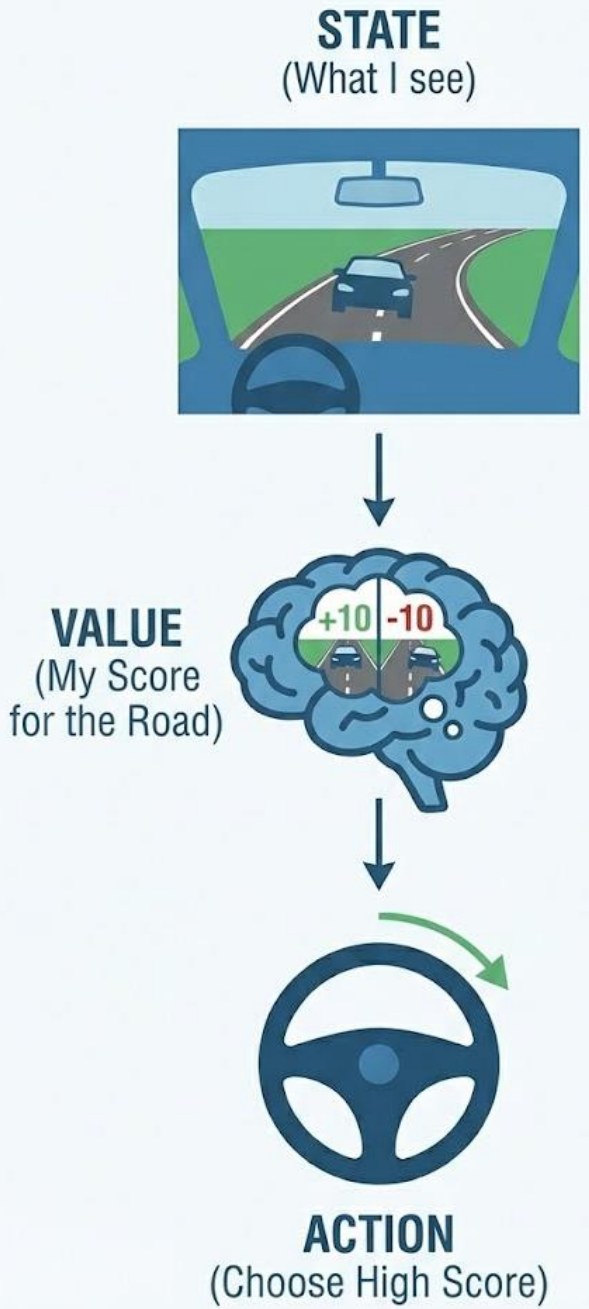
Policy-based Methods

- **Learning process**: for each iteration:
  1. Use policy π_θ to interact with the environment
  2. Collect a trajectory $\tau = \{s_1, a_1, \dots, s_T, a_T\}$
  3. Gradient descent $\theta^{i+1} = \theta^i - \alpha \nabla J(\theta)$.



Value-based Methods

- **Key idea**: Focus on the **potential** of a specific move in the current situation (the “value”).
  - When a truck is in front of us, “merging left" is worth more than "staying behind a truck" because it unlocks better future options.
- We learn to recognize high-value states and actions (merging left) vs. low-value ones (stuck behind a slow truck).
  - Act by simply choosing the move that yields the highest predicted long-term value.
- **Technically**: learn a Q value $Q(s, a)$ to represent the value of each action a under each state s.



Value-based Methods

- **Learning process**: for each iteration:
  1. Choose action a under state s by $a = \arg\max Q(s, a)$
  2. Take action a, get reward r and next state s
  3. Update Q value:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

		State Value $V(s)$	ACTIONS (a)			
			Stay (Continue)	Merge Left	Merge Right	Brake (Slow Down)
STATES (s)	Center Lane (Clear Road)	80	80	75	70	60
	Exit Lane (Slow Traffic)	40	40	55	30	35
	Stuck Behind Truck (Right Lane)	20	20	85	15	25

Value-based Methods

- But how do we know those Q values at first?
  - We have no idea whether staying is the best action when we are stuck behind a truck.
- We need to **explore** first before **exploiting** these values!

		State Value $V(s)$	ACTIONS (a)			
			Stay (Continue)	Merge Left	Merge Right	Brake (Slow Down)
STATES (s)	Center Lane (Clear Road)	???	???	???	???	???
	Exit Lane (Slow Traffic)	???	???	???	???	???
	Stuck Behind Truck (Right Lane)	20	20	???	15	???

Is this the best?

Value-based Methods

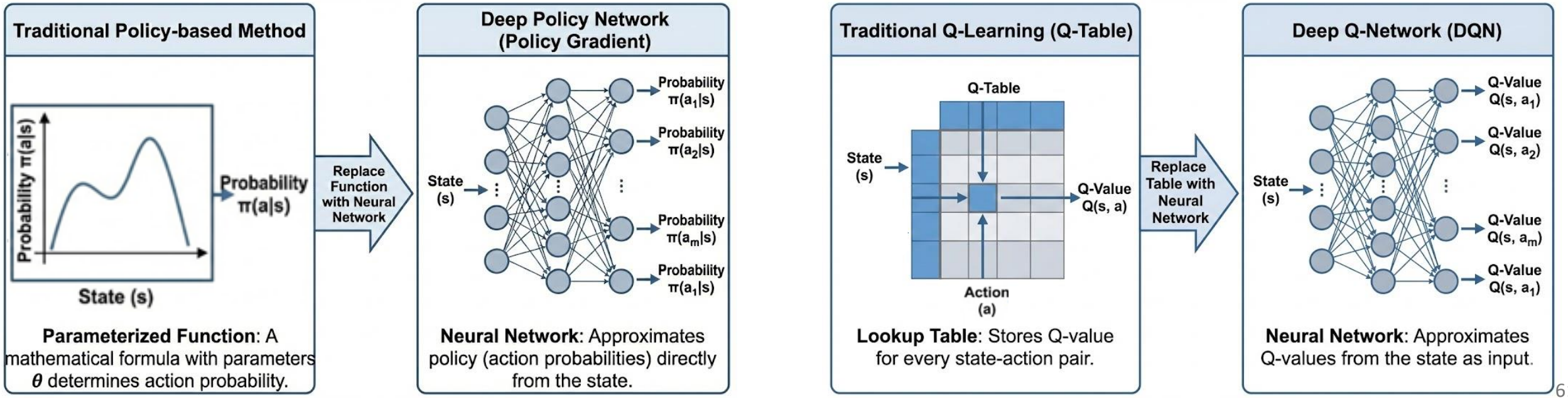
- **Learning process**: for each iteration:
  1. Choose action a using ϵ −greedy policy from $Q(s, a)$
  2. Take action a, get reward r and next state s
  3. Update Q value:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

- ϵ −greedy policy
  - **Exploration**: With probability ϵ , the agent chooses an action at random.
  - **Exploitation**: With probability $1 - \epsilon$, the agent chooses the action with the highest current estimated value in the table ($a = \arg\max Q(s, a)$).

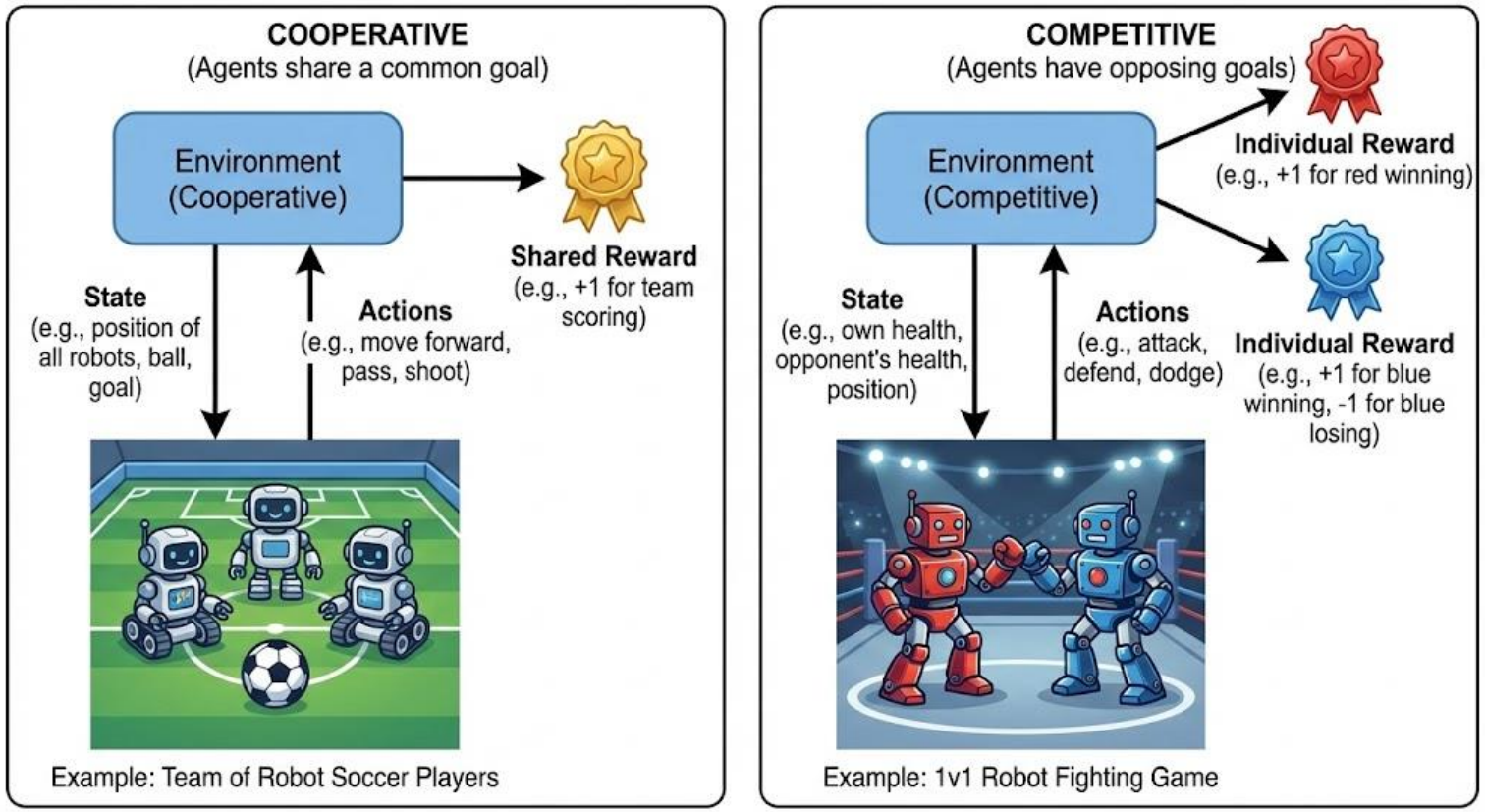
Deep Reinforcement Learning

- These methods are limited to a small environment as they require a dedicated memory slot for every possible scenario.
- Deep neural networks have been largely adopted to help handle high-dimensional spaces in the real world.
  - E.g., To capture road condition for autonomous driving, a single frame of video (2048×1080 pixels) contains over 2 million dimensions as the state.



Multi-agent Reinforcement Learning

- Sometimes there are multiple decision makers (agents).
- RL can be extended to multi-agent RL where multiple agents simultaneously take an action to maximize a shared reward (cooperative) or their own reward (competitive).



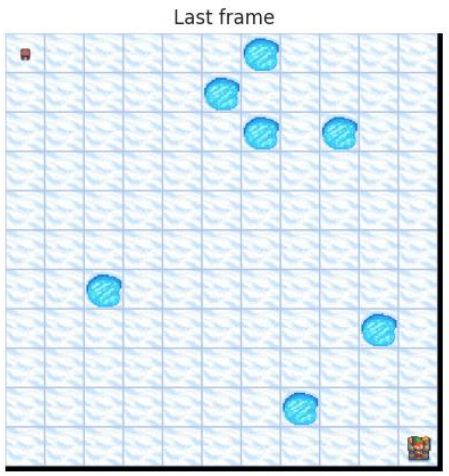
Reinforcement Learning – Python Example

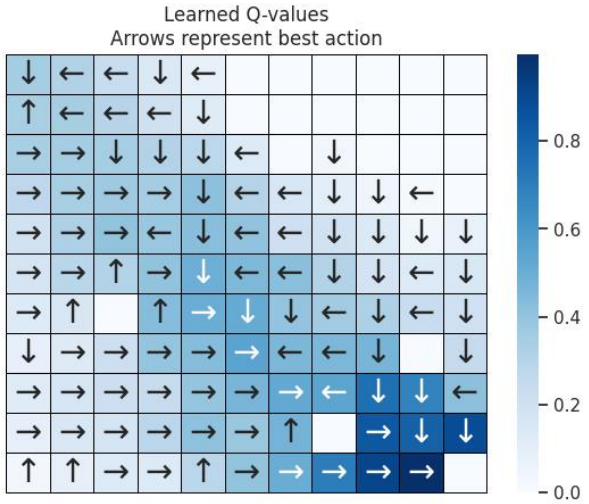
- Reinforcement learning often do not require a “dataset”.
- It requires an environment to tell the agent the reward and state transition after performing an action.
- In Python, this is powered by the Gynnasium library.
  - It provides many existing environment (e.g., CartPole, FrozenLake, Atari)
  - User can also define custom environment for their applications (e.g., inventory management, production scheduling)

Reinforcement Learning – Python Example

Gymnasium

- **Environment**: Frozen Lake
- **Model**: REINFORCE (vanilla policy-based method), Q-Learning (vanilla value-based method)
- **Key packages:** Gymnasium, PyTorch





From https://www.gymlibrary.dev/environments/toy_text/frozen_lake/

Review Questions

- **Module 1: Graph Neural Network**
  - How does message passing GNN capture the connection in a graph?
  - How do graph transformers handle long-range dependencies?
- **Module 2: Time Series Forecasting**
  - What are the three types of TSFM based on the architecture?
  - What's the benefit of adopting TSFM for time series forecasting compared to RNN/LSTM/GRU?
- **Module 3: Reinforcement Learning**
  - Why can't the production scheduling problem be solved by supervised learning?
  - Why do we need ϵ −greedy in value-based method?