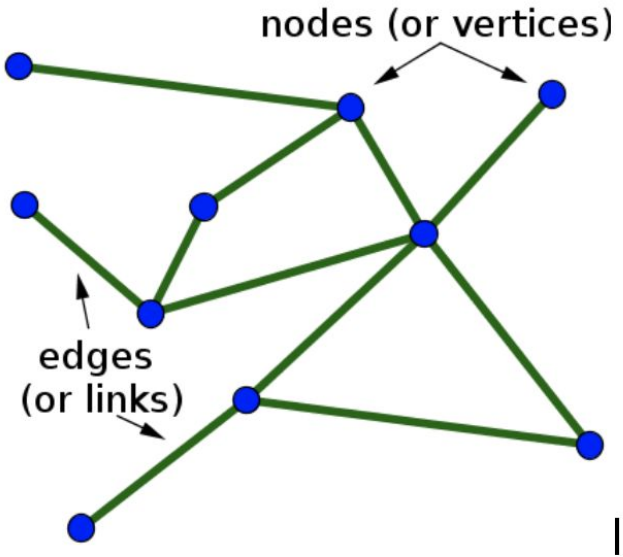
Graph Neural Networks



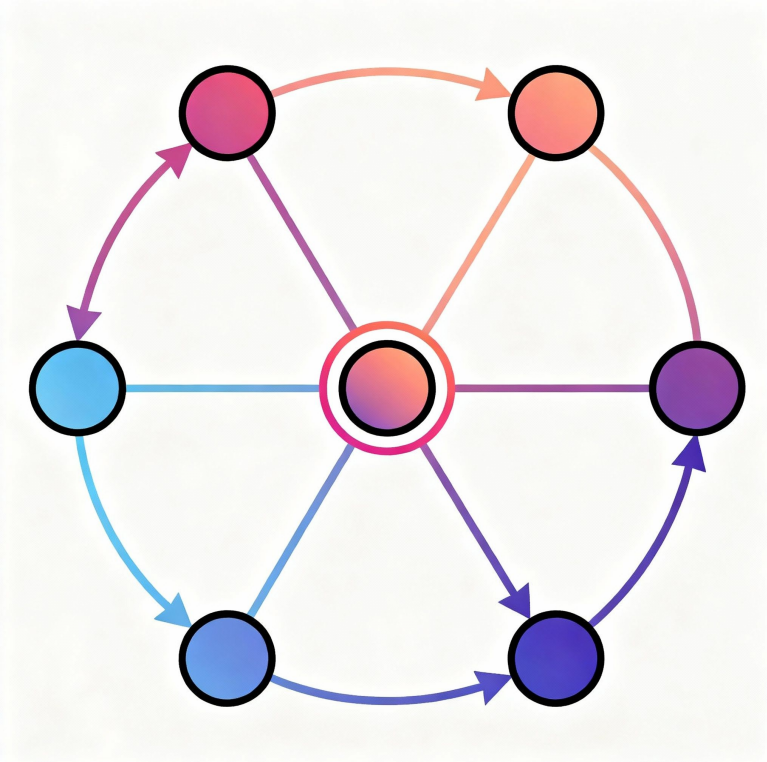
Introduction to GNNs

Everyday intuition: in social networks, a person's interests can be inferred not only from their profile but from friends and interactions, which is exactly the kind of relational signal graphs encode.

Definition: *a GNN is a neural architecture that learns over graph-structured data by propagating and aggregating information along edges while respecting permutation invariance.*

Core mechanism: message passing updates node states by combining their features with aggregated neighbor messages across layers or "hops."

Why now: graphs are ubiquitous across domains, and GNNs unify successful ideas from deep learning with relational inductive biases to achieve strong performance on structured tasks.



Applications

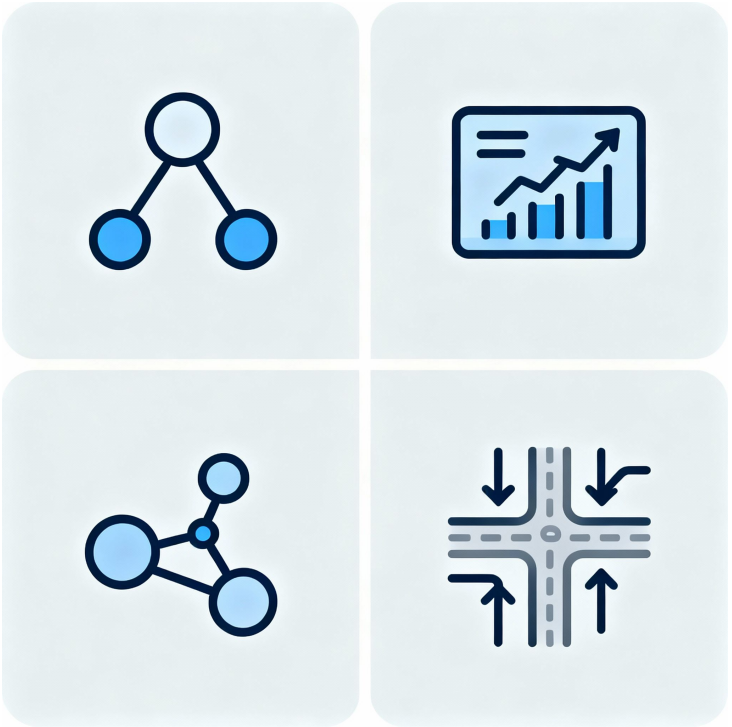
Recommender systems: propagate preferences over user–item graphs to suggest relevant products or content based on multi-hop similarity and interactions.

Fraud and risk in transaction networks: capture suspicious patterns like dense rings, star structures, or unusual multi-hop flows that evade local detectors.

Drug discovery and materials: predict molecular properties by passing messages over atoms and bonds, achieving strong benchmarks in chemistry.

Traffic and mobility: forecast speeds and demand by modeling road connectivity and temporal dynamics with spatio-temporal GNNs.

Time series with relations: learn dependencies among sensors or entities (e.g., power grids, multivariate KPIs) by constructing graphs and applying GNN4TS methods.



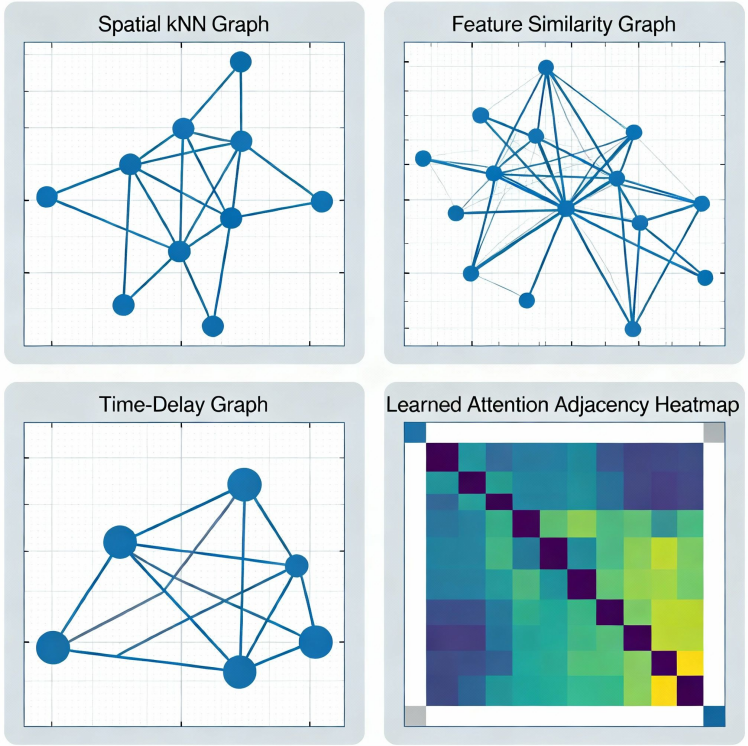
Why graph construction matters

Big idea: Treat graphs as the “wiring diagram” of your problem; if edges don’t reflect real dependencies, even strong GNNs underperform.

Explicit structures (already in data):

- **Recommenders**: users–items as bipartite nodes; edges = interactions with types/weights.
- **Molecules/Proteins:** nodes = atoms/amino acids; edges = bonds/contacts, typed/directed as needed.
- **Traffic**: nodes = sensors/road segments; edges = intersections/adjacency; attributes on nodes/edges.

Design checklist: task signal → node granularity → edge logic (topology/interaction/similarity/time) → attributes → sparsity/validation.



Building graphs from implicit structure

Computer Vision options:

- Grid graphs: image → patches; connect neighbors (simple, uniform).
- Scene/skeleton graphs: objects/joints as nodes; spatial edges capture relations/pose.
- Semantic graphs: cluster features; connect by similarity (kNN) and proximity; videos add temporal edges.

NLP and programs:

- Text graphs: nodes = words/sentences/docs; edges by co-occurrence/location/similarity.
- Syntactic graphs: dependency/constituency edges for grammatical structure.
- Semantic/knowledge/hybrid: typed entities/relations; mix structures to fuse signals.
- Programs: unify AST, control/data flow, and calls as a heterogeneous directed graph.

Pitfalls → fixes: over-dense kNN → sparsify/threshold; missing direction/time → add; heterogeneity ignored → typed edges; noisy similarity → robust metrics/learnable weights.

Why Representing Graphs Isn’t That Simple!

Real-world systems like social networks, biological networks, and web links naturally form graphs.

Traditional representation: G=(V,E)

G=(V,E) — where V = nodes and E = edges.

- But as graphs grow (**billions** of nodes), this model breaks down due to:
- **High computational cost:** e.g., Finding shortest paths is combinatorial.
- **Low parallelizability:** Inter-node dependencies cause heavy communication overhead.
- **Poor ML compatibility:** Graphs lack independent, fixed-size vector forms.

Turning Relationships into Vectors

Goal: Learn low-dimensional embeddings that preserve structure while enabling ML tasks.

Two Core Objectives:

- Graph Reconstruction – If two nodes are connected, their vectors stay close.
- Graph Inference – Predict unseen links, classify nodes, or visualize clusters.

In this learned space, geometric distance = relationship strength.

Enables link prediction, node classification, and community detection.

From Classical Embeddings to Neural Graphs

Traditional Graph Embedding (e.g., Isomap, LLE, Laplacian Eigenmaps):

- Focus on dimensionality reduction and structure preservation.
- Computationally expensive → struggles with large graphs.

Modern Graph Embedding:

- Incorporates richer information — side attributes, network properties.
- Techniques: Matrix factorization, random walks, deep models.
- Enables meaningful inference (link prediction, similarity, influence).

Graph Neural Networks (GNNs):

- Directly learn node embeddings via message passing.
- Most scalable and expressive modern approach.