L20: TASK ALLOCATION & MARKETS

Market-Based Approaches (Auctions) Robots act as self-interested agents maximizing individual utility (profit).

- **Utility Function:** $U_i(T) = R(T) - C_i(T)$, where R is reward and C is cost.
- **Objective Functions (Team Metrics):**
  - **MiniSum (Efficiency):** Min total cost, $\sum_i \sum_{j \in A_i} c_{ij}$. (e.g., total fuel).
  - **MiniMax (Makespan):** Min max cost, $\min(\max_i \sum_{j \in A_i} c_{ij})$. (e.g., mission time).
  - **MiniAvg (flowtime):** Min avg cost, $\min(avg_i \sum_{j \in A_i} c_{ij})$. (e.g., mission time).

Auction Types

1. **Parallel Auctions:** All tasks auctioned simultaneously. No dependencies. Bid is an approximate Marginal Cost.
   - *Pros:* Fastest. *Cons:* Robots ignore synergies (interference/coupling). High risk of sub-optimal allocation for complex tasks.
2. **Combinatorial Auctions:** Bids submitted on *bundles* of tasks ($S \subseteq T$).
   - Robot i bids $b_i(S)$ for subset S.
   - **Winner Determination Problem (WDP):** The auctioneer finds non-overlapping allocation maximizing revenue.

$$\max \sum_i b_i(S_i) \quad \text{s.t.} \quad S_i \cap S_j = \emptyset$$

   - *Complexity:* Equivalent to Weighted Set Packing (NP-Hard).
3. **Sequential Auctions (Single-Item):** Tasks auctioned one by one. Each robot bids accounting for dependencies with its tasks so far. This is the most important type.
   - **Greedy Bidding:** Robot i bids its exact *Marginal Cost* to add task t to its current schedule J_i:

$$MC(t, J_i) = Cost(J_i \cup \{t\}) - Cost(J_i)$$

   - *Pros:* Polynomial time ($O(N \cdot T)$). Good for dynamic environments.

Bidding rule	Team Performance Criterion					
	MiniSum Lower - Upper		*MiniMax* Lower - Upper		*MiniAvg* Lower - Upper	
BidSumPath	1.5	2	r	$2r$	$\frac{n+1}{2}$	$2n$
BidMaxPath	r	$2r$	$\frac{r+1}{2}$	$2r$	$\Omega(n^{1/3})$	$2n$
BidAvgPath	n	$2n^2$	$\frac{r+1}{2}$	$2n^2 r$	$\Omega(n^{1/3})$	$2n^2$

Figure 1: Auction Types approximation ratios

Emergent Task Allocation (Swarm Intelligence) Coordination via **Stigmergy** (indirect communication through environment). No central controller.

1. Ant Corpse Clustering / Sorting Agents pick up items from low-density areas and drop them in high-density areas.

- f: Local density of similar items in neighborhood.
- k_1, k_2: Threshold constants.
- **Pick-up Probability (P_p):** Decreases as density f increases.
- **Drop Probability (P_d):** Increases as density f increases.

2. Response Threshold Models Probabilistic division of labor based on stimulus s_j (task demand) and internal threshold θ_{ij}.

- **Response Probability:**

$$P(\text{act}) = \frac{s_j^n}{s_j^n + \theta_{ij}^n}$$

- $n > 1$: Determines steepness (step-function like for high n). Usually $n = 2$.
- **Adaptive Thresholds (RL):**
  - If agent performs task j: $\theta_{ij} \leftarrow \theta_{ij} - \Delta$ (Specialization).
  - If agent is idle/fails: $\theta_{ij} \leftarrow \theta_{ij} + \delta$ (Forgetting).

  The sum of the above 2 is the total change in threshold. Adaptive Threshold is considered a hybrid of response threshold and reinforcement learning.

L21: RL & MACHINE LEARNING IN MRTA

Why Machine Learning for MRTA? Standard Market methods rely on *known* cost functions $C(t)$. In reality, costs are uncertain (e.g., hidden debris, traffic).

- **Goal:** Learn to estimate costs, learn bidding strategies, or learn allocation policies directly from experience.

Reinforcement Learning (RL) Approaches Agents learn policy $\pi(s)$ to maximize expected cumulative reward.

1. The Credit Assignment Problem (Reward Shaping) How do we reward individual agents for team behavior?

- **Global Reward $G(z)$:** Team performance (e.g., Total Time).
  - *Pros:* Aligned with goal. *Cons:* High noise. "Did I succeed, or did the team carry me?" Slow learning ($1/N$).
- **Local Reward $L_i(z)$:** Individual performance (e.g., Tasks I did).
  - *Pros:* Fast learning. *Cons:* Greedy/Suboptimal. Robots steal tasks or ignore cooperation.
- **Difference Reward $D_i(z)$:** The "Marginal Contribution".

$$D_i(z) = G(z) - G(z_{-i})$$

  - z: Full system state. z_{-i}: System state *without* agent i.

2. RL for Bidding (Adaptive Auctions) Agents use Q-learning to value tasks instead of hardcoded heuristics.

- **State s:** Context (Battery, Location, Current Load).
- **Action a:** Bid value b or Task Selection.
- **Q-Update:**

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') \right)$$

- **Outcome:** Agents learn to bid low on tasks they are bad at (high cost) and high on tasks they are good at, even if the math model is unknown.

Figure 2: calm down with this funny cat

Deep Learning / GNNs in MRTA

- Not strictly RL, but related.
- **Problem:** WDP is NP-Hard. Auctions are slow (communication).
- **Idea:** Train a Graph Neural Network (GNN) to predict the optimal assignment matrix A_{ij} in one shot.
- **Input:** Spatial graph (Nodes=Robots/Tasks, Edges=Distances).
- **Output:** Probability distribution over allocations.
- **Key Advantage:** "Deep models learn relational reasoning implicitly." They see spatial patterns (e.g., clusters) and assign teams without explicit combinatorial search.

L22: WISDOM OF CROWDS

Core Concept Aggregating judgments of a group often outperforms individual experts.

Condorcet Jury Theorem (Binary Choice) Assume N voters, binary decision (Correct/Wrong).

- Each voter is independent and has probability $p > 0.5$ of being correct iff

$$\lim_{N \to \infty} P(\text{Majority is Correct}) = 1$$

Diversity Prediction Theorem (Continuous) Each individual estimate can be written as: $s_i = \theta + b_i + \epsilon_i$, where:

- θ: the true value,
- b_i: individual bias (systematic shift),
- ϵ_i: random noise (zero-mean).

$$\underbrace{(c-\theta)^2}_{\text{Collective Error}} = \underbrace{\frac{1}{N}\sum_{i=1}^{N}(s_i-\theta)^2}_{\text{Avg Individual Error}} - \underbrace{\frac{1}{N}\sum_{i=1}^{N}(s_i-c)^2}_{\text{Diversity}}$$

- **Collective Error:** Squared error of the mean.
- **Avg Individual Error:** How accurate the average person is.
- **Diversity:** The variance of the opinions.
- **Implication:** Collective error is *always* less than or equal to the average individual error. Diversity ($Var(s)$) reduces error "for free".

Conditions for Wisdom (Surowiecki)

1. **Diversity of Opinion:** Each person has some private information.
2. **Independence:** Opinions are not determined by others.
3. **Decentralization:** People specialize/draw on local knowledge.
4. **Aggregation:** Mechanism to turn private judgments into collective decision (voting, markets, averaging).

Failures (Madness of Crowds)

- **Information Cascades:** Sequential decision making where later people ignore their own info to follow predecessors. Violates Independence.
- **Social Influence/Herding:** Correlated errors prevent variance reduction.

L23-24: DISTRIBUTED CONSENSUS

1. Graph Theoretic Formulation Consider a network of N agents.

- **Graph:** $G = (V, E)$. $V = \{1, \dots, N\}$, $E \subseteq V \times V$.
- **Adjacency Matrix (A):** $a_{ij} > 0$ if $(j, i) \in E$, else 0.
- **Degree Matrix (D):** Diagonal, $d_{ii} = \deg_{in}(i) = \sum_j a_{ij}$.
- **Laplacian Matrix (L):** $L = D - A$.

Laplacian Properties:

- **Row Sums:** $L\mathbf{1} = \mathbf{0}$. ($\mathbf{1}$ is eigenvector for $\lambda_1 = 0$).
- **Spectrum:** $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$.
- **Symmetry:** If G undirected, L symmetric $\implies$ real eigenvalues.
- **Algebraic Connectivity (λ_2):** G is connected iff $\lambda_2 > 0$.
- **Eigenvalue Bound:** $\lambda_N \leq 2\Delta_{max}$ (where Δ is max degree).

2. Continuous-Time Consensus Protocol: Move towards neighbors.

Matrix Dynamics:

$$\dot{x}(t) = -Lx(t)$$

Solution Trajectory:

$$x(t) = e^{-Lt}x(0)$$

Eigen-Decomposition (for undirected G): Let v_k be eigenvectors of L with eigenvalues λ_k.

$$x(t) = \sum_{k=1}^{N} c_k e^{-\lambda_k t} v_k$$

Convergence State (Average Consensus):

$$x(\infty) = (\mathbf{1}^T x(0)/N)\mathbf{1} = \alpha\mathbf{1}, \quad \alpha = \text{Ave}(x(0))$$

Convergence Rate: Bounded by second smallest eigenvalue λ_2 (Fiedler value).

$$\|x(t) - \alpha\mathbf{1}\| \leq \|x(0)\|e^{-\lambda_2 t}$$

Time constant $\tau = 1/\lambda_2$.

3. Discrete-Time Consensus Update rule with step size ϵ:

$$x_i(k+1) = x_i(k) + \epsilon \sum_{j \in N_i} a_{ij}(x_j(k) - x_i(k))$$

Matrix Dynamics:

$$x(k+1) = (I - \epsilon L)x(k) = Px(k)$$

where P is the Consensus Matrix. **Properties of P:**

- **Row Stochastic:** $P\mathbf{1} = \mathbf{1}$ (Rows sum to 1).
- **Eigenvalues of P:** $\mu_i = 1 - \epsilon\lambda_i$.
- **Stability Condition:** Need $|\mu_i| \leq 1$ for all i.

$$0 < \epsilon < \frac{2}{\lambda_{max}} \quad \text{or} \quad \epsilon < \frac{1}{\Delta_{max}}$$

- **Doubly Stochastic:** If G undirected, P is symmetric (Rows & Cols sum to 1). $\implies$ Average is preserved.

4. Non-Linear Consensus: Kuramoto Model Used for synchronization of phases $\theta_i \in S^1$.

$$\dot{\theta}_i = \omega_i + \frac{K}{N} \sum_{j=1}^{N} \sin(\theta_j - \theta_i)$$

- ω_i: Natural frequency of oscillator i.
- K: Coupling strength.
- **Order Parameter (r):** Measures coherence.

$$re^{i\psi} = \frac{1}{N} \sum_{j=1}^{N} e^{i\theta_j}$$

- $r \approx 0$ (Incoherent), $r \approx 1$ (Synchronized).
- **Critical Coupling K_c:** Phase transition occurs at K_c.

5. Robotic Flocking / Formation Control Combining consensus with alignment and cohesion. State vectors: Position x_i, Heading θ_i. **Heading Control (Kuramoto-style):**

$$\dot{\theta}_i = \omega_i + \sum_{j \in N_i} K_{ij} \sin(\theta_j - \theta_i)$$

Position Control (Kinematics):

$$\dot{x}_i = v_i \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}$$

Formation Control Law: To maintain formation distance d_{ij}:

$$u_i = -\sum_{j \in N_i} (\|x_i - x_j\|^2 - d_{ij}^2)(x_i - x_j)$$

This acts as a spring-mass system minimizing potential energy.

6. Comparison

Feature	Consensus (Single Int)	Kuramoto
State Space	$\mathbb{R}^n$ (Euclidean)	S^1 (Angular)
Interaction	Linear Diff $(x_j - x_i)$	Sinusoidal $\sin(\theta_j - \theta_i)$
Equilibrium	$x_i = x_j$ (Point)	$\dot{\theta}_i = \dot{\theta}_j$ (Sync)
Application	Rendezvous, Sensor Fusion	Flocking, Clocks

L25-L26: NEURAL NETWORKS (BASICS)

The Perceptron Linear binary classifier. Inputs x, Weights w, Bias b.

$$z = w \cdot x + b = \sum w_i x_i + b$$

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{else} \end{cases}$$

Limitation: Can only solve linearly separable problems (No XOR).

Activation Functions $\phi(z)$ Required to stack layers (Linear of Linear is still Linear).

- **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$. Range $(0, 1)$.
  - Deriv: $\sigma(z)(1 - \sigma(z))$.
  - *Prob:* Vanishing gradient for large $|z|$. Not zero-centered.
- **Tanh:** $\tanh(z)$. Range $(-1, 1)$. Zero-centered.
  - Deriv: $1 - \tanh^2(z)$. Still vanishes.
- **ReLU:** $\max(0, z)$. Range $[0, \infty)$.
  - Deriv: 1 if $z > 0$, 0 if $z < 0$.
  - *Pros:* Fast, no vanishing gradient in + region.
  - *Cons:* Dead ReLU (if weights push $z < 0$ always).
- **Softmax:** $p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$. For multi-class probability.

Loss Functions $J(\theta)$

- **MSE (Regression):** $J = \frac{1}{N}\sum(y - \hat{y})^2$.
- **Cross-Entropy (Classification):** $J = -\sum_k y_k \log(\hat{y}_k)$.
- **Hinge Loss (SVM):** $J = \max(0, 1 - y \cdot \hat{y})$.

Backpropagation (Chain Rule) Goal: Compute $\nabla_w J$. Example: $x \xrightarrow{w} z \xrightarrow{\phi} a \xrightarrow{Loss} J$.

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

The Algorithm: 1. **Forward Pass:** Compute $a^{(l)}$ for all layers. 2. **Backward Pass:** Compute error $\delta^{(L)}$ at output.

$$\delta^{(L)} = \nabla_a J \odot \phi'(z^{(L)})$$

3. **Propagate:** $\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \odot \phi'(z^{(l)})$. 4. **Gradients:** $\frac{\partial J}{\partial W^{(l)}} = \delta^{(l)}(a^{(l-1)})^T$.

Optimization

- **SGD:** $\theta \leftarrow \theta - \eta \nabla J_{sample}$. High variance, escapes local minima.
- **Momentum:** $v_t = \gamma v_{t-1} + \eta \nabla J$. $\theta \leftarrow \theta - v_t$. Builds velocity in consistent directions.

L27-L28: TUNING & CNNS

Generalization & Regularization

- **Overfitting:** Low Train Loss, High Test Loss. Model learns noise.
- **Underfitting:** High Train Loss. Model too simple.

Techniques to fix Overfitting:

1. **Dropout:** Randomly zero out neurons with prob p during training.
   - Training: $\tilde{h} = h \odot \text{Bernoulli}(1 - p)$.
   - Testing: Scale weights by $(1 - p)$ (or scale up during train).
   - Effect: Prevents co-adaptation, acts as ensemble of 2^N nets.
2. **Early Stopping:** Monitor Validation Loss. Stop when it rises.
3. **Data Augmentation:** Flip, rotate, crop images to increase N.

Data Preprocessing

- **Normalization:** $(x - \mu)/\sigma$. Ensures gradients scale equally for all weights.
- **Weight Initialization:**
  - *Zero:* Fails (neurons symmetric, learn same thing).
  - *Xavier (Glorot):* Var $= 2/(n_{in} + n_{out})$. Good for Sigmoid/Tanh.
  - *He Init:* Var $= 2/n_{in}$. Good for ReLU.

Cross-entropy loss

logits = raw scores

Single sample cross-entropy loss between the true distribution y and the predicted distribution $\hat{y}$:

$$\text{CEL}(x, \mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{\text{Classes}} y_j \log(\hat{y}_j)$$

For the true class c : $\quad \text{CEL}_i = -\log(\hat{y}_c)$

Cross-entropy loss over the whole dataset:

$$\text{CEL} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{\text{Classes}} y_{ij}\,\log(\hat{y}_{ij})$$

- Cross-entropy measures how different the true class distribution $\mathbf{y}$ is from the predicted distribution $\hat{\mathbf{y}}$; it increases as predictions become less confident or incorrect.
- For a single sample, only the log–probability of the true class matters, since $y_c = 1$ and all other $y_j = 0$.
- With a Softmax output, $\hat{y}_j \in (0, 1)$ and never equals zero, preventing issues such as $\log(0)$ and ensuring stable gradients.
- Organize learning in epochs, where each epoch is a full sweep of the training set using SGD, based on the chosen batch sizes

Convolutional Neural Networks (CNNs) **Goal:** Process grid-like topology (Images) efficiently. **Key Inductive Biases:**

1. **Locality / Spatial Coherence:** Neurons connect only to a local region of the input volume (Receptive Field).
2. **Translation Invariance (Parameter Sharing):** The same feature detector (kernel) is useful at different positions. Weights are shared across space.
3. **Hierarchical Feature Learning:** Low-level (Edges) → Mid-level (Shapes) → High-level (Objects).

The Convolution Operation Input Volume: $W_1 \times H_1 \times D_1$. Kernel (Filter): $F \times F \times D_1$ (Depth must match input). Number of Filters: K. Stride: S. Padding: P. **Formula:**

$$(I * K)_{ij} = \sum_{m=0}^{F-1}\sum_{n=0}^{F-1}\sum_{d=0}^{D_1-1} I(i \cdot S + m, j \cdot S + n, d) \cdot K(m, n, d) + b$$

Output Dimensions ($W_2 \times H_2 \times D_2$):

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K \quad \text{(Number of Filters)}$$

Padding Types:

- **Valid:** No padding ($P = 0$). Output size shrinks.
- **Same:** Pad such that output size equals input size (if $S = 1$). Requires $P = (F - 1)/2$.

Parameter Count: Each filter has $F \cdot F \cdot D_1 + 1$ (bias) parameters. Total Params $= K \cdot (F^2 D_1 + 1)$.

Pooling Layers Downsampling operation to reduce spatial size (W, H).

- Operates independently on every depth slice.
- **Max Pooling:** Take max value in window. Preserves dominant features.
- **Average Pooling:** Take average. Smooths features.
- **Global Average Pooling:** Averages entire $W \times H$ map into 1 number per channel. Used before final classification to replace Flatten layer.
- **Properties:** No learnable parameters. Improves invariance to small shifts/distortions.

Transfer Learning Using a pre-trained network (e.g., on ImageNet, 1.2M images) for a new task with limited data. **Scenarios based on Dataset Size:**

1. **Fixed Feature Extractor:**
   - Use pre-trained CNN as a frozen vector generator.
   - Remove the last fully-connected (FC) layer.
   - Train a new linear classifier (or small MLP) on top of the extracted features.
   - *Use when:* New dataset is small + similar to original.
2. **Fine-Tuning:**
   - Initialize with pre-trained weights.
   - Unfreeze some top layers (or all layers) and train with a very small learning rate.
   - *Use when:* New dataset is large, or different from original.

L29: GRAPH NEURAL NETWORKS (GNNS)

1. Motivation & Network Modeling

- **Why GNNs?** Images are Euclidean (grid); Graphs are Non-Euclidean (irregular connections).
- **Properties:**
  - **Degree k_i:** Number of connections. Dist $P(k)$.
  - **Clustering Coeff C_i:** $2E_i/k_i(k_i - 1)$. Fraction of connected neighbors.
  - **Avg Path Length $\langle h \rangle$:** Mean distance between node pairs.
  - **Small-World:** High C, Low $\langle h \rangle$. (Watts-Strogatz model).
- **Permutation Invariance:** Re-ordering nodes $(v_1, v_2 \to v_2, v_1)$ should not change graph output.

2. Message Passing Neural Networks (MPNN) General framework for spatial GNNs. **The Core Loop (for layer l):**

1. **Message Computation:** For each edge (u, v):

$$m_{u\to v}^{(l)} = \text{MSG}(h_u^{(l-1)}, h_v^{(l-1)}, e_{uv})$$

2. **Aggregation:** Collect messages from neighborhood $\mathcal{N}(v)$.

$$m_v^{(l)} = \text{AGG}(\{m_{u\to v}^{(l)} : u \in \mathcal{N}(v)\})$$

   Aggregators: Sum, Mean, Max. Must be permutation invariant.
3. **Update:** Update node state.

$$h_v^{(l)} = \text{UPDATE}(h_v^{(l-1)}, m_v^{(l)})$$

3. Graph Convolutional Network (GCN) Specific MPNN variant using spectral approximation.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$

- $\tilde{A} = A + I_N$ (Adjacency with self-loops).
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ (Degree matrix).
- **Renormalization:** $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$ normalizes messages. Without it, features explode (if unnormalized) or vanish (if row-normalized).
- **Effect:** Isotropic smoothing of features over neighbors.

4. GraphSAGE (Inductive Learning) Designed for large graphs where we cannot fit full A in memory.

- **Neighbor Sampling:** Sample fixed size neighborhood (e.g., 10 neighbors) to keep computation constant.
- **Aggregation:**

$$h_{\mathcal{N}(v)}^{(l)} = \text{AGG}_l(\{h_u^{(l-1)}, \forall u \in \mathcal{N}(v)\})$$

  Aggregators: Mean, LSTM, Max-Pooling.
- **Update:** Concat self + neighbor info.

$$h_v^{(l)} = \sigma\left(W^{(l)} \cdot [h_v^{(l-1)} \| h_{\mathcal{N}(v)}^{(l)}]\right)$$

- **Inductive:** Can generate embeddings for *unseen* nodes (unlike GCN/Transductive).

5. Graph Attention Network (GAT) Assigns different importance α_{uv} to different neighbors.

1. **Attention Mechanism:**

$$e_{uv} = \text{LeakyReLU}(\vec{a}^T[Wh_u \| Wh_v])$$

2. **Normalization (Softmax):**

$$\alpha_{uv} = \frac{\exp(e_{uv})}{\sum_{k\in\mathcal{N}(u)}\exp(e_{uk})}$$

3. **Weighted Aggregation:**

$$h_u^{(l+1)} = \sigma\left(\sum_{v\in\mathcal{N}(u)} \alpha_{uv} W h_v^{(l)}\right)$$

4. **Multi-Head Attention:** Concat results from K independent heads to stabilize learning.

6. Tasks & Readout

- **Node Classification:** Use final embedding $h_v^{(L)}$. Loss: Cross-entropy on labeled nodes.
- **Link Prediction (Edge-level):** Predict existence of edge (u, v).

$$y_{uv} = \text{MLP}(h_u \| h_v) \quad \text{or} \quad h_u^T h_v$$

- **Graph Classification (Graph-level):** Aggregate all nodes to get h_G.

$$h_G = \text{READOUT}(\{h_v^{(L)} | v \in G\})$$

  Methods: Global Sum/Mean Pooling, Hierarchical Pooling (DiffPool).

7. Issue: Over-smoothing In deep GCNs, repeated averaging makes all node embeddings converge to the same value. **Solutions:** Skip connections (Residuals), DropEdge.

L30: FEDERATED LEARNING (FL)

1. Motivation & Concept

- **Core Idea:** "Bring code to data, not data to code."
- **Why?** Data is distributed (phones, hospitals), privacy-sensitive, bandwidth-constrained.
- **Definition:** Agents collaborate to train a global model w without sharing raw data D_k.

$$\min_w \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where } F_k(w) = \frac{1}{n_k}\sum_{i\in D_k} \ell_i(w)$$

2. FL Architectures

- **Centralized FL:** Server coordinates, clients train. (Standard).
- **Decentralized FL:** P2P gossip learning, no central server.

- **Cross-Device:** Millions of mobile/IoT devices, unreliable, low bandwidth.
- **Cross-Silo:** Few organizations (hospitals/banks), high reliability, massive data per client.

3. Algorithms

A. Federated SGD (FedSGD)

- Clients compute exact gradient $\nabla F_k(w_t)$ on full local data.
- Server updates: $w_{t+1} = w_t - \eta \sum \frac{n_k}{n} \nabla F_k$.
- **Cons:** Very high communication overhead (per step).

B. Federated Averaging (FedAvg) The de-facto standard. Clients perform *multiple* local epochs (E). **Protocol Loop:**

1. **Broadcast:** Server sends global model w_t to subset of clients S_t.
2. **Local Training:** Each client $k \in S_t$:
   - $w_t^k \leftarrow w_t$
   - Run E epochs of SGD: $w^k \leftarrow w^k - \eta \nabla \ell(w^k)$
   - Send update $\Delta w_k = w_{final}^k - w_t$ (or just weights) to server.
3. **Aggregation:** Server averages updates:

$$w_{t+1} \leftarrow w_t + \sum_{k \in S_t} \frac{n_k}{n} \Delta w_k$$

Properties: Increases computation per communication round. Converges on IID data.

C. Federated Distillation (FD) Model-Agnostic FL.

- **Problem:** Sending weights is expensive (GBs for LLMs).
- **Idea:** Exchange model outputs (logits/soft labels) on a shared public dataset.
- **Mechanism:** Clients act as Teachers, Server as Student.

$$L_{distill} = KL(P_{teacher}(y|x)||P_{student}(y|x))$$

- **Pros:** Very low bandwidth. Heterogeneous client architectures allowed.

4. Key Challenges

- **Systems Heterogeneity (Stragglers):** Devices have different CPU/Battery. Server must wait for slowest device in synchronous FL. **Fix:** Asynchronous aggregation (FedAsync), Timeouts.
- **Statistical Heterogeneity (Non-IID Data):** $P_k(x, y) \neq P(x, y)$. (e.g., User A types "Good morning", User B "Bon jour"). **Effect:** Client Drift. Local updates move towards local minima, average is not global minimum. **Fix:** FedProx (add proximal term $\frac{\mu}{2}\|w - w_t\|^2$ to local loss).
- **Privacy Leakage:** Gradients can leak data (Model Inversion). **Fix:** Differential Privacy (Clip gradients + Add Gaussian noise), Secure Aggregation (MPC).

L31: AGENTIC AI

1. Definitions & Paradigm

- **Generative AI:** Reactive. User Prompt → Output. Stateless.
- **Agentic AI:** Proactive. Goal → Plan → Act → Iterate. Stateful.
- **Core Traits:** Autonomy (Initiative), Tool Use (API/Code), Persistence (Memory).

2. The Agentic Loop (Cognitive Architecture) Iterative process (Observe-Think-Act).

1. **Perception:** Read environment, user input, tool outputs.
2. **Memory:**
   - **Short-term:** Context window (conversation history).
   - **Long-term:** Vector DB / RAG (retrieval).
3. **Reasoning (The LLM):**
   - **Decomposition:** Break complex goal into sub-steps.
   - **Self-Reflection:** Critique past actions ("Did that work?").
4. **Planning:** Sequence actions (Chain of Thought).
5. **Action:** Execute tool calls (Python REPL, Search, Calculator).

3. ReAct Framework (Reasoning + Acting) Prompting strategy to interleave thought and action. **Template:**

- **Thought:** Analyze the current state and decide what to do.
- **Action:** Specific tool call (e.g., 'search("weather Doha")').
- **Observation:** Output from the tool (e.g., "35C").
- **Thought:** Process observation. "I have the info."
- **Action:** Final Answer.

Why? Thoughts help maintain reasoning trace; Actions ground LLM in reality.

4. Tool Use (Function Calling) LLMs don't just output text; they output structured API calls.

- **Input:** Schema of available tools (Name, Args, Docstring).
- **Process:** LLM generates JSON arguments. Runtime executes code.

5. Multi-Agent Systems (Swarm 2.0) Decomposing a single "Super Agent" into specialized roles.

- **Analogy:** Software equivalent of MRTA.
- **Roles:**
  - **Planner/Manager:** Breaks down user request, delegates.
  - **Executor/Coder:** Writes code/performs task.
  - **Reviewer/Critic:** Checks output, gives feedback.
- **Coordination:** Agents communicate via chat messages (Message Passing).
- **Advantages:** Specialization (Prompt Engineering per role), Error Isolation, Parallelism.
- **Frameworks:** AutoGen, CrewAI, LangGraph.

6. Future Outlook & Risks

- **Integration:** Agents controlling physical robots (VLA: Vision-Language-Action models).
- **Alignment Risk:** Autonomous loops can diverge from intent.
- **Safety:** Infinite loops, Resource exhaustion, Unintended API calls.