NASA GES-DISC

LINK PREDICTION AND CONTENT RECOMMENDATION

Sagar Sheth

MEET THE TEAM

Data Engineer

ML Engineer

Data Analyst

Tester

AI Assistant

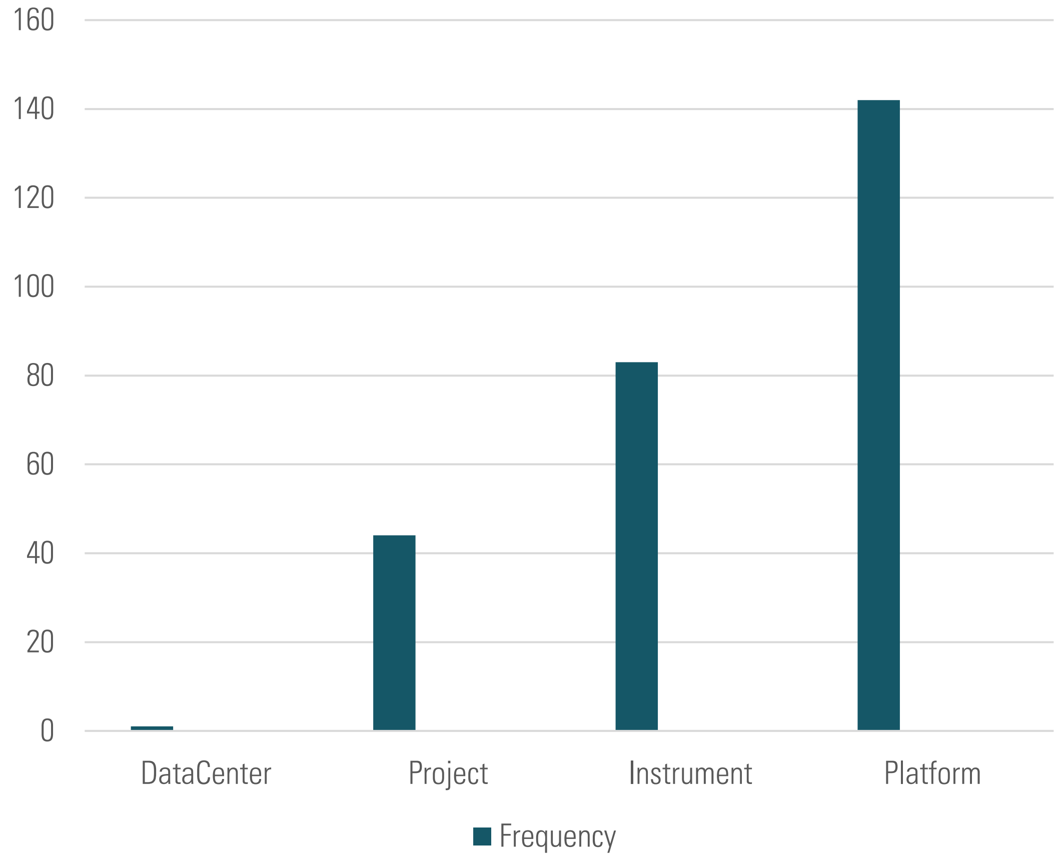
INTRODUCTION

- The NASA Goddard Earth Sciences Data and Information Services Center (GES DISC) has developed a knowledge graph to enhance dataset discovery and research reproducibility.
- This graph connects various entities such as datasets, publications, instruments, platforms, authors, and science keywords, facilitating improved navigation and understanding of the relationships between these components.
- By integrating machine learning techniques like link prediction, we can identify missing associations between datasets and science keywords. This approach enhances metadata completeness, thereby improving dataset discoverability.
- Additionally, by utilizing GNN's with RAG one can augment and enrich recommendations with more relevant content.

EXPLORING THE DATA

LABELS

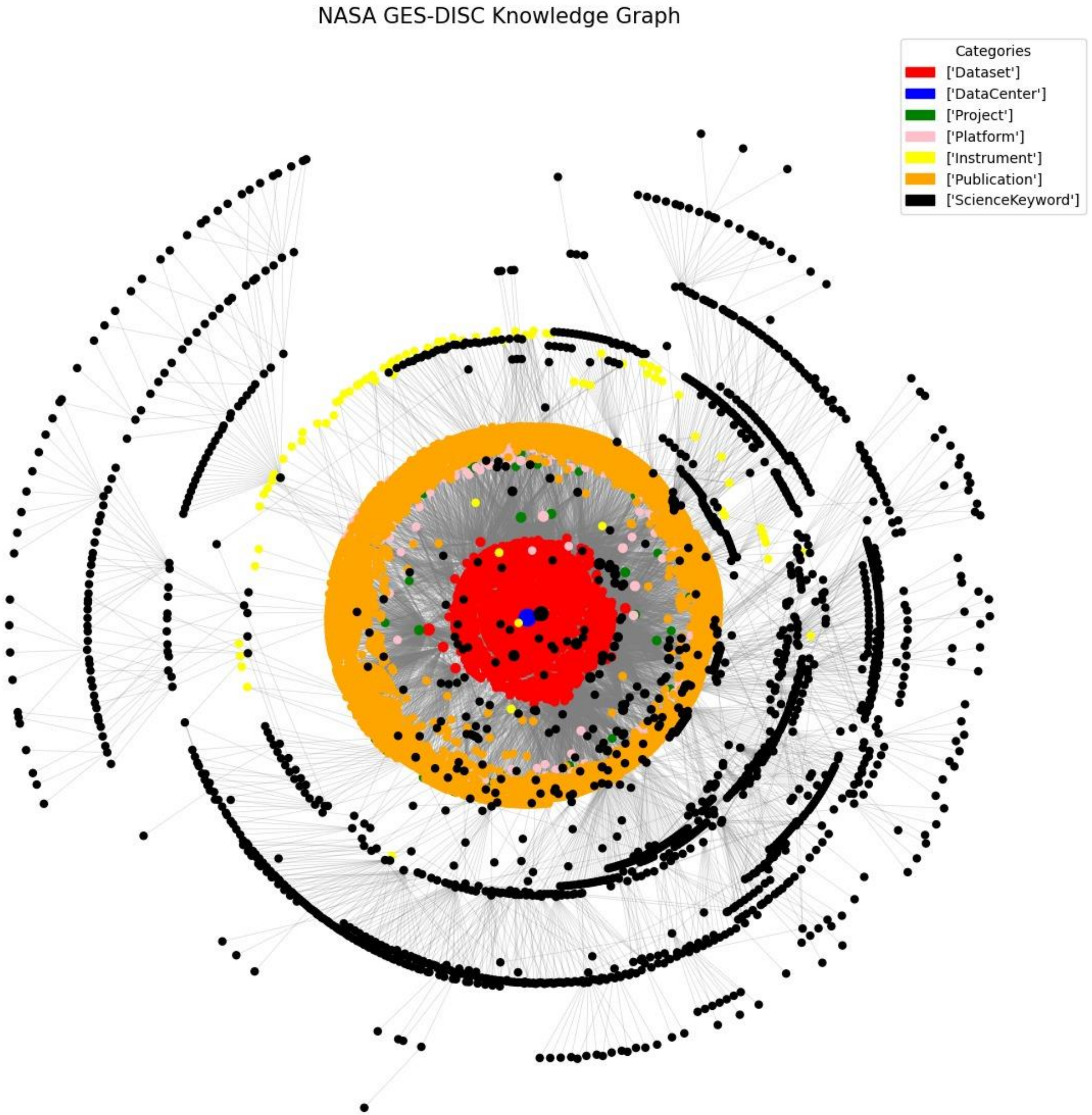
Label	Frequency
DataCenter	1
Project	44
Instrument	83
Platform	142
Dataset	1300
ScienceKeyword	1609
Publication	2584



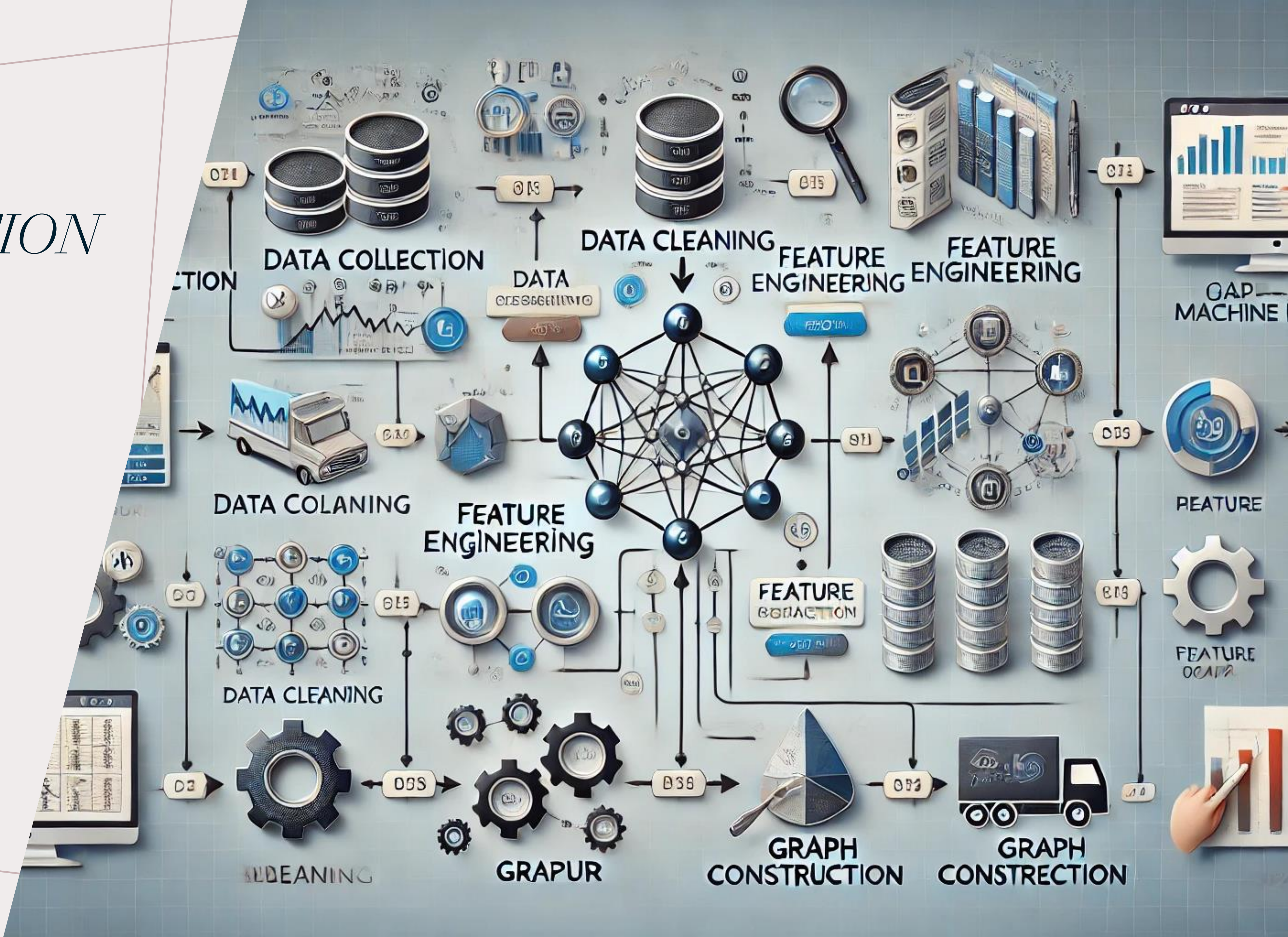
NODE RELATIONSHIPS

Source Label	Relationship Type	Target Label	Connection Count
DataCenter	Has Dataset	Dataset	1300
Dataset	Has Platform	Platform	1519
Dataset	Has Sciencekeyword	ScienceKeyword	4015
Dataset	Of Project	Project	1325
Platform	Has Instrument	Instrument	215
Publication	Uses Dataset	Dataset	3623
ScienceKeyword	Subcategory Of	ScienceKeyword	1823

GRAPH VISUALIZATION



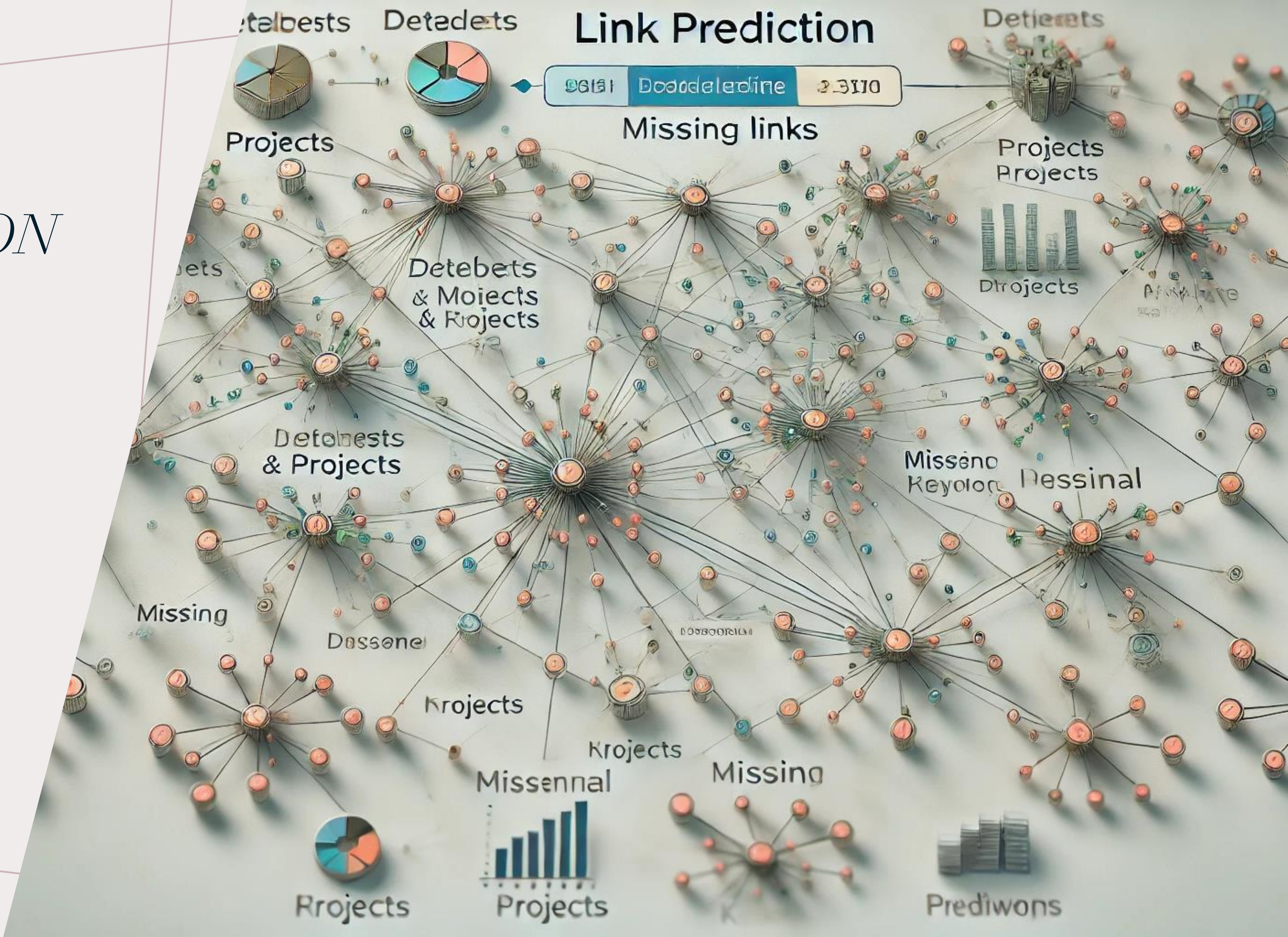
DATA PREPARATION



DATA PREPROCESSING FOR MODELLING

- Using Node2Vec, generated vector embeddings for vanilla Logistic regression and Neural Nets.
- Using NetworkX library to store the Knowledge Graph as a Graph in Python.
- Based on the Label on the Node, generated different abstracts derived from the properties column
- Using GPT2's tokenizer Generated embeddings for the nodes from the user defined abstracts.

LINK PREDICTION



LINK PREDICTION

Generating node embeddings using various methods such as node2vec, LLM representation., etc.

Leveraging this embedding and the knowledge graph to then develop a Graph neural Network.

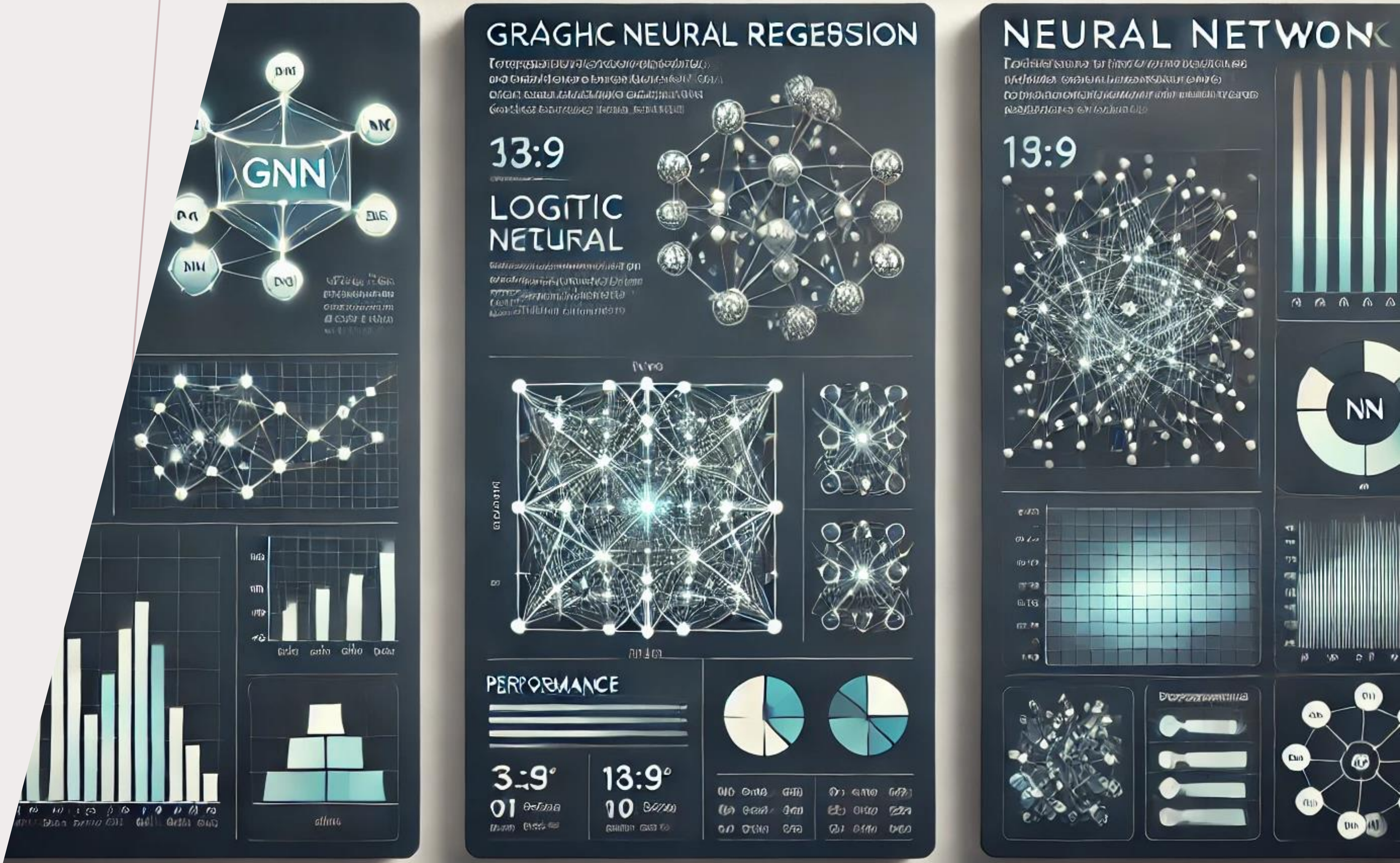
Evaluation: Based on data provided in the Test and validation dataset, the model will be evaluated on the following metrics

- Accuracy
- F1
- Precision
- Recall

GNN ARCHITECTURE

```
class GCN(nn.Module):

    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GCN, self).__init__()

        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.conv3 = GCNConv(hidden_channels, hidden_channels)
        self.conv4 = GCNConv(hidden_channels, out_channels)

    def forward(self, x, edge_index):
        self.dropout = nn.Dropout(0.5)
        x = torch.relu(self.conv1(x, edge_index))
        x = self.dropout(x)
        x = torch.relu(self.conv2(x, edge_index))
        x = self.dropout(x)
        x = torch.relu(self.conv3(x, edge_index))
        x = self.dropout(x)
        return self.conv4(x, edge_index)

class LinkPredictor(nn.Module):

    def __init__(self, in_channels):
        super(LinkPredictor, self).__init__()

        self.lin = nn.Linear(in_channels, 1)

    def forward(self, z, edge_index):
        z_i = z[edge_index[0]]
        z_j = z[edge_index[1]]

        return torch.sigmoid((z_i * z_j).sum(dim=-1))
```

RESULTS


GNN
GRAGHC NEURAL REGESSION
13:9
LOGITIC
NETURAL
PERPORMANCE
3:9
13:9
01
10
NEURAL NETWON
13:9
NN

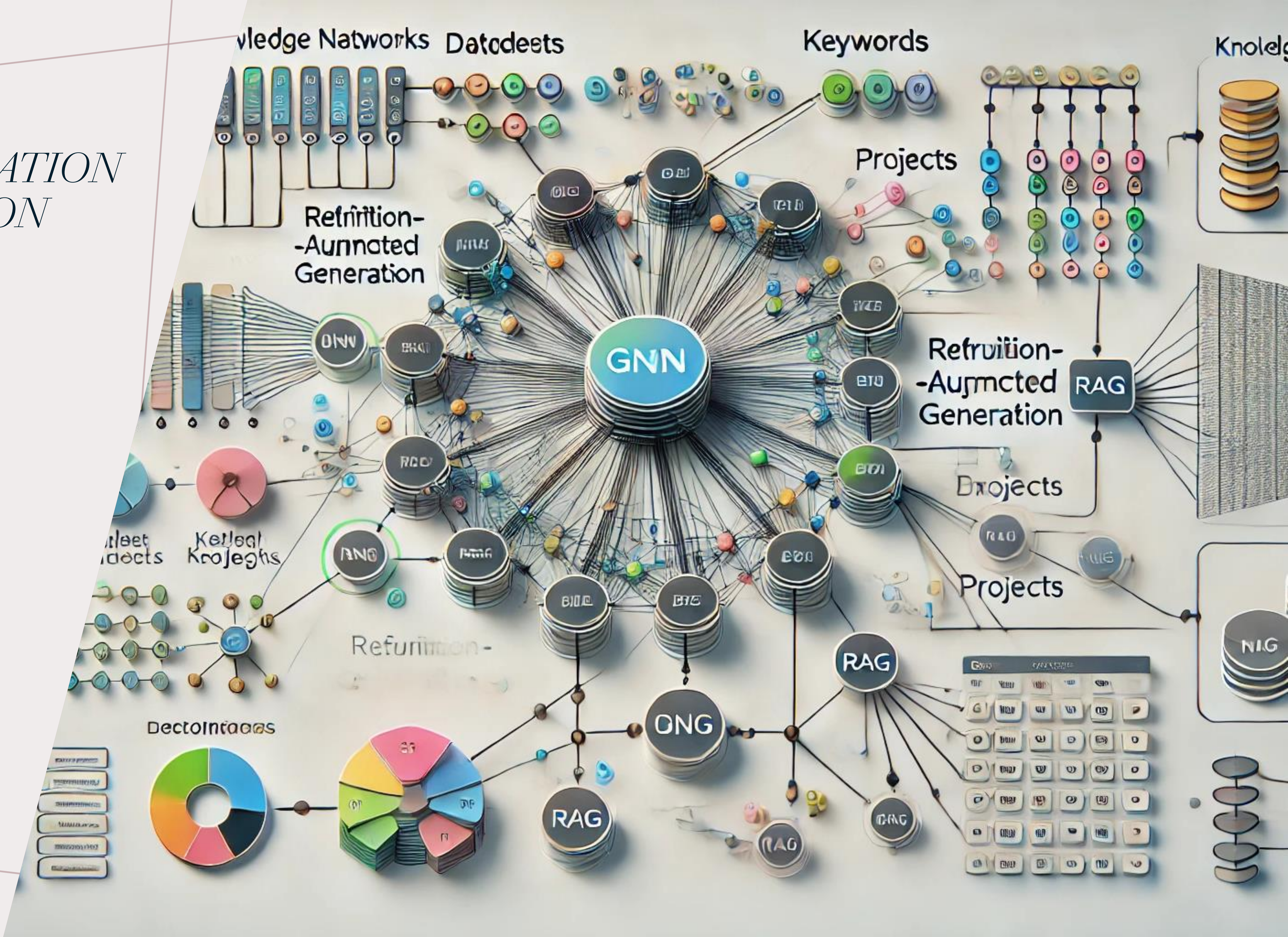
TEST DATASET

Method	Accuracy	Recall	Precision	F1
Jaccard Coefficient (Baseline)	0.701	0.701	1	N/a
Node2Vector Logistic Regression(Edge Embedding)	0.842	0.842	1	0.91
Node2Vector NN(Vector Embedding)	0.05	0.05	1	0.10
GNN	0.991	0.991	1	0.995

VALIDATION DATASET

Method	Accuracy	Recall	Precision	F1
Jaccard Coefficient (Baseline)	0.71	0.71	1	0.83
Node2Vector Logistic Regression(Edge Embedding)	0.80	0.80	1	0.89
Node2Vector NN(Vector Embedding)	0.20	0.20	1	0.33
GNN	0.994	0.994	1	0.997

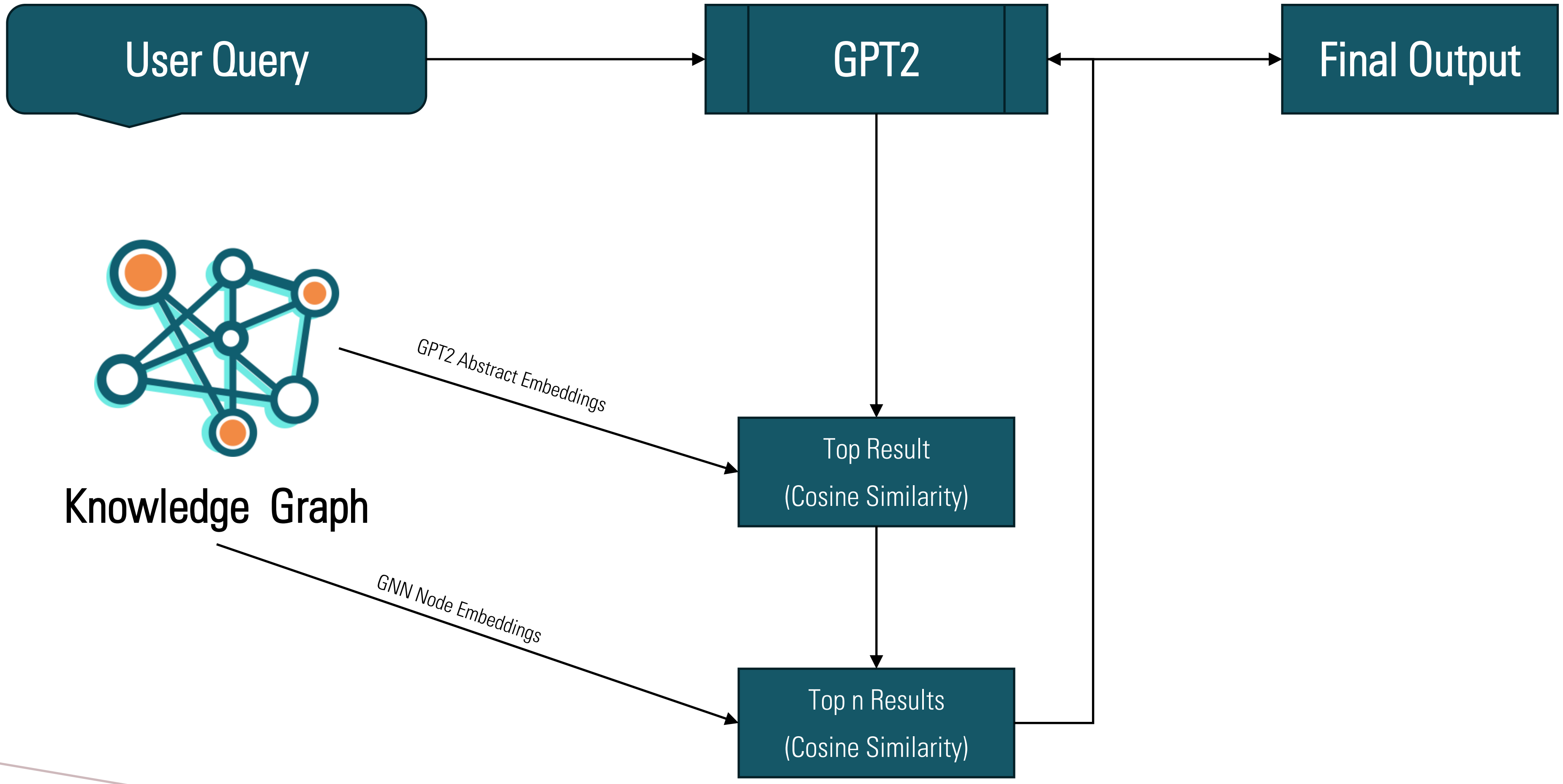
RECOMMENDATION AUGMENTATION USING RAGS



RAG IMPLEMENTATION

- **Prepare the Data for RAG**
  - Graph Creation
  - Collect text-based data sources(publications, dataset descriptions, and project details)
- **Prepare Node Embeddings for GNN Model & Train Model**
- **Set Up the Retrieval System**
- **Retrieval Phase - Find Relevant Context**
- **Generation Phase - Use Generative Model for Enrichment**
- **Evaluation and Testing**
  - Check retrieval accuracy by manually reviewing whether the retrieved documents are relevant.

RAG LOGICAL FLOW

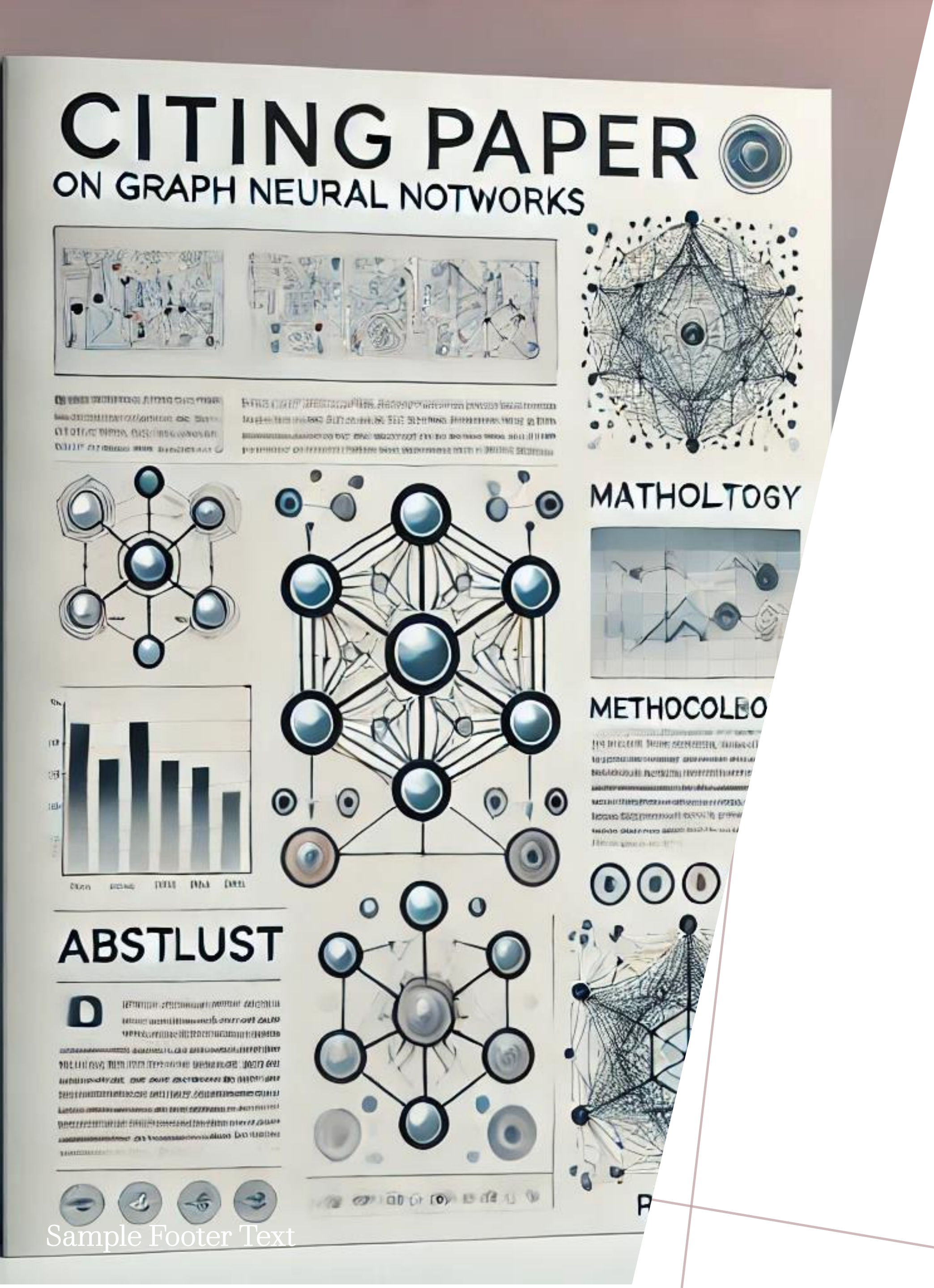


RAG RESULTS

- Implementing the RAG using the Context from the GNN and GPT2 was a failure.
- GPT2 model is not complex enough to handle large input token sizes and has a smaller output token size length.
- Alternatively, using GPT2 to embed the Query and using cosine similarity to get the top 10 results within graph gave better results.
- Due to Memory & Cost considerations Larger & newer LLM's were not used to implement the RAG.

FUTURE WORKS

- Implement the RAG using Larger LLM’s.
- Using the Larger Dataset, creating a foundational GNN Model.



PREVIOUS WORK

- GNN
  - How Powerful are Graph Neural Networks?
    - [arXiv:1810.00826v3]
  - Graph Neural Networks: A Review of Methods and Applications
    - [arXiv:1812.08434v6]
- GNN with RAG
  - GNN-RAG: Graph Neural Retrieval for Large Language Model Reasoning
    - [arXiv:2405.20139v1]
  - Don't Forget to Connect! Improving RAG with Graph-based Reranking
    - [arXiv:2405.18414v1]


ANY QUESTIONS
ANY
QUESTIONS
ANY
QUESTIONS
ANY
QUESTIONS
ANY
QUESTIONS
ANY
ANY
QUESTIONS
ANY


Thank you
THANK YOU