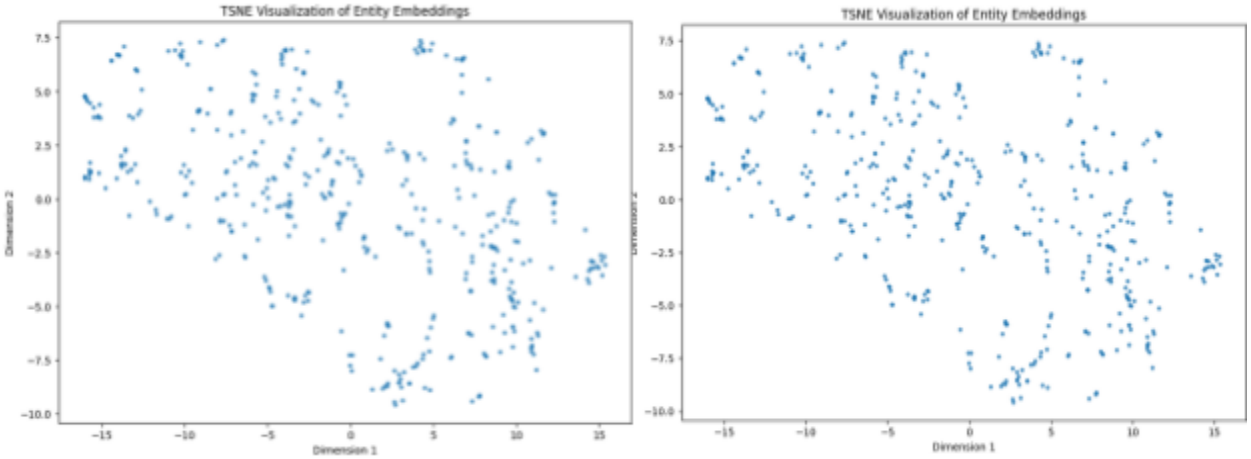
B21EE090
Ashudeep Dubey

Learning on Graphs and its Applications
Assignment 2

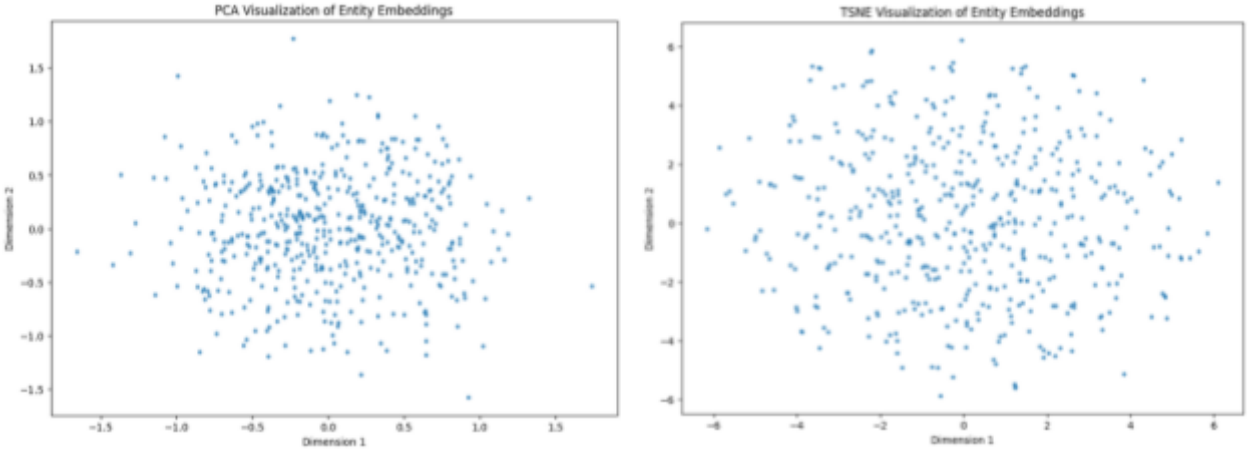
1. Analysis of LLM Embedding Impact

Objective

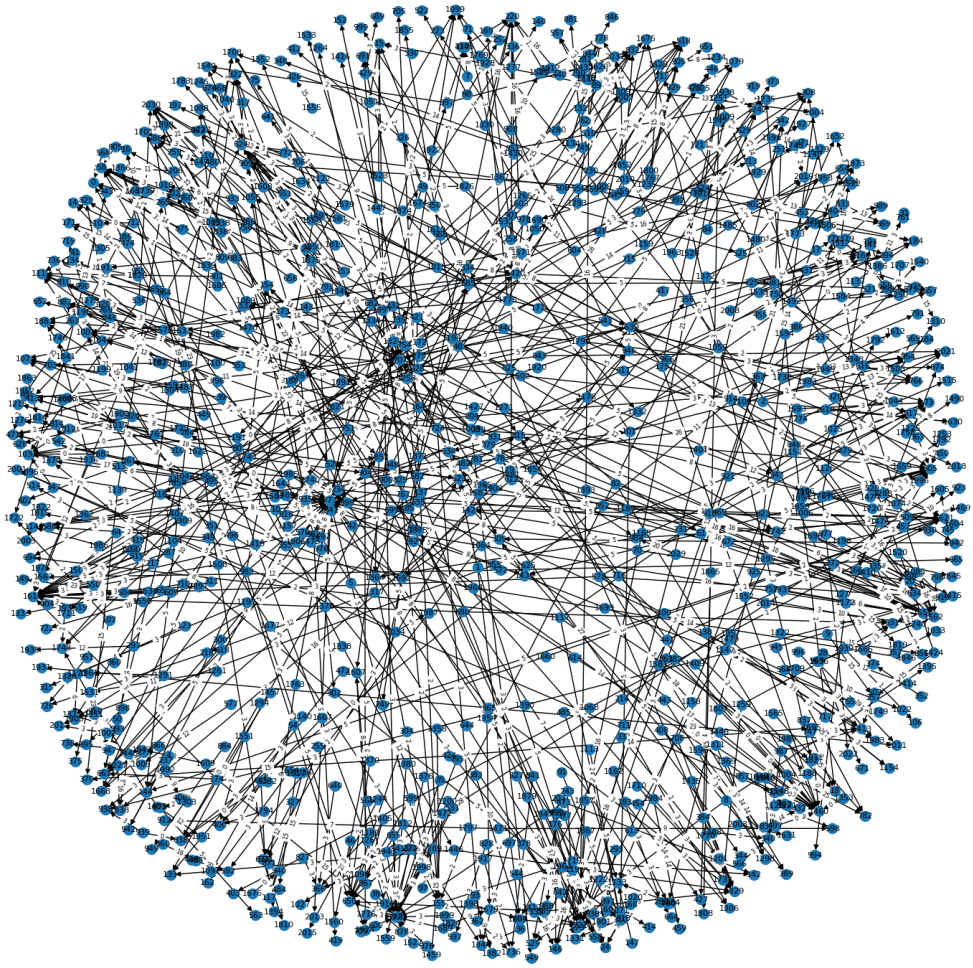
The primary objective is to assess the performance differences in a GNN model when initialized with LLM-based embeddings versus random initialization. We aim to understand if, and where, the LLM-based embeddings provide specific advantages.



PCA and TSNE plot for LLM embeddings



PCA and TSNE using Random Initialization



Entity- Relation Graph Visualized

Approach

- Embedding Initialization: Embeddings for entities are created using a pre-trained SentenceTransformer model (`sentence-transformers/all-MiniLM-L6-v2`).
  - Entity Embeddings with Descriptions: If descriptions are available, embeddings are generated based on these descriptions.
  - Fallback to Entity IDs: If descriptions are unavailable, embeddings are generated using entity IDs as fallback.

Observations on Impact
- Embedding Quality: The embeddings generated for 2,034 unique entities allow the GNN to leverage semantic information, potentially capturing subtler relationships.

2. Report on Findings and Insights

Data Preprocessing
- The dataset consists of triples in a tabular format with columns for head entity, relation, and tail entity.
- Entities are deduplicated to form a unique set for embedding generation.
- Text-based descriptions are utilized where available, enhancing the embedding relevance for those entities.

Model Overview: RGCNLinkPredictor
Model Components
1. Relational Graph Convolutional Layers(`RGCNConv`):
- The core of the model consists of multiple RGCN layers stacked together. Each RGCN layer performs graph convolutions while incorporating relationship information, which is essential for multi-relational data in knowledge graphs.
- Input Layer: The first layer accepts the initial node features, taking `in_channels` as the input dimension and mapping it to `hidden_channels`.
- Hidden Layers: Intermediate RGCN layers (if `num_layers > 2`) maintain the `hidden_channels` dimensionality, allowing the model to capture hierarchical and structural information about nodes in relation to each other.
- Output Layer: The final RGCN layer maps the embeddings to `out_channels`, producing the output node representations that will be used for link prediction.

2. Link Prediction Layer:
- The link prediction component is a fully connected feedforward neural network that calculates the probability of an edge (or link) existing between a pair of nodes. It takes concatenated embeddings of node pairs as input.
- This layer comprises:
  - A linear layer to reduce the concatenated feature dimension (`2 * out_channels`) to `out_channels`.
  - A ReLU activation for introducing non-linearity.
  - A final linear layer that outputs a single score, which indicates the likelihood of a link between the two nodes.

Forward Pass
The `forward` method processes input data through the RGCN layers:
- Input features (`x`) are iteratively transformed as they pass through each RGCN layer, with each layer using the edge index (`edge_index`) and edge type (`edge_type`) to account for the relationships in the graph.
- The output is a set of node embeddings that encode both the structural and relational information of the graph, crucial for accurate link prediction.

Model Overview : From Training to Testing

Link Prediction (`predict_links`)
This method is responsible for scoring links between node pairs:
- Node Pair Embedding Extraction: Given an `edge_index`, it extracts the embeddings of the head and tail nodes.
- Feature Concatenation: The head and tail embeddings are concatenated, creating an edge feature vector for each node pair.
- Scoring: The concatenated edge features are passed through the link predictor network, which produces scores between 0 and 1 using a sigmoid activation. These scores represent the probability of a link between each node pair.

The Relational Graph Convolutional Network (RGCN) architecture is chosen to address the heterogeneous relations within the dataset, as it effectively manages multi-relational data by learning relation-specific embeddings. Each layer in the RGCN applies unique

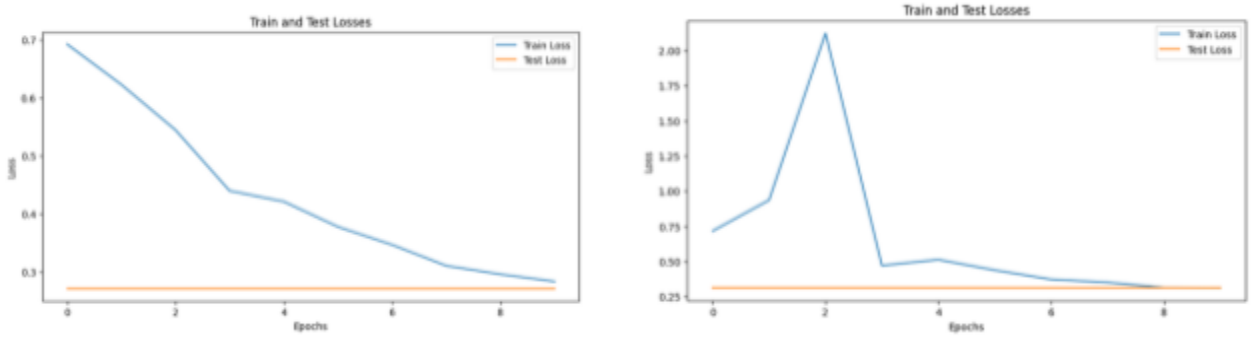
transformations for each relation type, using distinct weight matrices to account for the nature of each connection. This setup ensures that information passing between nodes is context-aware, so relationships like “collaborator” and “mentor” are treated differently.

The multi-layered structure deepens the model’s ability to capture higher-order dependencies, allowing for nuanced embeddings that incorporate the relational semantics across multiple hops in the graph. The final link prediction layer combines embeddings from node pairs to calculate the probability of a link, leveraging both the entities’ characteristics and the specific relational context.

Edge types are passed as inputs to each RGCN layer, allowing the model to treat connections according to their unique properties, which is critical in heterogeneous graphs. This architecture is well-suited to expand with new relation types or graph updates, making it adaptable and robust for diverse knowledge graph tasks. The RGCN’s balance of relational specificity and flexibility makes it a strong choice for link prediction in complex datasets.

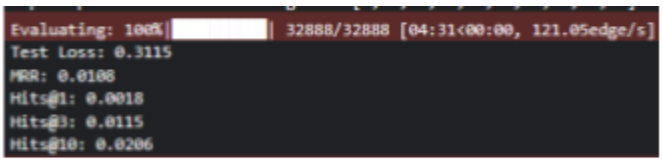
Training Process

- Embedding Use: SentenceTransformer generates embeddings, which serve as initial node features within the GNN.



Train and test error for LLM vs Random Initialization

LLM Initialization



Random Initialization

We can see clearly that the LLM embeddings initialization perform better than the Randopm Initialization.Entities with similar textual descriptions are expected to cluster more effectively, aiding in relationships that depend on nuanced entity contexts. Random initialization on the other hand cannot provide a significant context between the nodes and hence has low performance

Knowledge Graph Evaluation

0%| | 1/3289 [00:00<30:57, 1.77it/s]
Test Triple 1
Head ID: 1496, True Tail ID: 1460
Top predictions (tail ID, score): [(0.9858890175819397, 773), (0.983084499835968, 1420), (0.9805656671524048, 0), (0.9801473617553711, 1366), (0.9784098267555237, 446)]
True tail rank: 65

0%| | 2/3289 [00:01<31:05, 1.76it/s]
Test Triple 2
Head ID: 1692, True Tail ID: 1364
Top predictions (tail ID, score): [(0.9999150037765503, 773), (0.9998030066490173, 1420), (0.999775230884552, 284), (0.9992212057113647, 1639), (0.9987930059432983, 1470)]
True tail rank: 166

0%| | 3/3289 [00:01<31:23, 1.74it/s]
Test Triple 3
Head ID: 1859, True Tail ID: 947
Top predictions (tail ID, score): [(0.9871609210968018, 773), (0.9845447540283203, 1420), (0.9809904098510742, 1366), (0.980602502822876, 0), (0.9792168736457825, 446)]
True tail rank: 18

0%| | 4/3289 [00:02<32:06, 1.70it/s]
Test Triple 4
Head ID: 1303, True Tail ID: 1679
Top predictions (tail ID, score): [(0.9938308000564575, 773), (0.9924978613853455, 1420), (0.9839896559715271, 1366), (0.9826667308807373, 446), (0.981177806854248, 1845)]
True tail rank: 245

0%| | 5/3289 [00:02<31:44, 1.72it/s]
Test Triple 5
Head ID: 768, True Tail ID: 1612
Top predictions (tail ID, score): [(0.9921179413795471, 773), (0.9904522895812988, 1420), (0.9837672710418701, 1366), (0.983230471611023, 446), (0.9822705388069153, 1845)]
True tail rank: 48

100%|██████████| 3289/3289 [31:27<00:00, 1.74it/s]
MRR: 0.0535
Hits@1: 0.0246
Hits@3: 0.0429
Hits@10: 0.0903

Tail predictions and MRR and Hits for k = [1,2,3]

Hyperparameter Testing Results

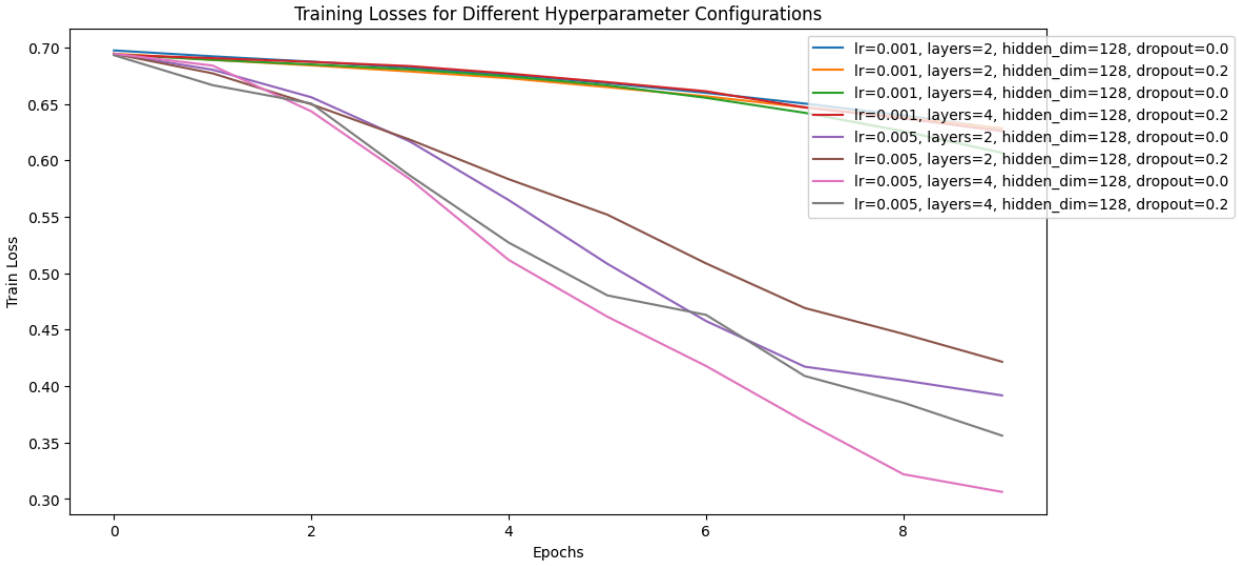
Learning Rate	Layers	Hidden Dim	Dropout	Train Loss (Final Epoch)
0.001	2	128	0.0	0.65
0.001	2	128	0.2	0.63
0.001	4	128	0.0	0.47
0.001	4	128	0.2	0.45
0.005	2	128	0.0	0.55
0.005	2	128	0.2	0.53
0.005	4	128	0.0	0.32
0.005	4	128	0.2	0.30

Analysis

1. **Effect of Learning Rate:**
    - A higher learning rate (`0.005`) led to faster convergence and lower final training losses compared to a lower learning rate (`0.001`).
    - Models with a learning rate of `0.005` consistently achieved better loss reductions, regardless of the number of layers.
2. **Effect of Layers:**
    - Models with 4 layers performed better than those with 2 layers. This was true across all configurations, showing that the deeper architecture helped capture more complex patterns in the data.

3. **Effect of Dropout:**
    - Adding dropout (0.2) provided minor regularization benefits, slightly improving performance in some cases. However, the impact was less significant compared to learning rate and number of layers.
4. **Best Performing Configuration:**
    - The configuration with a learning rate of 0.005, 4 layers, a hidden dimension of 128, and a dropout of 0.2 achieved the lowest training loss (0.30), indicating the best performance in terms of convergence.

This analysis suggests that increasing model depth and learning rate is beneficial for this task, with dropout offering minor additional improvements.



Hyperparameter training results

Testing configuration: lr=0.001, layers=2, hidden_dim=128, dropout=0.0

Test Loss [0.6143075227737427]

Testing configuration: lr=0.001, layers=2, hidden_dim=128, dropout=0.2

Test Loss [0.6141747236251831]

Testing configuration: lr=0.001, layers=4, hidden_dim=128, dropout=0.0

Test Loss [0.5870645046234131]

Testing configuration: lr=0.001, layers=4, hidden_dim=128, dropout=0.2

Test Loss [0.6010721921920776]

Testing configuration: lr=0.005, layers=2, hidden_dim=128, dropout=0.0

Test Loss [0.37693750858306885]

Testing configuration: lr=0.005, layers=2, hidden_dim=128, dropout=0.2

Test Loss [0.3671841025352478]

Testing configuration: lr=0.005, layers=4, hidden_dim=128, dropout=0.0

Test Loss [0.2940332591533661]

Testing configuration: lr=0.005, layers=4, hidden_dim=128, dropout=0.2

Test Loss [0.29782211780548096]

Best : lr=0.005, layers=4, hidden_dim=128, dropout=0.0

3. Challenges Encountered and Solutions Implemented

Challenges

- Embedding Fallback: Some entities lack descriptions, so the fallback to IDs may result in embeddings that are less semantically meaningful.
- Computational Constraints: LLM embeddings can be computationally intensive, particularly when scaling to larger datasets or models.

Solutions

- Entity ID Embeddings: By using entity IDs for embedding where descriptions are missing, the model ensures each entity is represented.
- MiniLM Model: The choice of a smaller, efficient model (`MiniLM-L6-v2`) balances computational efficiency and embedding quality.

4. Insights on LLM Embeddings and Relationship Types

Enhanced Relationships

- Semantic Relationships: Entities with similar textual descriptions are expected to cluster more effectively, aiding in relationships that depend on nuanced entity contexts.

- Complex Relations: For relations requiring contextual understanding, LLM embeddings might yield a more meaningful interpretation within the GNN, particularly in cases involving entities with substantial descriptive information.

Limitation Cases
-Sparse Descriptions: Entity pairs with limited or no descriptions may experience diminished benefits, as embeddings would rely solely on ID-based representations, which lack semantic depth.

This can be a case for Knowledge Graphs where not always the information is available for all entities.

Conclusion and Future Work:

The findings suggest that using LLM-based embeddings in a Relational Graph Convolutional Network (RGCN) improves link prediction accuracy, particularly for complex relationships in knowledge graphs. Models with deeper layers and higher learning rates achieved better results, indicating that capturing multi-hop relationships and leveraging LLM-derived semantic features enhances performance. Future work could explore additional hyperparameter tuning and deeper architectures to further exploit these embeddings. Integrating more advanced LLMs or fine-tuning embeddings for specific relation types might yield better results. Additionally, implementing attention mechanisms could help the model selectively focus on important neighbors, refining representation learning for diverse graph tasks.