Knowledge Graphs

🧠 Knowledge Graphs — Representations, Completion, and Properties

🌍 1. What is a Knowledge Graph (KG)?

A **Knowledge Graph (KG)** is a structured way to **store information as relationships between entities**.

Instead of plain text, it represents knowledge in the form of **triples**:

$$(\text{Head Entity}, \text{Relation}, \text{Tail Entity})$$

or simply:

$$(h, r, t)$$

Example:

- ("Albert Einstein", *bornIn*, "Ulm")
- ("Paris", *isCapitalOf*, "France")
- ("Water", *hasProperty*, "Liquid")

Each **node** is an *entity*, and each **edge** is a *relation*.

🧩 2. Why We Use Knowledge Graphs

In natural text, knowledge is *unstructured*.

Example:

> "Einstein was born in Ulm, Germany in 1879."

A machine finds this hard to process directly.

But a knowledge graph turns it into structured data:

```
[Einstein] --(bornIn)⟶ [Ulm]
[Einstein] --(bornOn)⟶ [1879]
[Ulm] --(locatedIn)⟶ [Germany]
```

✅ Easy for algorithms to:

- Search
- Reason (e.g., infer new facts)
- Link with other knowledge sources
- Use for retrieval in **Generative AI**

⚙️ 3. Structure of a Knowledge Graph

Component	Description	Example
Entities (Nodes)	Objects, people, places, concepts	“Einstein”, “Ulm”, “Germany”
Relations (Edges)	Connections between entities	*bornIn*, *locatedIn*, *isPartOf*
Triples (Facts)	Combination of two entities + relation	(“Einstein”, *bornIn*, “Ulm”)

Each triple is a **fact** in the knowledge base.

🧩 4. Mathematical Representation

Knowledge graphs can be represented as:

- A **directed labeled graph**

  where edges are labeled with relation types.

$G = (E, R, T)$

where

- (E): set of entities
- (R): set of relations
- $(T \subseteq E \times R \times E)$: set of triples (facts)

🧮 5. Knowledge Graph Representations

Because graphs are **discrete**, we often convert them into **continuous vector representations (embeddings)** for ML models.

a. One-Hot Representation

Each entity and relation is represented as a one-hot vector.

- Simple but inefficient (very large and sparse).

b. Embedding-Based Representation

Each entity (e_i) and relation (r_k) is embedded into a **dense vector space**.

Goal: represent each fact (h, r, t) in a way that **true triples score higher** than false ones.

Different models define the *scoring function* differently.

🧠 6. Common Knowledge Graph Embedding Models

Model	Core Idea	How Relations are Modeled	Key Properties / Notes
TransE (2013)	Represents relations as translations in vector space.	`h + r ≈ t` — the head plus relation vector should approximate the tail vector.	Simple and efficient; struggles with complex relations (1-to-N, N-to-1, N-to-N).
TransH (2014)	Projects entities onto a relation-specific hyperplane before translation.	Each relation has a normal vector defining its hyperplane.	Handles multi-valued relations better than TransE.
TransR / TransD (2015)	Entities and relations exist in different spaces; relation-specific projection.	Projects entity embeddings into relation space before applying translation.	Captures relation-specific features; more flexible than TransE/H.
DistMult (2015)	Uses a bilinear scoring function with diagonal matrices.	`score(h, r, t) = hᵀ diag(r) t`	Efficient; assumes symmetric relations, limiting expressiveness.

Model	Core Idea	How Relations are Modeled	Key Properties / Notes
ComplEx (2016)	Extends DistMult into complex vector space.	Uses complex-valued embeddings and Hermitian dot product.	Models asymmetric relations (e.g., “parentOf”); strong empirical performance.
RESCAL (2011)	Uses a full matrix per relation to capture pairwise interactions.	$\text{score}(h, r, t) = h^\top R\, t$ where R is a dense matrix.	Very expressive but computationally heavy for large KGs.
HolE (2016)	Combines circular correlation of entity vectors to encode interactions.	Compresses interactions efficiently using circular correlation.	Combines expressiveness of RESCAL with efficiency of simpler models.
RotatE (2019)	Models relations as rotations in complex space.	$t = h \odot r$, where $\odot$ denotes element-wise rotation in complex plane.	Captures symmetry, antisymmetry, inversion, and composition.
SimplE (2018)	Extends CP decomposition for bidirectional relations.	Each entity has two embeddings (as head and as tail).	Simple and interpretable; handles inverse relations well.
TuckER (2019)	Based on Tucker tensor decomposition.	Learns shared core tensor for all relations.	High expressiveness with fewer parameters than RESCAL.

These embeddings make it possible to perform reasoning, completion, and similarity search efficiently.

🔍 7. Knowledge Graph Completion (KGC)

💬 What It Is

KGs are often incomplete — many real-world facts are missing.

Knowledge Graph Completion aims to **predict missing facts**, such as:

> Given (“Einstein”, bornIn, ?), predict “Ulm”.

This is done by:

1. Learning embeddings for entities and relations.
2. Using a **scoring function** to estimate if a triple is true.

⚙️ Example: Using TransE

For triple (h, r, t):

$$\text{Score}(h, r, t) = -||h + r - t||$$

- If score is high (small distance), triple is likely true.
- If low, triple is likely false.

This allows you to infer new facts like:

> (“Shakespeare”, bornIn, “Stratford”)
>
> (“Shakespeare”, *wrote*, “Hamlet”)
>
> → maybe infer (“Hamlet”, *writtenBy*, “Shakespeare”).

🧩 8. Types of Knowledge Graph Completion Tasks

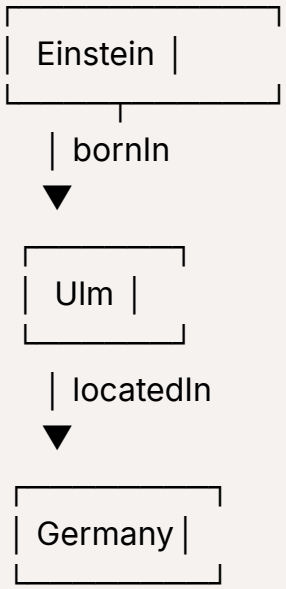
Task	Example	Goal
Link Prediction	(“Paris”, *isCapitalOf*, ?)	Predict missing tail/head entity
Relation Prediction	(“Einstein”, ?, “Ulm”)	Predict missing relation
Entity Typing	(“Einstein”, *isA*, ?)	Predict entity type (Person, Scientist)

🧠 9. Knowledge Graph Properties

Property	Meaning	Example / Effect
Directed	Each edge has a direction	(Einstein → bornIn → Ulm)
Labeled	Edges have semantic meaning	*bornIn*, *isPartOf*
Heterogeneous	Different entity and relation types	Person, City, Country

Property	Meaning	Example / Effect
Incomplete	Missing facts	May not know where “Tesla” was born
Dynamic (in some KGs)	Knowledge updates over time	“PresidentOf(USA)” changes
Multirelational	Many kinds of relationships exist between same nodes	(Einstein, *livedIn*, Germany), (Einstein, *bornIn*, Germany)

🧩 10. Graph Visualization Example



⚡ 11. Applications of Knowledge Graphs

✅ **Search Engines (Google KG, Bing)** → better entity understanding

✅ **Question Answering** → use graph reasoning to find answers

✅ **Recommendation Systems** → infer related items

✅ **RAG Systems (LLMs)** → provide factual grounding

✅ **Medical / Legal / Finance Domains** → structure expert knowledge

🧾 12. Key Takeaways

- A **Knowledge Graph** represents facts as *(head, relation, tail)* triples.
- Entities and relations can be **embedded** for machine learning.
- **Knowledge Graph Completion** fills in missing facts.
- Different models (TransE, RotatE, ComplEx) define how to model relationships.
- Properties like **directionality, heterogeneity, and incompleteness** make KGs powerful yet complex.
- They are the **backbone of reasoning and factual retrieval** in AI systems.

⚠️ Limitations of Knowledge Graphs

🧩 1. Incompleteness

💬 Explanation

Most real-world knowledge graphs (like Wikidata or Freebase) are **incomplete** — they cover only a **fraction of true facts**.

For example, they might know that:

> (“Einstein”, bornIn, “Ulm”)
>
> but not
>
> (“Einstein”, *receivedAward*, “Nobel Prize”)

⚙️ Why this happens

- It’s hard to manually add all real-world facts.
- Automatic extraction from text (information extraction) is imperfect.
- The world changes constantly — KGs can’t update in real time.

📉 Impact

→ Leads to **low recall** (many true facts missing).

→ Causes **incorrect inferences** or “unknown = false” assumptions.

🧩 2. Scalability and Maintenance

💬 Explanation

As KGs grow (millions of entities, billions of edges),

storing, updating, and reasoning over them becomes **computationally heavy**.

⚙️ Challenges

- **Storage cost** grows quadratically with number of entities.
- **Reasoning algorithms** (e.g., path finding, rule mining) become very slow.
- Updating or deleting knowledge is complex (dependencies between facts).

📉 Impact

→ Limits **real-time reasoning** and **dynamic knowledge updates**.

→ Difficult to maintain in fast-changing domains (e.g., finance, news).

🧩 3. Data Quality and Noise

💬 Explanation

KGs often combine data from multiple sources (web, databases, text).

This leads to:

- Duplicate entities (e.g., “NYC” vs “New York City”)
- Conflicting facts
- Incorrect relations (bad extraction)

⚙️ Example

- (“Barack Obama”, *bornIn*, “Hawaii”) ✅
- (“Barack Obama”, *bornIn*, “Kenya”) ❌ (error due to misinformation)

📉 Impact

→ Leads to **ambiguous reasoning** and **false conclusions**.

→ Makes training embeddings unstable or misleading.

🧩 4. Lack of Context and Temporal Information

💬 Explanation

KGs store facts as **static triples** — but real-world knowledge is **contextual and time-dependent**.

Example:

- (“Biden”, *isPresidentOf*, “USA”) → True in 2025
- (“Trump”, *isPresidentOf*, “USA”) → True in 2019

Without timestamps, both may appear “true” simultaneously.

📉 Impact

→ KGs cannot handle **temporal or situational reasoning**.

→ Hard to model changing facts, evolving entities, or event-based knowledge.

🧩 5. Limited Expressiveness

💬 Explanation

Each triple (head, relation, tail) captures a **simple binary relationship**.

But real-world facts are often **more complex** (involving multiple entities or conditions).

Example:

> “Alice gave Bob a book on Tuesday.”
>
> needs at least 4 pieces of info (giver, receiver, object, time).
>
> A triple like (Alice, *gaveTo*, Bob) misses the object and time context.

📉 Impact

→ KGs struggle to represent **n-ary relations** (relations with more than 2 entities).

→ Difficult to model **events, quantities, conditions, or causes**.

🧩 6. Difficulty in Automatic Construction

💬 Explanation

Building large-scale KGs automatically from raw text involves **Information Extraction (IE)** —

tasks like entity recognition, relation extraction, and linking.

These steps are **error-prone**, especially for:

- Ambiguous language
- Implicit relations (“He founded the company” → who is “He”?)
- Sarcasm, negation, or multi-sentence facts

📉 Impact

→ Results in **incomplete** or **incorrect graphs**.

→ Requires constant human supervision or validation.

🧩 7. Poor Generalization and Reasoning

💬 Explanation

KGs store **explicit facts**, not reasoning rules.

They cannot easily **infer new knowledge** unless supported by specialized algorithms or embeddings.

Example:

> (Paris, isCapitalOf, France)
>
> (France, *isInContinent*, Europe)
>
> KG cannot automatically infer:

> (Paris, *isInContinent*, Europe)

unless explicitly trained with logical or embedding-based reasoning models.

📉 Impact

→ Weak at **multi-hop reasoning** and **commonsense inference**.

→ Need external models (like Graph Neural Networks or LLMs) to infer new facts.

🧩 8. Lack of Uncertainty Handling

💬 Explanation

Each triple is usually stored as **true** or **false**, with no degree of confidence.

But in real knowledge:

> ("Chocolate", causes, "Happiness") → sometimes true, sometimes not.

📉 Impact

→ KGs can't express **probabilistic or fuzzy relationships**.

→ Makes them brittle for real-world reasoning where uncertainty exists.

🧩 9. Integration with Natural Language

💬 Explanation

While KGs are structured, natural language is unstructured.

Bridging the two — e.g., connecting sentences from documents to KG entities — requires complex **entity linking** and **semantic parsing**.

📉 Impact

→ Hard for LLMs or retrieval systems to directly "read" from a KG.

→ Limits the use of KGs in conversational AI unless hybridized (e.g., RAG with KGs).

🧩 10. Bias and Representation Limitations

💬 Explanation

KGs inherit biases from their data sources (like Wikipedia or news).

They might:

- Overrepresent Western knowledge
- Underrepresent minority groups or regions
- Include biased or outdated information

📉 Impact

→ Leads to **biased AI systems** that reproduce these imbalances.

→ Reduces fairness and reliability in reasoning.

🧾 11. Summary Table — Limitations of Knowledge Graphs

Category	Limitation	Example / Effect
Data Coverage	Incompleteness	Missing facts, low recall
Performance	Scalability	Too large to maintain or reason efficiently
Quality	Noisy or conflicting data	Duplicate or incorrect facts
Expressiveness	Limited to triples	Hard to model complex or temporal facts
Construction	Hard to automate	Requires accurate extraction and linking
Reasoning	Weak inference	Cannot derive new facts automatically
Flexibility	No uncertainty or context	All facts are “hard” true/false
Integration	Hard to connect with text	Complex mapping between natural language and KG
Fairness	Bias in knowledge	Overrepresentation or missing groups

💡 12. In Summary

> Knowledge Graphs are powerful for structured reasoning,
>
> but **rigid, incomplete, and hard to scale** in dynamic, real-world settings.

They need:

- Better automatic construction tools
- Contextual and temporal extensions
- Integration with **neural models** (like LLMs) for flexible reasoning

This is why **modern GenAI systems** often combine KGs with **vector retrieval or language models** →

to get the **best of both worlds** (structure + meaning).

🧭 Need for Specialized Knowledge Graphs

1️⃣ What are "Specialized Knowledge Graphs"?

Definition:

Specialized Knowledge Graphs (SKGs) are **domain-specific** or **task-focused** graphs designed to capture **knowledge within a particular field**, instead of representing the entire world (like Wikidata or DBpedia).

They encode **entities, relationships, and facts** relevant to a **specific area of expertise**, such as:

- **Biomedical KGs** – e.g., diseases, genes, drugs, symptoms
- **Financial KGs** – e.g., companies, assets, transactions, markets
- **Legal KGs** – e.g., cases, precedents, laws, judgments
- **Industrial KGs** – e.g., sensors, machines, process events

---

2️⃣ Why Do We Need Specialized KGs?

🧠 2.1. Complexity and Context

General-purpose KGs (like Wikidata) capture **broad** knowledge but **lack depth**.

In contrast, specialized domains (medicine, law, science) require **rich semantics**, **precise relations**, and **context-aware reasoning**.

Example:

A biomedical KG needs to know:

> (DrugA, inhibits, ProteinB)
>
> (GeneC, *expresses*, ProteinB)

Such relations are too fine-grained for generic graphs.

📊 2.2. High Domain Precision

Specialized domains require:

- **Strict vocabularies** (ontology alignment)
- **Precise terminology** (e.g., ICD codes, chemical IDs)
- **Verified sources** (scientific literature, databases)

Reason:

General KGs often contain ambiguous or noisy data, which is unacceptable in domains like medicine or finance.

🧩 2.3. Efficient Retrieval and Reasoning

Domain KGs can be optimized for **specific query types** and **reasoning patterns**, such as:

- “Which drugs target this protein?”
- “Which companies are linked to this regulation?”

Because the domain scope is narrow, specialized KGs allow:

- Faster traversal and inference
- Better query accuracy

- Easier integration with domain-specific models

🔍 2.4. Handling Complex Relationships

Many fields involve **n-ary or hierarchical relations** (more than two entities per fact).

For instance, in healthcare:

> (DrugA, treats, DiseaseB, underCondition, AgeGroup>60)

General KGs can't represent this efficiently, but specialized KGs can design **custom schemas** or **event-based nodes** to handle such facts.

📚 2.5. Integration with Domain Data Sources

Each field already has rich **structured and semi-structured data**, like:

- Electronic Health Records (EHRs)
- Scientific papers (PubMed)
- Legal judgments
- Sensor logs

A specialized KG can directly link to these sources, forming a **semantic layer** that connects structured data with raw content.

⚙️ 2.6. Improved Model Training for Domain LMs

When combined with **domain-specific language models (LLMs)**, SKGs:

- Provide **explicit, structured knowledge**
- Reduce **hallucination** in responses
- Enable **fact-grounded generation**

Example:

In Biomedical GenAI → BioGPT + BioKG improves factual correctness of answers.

🧱 2.7. Explainability and Trust

Specialized KGs provide **transparent reasoning chains**, which are critical in:

- Healthcare (why was this diagnosis suggested?)
- Finance (why was this company flagged?)
- Law (which precedents led to this judgment?)

Because SKGs store explicit relations, they make AI systems more **interpretable and auditable**.

🌐 2.8. Dynamic Knowledge and Continuous Updates

In specialized fields, new information is produced rapidly:

- New drugs are discovered weekly
- Financial markets change daily
- New regulations are introduced frequently

SKGs allow **focused, periodic updates** rather than re-indexing all global knowledge — improving maintainability.

3️⃣ Key Advantages Over General KGs

Aspect	General KGs	Specialized KGs
Scope	Broad (general world knowledge)	Narrow (focused on one domain)
Depth	Shallow, limited context	Deep, with rich semantics
Accuracy	Mixed quality	High precision, curated data
Schema	Fixed or simple triple-based	Customized schema & ontologies
Reasoning	Generic, limited depth	Domain-optimized reasoning
Applications	Search, general QA	Expert systems, domain LLMs, diagnostics

---

4️⃣ Examples of Specialized Knowledge Graphs

Domain	Example SKG	Purpose
Biomedical	BioKG, Hetionet, OpenBioLink	Drug–disease–gene interactions
Finance	Refinitiv KG, BloombergKG	Entity linking, fraud detection
Legal	OpenLegalData KG	Case law relationships, precedent retrieval
Education	ConceptNet (partially domain-focused)	Linking educational concepts
Industrial/IoT	Siemens MindSphere KG	Asset tracking, predictive maintenance

5️⃣ Summary

> Specialized Knowledge Graphs are domain-tailored extensions of general KGs.
>
> They are essential when **accuracy, context, reasoning, and interpretability** are critical.

In short:

They bridge the gap between **structured symbolic reasoning** and **domain-specific generative AI** —making them foundational for **RAG systems, scientific discovery, and trustworthy AI** in specialized fields.

🧮 Knowledge Graphs as a 3-Order Tensor

1️⃣ Recap — What is a Knowledge Graph?

A **Knowledge Graph (KG)** represents information as **triples** of the form:

(h, r, t)

where:

- **h** = head entity (subject)
- **r** = relation (predicate)
- **t** = tail entity (object)

Example:

> (“Paris”, isCapitalOf, “France”)

Each triple indicates a **fact** connecting two entities via a relation.

2️⃣ Tensor Representation — The Core Idea

A **tensor** is a multidimensional array — a generalization of matrices:

Object	Dimension	Example
Scalar	0D	(a)
Vector	1D	$([a_1, a_2, a_3])$
Matrix	2D	(A_{ij})
Tensor	**3D or higher**	(X_{ijk})

A **Knowledge Graph** can be seen as a **3-dimensional (3rd-order) tensor**, where the **three axes** correspond to:

1. **Head entities**
2. **Relations**
3. **Tail entities**

3️⃣ Defining the Tensor

Let’s define:

- Number of entities → **n**
- Number of relations → **m**

Then, the KG can be represented as a **binary tensor**:

$\mathcal{X} \in 0, 1^{n \times m \times n}$

where each entry:

$$\mathcal{X}_{h,r,t} = \begin{cases} 1, & \text{if the triple } (h, r, t) \text{ exists in the KG} \\ 0, & \text{otherwise} \end{cases}$$

So, if the fact “Paris isCapitalOf France” is true,

$\mathcal{X}_{\text{Paris,isCapitalOf,France}} = 1$

and for unrelated triples, the value is 0.

---

4 Intuition — The KG as a 3D Cube

Imagine a **3D cube (tensor)**:

- The **x-axis** represents **head entities (h)**
- The **y-axis** represents **relations (r)**
- The **z-axis** represents **tail entities (t)**

Each small cube (cell) in this 3D space corresponds to one possible triple ((h, r, t)).

- **Value = 1** → triple exists (true fact)
- **Value = 0** → triple doesn’t exist (unknown or false)

Visually:

Each “slice” along one relation (r) gives an **adjacency matrix** between entities for that relation.

---

5 Relation-wise Adjacency Matrices

For each relation (r_k), we can take a **2D slice** of the tensor:

$X_{:r_k:} \in 0, 1^{n \times n}$

This matrix shows how all entities are connected via that specific relation (r_k).

Example (for *isCapitalOf* relation):

	France	Italy	Germany
Paris	1	0	0
Rome	0	1	0
Berlin	0	0	1

Each "1" indicates that the corresponding city–country pair is connected by *isCapitalOf*.

So the full KG tensor is a **stack** of such adjacency matrices — one per relation.

6️⃣ Motivation — Why Represent KGs as a Tensor?

Because this view allows us to use **tensor factorization techniques** to learn **embeddings** (vector representations) of entities and relations.

Goal:

- Approximate the true tensor ($\mathcal{X}$) (which is mostly sparse)
- Learn dense, low-dimensional representations (h, r, t)
- Such that:

  $f(h, r, t) \approx \mathcal{X}_{h,r,t}$

  where (f) is a scoring function (model-specific).

7️⃣ Connection to Knowledge Graph Embeddings

When we **factorize the KG tensor**, we get embeddings such that:

$\mathcal{X}_{h,r,t} \approx f(\mathbf{h}, \mathbf{r}, \mathbf{t})$

where:

- ($\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$) are **entity embeddings**
- $\mathbf{r} \in \mathbb{R}^d$ $(or (\mathbb{R}^{d \times d}))$ is a **relation embedding**
- ($f(\cdot)$) is the **scoring function** specific to the embedding model (e.g., TransE, DistMult, ComplEx, etc.)

So:

- **Tensor decomposition → Embedding learning**
- **Tensor reconstruction → Knowledge completion (predicting missing triples)**

8️⃣ Advantages of Tensor Representation

Aspect	Benefit
Mathematical structure	Enables linear algebraic modeling and optimization
Compact representation	Turns symbolic triples into numerical data
Link prediction	Missing entries can be inferred via low-rank approximation
Compatibility	Works seamlessly with matrix/tensor factorization frameworks
Foundation	Forms the basis for all modern embedding models (TransE, DistMult, etc.)

9️⃣ Summary

> Representing a Knowledge Graph as a 3-order tensor gives a mathematical and computational framework for encoding, decomposing, and completing knowledge.

Key Takeaway:

$$\text{Knowledge Graph} = \text{3D Tensor}\ (\text{Entities} \times \text{Relations} \times \text{Entities})$$

This tensor view bridges:

- **Symbolic logic** (triples) ↔️ , **Continuous learning** (embeddings)

  and underlies nearly all **Knowledge Graph Embedding (KGE)** models.

🧮 Factoring the Knowledge Graph Tensor

1️⃣ Recap — The Knowledge Graph as a Tensor

A Knowledge Graph can be represented as a **3rd-order tensor**

$\mathcal{X} \in 0, 1^{n_e \times n_r \times n_e}$

where:

- (n_e): number of entities
- (n_r): number of relations
- ($\mathcal{X}_{h,r,t} = 1$) if the triple ((h, r, t)) exists, otherwise 0.

This tensor is **sparse** (most entries are 0) because only a small fraction of possible triples are true.

2️⃣ Motivation — Why Factor the Tensor?

The goal of **factoring (decomposing)** the tensor is to:

- **Find low-dimensional embeddings** for entities and relations
- **Predict missing links** (i.e., fill in unknown or missing tensor entries)
- **Compress and generalize** the knowledge in the graph

In other words, we approximate:

$\mathcal{X}_{h,r,t} \approx f(\mathbf{h}, \mathbf{r}, \mathbf{t})$

where:

- ($\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$): entity embeddings
- ($\mathbf{r}$): relation embedding
- (f): a *scoring function* that estimates how likely the triple is true.

3️⃣ Concept — Tensor Factorization

Tensor factorization is analogous to **matrix factorization**, but for 3D data.

- In matrix factorization:

  $A \approx U \Sigma V^\top$

  where (A) is an (m $\times$ n) matrix.

- In tensor factorization:

$$\mathcal{X} \approx \text{Combination of smaller latent components}$$

The goal is to express the big, sparse tensor ($\mathcal{X}$) as the interaction of **low-rank latent factors** representing entities and relations.

4️⃣ Main Factorization Approaches

There are several classical tensor factorization models used for Knowledge Graphs.

Here's how they differ in how they *define* the function ($f(\mathbf{h}, \mathbf{r}, \mathbf{t})$):

🧩 (a) RESCAL (2011) — Bilinear Tensor Factorization

RESCAL treats each relation as a **full matrix** ($R_r \in \mathbb{R}^{d \times d}$)

and each entity as a vector ($\mathbf{e}_i \in \mathbb{R}^d$).

Scoring function:

$$f(h, r, t) = \mathbf{h}^\top R_r \mathbf{t}$$

Interpretation:

- The tensor ($\mathcal{X}$) is approximated as:

  $$\mathcal{X} \approx E \times R \times E^\top$$

  where (E) stacks all entity embeddings.

Characteristics:

- Very expressive (captures all pairwise interactions).
- High computational cost (each (R_r) is (d $\times$ d)).

🧩 (b) DistMult (2015) — Simplified Bilinear Form

Simplifies RESCAL by using a **diagonal relation matrix**.

$$f(h, r, t) = \mathbf{h}^\top \text{diag}(\mathbf{r}) \mathbf{t}$$

or equivalently:

$$f(h, r, t) = \sum_i h_i, r_i, t_i$$

Characteristics:

- Very efficient (only element-wise products).
- Cannot model asymmetric relations (since (f(h,r,t) = f(t,r,h))).

🧩 (c) ComplEx (2016) — Complex-valued Factorization

Extends DistMult into the **complex space**.

Entities and relations are complex vectors: ($\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^d$).

Scoring function:

$$f(h, r, t) = \mathrm{Re}(\mathbf{h}^\top \mathrm{diag}(\mathbf{r}) \overline{\mathbf{t}})$$

where ($\overline{\mathbf{t}}$) is the complex conjugate.

Characteristics:

- Handles asymmetric relations naturally.
- Maintains efficiency similar to DistMult.

🧩 (d) TuckER (2019) — Tucker Decomposition

Based on **Tucker tensor decomposition**, where each mode (entity head, relation, entity tail) is projected via a shared **core tensor** ($\mathcal{W}$).

Scoring function:

$$f(h, r, t) = \mathcal{W} \times_1 \mathbf{h} \times_2 \mathbf{r} \times_3 \mathbf{t}$$

where ($\times_n$) denotes mode-n tensor multiplication.

Characteristics:

- Very expressive (generalizes several earlier models).
- Can share parameters effectively across relations.

5️⃣ Training Objective

The goal of training is to adjust the embeddings so that **true triples score higher** than **false ones**.

Common loss functions:

- **Margin ranking loss**:
  $L = \sum_{(h,r,t)\in \text{Pos}} \sum_{(h',r,t')\in \text{Neg}} \max(0, \gamma + f(h', r, t') - f(h, r, t))$
- **Binary cross-entropy loss** (for probabilistic models):
  $L = -y \log \sigma(f(h, r, t)) - (1 - y) \log(1 - \sigma(f(h, r, t)))$
  where (y=1) for true triples and (0) for negatives.

---

6 Interpreting the Factorization

Factoring the tensor means:

- **Each entity** is represented by a **vector of latent features**.
- **Each relation** defines how those features interact.
- **The score function** tells how compatible a triple is according to these features.

In practice, the factorized model can **reconstruct known triples** and **predict new ones** — i.e., perform **link prediction** or **knowledge graph completion**.

---

7 Summary Table

Model	Mathematical Form	Key Property
RESCAL	$(f(h, r, t) = h^\top R_r t)$	Full relation matrices; expressive but heavy
DistMult	$(f(h, r, t) = \sum_i h_i r_i t_i)$	Efficient; assumes symmetry
ComplEx	$(f(h, r, t) = \text{Re}(h^\top \text{diag}(r) \bar{t}))$	Models asymmetry using complex space
TuckER	$(f(h, r, t) = \mathcal{W} \times_1 h \times_2 r \times_3 t)$	General, parameter-efficient decomposition

---

8 Key Takeaways

- **Factoring the KG tensor** = approximating a huge sparse tensor by low-rank latent representations.
- This factorization gives us **embeddings** for entities and relations.
- The embeddings allow:
  - Predicting missing facts
  - Clustering and reasoning
  - Integration with neural or generative models

> In short:
>
> **Tensor factorization transforms symbolic triples into continuous vector spaces**,
>
> making Knowledge Graphs usable for machine learning and GenAI systems.

🧠 Graph Neural Networks (GNNs)

1. Motivation: Why GNNs?

Traditional neural networks (like CNNs and RNNs) are designed for **structured data**:

- CNNs → Grid-like data (images)
- RNNs → Sequential data (text, time series)

But **many real-world datasets** are not structured this way.

They are **graphs**, e.g.:

- Social networks → Users connected by friendships
- Knowledge graphs → Entities connected by relations
- Molecules → Atoms connected by bonds

👉 Graphs have **nodes (entities)** and **edges (relationships)** — irregular, unordered structures.

GNNs are designed to **learn representations (embeddings)** for nodes, edges, or entire graphs.

---

2. What is a Graph?

A graph is defined as:

G = (V, E)

Where:

- (V) = Set of **nodes/vertices**
- (E) = Set of **edges** connecting pairs of nodes

Each node (v_i) may have **features** (x_i), and edges may also have attributes.

---

3. Core Idea of GNNs

The goal of GNNs is to **learn a representation (embedding)** for each node (and optionally for the whole graph) by **aggregating information from its neighbors**.

Key Intuition:

> Each node learns about its context (neighbors) by passing and receiving messages.

This is known as **Message Passing** or **Neighborhood Aggregation**.

---

4. Message Passing Framework

At each layer (l), for every node (v):

1. **Message Aggregation:**

   Collect messages from neighboring nodes:

   $$m_v^{(l)} = \text{AGGREGATE}\left(h_u^{(l-1)} : u \in \mathcal{N}(v)\right)$$

   where ($\mathcal{N}(v)$) is the set of neighbors of (v).

2. **Update Step:**

Combine the node's previous representation with aggregated messages:

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

After several layers, each node embedding ($h_v^{(L)}$) captures information from its multi-hop neighborhood.

---

5. Typical GNN Layer

One common implementation (Graph Convolutional Network — GCN):

$$h_v^{(l)} = \sigma\left(W \cdot \text{AGG}\left(h_v^{(l-1)} \cup h_u^{(l-1)}, \forall u \in \mathcal{N}(v)\right)\right)$$

Where:

- (W): Learnable weight matrix
- (σ): Activation function (like ReLU)
- AGG can be **mean**, **sum**, or **max pooling**

---

6. Graph-Level vs Node-Level Tasks

Task Type	Goal	Example
Node Classification	Predict label for each node	Predict user interests in a social graph
Link Prediction	Predict whether an edge exists between nodes	Knowledge graph completion
Graph Classification	Predict label for the entire graph	Classify molecules as toxic/non-toxic
Community Detection	Cluster nodes based on learned embeddings	Detect groups of friends in social networks

---

7. Popular GNN Architectures

Model	Key Idea	Main Feature
GCN (Graph Convolutional Network)	Extends convolution to graphs	Simple aggregation using normalized adjacency

Model	Key Idea	Main Feature
GraphSAGE	Learns aggregation functions (mean, LSTM, pooling)	Efficient for large graphs
GAT (Graph Attention Network)	Uses attention weights for neighbors	Learns importance of neighbors
GIN (Graph Isomorphism Network)	Powerful architecture matching graph isomorphism test	Strong discriminative power

8. Advantages of GNNs

- Handles **irregular graph structures** naturally
- Learns **context-aware embeddings** (nodes know their neighborhood)
- Works well for **relational data** (like Knowledge Graphs, molecules)
- Supports **inductive learning** — generalizing to unseen nodes/graphs

9. Challenges and Limitations

- **Over-smoothing:** Too many layers cause all nodes to have similar embeddings
- **Scalability:** Message passing can be expensive for large graphs
- **Dynamic Graphs:** Harder to handle when the structure changes over time
- **Interpretability:** Hard to understand learned relationships

10. Summary

Concept	Meaning
Goal	Learn node or graph representations from connectivity + features
Mechanism	Message passing and aggregation
Applications	Social networks, molecules, recommender systems, knowledge graphs
Key Models	GCN, GraphSAGE, GAT, GIN

Concept	Meaning
Limitations	Over-smoothing, scalability, interpretability

Excellent — this is a perfect continuation after GNNs.

Let's now move into one of the **most advanced architectures for graphs** — where the power of **Transformers** meets the **structure of Graphs**.

🧠 Graph Transformers

(Detailed conceptual lecture notes — simplified and structured)

1. Motivation: Why Graph Transformers?

Graph Neural Networks (GNNs) are great at **local aggregation** — each node gathers information from its **neighbors** using message passing.

However, they have **limitations**:

- **Locality bias:** A node can only access distant information after multiple layers.
- **Over-smoothing:** Too many layers make all node embeddings similar.
- **Difficulty with long-range dependencies:** Nodes far apart in the graph influence each other weakly.

👉 Transformers, on the other hand, excel at modeling **global relationships** via **self-attention**, where every token (or node) can directly attend to every other.

Thus, **Graph Transformers = Graph structure + Transformer attention mechanism**,

combining **local structure** and **global reasoning**.

2. Recap: Transformer Core Idea

Transformers use **self-attention** to learn dependencies between tokens:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where:

- (Q), (K), (V): Query, Key, Value matrices
- Each token attends to others with learned weights.

The challenge:

In graphs, nodes are **not sequential** and **have no fixed order** — so we need to **adapt self-attention** to graph structures.

---

3. Core Idea of Graph Transformers

🎯 Goal:

Enable **global attention** among all nodes while still **preserving graph structure** (edges, neighborhoods).

Each node computes attention not just based on node features, but also on:

- Its **connectivity** (edges)
- **Positional or structural encodings** (e.g., node distance, degree)

$$\text{Attention}(i,j) = \text{softmax}\left(\frac{(Q_i K_j^T + E_{ij})}{\sqrt{d_k}}\right)$$

Here, (E_{ij}) encodes the **edge or positional bias** between node (i) and node (j).

---

4. Architecture Overview

A **Graph Transformer layer** consists of:

1. **Input Embeddings:**

   Each node has a feature vector (x_i)

   and optionally an edge embedding (e_{ij}).

2. **Positional / Structural Encoding:**

   Unlike text, graphs have no order — so we use:

   - Shortest-path distance
   - Laplacian eigenvectors

- Node degrees
- Random walk encodings to provide position-like context.

3. **Self-Attention Layer:**

   Nodes attend to all (or nearby) nodes using:

$$\alpha_{ij} = \text{softmax}\left(\frac{Q_i K_j^T + \phi(E_{ij})}{\sqrt{d}}\right)$$

   where ($\phi(E_{ij})$) encodes edge features.

4. **Feed-Forward Layer:**

   Standard MLP applied to each node's attended representation.

5. **Residual & Normalization:**

   Same as in Transformers — for stability and gradient flow.

---

5. Variants of Graph Transformers

Model	Key Idea	Notable Feature
Graphormer (Microsoft, 2021)	Adds **spatial encoding** (shortest path distance) and **edge encoding**	State-of-the-art for molecule graphs
SAN (Structure-Aware Transformer)	Adds **structural bias** using adjacency matrix	Keeps global attention but aware of local connections
Graph-BERT	Replaces message passing entirely with transformer layers	Uses Laplacian positional embeddings
TokenGT	Treats **edges as tokens** too	Enables edge-level reasoning
GPS (Graph Transformer with GNN layers)	Combines local GNN aggregation + global transformer attention	Best of both worlds

---

6. Comparison: GNN vs Graph Transformer

Aspect	GNN	Graph Transformer
Information Flow	Local (neighbors only)	Global (all nodes)
Positional Encoding	Implicit via adjacency	Explicit (Laplacian, distance, degree)
Scalability	Better on large sparse graphs	Costly ($O(N^2)$ attention)
Expressiveness	Local features, good for small graphs	Captures long-range and global dependencies
Best Used For	Node classification, link prediction	Molecular graphs, document graphs, scene graphs

7. Applications

- **Molecular property prediction** (Drug discovery)
- **Knowledge graph reasoning** (complex relation inference)
- **Program graph analysis** (code understanding)
- **Scene graph understanding** (vision + NLP tasks)
- **Recommendation systems** (modeling user-item graphs globally)

8. Limitations

Challenge	Reason
Scalability	Attention is $O(N^2)$ for N nodes
Positional encoding choice	No universal “best” for all graph types
Overfitting	Transformers have many parameters — need large datasets
Sparse graphs	Global attention might ignore sparsity and structure bias

9. Summary

Concept	Meaning
Goal	Combine graph structure with global attention mechanism
Key Mechanism	Self-attention with edge & positional encodings
Advantage	Models long-range dependencies and global structure
Popular Models	Graphormer, Graph-BERT, SAN, GPS
Main Limitation	Quadratic complexity and overfitting on small graphs