



# Graph Neural Networks for Ontology Relation Extraction

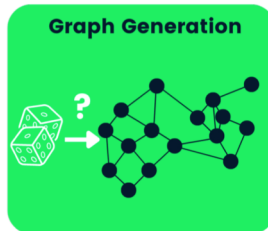
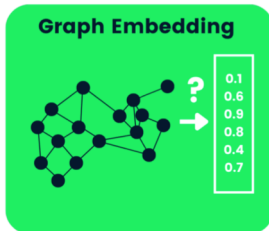
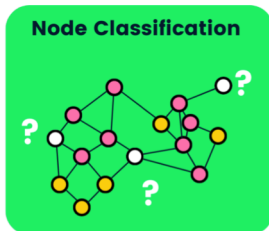
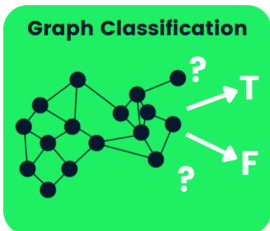
Renato Vuković

Dialogue Systems and Machine Learning

- Introduction to graph neural networks (GNNs)
- GNNs in natural language processing (NLP)
- Large language models (LLMs) and GNNs
- LLMs and knowledge graphs (KGs)
- Application of GNNs to ontology relation extraction

- GNNs capture information from **graph structures**
- They can be utilised on a variety of tasks that include graph structural information
  - Node-focussed and graph-focussed tasks, e.g. node or edge classification
  - Applications in information extraction, knowledge graph reasoning, syntactic parsing, etc.
- GNNs mostly rely on a **message-passing algorithm** where a node's embedding is based on neighbouring embeddings

# Graph Neural Networks (GNNs)



- **Graph Classification:** classify graphs into various categories.
- **Node Classification:** predict node labels based on neighbouring node labels
- **Link Prediction:** predict the link between a pair of nodes in a graph with an incomplete adjacency matrix
- **Community Detection:** divide nodes into various clusters based on edge structure.
- **Graph Embedding:** maps graphs into vectors, preserving the relevant information on nodes, edges, and graph structure.
- **Graph Generation:** generate a new but similar graph structure based on a sample graph distribution



- GNNs learn embeddings for each node in the graph and **aggregate the node embeddings** to produce the graph embeddings
- The learning process of node embeddings utilises graph structure and input node embeddings:  $\mathbf{h}_i^{(l)} = f_{\text{filter}}(\mathbf{A}, \mathbf{H}^{(l-1)})$
- Here:
  - $\mathbf{A} \in \mathbb{R}^{n \times n}$  is the adjacency matrix of the graph that can be binary or weighted
  - $\mathbf{H}^{(l-1)} = \{\mathbf{h}_1^{(l-1)}, \dots, \mathbf{h}_n^{(l-1)}\} \in \mathbb{R}^{n \times d}$  denotes the input node embeddings at the  $l - 1$ -th GNN layer and  $\mathbf{H}^{(l)}$  are the updated node embeddings,  $d$  is the dimension of  $\mathbf{h}_i^{(l-1)}$
  - $f_{\text{filter}}(\cdot, \cdot)$  is a graph filter function  $\rightarrow$  main difference in GNN methods
- The graph embeddings are tuned for inference on downstream tasks

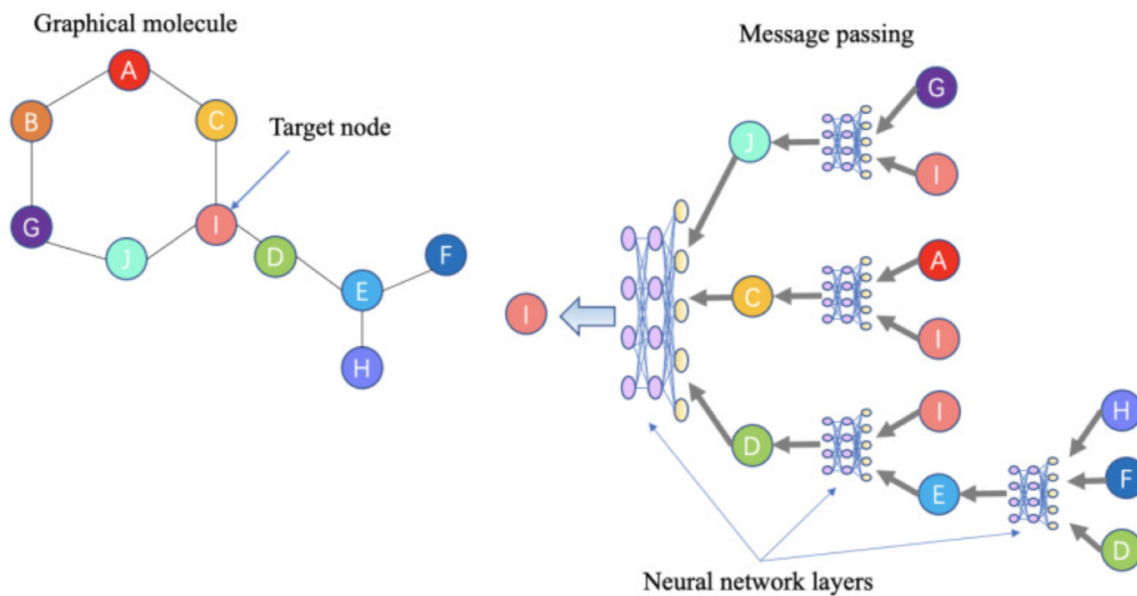
## ■ Spatial:

- Update the node embeddings via **message-passing** from spatially close nodes
- The embedding of a node is computed based on the embeddings of its  **$k$ -hop neighbouring nodes** in a message-passing neural network (MPNN; Gilmer et al., 2017)
- It is also possible to sample  $k$ -hop neighbours for each node to reduce the computational cost (GraphSage; Hamilton et al., 2017)

## ■ Attention:

- Originally GNNs do not dynamically adapt the importance of edges when computing node embeddings
- **Attention-based GNNs** such as the graph attention network (GAT; Veličković et al, 2018) assign weights to edges
- Important neighbouring nodes get higher attention scores during embedding computation and hence influence the final node embedding more

# Message-passing Neural Network (MPNN)



## ■ Spectral:

- Based on graph signal processing and spectral graph theory
- Graph Convolutional Network (GCN; Kipf and Welling, 2016) generalises a convolutional network to graphs
  - Apply **filters** to the multiplication of the eigenvector matrix of the graph and the node embeddings
- The filter captures information about the neighbourhood of a node based on the adjacency matrix

## ■ Recurrent:

- Compute the embedding at time step  $t$  based on prior embeddings
- Take into account the direction of nodes and the edge type
- Utilise a gated recurrent unit (GRU; Cho et al., 2014)

# Graph Convolutional Network (GCN)

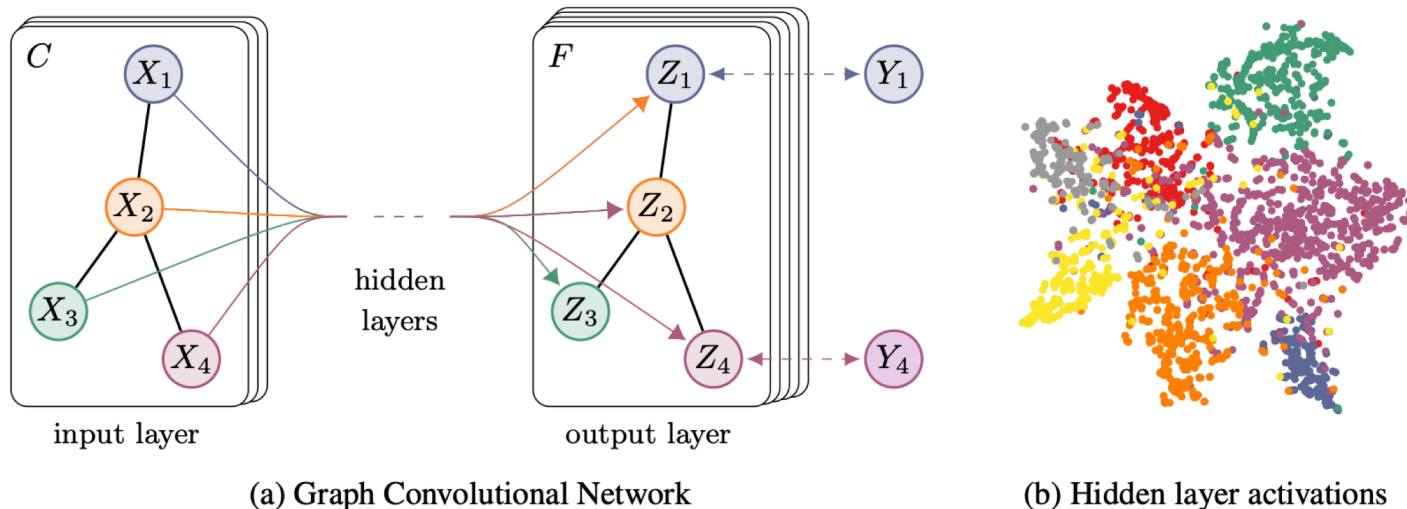


Figure 1: *Left*: Schematic depiction of multi-layer Graph Convolutional Network (GCN) for semi-supervised learning with  $C$  input channels and  $F$  feature maps in the output layer. The graph structure (edges shown as black lines) is shared over layers, labels are denoted by  $Y_i$ . *Right*: t-SNE (Maaten & Hinton, 2008) visualization of hidden layer activations of a two-layer GCN trained on the Cora dataset (Sen et al., 2008) using 5% of labels. Colors denote document class.

- The **message-passing** algorithm for computing node embeddings based in the  $l$ -th layer on their neighbourhood can be expressed as follows:
  - $h_i^{(l)} = \text{activation}(W \cdot (h_i^{(l-1)}, \text{AGGREGATE}(h_j^{(l-1)}, \forall j \in \mathcal{N}_i)))$
  - where the embedding of the  $i$ -th node is computed based on an activation function, weight and aggregation function to take into account neighbourhood information in  $\mathcal{N}_i$
- The aggregation function is the **main distinction** in different GNN architectures

- For the **Graph Attention Network** the aggregation function is the weighted sum over the first order neighbours of a node:
  - $h_i^{(l)} = \sigma(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \cdot W h_j^{(l-1)})$
  - where the attention weight  $\alpha_{ij}$  denotes the importance of node  $n_j$  for  $n_i$ , which is adapted during training

# Graph Attention Network (GAT)

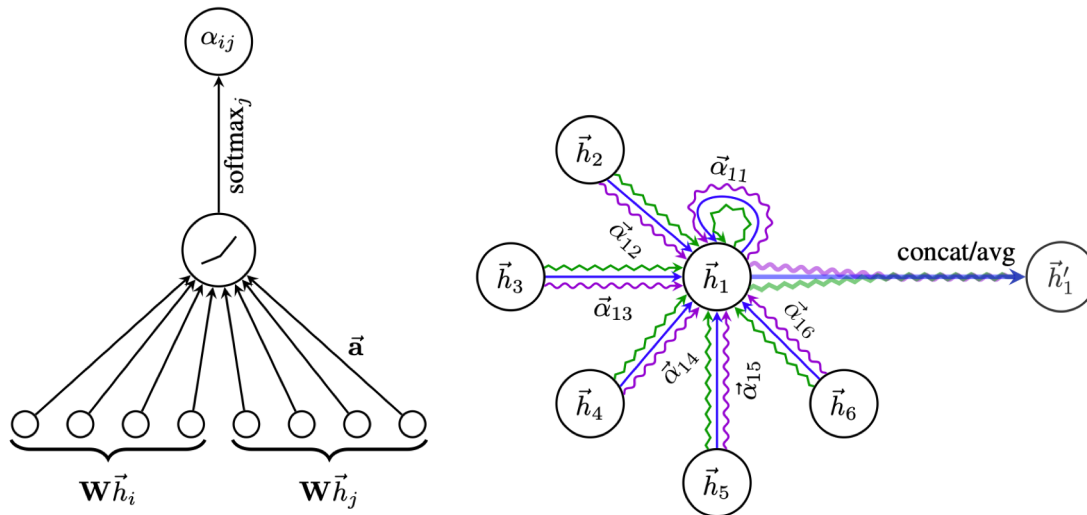


Figure 1: **Left:** The attention mechanism  $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$  employed by our model, parametrized by a weight vector  $\vec{a} \in \mathbb{R}^{2F'}$ , applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with  $K = 3$  heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain  $\vec{h}'_1$ .



# GNN Downstream Tuning

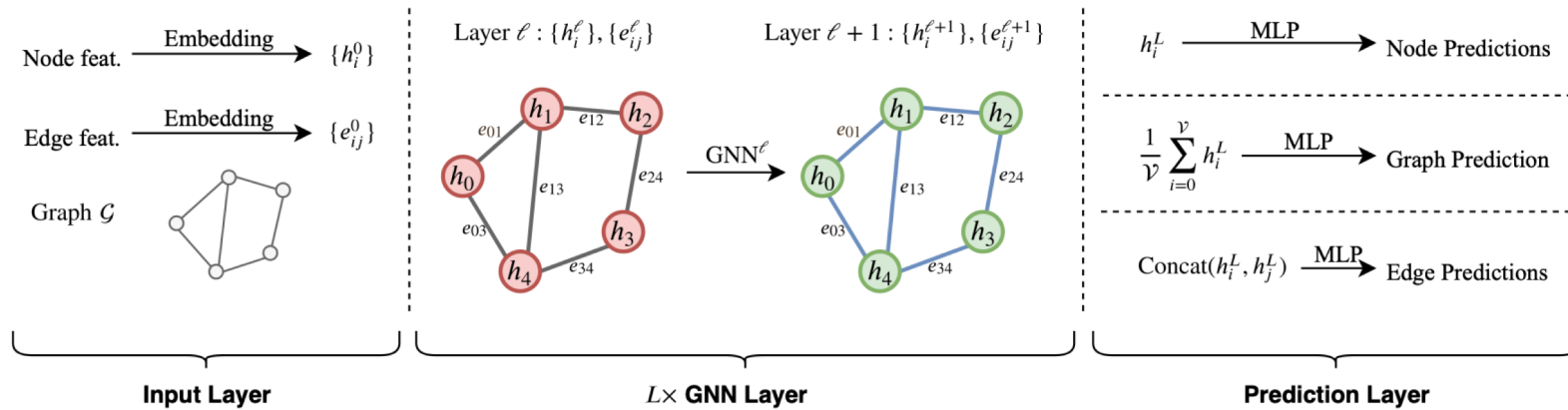


Figure 3: A standard experimental pipeline for GCNs, which embeds the graph node and edge features, performs several GNN layers to compute convolutional features, and finally makes a prediction through a task-specific MLP layer.

- Generate a **higher-level embedding** of nodes in a graph
- Reduce the number of nodes in a graph by aggregating different node embeddings
- The pooling operation is given by:  $A', \mathbf{H}' = f_{\text{pool}}(A, H)$ 
  - $A \in \mathbb{R}^{n \times n}$  and  $A' \in \mathbb{R}^{n' \times n'}$  are the adjacency matrices before and after graph pooling
  - $\mathbf{H} \in \mathbb{R}^{n \times d}$  and  $\mathbf{H}' \in \mathbb{R}^{n' \times d'}$  are the node embeddings before and after graph pooling
  - $n'$  is set to 1 in most cases to get **one embedding** for the entire graph

- Generate **graph-level representations** for graph-focused downstream tasks, e.g. graph classification
- Node embeddings are sufficient for node-focused tasks, however, for graph-focused tasks, a **representation of the entire graph** is required.
- Pooling summarises the node embedding information and the graph structure information
- Examples for graph pooling layers:
  - **Flat graph pooling:**
    - Use a fully connected layer on the node embeddings and then do max or average pooling
  - **Hierarchical graph pooling:**
    - Aggregate the node embeddings step by step to learn the graph-level embedding
    - Sub-sample the most important nodes or combine nodes to form supernodes until a final graph-representation is reached

## ■ Static graph construction:

- Construct the graph structures by leveraging existing relation parsing tools (e.g., dependency parsing) or manually defined rules/annotation → **adjacency matrix**
- The adjacency matrix **augments the raw text** with rich structured information.
- E.g. *knowledge graphs* where a graph consists of entities as nodes and relations as edges  $\mathcal{G} = (V, E)$  with relational triplets (entity<sub>1</sub>, relation, entity<sub>2</sub>)

## ■ Dynamic graph construction:

- Do graph **construction and representation** learning **jointly** instead of relying on an annotation or previously predicted information
- Use a graph similarity metric learning component for **learning a weighted adjacency matrix** by considering pair-wise node similarity in the embedding space
- A graph **sparsification component** is used for extracting a sparse graph from the learned fully-connected graph
  - e.g. keeping only the highest weight edges by applying an activation function

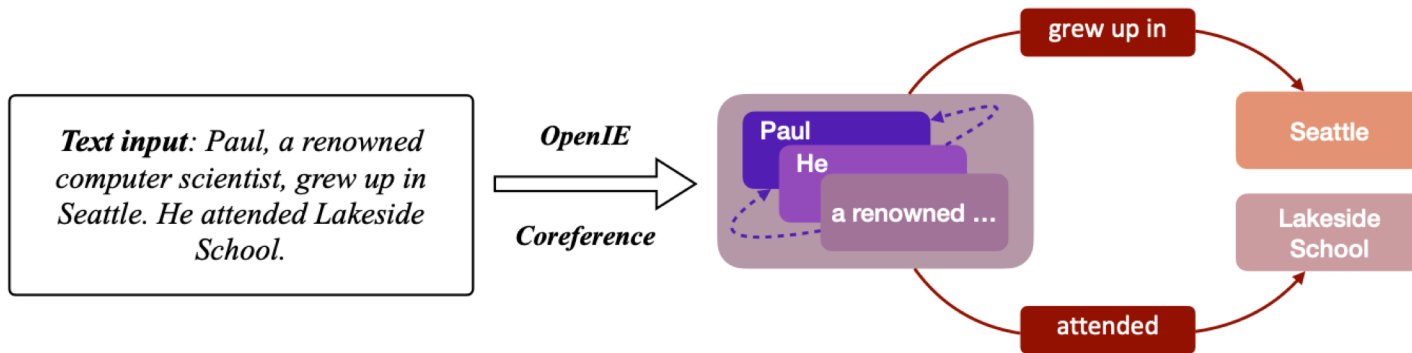


Figure 4: An example for IE graph construction which contains both the Co-reference process and the Open Information Extraction process.

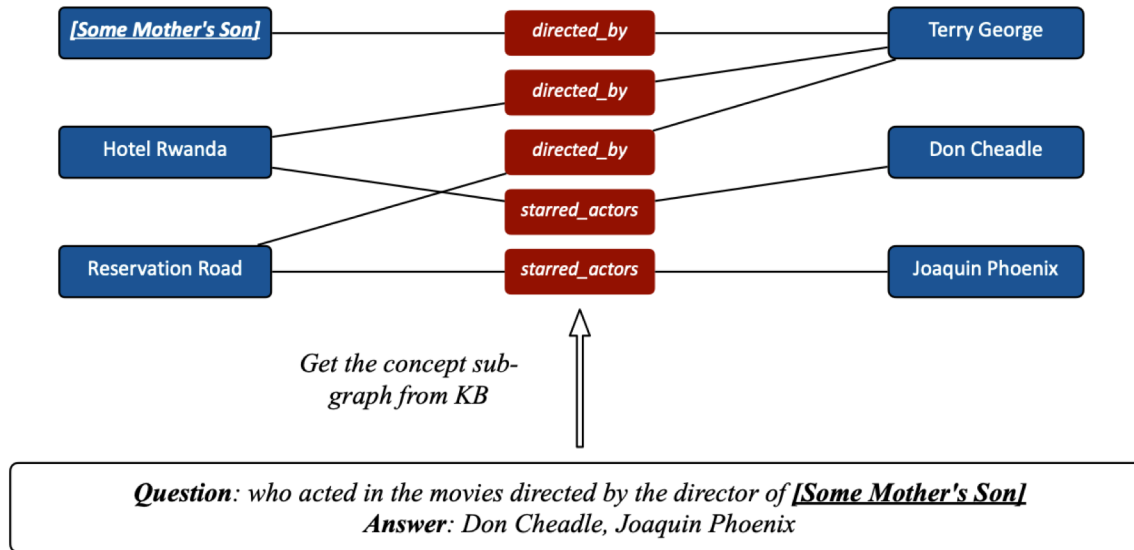


Figure 5: An example for knowledge graph construction, where the knowledge base (KB) used and the generated concept graph are both from the dataset *MetaQA* (Zhang et al., 2018a).

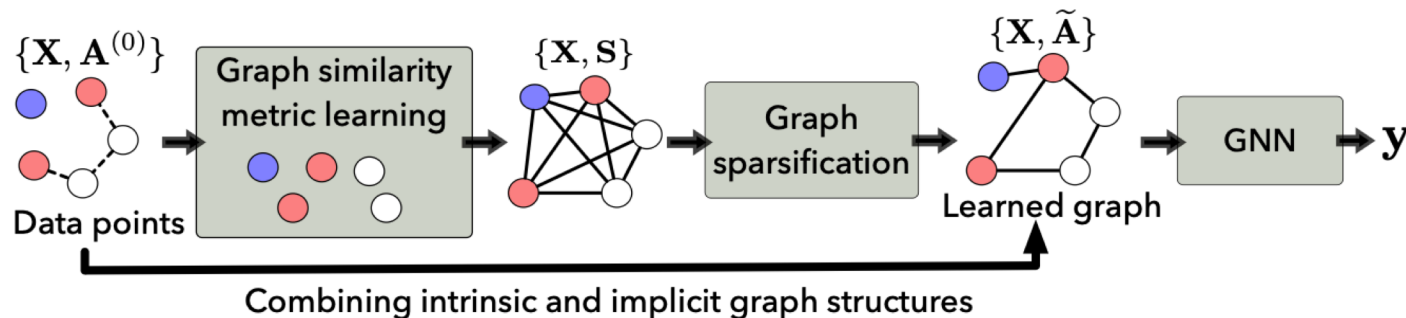


Figure 10: Overall illustration of dynamic graph construction approaches. Dashed lines (in data points on left) indicate the optional intrinsic graph topology.

- Transformers can be considered GNNs, which operate on a **fully connected dynamic graph** constructed by employing the self-attention mechanism
- Although Transformers can be applied to natural text easily and build the graph in the background they cannot be applied to more complex data such as graphs directly
- GNNs can only be applied to **pre-structured graph data**
  - self-supervised pre-training, one of the main advantages of transformers, is hence impossible



- *Graph transformers* (Yun et al., 2019) adapt a structure-aware self-attention mechanism to combine the advantages of GNNs and transformers
- Rely on attention to utilise the original **graph topology information**
  - Possibly not the best way to explore the original graph input information, especially when graph inputs are multi-relational and heterogeneous graphs.
- learn a soft selection of edge types and composite relations for generating useful multi-hop connections, called *meta-paths*

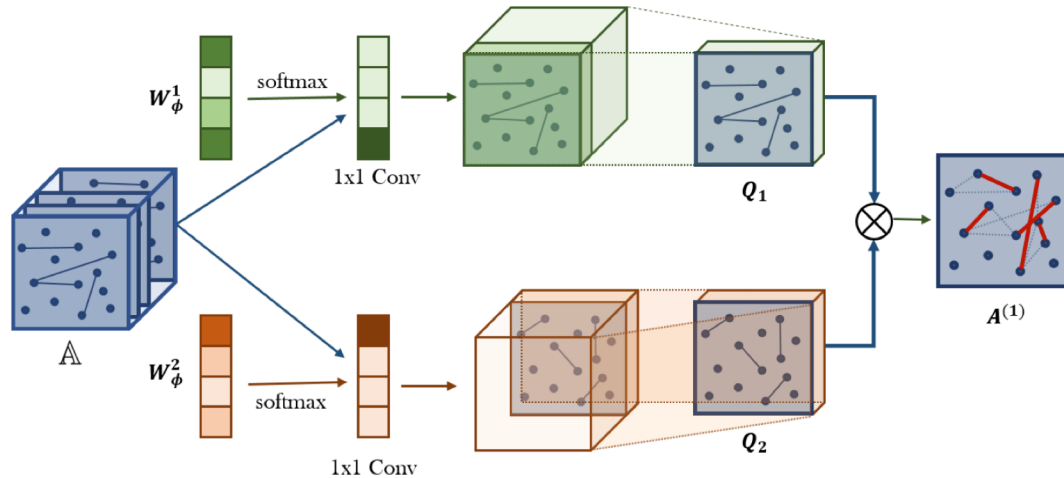


Figure 1: **Graph Transformer Layer** softly selects adjacency matrices (edge types) from the set of adjacency matrices  $\mathbb{A}$  of a heterogeneous graph  $G$  and learns a new meta-path graph represented by  $A^{(1)}$  via the matrix multiplication of two selected adjacency matrices  $Q_1$  and  $Q_2$ . The soft adjacency matrix selection is a weighted sum of candidate adjacency matrices obtained by  $1 \times 1$  convolution with non-negative weights from  $\text{softmax}(W_\phi^1)$ .

# Graph Transformer Networks (GTN)

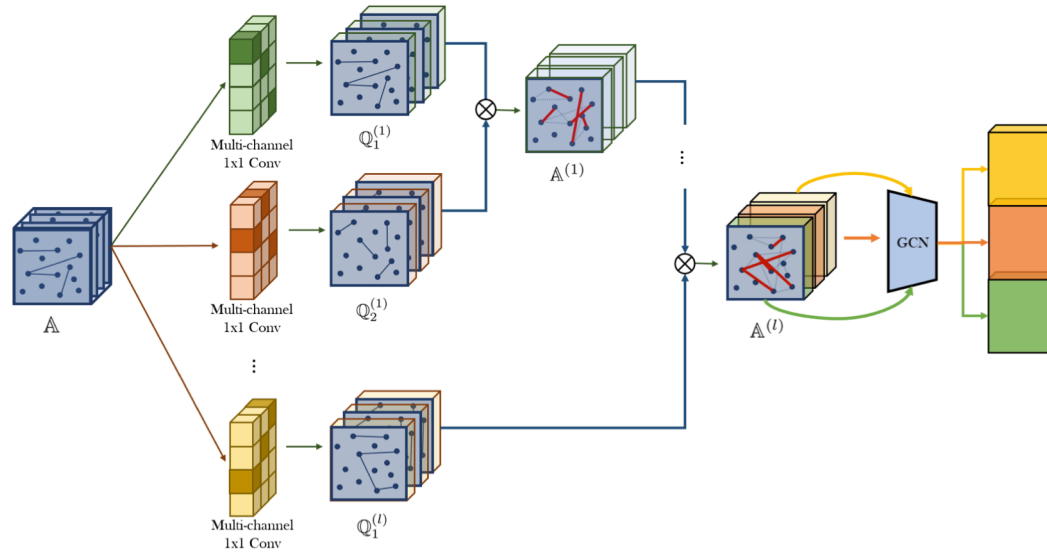


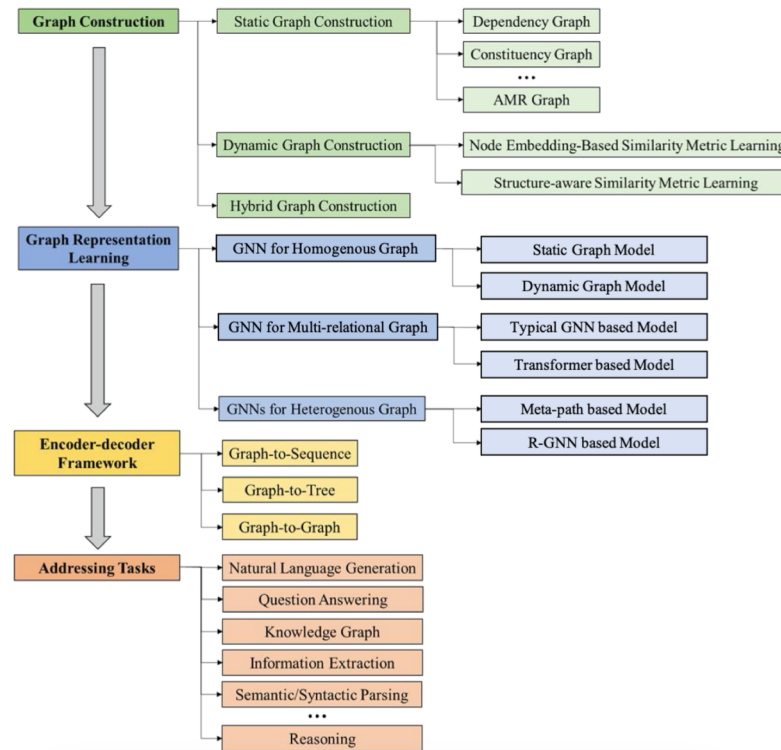
Figure 2: Graph Transformer Networks (GTNs) learn to generate a set of new meta-path adjacency matrices  $\mathbb{A}^{(l)}$  using GT layers and perform graph convolution as in GCNs on the new graph structures. Multiple node representations from the same GCNs on multiple meta-path graphs are integrated by concatenation and improve the performance of node classification.  $Q_1^{(l)}$  and  $Q_2^{(l)} \in \mathbf{R}^{N \times N \times C}$  are intermediate adjacency tensors to compute meta-paths at the  $l$ th layer.

# GNN Time Complexity

Approach	Category	Inputs	Pooling	Readout	Time Complexity
GNN* (2009) [15]	RecGNN	$A, X, X^e$	-	a dummy super node	$O(m)$
GraphESN (2010) [16]	RecGNN	$A, X$	-	mean	$O(m)$
GGNN (2015) [17]	RecGNN	$A, X$	-	attention sum	$O(m)$
SSE (2018) [18]	RecGNN	$A, X$	-	-	-
Spectral CNN (2014) [19]	Spectral-based ConvGNN	$A, X$	spectral clustering+max pooling	max	$O(n^3)$
Henaff et al. (2015) [20]	Spectral-based ConvGNN	$A, X$	spectral clustering+max pooling	-	$O(n^3)$
ChebNet (2016) [21]	Spectral-based ConvGNN	$A, X$	efficient pooling	sum	$O(m)$
GCN (2017) [22]	Spectral-based ConvGNN	$A, X$	-	-	$O(m)$
CayleyNet (2017) [23]	Spectral-based ConvGNN	$A, X$	mean/graculus pooling	-	$O(m)$
AGCN (2018) [40]	Spectral-based ConvGNN	$A, X$	max pooling	sum	$O(n^2)$
DualGCN (2018) [41]	Spectral-based ConvGNN	$A, X$	-	-	$O(m)$
NN4G (2009) [24]	Spatial-based ConvGNN	$A, X$	-	sum/mean	$O(m)$
DCNN (2016) [25]	Spatial-based ConvGNN	$A, X$	-	mean	$O(n^2)$
PATCHY-SAN (2016) [26]	Spatial-based ConvGNN	$A, X, X^e$	-	sum	-
MPNN (2017) [27]	Spatial-based ConvGNN	$A, X, X^e$	-	attention sum/set2set	$O(m)$
GraphSage (2017) [42]	Spatial-based ConvGNN	$A, X$	-	-	-
GAT (2017) [43]	Spatial-based ConvGNN	$A, X$	-	-	$O(m)$
MoNet (2017) [44]	Spatial-based ConvGNN	$A, X$	-	-	$O(m)$
LGCN (2018) [45]	Spatial-based ConvGNN	$A, X$	-	-	-
PGC-DGCNN (2018) [46]	Spatial-based ConvGNN	$A, X$	sort pooling	attention sum	$O(n^3)$
CGMM (2018) [47]	Spatial-based ConvGNN	$A, X, X^e$	-	sum	-
GAAN (2018) [48]	Spatial-based ConvGNN	$A, X$	-	-	$O(m)$
FastGCN (2018) [49]	Spatial-based ConvGNN	$A, X$	-	-	-
StoGCN (2018) [50]	Spatial-based ConvGNN	$A, X$	-	-	-
Huang et al. (2018) [51]	Spatial-based ConvGNN	$A, X$	-	-	-
DGCNN (2018) [52]	Spatial-based ConvGNN	$A, X$	sort pooling	-	$O(m)$
DiffPool (2018) [54]	Spatial-based ConvGNN	$A, X$	differential pooling	mean	$O(n^2)$
GeniePath (2019) [55]	Spatial-based ConvGNN	$A, X$	-	-	$O(m)$
DGI (2019) [56]	Spatial-based ConvGNN	$A, X$	-	-	$O(m)$
GIN (2019) [57]	Spatial-based ConvGNN	$A, X$	-	sum	$O(m)$
ClusterGCN (2019) [58]	Spatial-based ConvGNN	$A, X$	-	-	-

- Complexity is mostly linear wrt. the number of edges  $m$  in a graph
- For some architectures it is quadratic or cubic wrt. the number of nodes  $n$

# GNNs in NLP



- KGs capture **entities and relations** from unstructured data
- Use relation triples (*entity1, relation, entity2*)
- GNNs can be applied for KG embedding to represent the knowledge
- This representation can be used for KG completion to find missing relations
- The representation can also be used for KG alignment where two different KGs have to be matched/aligned to be used in one system

- KG completion can be solved using an encoder-decoder framework with GNNs
- Here, the **neighbourhood information** of an entity can be encoded using a GNN
- The encoder maps each entity to a real-valued vector and the relation can be represented as an embedding or matrix
  - Because of message-passing each node embedding contains information about the neighbours
- The decoder can be regarded as a scoring function
  - Score how likely each edge is
- Normally the models are trained using negative sampling, which randomly corrupts either the subject or the object of a relation triple

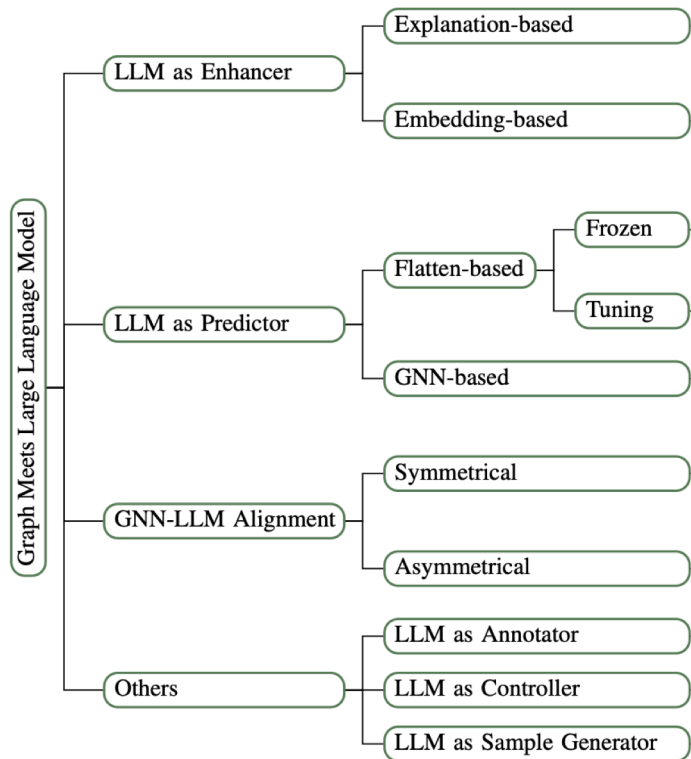


- For KG alignment GNN models are used to learn representations of the entities and relations in different KGs
  - The entity/relation alignment can be performed by computing a distance between two entities or relations
- The distance measuring functions are mainly based on L1 norm, L2 norm or feed-forward neural networks

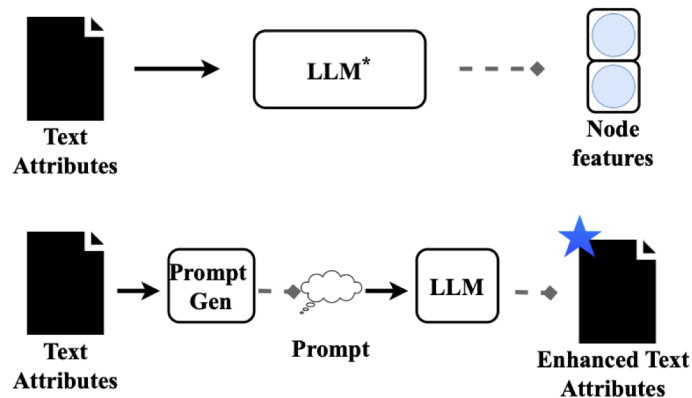
- For IE GNNs have been widely used to model the interaction between **entites and relations** in text
- IE is composed of named entity recognition (NER) and relation extraction (RE)
- GNN-based IE approaches normally operate via a pipeline approach:
  1. Construct a text graph
  2. Recognise entities
  3. Predict the relations plus the relation types between the entities
- It is also possible to *jointly learn* NER and RE
  - Take advantage of the interaction between these two subtasks and to reduce the risk of error propagation

# GNNs and LLMs

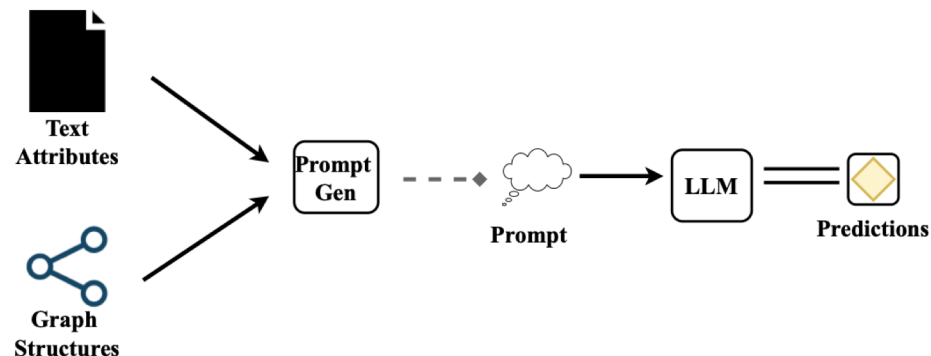
- LLMs can:
  - serve as **enhancers** of GNNs by using LM embeddings or textual outputs as node features
  - serves as **predictors** by taking graph representations (embeddings, flattened graphs) as input for predictions
  - be **aligned to graph structures** via
    - Contrastive training with graph embeddings
    - Iterative training
    - Graph-nested structures for joint training
    - Distillation where GNNs serve as teachers for the LLM to learn awareness for graph structures



# LLM Enhancement vs Prediction



(a) An illustration of LLMs-as-Enhancers, where LLMs pre-process the text attributes, and GNNs eventually make the predictions. Three different structures for this pipeline are demonstrated in Figure 2.



(b) An illustration of LLMs-as-Predictors, where LLMs directly make the predictions. The key component for this pipeline is how to design an effective prompt to incorporate structural and attribute information.

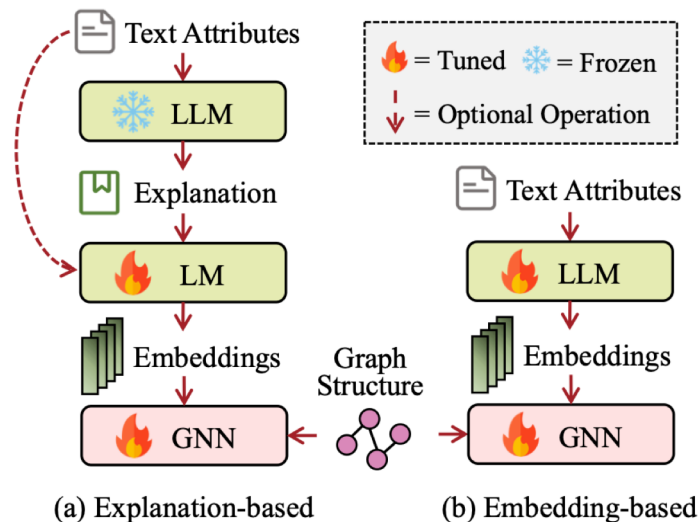


Figure 3: The illustration of LLM-as-enhancer approaches: **a) explanation-based enhancement**, which uses LLMs to generate explanations of text attributes to enhance text embeddings; **b) Embedding-based enhancement**, which directly obtains text embeddings by LLMs as initial node embeddings.

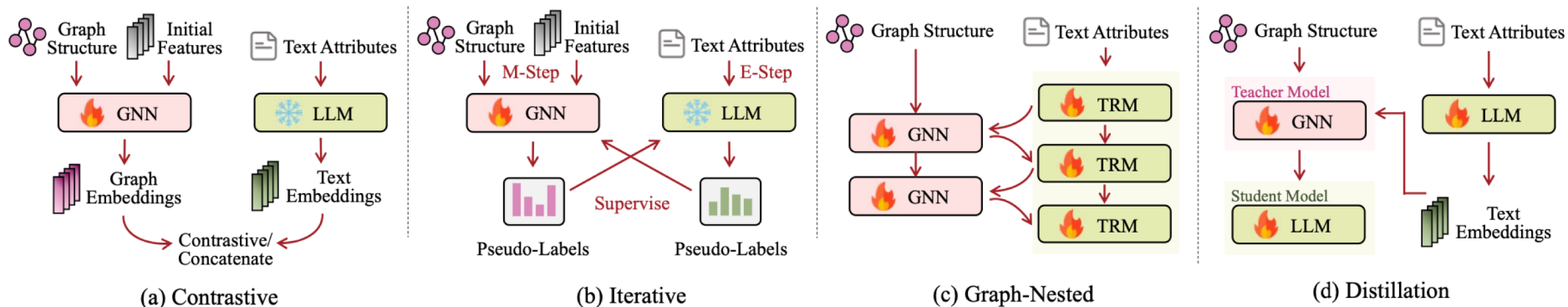


Figure 5: The illustration of GNN-LLM-Alignment approaches: **a) Contrastive**, symmetric alignment which applies concatenation or contrastive learning to graph embeddings and text embeddings; **b) Iterative**, belongs to symmetric alignment, aiming to implement iterative interactions on embeddings of two modalities; **c) Graph-nested**, a symmetric alignment which interweaves GNNs with Transformers and **d) Distillation**, belongs to asymmetric alignment, which uses GNN as a teacher to train language models to be graph-aware.



# LLM-Enhancer Comparison on Node Classification

- High labelling ratio, e.g. 60% of data for training, 20% validation and 20% test set

Table 2: Experimental results for feature-level *LLMs-as-Enhancers* on CORA and PUBMED with a high labeling ratio. We use yellow to denote the best performance under a specific GNN/MLP model, green the second best one, and pink the third best one.

	CORA			PUBMED		
	GCN	GAT	MLP	GCN	GAT	MLP
<b>Non-contextualized Shallow Embeddings</b>						
TF-IDF	90.90 ± 2.74	90.64 ± 3.08	83.98 ± 5.91	89.16 ± 1.25	89.00 ± 1.67	89.72 ± 3.57
Word2Vec	88.40 ± 2.25	87.62 ± 3.83	78.71 ± 6.32	85.50 ± 0.77	85.63 ± 0.93	83.80 ± 1.33
<b>PLM/LLM Embeddings without Fine-tuning</b>						
Deberta-base	65.86 ± 1.96	79.67 ± 3.19	45.64 ± 4.41	67.33 ± 0.69	67.81 ± 1.05	65.07 ± 0.57
LLama 7B	89.69 ± 1.86	87.66 ± 4.84	80.66 ± 7.72	88.26 ± 0.78	88.31 ± 2.01	89.39 ± 1.09
<b>Local Sentence Embedding Models</b>						
Sentence-BERT(MiniLM)	89.61 ± 3.23	90.68 ± 2.22	86.45 ± 5.56	90.32 ± 0.91	90.80 ± 2.02	90.59 ± 1.23
e5-large	90.53 ± 2.33	89.10 ± 3.22	86.19 ± 4.38	89.65 ± 0.85	89.55 ± 1.16	91.39 ± 0.47
<b>Online Sentence Embedding Models</b>						
text-ada-embedding-002	89.13 ± 2.00	90.42 ± 2.50	85.97 ± 5.58	89.81 ± 0.85	91.48 ± 1.94	92.63 ± 1.14
Google Palm Cortex 001	90.02 ± 1.86	90.31 ± 2.82	81.03 ± 2.60	89.78 ± 0.95	90.52 ± 1.35	91.87 ± 0.84
<b>Fine-tuned PLM Embeddings</b>						
Fine-tuned Deberta-base	85.86 ± 2.28	86.52 ± 1.87	78.20 ± 2.25	91.49 ± 1.92	89.88 ± 4.63	94.65 ± 0.13
<b>Iterative Structure</b>						
GLEM-GNN	89.13 ± 0.73	88.95 ± 0.64	N/A	92.57 ± 0.25	92.78 ± 0.21	N/A
GLEM-LM	82.71 ± 1.08	83.54 ± 0.99	N/A	94.36 ± 0.21	94.62 ± 0.14	N/A

- Fine-tuning LMs jointly works well with enough labels
- The choice of the embeddings is not so important if enough labels are available

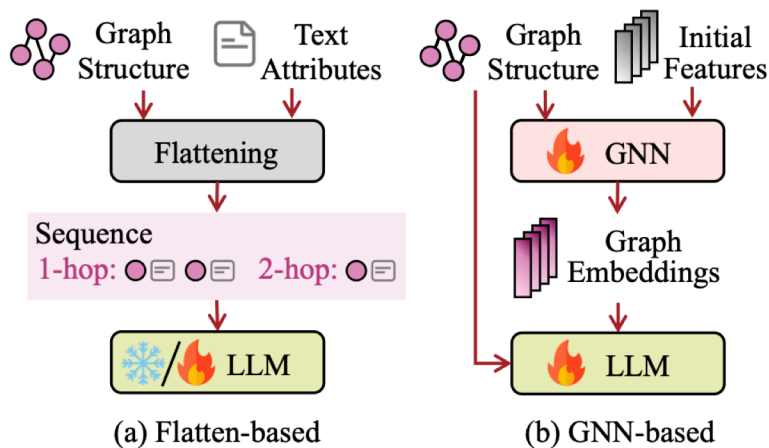
# LLM-Enhancer Comparison on Node Classification

- Low labelling ratio: train only on a few labelled nodes, e.g. 20 from each node class for training, 500 nodes for validation and 1000 for testing

Table 1: Experimental results for feature-level *LLMs-as-Enhancer* on CORA and PUBMED with a low labeling ratio. Since MLPs do not provide structural information, it is meaningless to co-train it with PLM (with their performance shown as N/A). We use yellow to denote the best performance under a specific GNN/MLP model, green the second best one, and pink the third best one.

	CORA			PUBMED		
	GCN	GAT	MLP	GCN	GAT	MLP
<b>Non-contextualized Shallow Embeddings</b>						
TF-IDF	81.99 ± 0.63	82.30 ± 0.65	67.18 ± 1.01	78.86 ± 2.00	77.65 ± 0.91	71.07 ± 0.78
Word2Vec	74.01 ± 1.24	72.32 ± 0.17	55.34 ± 1.31	70.10 ± 1.80	69.30 ± 0.66	63.48 ± 0.54
<b>PLM/LLM Embeddings without Fine-tuning</b>						
Deberta-base	48.49 ± 1.86	51.02 ± 1.22	30.40 ± 0.57	62.08 ± 0.06	62.63 ± 0.27	53.50 ± 0.43
LLama 7B	66.80 ± 2.20	59.74 ± 1.53	52.88 ± 1.96	73.53 ± 0.06	67.52 ± 0.07	66.07 ± 0.56
<b>Local Sentence Embedding Models</b>						
Sentence-BERT(MiniLM)	82.20 ± 0.49	82.77 ± 0.59	74.26 ± 1.44	81.01 ± 1.32	79.08 ± 0.07	76.66 ± 0.50
e5-large	82.56 ± 0.73	81.62 ± 1.09	74.26 ± 0.93	82.63 ± 1.13	79.67 ± 0.80	80.38 ± 1.94
<b>Online Sentence Embedding Models</b>						
text-ada-embedding-002	82.72 ± 0.69	82.51 ± 0.86	73.15 ± 0.89	79.09 ± 1.51	80.27 ± 0.41	78.03 ± 1.02
Google Palm Cortex 001	81.15 ± 1.01	82.79 ± 0.41	69.51 ± 0.83	80.91 ± 0.19	80.72 ± 0.33	78.93 ± 0.90
<b>Fine-tuned PLM Embeddings</b>						
Fine-tuned Deberta-base	59.23 ± 1.16	57.38 ± 2.01	30.98 ± 0.68	62.12 ± 0.07	61.57 ± 0.07	53.65 ± 0.26
<b>Iterative Structure</b>						
GLEM-GNN	48.49 ± 1.86	51.02 ± 1.22	N/A	62.08 ± 0.06	62.63 ± 0.27	N/A
GLEM-LM	59.23 ± 1.16	57.38 ± 2.01	N/A	62.12 ± 0.07	61.57 ± 0.07	N/A

- Fine-tuning LLMs jointly does not work in the low label ratio set-up
- The choice of the embeddings impacts performance significantly
- Sentence embedding models work best



- **Textual input:** flatten the graph structure and input the  $k$ -hop neighbours into the LLM
- **Embedding input:** use graph embeddings as input to an LLM

Figure 4: The illustration of LLM-as-predictor approaches: **a) Flatten-based prediction**, which incorporates graph structure with LLMs via different flattening strategies; **b) GNN-based prediction**, utilizing GNNs to capture structural information for LLMs.

- Graph neural prompting (GNP) for using **graph information as input** to LLMs and apply it to question answering → LLM as predictor
- Question: “Can we learn beneficial knowledge from KGs and integrate them into pre-trained LLMs?”
- GNP retrieves and encodes the knowledge to derive the Graph Neural Prompt, an **graph embedding vector** that can be sent into LLMs

1. GNP first utilises a GNN to capture and encode the **intricate graph knowledge** into entity/node embeddings
2. A cross-modality pooling module is present to determine the **most relevant node embeddings in relation to the text input**
  - consolidate these node embeddings into a holistic **graph-level embedding**
3. GNP encompasses a domain projector to include the **domain information** in the neural prompt
4. A self-supervised **link prediction** objective is introduced to enhance the model comprehension of relationships between entities and capture graph knowledge

# Graph Neural Prompting

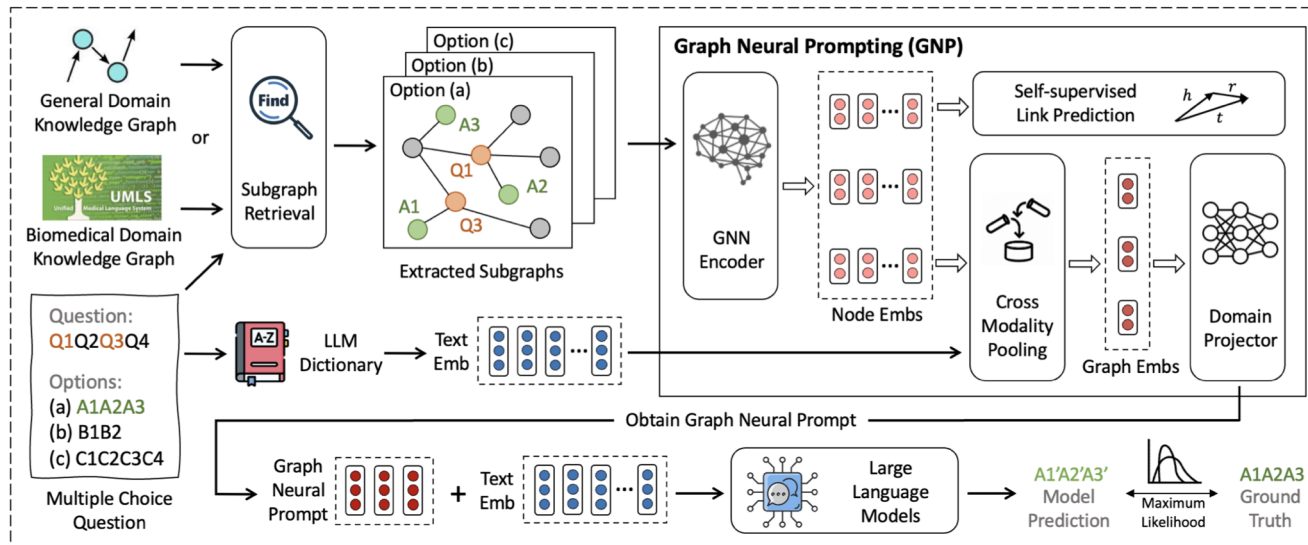
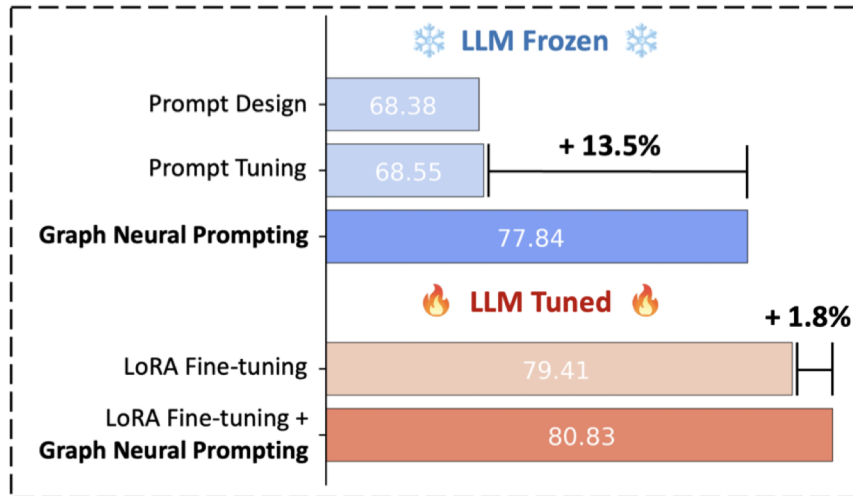


Figure 2: The overall framework. Given a multiple choice question, we first retrieve subgraphs from the knowledge graph based on the entities in the question and options. We then develop Graph Neural Prompting (GNP) to encode the pertinent factual knowledge and structural information to obtain the Graph Neural Prompt. GNP contains various designs including a GNN, a cross-modality pooling module, a domain projector, and a self-supervised link prediction objective. Later, the obtained Graph Neural Prompt is sent into LLM for inference along with the input text embedding. We utilize the standard maximum likelihood objective for downstream task adaptation, while LLM is kept frozen or tuned depending on different experimental settings.

- The graph neural prompt is the **input to the LLM** together with the embedding of the input text.
- Extract subgraphs from huge **general knowledge graphs** based on the input text
- For the contextual subgraph use **entity linking** with the text input
  - include for each entity node the *2-hop neighbours* together with their relations
- Either *fine-tune* the LLM together with the GNN or keep the LLM weights *frozen*.



- When keeping the **LLM frozen** GNP significantly outperforms other methods for QA
- Almost reaches LoRA fine-tuning level
- Compared to fine-tuning the, utilising GNP in addition improves performance further

Figure 1: Result comparison across LLM Frozen (parameters unchanged) and LLM Tuned (parameters updated) settings. The proposed Graph Neural Prompting significantly improves the performance. Reported results are averaged across six datasets on two tasks for an 11B FLAN-T5 model.



- Adapt LLMs to handle graph structure in the form of ***natural language graph tokens***
- Train a projector that maps graph embeddings to **graph tokens** as input to the LLM
  - Train it self-supervisedly via graph matching task
  - Fine-tune on tasks with **graph-based prompt** for better performance
- The LLM learns to use the graph structure of a matched subgraph during inference which improves predictions
- Similar to GNP, however **natural language graph features** as input to LLM instead embeddings

- Natural language graphs are long inputs to LLMs
- The structural information cannot be captured well by LLMs


**(a) ChatGPT with Node Content Only** Token Length: 615


**Input:**  
**Abstract:** The use of lower precision has emerged as a popular technique ...  
**Title:** TiM-DNN: Ternary in-Memory accelerator for Deep Neural Networks  
**Question:** Which arXiv CS sub-category does this paper belong to? ...  
**Output:**  
cs.AR, cs.AI, cs.SY, cs.ET, cs.NE. The paper presents a hardware ...  
Therefore, the most likely category for this paper is **cs.AR** ...

**(b) ChatGPT with Node Content and Text-based Graph Structure** Token Length: 4649

**Input:**  
**Abstract:** The use of lower precision has emerged as a popular technique ...  
**Title:** TiM-DNN: Ternary in-Memory accelerator for Deep Neural Networks  
With it as central node (paper 0), a citation graph can be constructed.  
**The list of neighbors:** Paper 1: ..., ..., Paper 102: ...  
**The citation relations:** Paper 0 cites Paper 1, ..., ... cites Paper 102.  
**Question:** Which arXiv CS sub-category does this paper belong to? ...  
**Output:**  
Based on the title and Abstract, the paper is likely to belong:  
1. **cs.AR** (Hardware Architecture) ...

**(c) GraphGPT** Token Length: 750

**Input:**  
Given a citation graph:  **<graph>** where the 0th node is the target paper, with the following information:  
**Abstract:** The use of lower precision has emerged as a popular technique ...  
**Title:** TiM-DNN: Ternary in-Memory accelerator for Deep Neural Networks  
**Question:** Which arXiv CS sub-category does this paper belong to? ...  
**Output:**  
Based on the title and abstract, we can identify the following CS sub-categories that are most likely to be relevant: 1. **cs.LG** ...  
**Ground Truth:** cs.LG, Machine Learning



**Figure 1: Limitation of LLMs in understanding graph structural contexts with heavy reliance on textual data.**

# GraphGPT for Node Classification

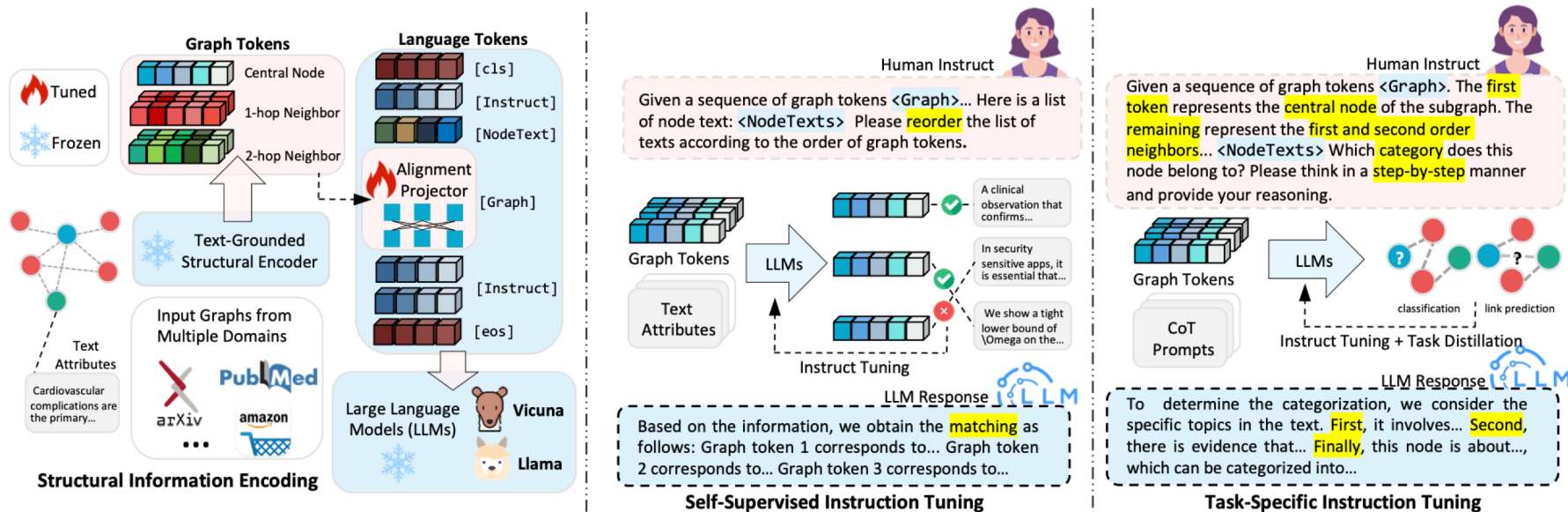


Figure 2: The overall architecture of our proposed GraphGPT with graph instruction tuning paradigm.

- Coreference resolution methods either work with **mention-pairs** that should be classified as coreferent or based on **entity-mentions**
- With entity-mentions use entity-level features including **non-local information**
- As coreferences normally span long distances in text it is difficult to define effective **global graph features**
- Using GNNs to capture **global coreference information** is a possible way to get *entity-centric features*
- Combining **local LM features** with a **coreference graph** to model more global information can improve coreference resolution performance

# Graph-based Coreference Resolution

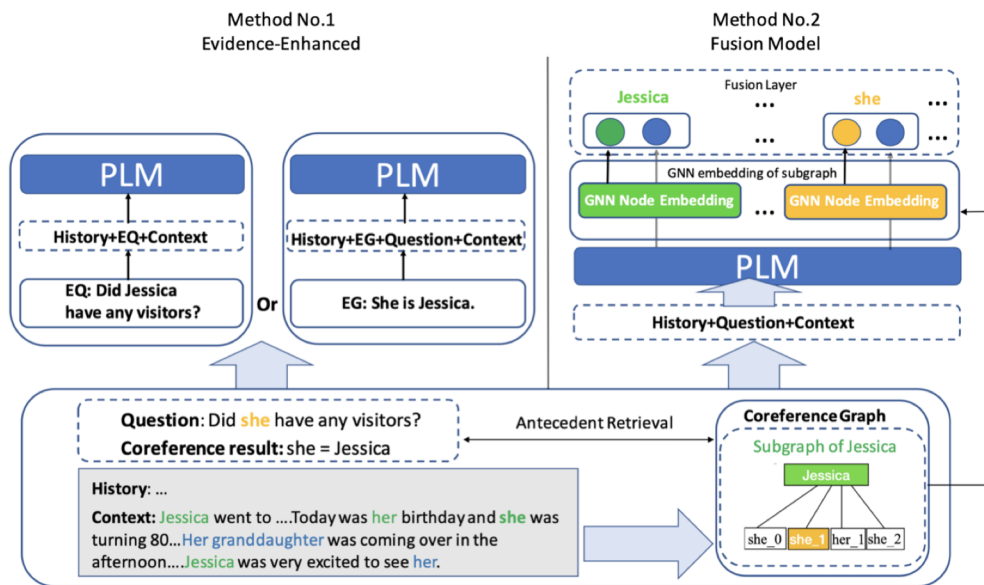
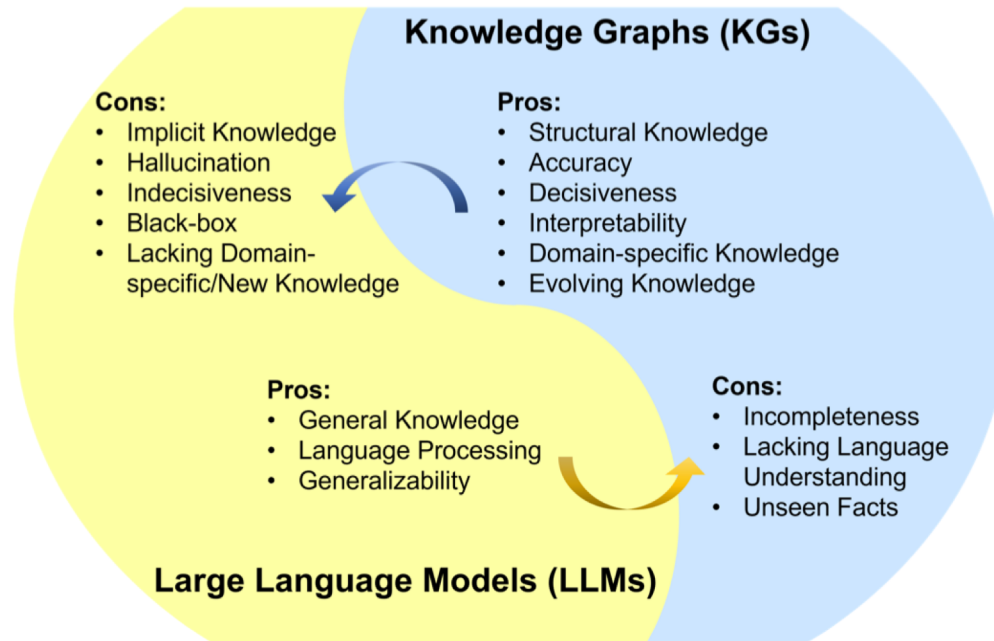


Figure 2: Overview of evidence-enhanced and fusion model. To answer current question, model should determine pronoun's referential entity through context or conversation history; graph-based coreference resolution can precisely determine dependency and add additional information to current question. Left part denotes textual level method of evidence-enhanced method. Right part denotes fusion model and fusion of PLM and graph embedding.

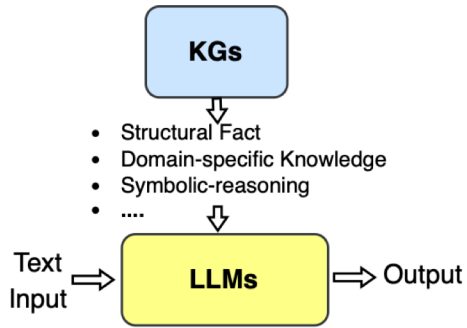
# Large Language Models and Knowledge Graphs

- LLMs lead to a paradigm shift in knowledge representation:
    - Do not rely solely on **explicit representation** of knowledge in knowledge graphs
    - Also use **parametric knowledge** in the parameters of language models
  - Use a hybrid representation of knowledge in both explicit and parametric form
  - Parametric knowledge comes with high recall, but low precision due to possible hallucinations
  - Explicit knowledge has high precision
- Trade-off between precision and recall when choosing parametric or explicit knowledge

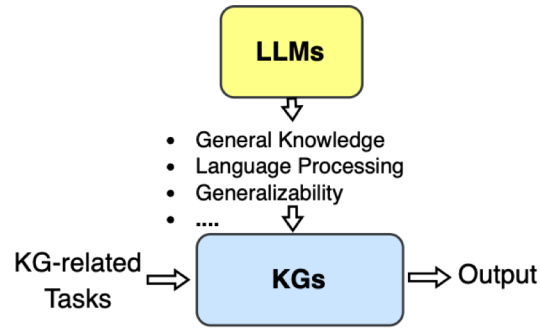




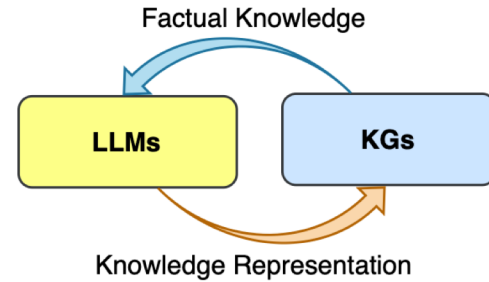
- Biases can be more easily removed in explicit knowledge, as it can be easily edited
- Knowledge is normally stored in text, which can be harnessed by LLMs and can thus increase the amount of available knowledge
- Two ways of combining LLMs and KGs:
  - **Explicit-knowledge-first:** LLMs can augment KGs and improve scalability, quality and utility
  - **Parametric-knowledge-first:** KGs can ground, and verify LLM generations to increase reliability and trust in LLM usage



a. KG-enhanced LLMs



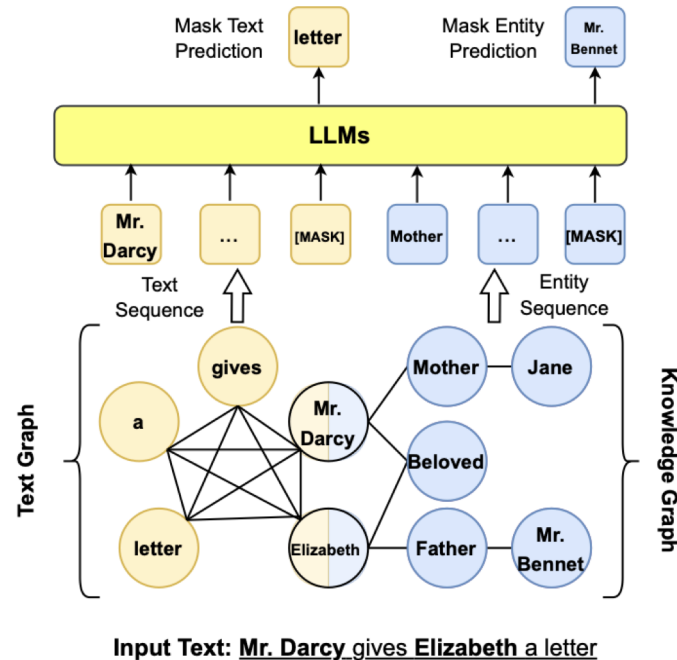
b. LLM-augmented KGs



c. Synergized LLMs + KGs

- LLMs can be used to expand KGs:
  - More scalable knowledge extraction, KG construction and alignment and ontology schema construction
- KGs can help with verification of LLM knowledge
  - Especially with problematic knowledge types, such as numerical values, long-tail entities and updating/editing knowledge

# Injecting KG in LLM



- For a range of reliability or safety-critical applications, **structured knowledge** remains indispensable
  - There are many ways in which KGs and LLMs can improve each other
- Some components might be obsolete, as LLMs perform better out of the box
- Regarding LLMs there exist a magnitude of exaggerated claims and expectations which should be critically examined
  - In particular, a fundamental fix to the so-called problem of **hallucinations** is not in sight.
- The advances triggered by LLMs enable to enter the field with significant shortcuts.

- Combination of LLMs with KGs is promising
- GNNs can be used for computing **KG representations** for completion and alignment
- KG alignment methods could be applied to unify the KGs predicted by LLMs on dialogue-level
  - LLMs cannot get a unified format for whole corpora because of limited context size
- KG completion can be used after predicting dialogue-level KGs e.g. to find more global relations
- **Explicit knowledge** can help in evaluating the ontology

- The information within Large Language Models (LLMs) quickly becomes outdated
  - Knowledge needs to be edited on a regular basis
- Existing knowledge editing methods often overlook the interconnected nature of facts, failing to account for the ripple effects caused by changing one piece of information
- Use graph-memory for knowledge editing:
  - Store and update information in **external knowledge graph** without changing the parametric knowledge in LLM
  - The LLM is prompted to **generate queries** to extract information from the KG
  - The KG updates are given externally

# Graph Memory-based Editing for LLMs

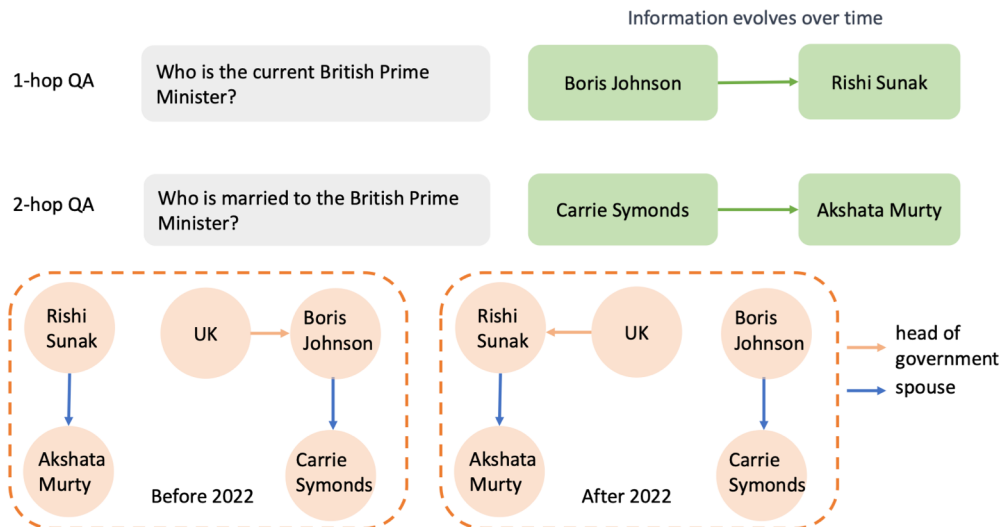
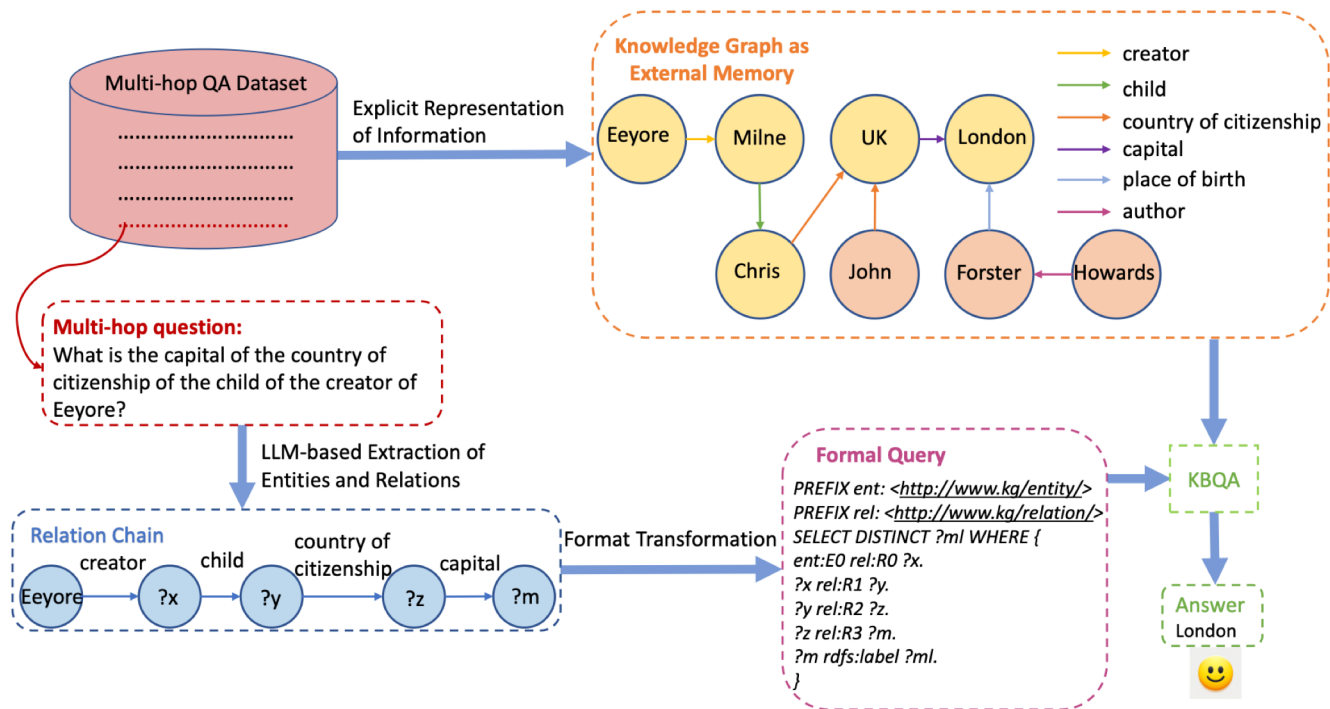


Figure 1: Dynamic nature of information: Changes over time may trigger subsequent modifications. For instance, a transition in the British Prime Minister, such as from Boris Johnson to Rishi Sunak, necessitates corresponding adjustments, like the change in the British Prime Minister’s spouse.



# Graph Memory-based Editing for LLMs



# GNNs for Ontology Relation Extraction

- Input: List of task-oriented dialogues  $D = \{d_1, d_2, \dots, d_n\}$  where each dialogue  $d_i$  comes with a list of terms  $T_i = \{t_1, \dots, t_k\}$
- Output: for each pair of terms  $(t_i, t_j)$  in a dialogue predict whether exactly one of these relations holds:
  - Domain-slot relation  $e_{ds}$  where the head entity is a domain and the tail entity is a slot belonging to the domain term in the respective dialogue
  - Slot-value relation  $e_{sv}$  where the head entity is a slot and the tail entity is a value belonging to the slot in the respective dialogue
  - Value-domain relation  $e_{vd}$  where the head entity is a value and the tail entity is a domain the slot belongs to
  - Equivalence relation  $e_{eq}$  where the two terms are from the same hierarchy level and equivalent in meaning (only undirected relation)

- **Intra-dialogue relations:** relations that can be inferred from single dialogues, as both terms occur the same dialogue at least once (*local information*)
- **Inter-dialogue relations:** relations where terms from different dialogues are connected (*global information*)

## Input: dialogue + term list

"user": "am looking for a place to to stay that has cheap price range it should be in a type of hotel"  
"system": "okay , do you have a specific area you want to stay in ?"  
"user": "no , i just need to make sure it ' s cheap . oh , and i need parking"  
"system": "i found 1 cheap hotel for you that includes parking . do you like me to book it ?"  
"user": "yes , please . 6 people 3 nights starting on tuesday ."  
"system": "i am sorry but i wasn ' t able to book that for you for tuesday . is there another day you would like to stay or perhaps a shorter stay ?"  
"user": "how about only 2 nights ."  
"system": "booking was successful . reference number is : 7gawk763 . anything else i can do for you ?"  
"user": "no , that will be all . good bye ."  
"system": "thank you for using our services ."

## Output: relations between terms

- (domain, has slot, slot)
- (slot, has value, value)
- (value, has domain, domain)
- (term1, refers to same concept as, term2)

['parking', 'has value', 'yes'], ['yes', 'has domain', 'hotel'],  
**['book day', 'has value', 'tuesday'], ['1', 'refers to same concept as', 'me'],** ['7gawk763', 'has domain', 'hotel'],  
['hotel', 'has slot', 'price range'], ['hotel', 'has domain', 'hotel'],  
['1', 'has domain', 'hotel'], **['hotel', 'has slot', 'book stay'],**  
['book stay', 'has value', '3'], **['type', 'has value', 'hotel'],**  
['hotel', 'has slot', 'book day'], ['hotel', 'has slot', 'book people'],  
['hotel', 'has slot', 'parking'], ['hotel', 'has slot', 'choice'],  
['hotel', 'has slot', 'type'], ['choice', 'has value', '1'],  
['hotel', 'has slot', 'ref'], ['tuesday', 'has domain', 'hotel'], ['2', 'has domain', 'hotel'],  
['ref', 'has value', '7gawk763'], **['price range', 'has value', 'cheap'],** ['book people', 'has value', '6'],  
['6', 'has domain', 'hotel'], **['book stay', 'has value', '2'],**  
['hotel', 'has slot', 'area'], ['3', 'has domain', 'hotel'], ['cheap', 'has domain', 'hotel']

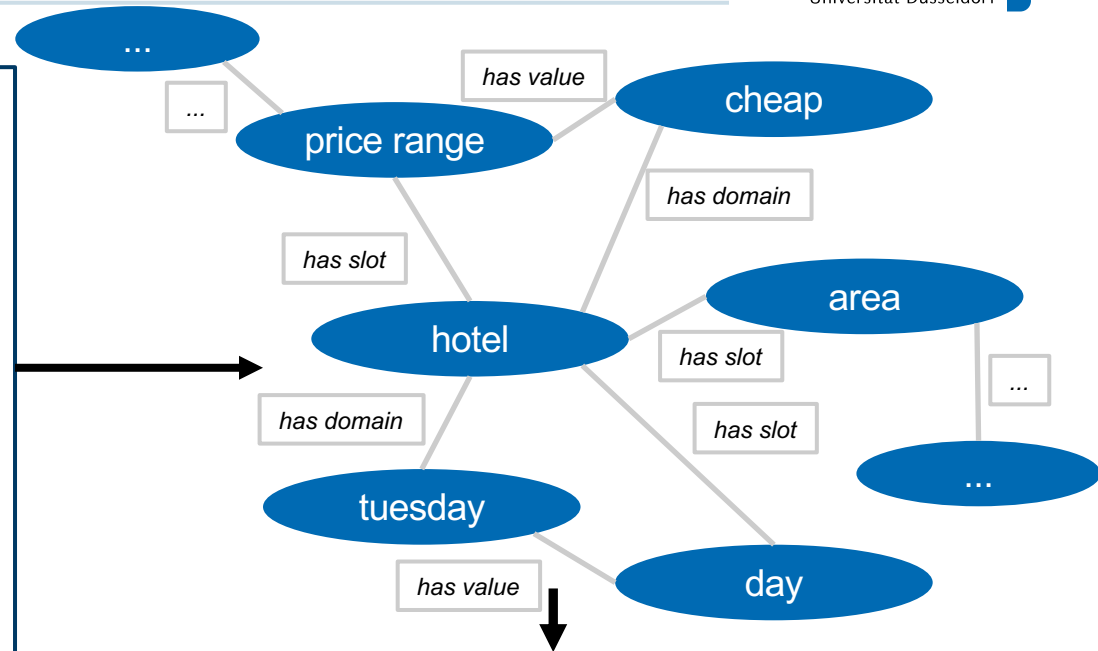
## Termlist

['price', 'day', 'parking', 'hotel', 'stay', 'reference', 'type', 'tuesday', 'area', 'people', '7gawk763', 'price range', '2', '1', 'me', '6', 'nights', 'reference number', '3', 'cheap']

# GNN for Intra-dialogue Ontology RE

Input: dialogue + term list

"user": "am looking for a place to to stay that has cheap price range it should be in a type of hotel"  
"system": "okay , do you have a specific area you want to stay in ?"  
"user": "no , i just need to make sure it ' s cheap . oh , and i need parking"  
"system": "i found 1 cheap hotel for you that includes parking . do you like me to book it ?"  
"user": "yes , please . 6 people 3 nights starting on tuesday ."  
"system": "i am sorry but i wasn ' t able to book that for you for tuesday . is there another day you would like to stay or perhaps a shorter stay ?"  
"user": "how about only 2 nights ."  
"system": "booking was successful . reference number is : 7gawk763 . anything else i can do for you ?"  
"user": "no , that will be all . good bye ."  
"system": "thank you for using our services ."



→ Train GNN based on the groundtruth graph to predict **intra-dialogue relations**  
→ possibly use the GNN embedding as additional/main input to LLM (neural prompt)

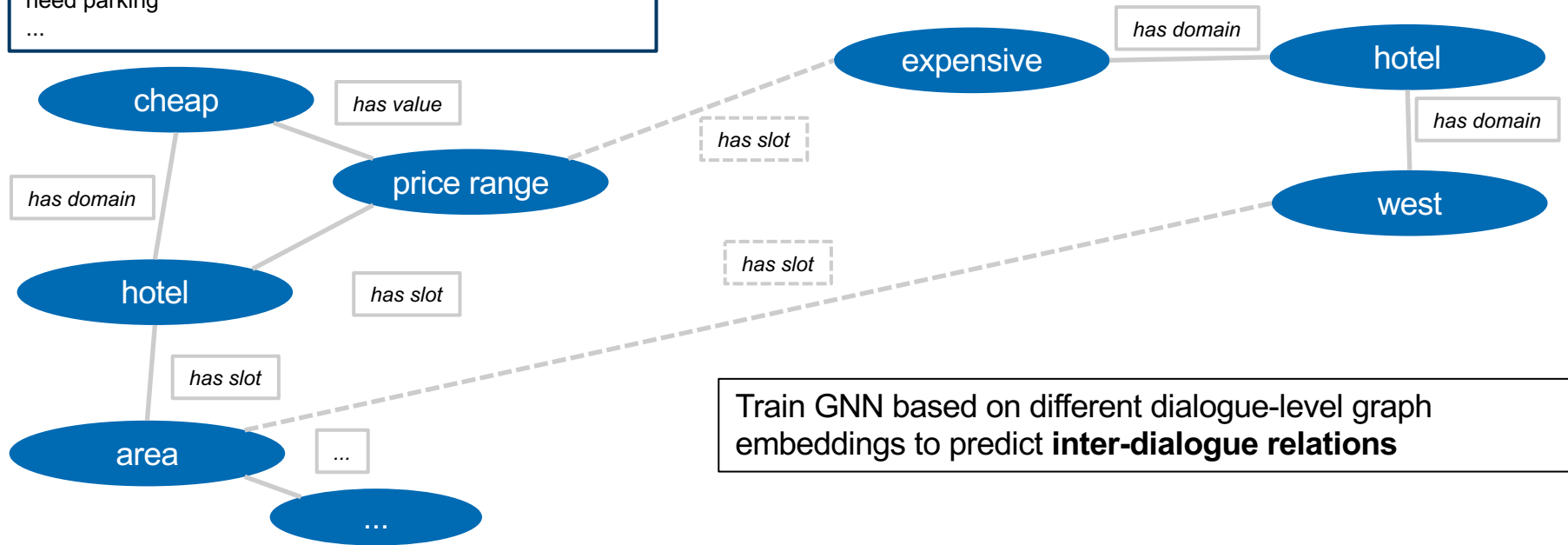
## Termlist

[ 'price', 'day', 'parking', 'hotel', 'stay', 'reference', 'type', 'tuesday', 'area', 'people', '7gawk763', 'price range', '2', '1', 'me', '6', 'nights', 'reference number', '3', 'cheap' ]

# GNN for Inter-dialogue Ontology RE

"user": "am looking for a place to to stay that has cheap **price range** it should be in a type of hotel"  
"system": "okay , do you have a specific **area** you want to stay in ?"  
"user": "no , i just need to make sure it ' s cheap . oh , and i need parking"  
...

"user": "could you suggest an **expensive** hotel in the **west**?"  
"system": "there are several options, do you need free wifi?"  
"user": "yes, free wifi sounds nice."  
...



## ■ Local:

- Predict intra-dialogue relations with a dialogue-level term graph to train GNN embeddings based on LM embeddings

## ■ Global:

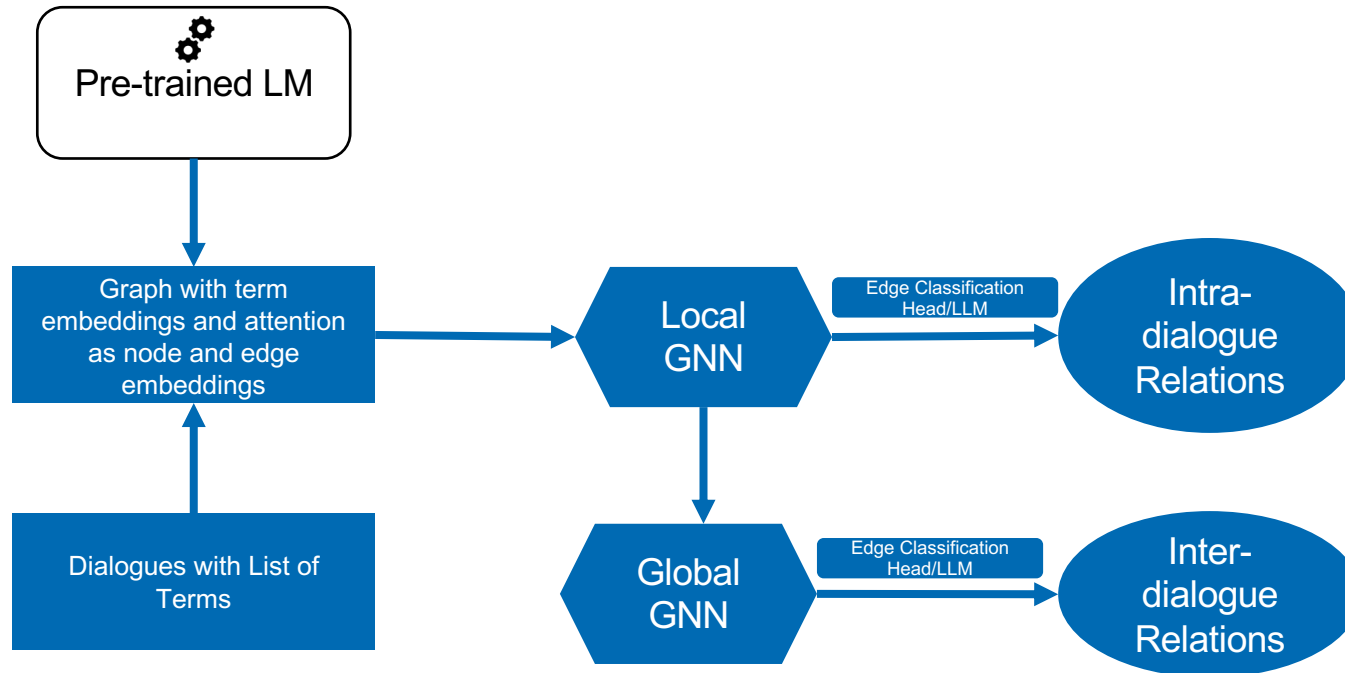
- Model gets a set of **dialogue-level graphs as input** and is trained to make inter-dialogue relations predictions between the graphs

## ■ Predict term relations with fine-tuned GNN with LM embedding input

- GNN with edge classification head
- GNN embeddings as LLM input possibly combined with dialogue embedding
- LLM with textual graph input



# GNN-based Approach



- The local model gets a term graph with LM features from the dialogue embedding as input and predicts **intra-dialogue relations**
- The global model gets a set of dialogue-level GNN embedded graphs as input and predicts **inter-dialogue relations**
- Possibly jointly have one GNN for both intra- and inter-dialogue relations
- Alternatively use LLM for local predictions and the GNN for global information

- The training data is constructed from the TOD data-set's annotation on groundtruth terms
- The model is trained to predict the relation type and direction for each edge/pair of terms in the dialogue
- First focus on intra-dialogue relations, then inter-dialogue
- A separate local and global model are trained
- The classification head gets the **node representations of two terms** as input and outputs a distribution over the **different relation classes**

- GCN, GAT or GTN, which are implemented in PyTorch Geometric<sup>1</sup> library
- Model a heterogeneous graph with different types of nodes and edges
- The **input embeddings** come from an LM → node and edge features
- It should be investigated whether it is better to fine-tune the LM jointly with the GNN or to keep it fixed
- Baselines:
  - Fine-tuned LM
  - Model with textual graph input for global relation prediction
- First focus on the same data distribution

- Graph neural networks can capture **structural information for graphs**
- Graph attention networks can capture different edge weights
- The resulting embeddings can be utilised for downstream tasks
- Possibly GNNs can be utilised for the task of **ontology relation extraction**
- Here, **intra- and inter-dialogue** relations might have to be **considered separately** with different models



## Thank you!

Looking forward to your questions.