Language from Links: Optimizing Graph Embeddings for Text Generation using Large Language Models

Samidha Verma
Yardi School of Artificial Intelligence
IIT Delhi
aiz238705@iitd.ac.in

Aryan Dua
Dept. Computer Science and Engineering
IIT Delhi
cs5200475@iitd.ac.in

Abstract

Graph-to-text generation focuses on creating coherent and natural language text from graph-based data. This study explores the application of decoder-only pretrained language models (PLMs), specifically GPT-2, and evaluates the effect of task-specific pretraining strategies on their performance in graph-to-text generation tasks. The work covers the domain of Wikipedia-based knowledge graphs, WEBNLG.

1 Introduction

Graph-to-text generation is a challenging task at the intersection of natural language processing (NLP) and graph representation learning. Converting structured knowledge from graphs into coherent, human-like text has applications in automatic report generation, knowledge summarization, and question answering. This report investigates the effectiveness of large pretrained language models (PLMs), focusing on GPT-2, in generating text from knowledge graphs. We also explore task-specific pretraining and fine-tuning strategies to enhance performance, particularly in domains such as Wikipedia-based knowledge graphs and scientific datasets.

Our contributions are threefold:

- **Model Adaptation:** We adapt GPT-2 to the graph-to-text task by integrating graph embeddings.
- **Task-specific Pretraining:** We introduce pretraining strategies using domain-specific corpora.
- **Evaluation:** We provide a comprehensive evaluation using standard datasets and metrics, comparing our approach with existing methods.

2 Related Work

Graph-to-text generation has been studied extensively in recent years, with approaches ranging from rule-based systems to deep learning models:

- **Rule-based Systems:** Early methods relied on templates and rules to convert graph data into text. These systems often lack flexibility and require extensive domain-specific knowledge.
- **Neural Network Models:** Sequence-to-sequence (Seq2Seq) architectures, particularly those using attention mechanisms, have shown promise. Models like Transformer and its variants have set new benchmarks.
- **Pretrained Language Models (PLMs):** More recently, PLMs such as BART and T5 have been fine-tuned for graph-to-text tasks. However, decoder-only models like GPT-2 have received less attention in this context.

Our work extends previous research by applying GPT-2 with graph embeddings and evaluating task-specific finetuning strategies, aiming for improved performance in specialized domains.

3 Problem Formulation

Task 1: The task of graph-to-text generation can be formulated as a conditional text generation problem where the goal is to generate a natural language sentence or paragraph Y, summarizing the graph's content.

$$Y = \{y_1, y_2, \ldots, y_m\}$$

given a structured graph with nodes (entities) and edges (relationships).

$$G = (V, E).$$

Here:

- $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of nodes in the graph.
- $E \subseteq V \times V$ represents the set of edges, with each edge $e \in E$ optionally labeled by a relation r.

Objective

Given G, the objective is to model the conditional probability:

$$P(Y \mid G; \theta)$$

where θ represents the parameters of the model. The generated text Y should be:

1. **Fluent**: Grammatically correct and coherent in natural language.
2. **Faithful**: Consistently reflect the semantics of the input graph G.

Formally, we seek a function $f : G \to Y$, where f is implemented using a pretrained language model adapted for this task.

Pretraining and Fine-tuning

Let

$$D = \{(G_i, Y_i)\}_{i=1}^{N}$$

be a dataset of N graph-text pairs, where each pair consists of a graph G_i and a corresponding reference text Y_i. The model is trained to maximize the conditional log-likelihood:

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{|Y_i|} \log P(y_t \mid y_{<t}, G_i; \theta)$$

where $y_{<t}$ denotes the sequence of tokens preceding y_t.

Task-Specific Adaptation

For task-specific pretraining, additional task-relevant data

$$D' = \{(G'_j, Y'_j)\}_{j=1}^{M}$$

is used to adapt the model. This involves:

1. **Language Model Adaptation (LMA)**: Training on textual data $\{Y'_j\}$ without considering graph structures, optimizing:

$$\mathcal{L}_{\text{LMA}}(\theta) = \sum_{j=1}^{M} \sum_{t=1}^{|Y'_j|} \log P(y'_t \mid y'_{<t}; \theta).$$

2. **Supervised Task Adaptation (STA)**: Fine-tuning on graph-text pairs D', optimizing:

$$\mathcal{L}_{\text{STA}}(\theta) = \sum_{j=1}^{M} \sum_{t=1}^{|Y'_j|} \log P(y'_t \mid y'_{<t}, G'_j; \theta).$$

4 Methodology

4.1 Task 1

In this task, we aimed to replicate and extend the work presented in the paper by (Ribeiro et al., 2021). The original study utilized BERT and T5 architectures to translate graphs into coherent English sentences. Our objective was to adapt and implement this methodology using the GPT-2 model (on the WebNLG Dataset).

4.1.1 Initial Implementation

Our initial approach involved modifying the existing codebase provided by the authors to make it compatible with GPT-2. However, the results were unsatisfactory:

- **Qualitative Assessment**: Generated outputs frequently deviated from the input graphs and contained numerous grammatical errors.
- **Quantitative Evaluation**: Performance metrics, including BLEU and ROUGE scores, indicated poor alignment with reference outputs.

4.1.2 Custom Implementation

To address these shortcomings, we implemented the approach from scratch, closely following the methodology outlined in the original paper. The key modifications and strategies applied were as follows:

1. **Tokenizer Adaptation**:
   - Three special tokens, `<H>`, `<R>`, and `<T>`, representing head, relation, and tail entities, respectively, were added to the GPT-2 tokenizer.
   - All edge labels were also incorporated into the tokenizer's vocabulary to ensure accurate representation of graph structures.
2. **Left Padding**:
   Given that GPT-2 is a decoder-only architecture, the padding mode was adjusted to `left` to align input sequences properly during training.

3. **System Instruction**:
   A system-level prompt, "Convert the triple to grammatically correct english sentence:," was prepended to the input. This contextual instruction significantly improved the model's performance, as reflected in BLEU scores.

4. **Training Configuration**:

   - **Optimizer**: We employed the Adam optimizer with an initial learning rate of 3×10^{-5}.
   - **Learning Rate Schedule**: A linear decay schedule without warm-up was used.
   - **Hyperparameter Selection**: Batch sizes and beam search sizes were optimized based on development set performance, selected from the sets $\{2, 4, 8\}$ and $\{1, 3, 5\}$, respectively. The model achieving the highest BLEU score on the development set was chosen for final evaluation.

4.1.3 Post-processing of Generations

While this custom implementation outperformed our initial attempt, achieving better qualitative coherence, the BLEU scores remained suboptimal. Consequently, we integrated successful strategies from our custom implementation back into the original codebase. Additionally, we applied post-processing techniques to the generated outputs, including:

- Removal of redundant text fragments.
- Trimming extraneous content beyond the generated sentence boundary.

These refinements led to a substantial improvement in quantitative performance, yielding a BLEU score of **23%** on the `test_all` dataset (Refer 1).

4.2 Task 2: Graph Embedding Generation

4.2.1 Objective

The objective of this task was to investigate whether generating specialized graph embeddings can enhance the performance of language models (LMs) when processing graph-structured data. We hypothesized that graph embeddings may retain more structural information compared to text-based representations. We tried two methods for this task.

4.2.2 Methodology - Method 1

1. In the first approach, each input graph was initially linearized into a specific format, `<H> node1, <R> edge, <T> node2`, where `node1`, `edge`, and `node2` represent the entities and relations present in the graph. The purpose of this linearization is to encode both the structural and semantic information of the graph in a text-based format that can be processed by natural language models.

2. To facilitate graph processing, a NetworkX graph object was created for each linearized graph. The node-level information, representing the entities in the graph, was encoded as node attributes, while the relation information, describing the connections between entities, was encoded as edge attributes (specifically, edge types). This representation captured the relationships between entities in a structured format suitable for subsequent graph processing tasks.

3. The NetworkX graph was then converted into a PyTorch Geometric (PyG) graph object, which is more suitable for graph neural network (GNN) operations. The PyG format enables efficient manipulation of graph-structured data and supports operations like message passing, which are essential for learning node and graph-level representations.

4. The graph was passed through a graph encoder, particularly GCN (Kipf and Welling, 2016), a neural network model that generates embeddings for each node in the graph. The node embeddings capture both the structural information (i.e., the position and connectivity of the node) and the semantic information (i.e., the entity represented by the node). To obtain a single representation for the entire graph, the embeddings of all nodes were aggregated using mean pooling, where the node embeddings were averaged to form a single graph-level embedding.

5. This graph embedding, which is a fixed-size vector representing the entire graph, was then passed along with a text prompt to the GPT-2 model. The GPT-2 model, which is a pre-trained language model, used the graph embedding (and the textual graph) to guide the generation of a target sentence. The idea was

that the graph's structural and semantic information, encoded in the graph embedding, would provide context for GPT-2 to generate a relevant textual description or prediction based on the given prompt.

6. Despite this structured approach, the performance of this method was not satisfactory. The graph-based embedding did not provide sufficient or relevant context for the GPT-2 model to effectively generate meaningful output. This highlighted the limitations of relying solely on a graph encoding approach for generating high-quality textual data without additional refinements in model architecture or training procedure.

4.2.3 Methodology - Method 2

1. **Graph Creation:** To generate embeddings that effectively capture the structure of graph-structured data, we first constructed a comprehensive global graph by aggregating individual linearized graphs from the dataset. Each linearized graph follows the format `<H> node1, <R> edge, <T> node2`, where `node1`, `edge`, and `node2` represent entities and relations within the graph.

   Node and Edge Representation:

   - Each component `node1`, `edge`, `node2` in a linearized graph is treated as a node in the overall graph.
   - We created directed edges to represent the relationships between these entities:
     - A directed edge is established from `node1` to `edge`.
     - Another directed edge connects `edge` to `node2`.
     - The node features were initialized with the average GPT-2 embeddings of all tokens corresponding to the text content of that node or edge.
     - Further, we assigned GPT2 embeddings to different edges as edge attributes. Specifically we encoded relations such as `'subject to relation'`, `'relation to object'`, `'subject to category'` and `'category to global'`.

   Global Graph Structure: To ensure comprehensive connectivity and facilitate information propagation across the graph, we introduced an entity registry and a global node:

   - The **entity registry** contains all nodes (entities) present in the dataset (nodes + edges in the linearized graph), ensuring unique representation.
   - A **global node** is introduced, which connects to all category nodes. Each category node, in turn, connects to all individual nodes (`node1` and `node2`) within that category. This hierarchical structure helps capture broader relationships and contextual information within the graph.

   Example: Consider the following linearized graphs representing relationships between entities:

   - `<H> Azerbaijan <R> leader <T> Artur_Rasizade`
   - `<H> Azerbaijan <R> leaderTitle <T> Prime_Minister`
   - `<H> Abilene_Regional_Airport <R> cityServed <T> Abilene,_Texas`

   The corresponding graph structure is depicted in Figure 1, illustrating how nodes and edges are connected to represent the relationships within the data.

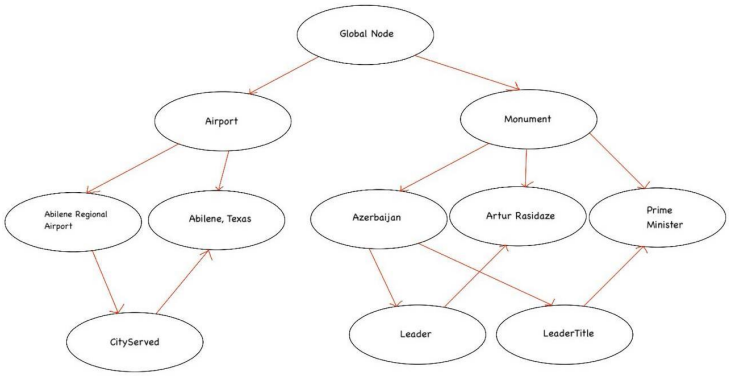


Figure 1: Graph representation of linearized data

2. **Graph Embedding Generation and Training:** The constructed graph is highly heterogeneous, containing both node and edge types. These are challenging graphs to encode; hence, we tried to leverage a specialized graph transformer architecture to encode the graph's structural and semantic information in

the highly heterogeneous graph space instead of resorting to vanilla GCN.

Graph Embeddings using GNN and Graph-Transformer: The first step in our approach involves learning graph embeddings using a graph transformer. We adopt the same graph transformer proposed in (Shi et al., 2020), which enables the extraction of high-quality graph embeddings that encapsulate both the graph structure and node semantics. The learned embeddings are then used as input to GPT-2 for text generation.

Unsupervised GNN Loss: We use an unsupervised loss function for the GNN training. Specifically, we define two types of edges for training:

- **Positive edges:** These are the actual edges in the graph, representing relationships between nodes (entities).
- **Negative edges:** These are randomly sampled non-edges, which are pairs of nodes that do not have a direct relationship.

This approach ensures that the GNN learns to differentiate between valid and invalid relationships, encoding both structural and semantic information in the graph. This is the loss function that we use: $J_G(z_u) = -\log\left(\sigma(z_u^\top z_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)}\left[\log\left(\sigma(-z_u^\top z_{v_n})\right)\right]$ where:

- σ is the sigmoid function,
- P_n is a negative sampling distribution,
- Q defines the number of negative samples,
- z_u and z_v are the embeddings for nodes u and v, respectively,
- v_n is a node that co-occurs near u on fixed-length random walks,
- $\mathbb{E}_{v_n \sim P_n(v)}$ denotes the expectation over negative samples v_n drawn from the distribution $P_n(v)$.

Explanation: This loss function is used to learn useful, predictive representations in a fully unsupervised setting. The goal is to apply a graph-based loss function to the output representations z_u for each node $u \in V$ and tune the weight matrices W^k of the aggregator functions via stochastic gradient descent. The loss function encourages nearby nodes to have similar representations, while ensuring that the representations of disparate nodes are highly distinct.

In this context, the representations z_u are generated from the features contained within a node's local neighborhood, rather than training a unique embedding for each node (via an embedding look-up). This approach is particularly useful for situations where node features are provided to downstream machine learning applications but are to be used only for a specific task (e.g., cross-entropy loss for classification).

Entity Registry and Node Identification: To facilitate the graph-based learning process, we maintain an **entity registry** that keeps track of the nodes in the graph. A dictionary is used to map each node to its corresponding node ID, which is crucial for efficiently retrieving node embeddings during the forward pass of the GNN.

Embedding Generation from GNN: During the forward pass through the GNN, we obtain an embedding for each node in the graph. For a given triple (node1, edge, node2), we can derive an embedding that combines the embeddings of the nodes and their relationship (edge). The resulting embedding is a concatenation of the node embeddings and edge embedding, forming a representation of the entire triple.

Integration with GPT-2: The generated triple embedding is then passed through a Multilayer Perceptron (MLP), which processes the 3D-dimensional embedding to obtain a vector suitable for input into GPT-2 (batch size * 3 * GPT embedding size). The MLP output is concatenated with a separator embedding followed by the target embeddings generated by GPT-2, which enables the model to effectively condition the text generation on the graph embeddings.

Training the MLP: The GNN is frozen during training, and only the MLP is trained. This ensures that the graph structure is preserved while the MLP learns to map the graph embeddings to an appropriate representation for

GPT-2.

Text Generation with GPT-2: For the text generation task, we concatenate the input embedding (which represents the graph triple) with the target text embedding. This concatenated vector is then passed to GPT-2, which generates the corresponding text description of the graph structure, similar to the task of soft prompting in language models. The model is fine-tuned using the standard objective $P(y|y_{<i})$, where y is the target sentence.

Handling Seen and Unseen Triples: During inference, the triples presented to the model can be either seen (previously encountered during training) or unseen (new triples). For seen triples, we directly use the embeddings learned by the GNN. For unseen triples:

- If the triple's relation has been encountered but the nodes are new, the triple is added to the graph.
- If the triple's relation and nodes are both new, the entire triple is added to the graph.

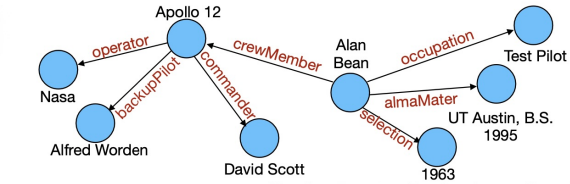
Once the graph is updated, the new embeddings are computed, and the GNN performs a forward pass to obtain the updated graph embeddings.

Generation of Descriptions: After the graph is updated and embeddings are recalculated, we concatenate the new graph embeddings with a separator token and feed the combined representation into GPT-2. The model then generates a text description of the newly added or updated graph structure, providing a natural language interpretation of the graph.

5 Experimental Setup

5.1 Dataset

For the purpose of this study, we focus on the WebNLG dataset (Gardent et al., 2017). Each instance in the WebNLG dataset consists of a knowledge graph (KG) derived from DBPedia (Auer et al., 2007) paired with one or more target sentences describing the graph. The test set is divided into two partitions: seen, which includes only DBPedia categories present in the training data, and unseen, which comprises categories not encountered during training. The combined test set is referred to as all.



Linearized representation: **<H>** Apollo 12 **<R>** backup pilot **<T>** Alfred Worden **<H>** Alan Bean **<R>** was a crew member of **<T>** Apollo 12 **<H>** Apollo 12 **<R>** operator **<T>** NASA **<H>** Alan Bean **<R>** occupation **<T>** Test pilot **<H>** Apollo 12 **<R>** commander **<T>** David Scott **<H>** Alan Bean **<R>** was selected by NASA **<T>** 1963 **<H>** Alan Bean **<R>** alma Mater **<T>** UT Austin B.S. 1955
Text: Alan Bean graduated from UT Austin in 1955 with a Bachelor of Science degree. He was hired by NASA in 1963 and served as a test pilot. Apollo 12's backup pilot was Alfred Worden and was commanded by David Scott.

Figure 2: Example of the linearised WebNLG graphs (Ribeiro et al., 2021), the input for the GPT-2 model, and the corresponding text.

In line with the baseline paper (Ribeiro et al., 2021), special tokens ⟨H⟩, ⟨R⟩, and ⟨T⟩ are added before the head entity, relation, and tail entity of each triple, respectively (refer Fig. 2) for Task 1 (refer section 4.1.3). Further, in order to combine actual structural information along with the textual information in the graph we utilize the graph structure as defined in section 4.2.3.

5.2 Metrics

BLEU Score: The Bilingual Evaluation Understudy (BLEU) score (Papineni et al., 2002) measures the similarity between machine-generated text and reference text using n-gram overlap. It is defined as:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \quad (1)$$

where p_n is the modified precision for n-grams of size n, w_n is the weight assigned to each n-gram, and N is the maximum n-gram length (typically 4). The brevity penalty (BP) is given by:

$$BP = \begin{cases} 1, & \text{if } c > r, \\ e^{(1-r/c)}, & \text{if } c \leq r, \end{cases} \quad (2)$$

where c is the length of the candidate text and r is the effective reference length. BLEU scores range from 0 to 1, with higher scores indicating better alignment with the reference.

ROUGE Score: The Recall-Oriented Understudy for Gisting Evaluation (ROUGE) (Lin, 2004) is a set of metrics that evaluate text quality based on the overlap between a candidate and reference. Key variants include:

ROUGE-N: Measures n-gram recall, defined as:

$$\text{ROUGE-N} = \frac{\sum_{S \in \text{Refs}} \sum_{\text{ng} \in S} \min(\text{C}_{\text{cand}}(\text{ng}), \text{C}_{\text{ref}}(\text{ng}))}{\sum_{S \in \text{Refs}} \sum_{\text{ng} \in S} \text{C}_{\text{ref}}(\text{ng})}. \tag{3}$$

where $\text{C}_{\text{cand}}(\text{ng})$ is the number of occurrences of an n-gram in the candidate, and $\text{C}_{\text{ref}}(\text{ng})$ is its count in the reference.

ROUGE-L: Measures the longest common subsequence (LCS) between the candidate and reference, capturing sentence-level structure. It is computed as:

$$\text{ROUGE-L} = F_\beta = \frac{(1+\beta^2) \cdot \text{R} \cdot \text{P}}{\beta^2 \cdot \text{R} + \text{P}}, \tag{4}$$

where recall (R) and precision (P) for LCS are:

$$\text{R} = \frac{\text{LCS}}{\text{Length of ref.}}, \quad \text{P} = \frac{\text{LCS}}{\text{Length of cand.}}. \tag{5}$$

The parameter β determines the relative importance of recall and precision (commonly set to $\beta = 1$).

Higher ROUGE-N and ROUGE-L scores signify greater similarity to the reference text.

5.3 Compute:

For Task 1, replicating results of (Ribeiro et al., 2021) with GPT-2 we used NVIDIA DGX Station with four V100 GPU cards having 128GB GPU memory, 256GB RAM, and a Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz running in Ubuntu 18.04.6 LTS.

All experiments for Task 2, incorporating graph structure, were performed on an Intel Xeon Gold 6248 processor with 96 cores and 1 NVIDIA A100 GPU with 40GB memory and 377 GB RAM with Ubuntu 18.04.

6 Results & Analysis

6.1 Task 1

Table 1 covers the results obtained for task 1 using GPT-2, followed by the results using T5 and BART as report in (Ribeiro et al., 2021). We see

Model	All	Seen	Unseen
GPT2	23	30.87	12.91
$\text{BART}_{\text{base}}$	53.11	62.74	41.53
$\text{BART}_{\text{large}}$	54.72	63.45	43.97
T5_{small}	56.34	65.05	45.37
T5_{base}	59.17	64.64	52.55
T5_{large}	59.70	64.71	53.67

Table 1: Comparison of GPT-2 with baseline models on BLEU scores on the WebNLG dataset.

that the decoder-only model GPT2 is not as good at converting the graph to the corresponding text. Although adding in the system instruction did help in getting better and comparatively more grammatical outputs vs. not using them, qualitatively, we observed that there is significant scope for improvement in the fluency and grammatical correctness of the implementation still. The rouge scores of our model are given in the App. A.

6.2 Task 2

- Although we have not yet conducted comprehensive testing, the model pipeline is fully implemented and operational.
- Below are some sample predictions generated by the model:
  1. "biased biased biased biased biased"
  2. "ancialancialancialancialancialancial"
  3. "610 610 610 610 610 610 610"
- The current outputs are either empty, repetitive, or nonsensical.
- To improve performance, we may need to combine text input embeddings with their corresponding graph embeddings. Using graph embeddings alone has not yielded meaningful learning outcomes.

Potential Improvements: Future enhancements to this methodology may involve tokenizing the triples themselves and concatenating the graph embeddings directly with these tokenized triples. Also, adding the system instruction proved useful in getting slightly more grammatical outputs in task 1, hence we can also add that for task 2. This could further improve the quality of the generated text and make the graph-to-text generation process more seamless.

7 Conclusion

In this study, we explored the application of the GPT-2 model for graph-to-text generation, replicating and extending the methodology from Ribeiro et al. (2020). Through an iterative process, we identified key challenges in adapting the existing codebase and addressed them by implementing custom solutions, such as tokenizer adaptations, left padding, and contextual system instructions. These enhancements led to significant improvements, culminating in a BLEU score of 23% on the `test_all` dataset.

Furthermore, we investigated the potential of specialized graph embeddings to enhance language model performance. Despite the structured approach and sophisticated graph-based methods, results indicated that additional architectural refinements and training strategies are necessary for substantial gains. Overall, our findings underscore the complexity of integrating graph structures with language models and highlight areas for future research, including more advanced graph encoding techniques and hybrid architectures.

Contribution

Samidha Verma:

1. Implemented task 1 by making changes to the repository mentioned in the codebase given to us. (This gave better results).
2. Implemented task 2 using Method 2 as described above.
3. Global Graph construction implementation
4. Report writing and PPT

Aryan Dua:

1. Implemented task 1 from scratch using recent packages.
2. Implemented task 2 using Method 1 as described above.
3. Preprocessing of input files
4. Report writing and PPT

References

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The Semantic Web*, pages 722–735, Berlin, Heidelberg. Springer Berlin Heidelberg.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain. Association for Computational Linguistics.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Leonardo F. R. Ribeiro, Martin Schmitt, Hinrich Schütze, and Iryna Gurevych. 2021. Investigating pretrained language models for graph-to-text generation. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*, pages 211–227, Online. Association for Computational Linguistics.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. 2020. Masked label prediction: Unified message passing model for semi-supervised classification.

A Appendix

A Task 1: ROUGE Score Evaluation

This section provides the detailed ROUGE score results for both seen and unseen data, as well as the combined performance across all data points. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics commonly used to evaluate the quality of text generation models by comparing generated outputs against reference texts.

A.0.1 Seen Data ROUGE Scores

The ROUGE scores for the seen data are as follows:

- **ROUGE-1:** Precision = 0.4558, Recall = 0.6257, F1 = 0.5125
- **ROUGE-2:** Precision = 0.2670, Recall = 0.3798, F1 = 0.3042

- **ROUGE-L:** Precision = 0.3467, Recall = 0.4876, F1 = 0.3935

A.0.2 Unseen Data ROUGE Scores

The ROUGE scores for the unseen data are as follows:

- **ROUGE-1:** Precision = 0.3033, Recall = 0.3971, F1 = 0.3333
- **ROUGE-2:** Precision = 0.1084, Recall = 0.1503, F1 = 0.1222
- **ROUGE-L:** Precision = 0.2220, Recall = 0.2960, F1 = 0.2455

A.0.3 Overall ROUGE Scores

The ROUGE scores across both seen and unseen data combined are as follows:

- **ROUGE-1:** Precision = 0.3828, Recall = 0.5163, F1 = 0.4268
- **ROUGE-2:** Precision = 0.1911, Recall = 0.2700, F1 = 0.2171
- **ROUGE-L:** Precision = 0.2870, Recall = 0.3959, F1 = 0.3227