The Graph Language: How Knowledge Graphs Speak to Large Language Models Supplemental Material

Giuseppe Pirrò

DeMaCS, University of Calabria
giuseppe.pirro@unical.it

1 Dataset Description

The diverse question-answering datasets in Table 1 provide a comprehensive testbed for evaluating GRALAN's ability to reason over structured knowledge and generate accurate responses. We now analyze the key characteristics of these datasets and explore why they are important for showcasing GRALAN's capabilities. First, we have the open-domain datasets: PQ-2hop, PQ-3hop, PQL-2hop, PQL-3hop, and MetaQA 1/2/3-hop. These datasets are built upon the Freebase and WikiMovies knowledge graphs, which contain a vast amount of factual information across various domains. Freebase, for example, is a large collaborative knowledge base consisting of data about real-world entities, such as people, places, and organizations, along with their properties and relationships. WikiMovies, on the other hand, focuses specifically on information related to movies, such as actors, directors, and plot details.

The PQ (Path Question) datasets involve answering questions that require multi-hop reasoning over the knowledge graph. For instance, PQ-2hop questions might ask "Who directed the movie that starred [actor]?", requiring the model to first find the movie the actor starred in and then identify the director of that movie. Similarly, PQL datasets introduce an additional challenge by requiring the model to figure out missing links in the knowledge graph before answering the question.

MetaQA datasets are designed to evaluate multi-hop question answering at different levels of difficulty. The 1-hop questions can be answered by directly querying the knowledge graph, while 2-hop and 3-hop questions require the model to traverse multiple edges to arrive at the answer. These datasets test GRALAN's ability to perform multi-step reasoning and navigate complex graph structures.

Next, we have the ZEROshotRE and ComplexWebQuestions datasets, which are built on top of the Wikidata 5M knowledge graph. Wikidata is a free and open knowledge base that covers a wide range of topics, including people, places, events, and concepts. ZEROshotRE focuses on relation extraction, testing the model's ability to identify and extract relations between entities in a zero-shot setting, where the model has not seen the specific relation during training. ComplexWebQuestions, as the name suggests, consists of complex questions that require a deeper understanding of the relationships between entities in the knowledge graph.

Lastly, we have the multiple-choice datasets: OBQA, ARC, RIDDLE, and PIQA. These datasets utilize the ConceptNet knowledge graph, which is a semantic network

Datasets	Task	KG	Underlying KG Stats			Question Sets		
			#Entity	#Relation	#Triple	#Train	#Valid	#Test
PQ-2hop	Open QA	Freebase	1,056	13	1,211	1,526	190	192
PQ-3hop	Open QA	Freebase	1,836	13	2,839	4,158	519	521
PQL-2hop	Open QA	Freebase	5,034	363	4,247	1,276	159	159
PQL-3hop	Open QA	Freebase	6,505	411	5,597	825	103	103
MetaQA 1-hop	Open QA	WikiMovies	43,234	9	134,741	96,106	9,992	9,947
MetaQA 2-hop	Open QA	WikiMovies	43,234	9	134,741	118,980	14,872	14,872
MetaQA 3-hop	Open QA	WikiMovies	43,234	9	134,741	114,196	14,274	14,274
ZEROshotRE	Open QA (LLM comparison)	Wikidata 5M	∼4.8M	822	20.6M	147,909	3,724	4,966
ComplexWebQuestions	Open QA (LLM comparison)	Wikidata 5M	∼4.8M	822	20.6M	27,734	3,480	3,531
OBQA	Multiple-choice QA	ConceptNet	∼2M	17	∼2.4M	4,957	500	500
ARC	Multiple-choice QA	ConceptNet	∼2M	17	∼2.4M	2,590	299	1,172
RIDDLE	Multiple-choice QA	ConceptNet	∼2M	17	∼2.4M	3,510	1,021	1,184
PIQA	Multiple-choice QA	ConceptNet	∼2M	17	∼ 2.4M	16,113	1,838	3,084

Table 1: Dataset statistics across different question-answering tasks.

that captures commonsense knowledge about the world. ConceptNet contains concepts and their relationships, such as "apple is a fruit" or "pencil is used for writing". The multiple-choice datasets test GRALAN's ability to reason over this commonsense knowledge to answer questions that require a deeper understanding of the world.

The OBQA (Open Book Question Answering) dataset consists of questions that require both understanding of a given text and commonsense reasoning to answer correctly. ARC (AI2 Reasoning Challenge) focuses on grade-school level science questions that often involve complex reasoning and knowledge beyond what is directly stated in the question. RIDDLE is a dataset of commonsense reasoning questions that evaluate the model's ability to infer implicit knowledge and understand the relationships between entities. PIQA (Physical Interaction Question Answering) tests the model's understanding of physical commonsense knowledge, such as how objects interact with each other.

By evaluating GRALAN on these diverse datasets, we can assess its ability to integrate knowledge from structured knowledge graphs with the reasoning capabilities of language models. The open-domain datasets challenge GRALAN to perform multi-hop reasoning and navigate large-scale knowledge graphs, while the multiple-choice datasets require commonsense reasoning and understanding of implicit relationships.

Moreover, the inclusion of train, valid, and test splits in these datasets[1] allows for a rigorous evaluation of GRALAN's performance. The train split is used to fine-tune the model on the specific task, the valid split is used for hyperparameter tuning and model selection, and the test split provides an unbiased assessment of the model's generalization ability.

2 Detailed Pseudocode

We start by providing the pseudocode for question-focused subgraph extraction. Then, the GRALAN training pseudocode outlines the process of training the GRALAN model

[1] https://github.com/michiyasunaga/dragon

Algorithm 1 QUESTIONFOCUSEDBFS

procedure QUESTIONFOCUSEDBFS(dataset, max_depth, min_k, k_fraction, query(subject, predicates))
  Load or process graph data into `adjacency_list`, `entity_dict`, `relation_dict`
  Load or compute `relatedness_matrix` and `pred_to_idx`
  Initialize empty sets: `nodes` $\leftarrow \emptyset$, `edges` $\leftarrow \emptyset$, `visited` $\leftarrow \emptyset$
  Initialize queue $Q \leftarrow [(subject, 0)]$
  while $Q \neq \emptyset$ **do**
    $(u, d) \leftarrow$ pop(Q)
    if $d <$ max_depth **then**
      Add u to `nodes`
      $\mathcal{N} \leftarrow []$ ▷ Gather neighbor similarities
      for each (v, r) in `adjacency_list`[u] **do**
        $s \leftarrow$ `relatedness_matrix`[relation][`pred_to_idx`[r]]
        Append (v, r, s) to $\mathcal{N}$
      $k \leftarrow \max\big(\text{min\_k}, \lfloor |\mathcal{N}| \times \text{k\_fraction} \rfloor\big)$
      $\mathcal{T} \leftarrow$ top k elements of $\mathcal{N}$ by similarity s
      for each $(v, r, \_)$ in $\mathcal{T}$ **do**
        Add edge (u, r, v) to `edges`
        if $v \notin$ `visited` **then**
          Push $(v, d + 1)$ into Q
          Add v to `visited`
  return `nodes`, `edges`

on a dataset $\mathcal{D}$ consisting of triplets (G_i, Q_i, A_i), where G_i represents a question-focused subgraph of an original knowledge graph, Q_i is a question, and A_i is the corresponding answer.

The TRAIN procedure takes the dataset $\mathcal{D}$ as input and iterates over a specified number of epochs. Within each epoch, the dataset is divided into batches $\mathcal{B}$ of size B. The purpose of using batches is to improve computational efficiency and enable parallel processing during training.

For each batch $\mathcal{B}$, the following steps are performed:

1. **Graph Encoding:** The GRAPHENCODER procedure is called to encode the knowledge graphs in the batch. It takes the batch $\mathcal{B}$ as input and initializes node embeddings $\mathbf{h}_i^{(0)}$ and relation embeddings $\mathbf{r}_k$ for each graph G_j in the batch. Then, it performs L layers of message passing using a Relational Graph Convolutional Network (RGCN) to update the node embeddings. The output of the GRAPHENCODER is a set of updated node embeddings $\mathbf{H} = \{\mathbf{h}_i^{(L)}\}_{i=1}^{|\mathcal{V}|}$ for each graph in the batch.

2. **Cross-Modal Alignment:** The CROSSMODALALIGNMENT procedure takes the node embeddings $\mathbf{H}$ and the question embeddings $\mathcal{B}_Q$ as input. For each question Q_j in the batch, it tokenizes and embeds the question to obtain $\mathbf{E}_{T_j}$. Then, it projects the node embeddings $\mathbf{H}_j$ and question embeddings $\mathbf{E}_{T_j}$ into a shared attention space to obtain query ($\mathbf{Q}_j$), key ($\mathbf{K}_j$), and value ($\mathbf{V}_j$) representations. It computes attention scores $\mathbf{A}_j$ and context-enhanced node representations $\mathbf{Z}_j$ using the attention mechanism. Finally,

Algorithm 2 GRALAN Training Pseudocode

procedure TRAIN($\mathcal{D} = \{(G_i, Q_i, A_i)\}_{i=1}^{N}$)
  for each epoch **do**
    for each batch $\mathcal{B} = \{(G_j, Q_j, A_j)\}_{j=1}^{B}$ **do**
      $\mathbf{H} \leftarrow$ GRAPHENCODER($\mathcal{B}$)
      $\mathbf{Z} \leftarrow$ CROSSMODALALIGNMENT($\mathbf{H}, \mathcal{B}_Q$)
      $\mathbf{X} \leftarrow$ LLMINPUTCONSTRUCTION($\mathbf{Z}, \mathcal{B}_Q$)
      $\mathbf{logits}, \mathbf{hidden\_states} \leftarrow$ LLMFORWARDPASS($\mathbf{X}$)
      $\mathbf{total\_loss} \leftarrow$ LOSSCOMPUTATION($\mathbf{logits}, \mathbf{hidden\_states}, \mathcal{B}_A$)
      Backpropagate $\mathbf{total\_loss}$ and update model parameters
procedure GRAPHENCODER($\mathcal{B} = \{(G_j, Q_j, A_j)\}_{j=1}^{B}$)
  for each graph $G_j \in \mathcal{B}$ **do**
    Initialize node embeddings $\mathbf{h}_i^{(0)}$ and relation embeddings $\mathbf{r}_k$
    for $l = 1$ to L **do**
      Update node embeddings $\mathbf{h}_i^{(l)}$ using RGCN
  return $\mathbf{H} = \{\mathbf{h}_i^{(L)}\}_{i=1}^{|\mathcal{V}|}$
procedure CROSSMODALALIGNMENT($\mathbf{H}, \mathcal{B}_Q$)
  for each question $Q_j \in \mathcal{B}_Q$ **do**
    Tokenize and embed Q_j to get $\mathbf{E}_{T_j}$
    Project $\mathbf{H}_j$ and $\mathbf{E}_{T_j}$ into shared attention space to get $\mathbf{Q}_j, \mathbf{K}_j, \mathbf{V}_j$
    Compute attention scores $\mathbf{A}_j$ and context-enhanced node representations $\mathbf{Z}_j$
    Refine $\mathbf{Z}_j$ using residual connection and normalization
  return $\mathbf{Z} = \{\mathbf{Z}_j\}_{j=1}^{B}$
procedure LLMINPUTCONSTRUCTION($\mathbf{Z}, \mathcal{B}_Q$)
  for each question $Q_j \in \mathcal{B}_Q$ **do**
    Compute query embedding $\mathbf{q}_j$ by mean-pooling question token embeddings
    Compute relevance scores β_{ij} between nodes and query
    Select top-K nodes based on relevance scores
    Assemble input sequence $\mathbf{X}_j$ with special tokens, global graph embedding, top-K node embeddings, and query embedding
  return $\mathbf{X} = \{\mathbf{X}_j\}_{j=1}^{B}$
procedure LLMFORWARDPASS($\mathbf{X}$)
  Forward $\mathbf{X}$ through the LLM
  Obtain logits and hidden states from the LLM output
  return $\mathbf{logits}, \mathbf{hidden\_states}$
procedure LOSSCOMPUTATION($\mathbf{logits}, \mathbf{hidden\_states}, \mathcal{B}_A$)
  for each answer $A_j \in \mathcal{B}_A$ **do**
    Extract hidden state corresponding to [ANSWER] token
    Classify based on answer hidden state to get logits
    Compute classification loss, LLM output loss, and regularization term
    Combine losses to get total loss for the sample
  return mean $\mathbf{total\_loss}$ over the batch

Algorithm 3 GRALAN Inference Pseudocode

```
procedure INFER(G, Q)
    H ← GRAPHENCODER(G)
    Z ← CROSSMODALALIGNMENT(H, Q)
    X ← LLMINPUTCONSTRUCTION(Z, Q)
    logits, hidden_states ← LLMFORWARDPASS(X)
    A ← ANSWEREXTRACTION(logits, hidden_states)
    return A
procedure ANSWEREXTRACTION(logits, hidden_states)
    Extract hidden state corresponding to [ANSWER] token
    Classify based on answer hidden state to get logits
    Select the answer(s) with the highest probability
    return A
```

it refines the node representations using a residual connection and normalization. The output of CROSSMODALALIGNMENT is a set of context-enhanced node representations $\mathbf{Z} = \{\mathbf{Z}_j\}_{j=1}^{B}$ for each question in the batch.

3. **LLM Input Construction:** The LLMINPUTCONSTRUCTION procedure takes the context-enhanced node representations $\mathbf{Z}$ and the question embeddings $\mathcal{B}_Q$ as input. For each question Q_j in the batch, it computes a query embedding $\mathbf{q}_j$ by mean-pooling the question token embeddings. It then computes relevance scores β_{ij} between the nodes and the query, and selects the top-K nodes based on these scores. Finally, it assembles an input sequence $\mathbf{X}_j$ for the LLM by combining special tokens, the global graph embedding, the top-K node embeddings, and the query embedding. The output of LLMINPUTCONSTRUCTION is a set of input sequences $\mathbf{X} = \{\mathbf{X}_j\}_{j=1}^{B}$ for each question in the batch.

4. **LLM Forward Pass:** The LLMFORWARDPASS procedure takes the input sequences $\mathbf{X}$ and forwards them through the pre-trained language model (LLM). It obtains the logits and hidden states from the LLM's output.

5. **Loss Computation:** The LOSSCOMPUTATION procedure takes the logits, hidden states, and the corresponding answers $\mathcal{B}_A$ for the batch. For each answer A_j in the batch, it extracts the hidden state corresponding to the [ANSWER] token and performs classification based on this hidden state to obtain logits. It then computes the classification loss, LLM output loss, and regularization term, and combines them to get the total loss for each sample. Finally, it returns the mean total loss over the entire batch.

6. **Backpropagation and Parameter Update:** The computed total loss is backpropagated through the model, and the model parameters are updated using an optimization algorithm (e.g., stochastic gradient descent) to minimize the loss.

The training process repeats these steps for multiple epochs, iteratively updating the model parameters to improve its performance on the question-answering task.

The GRALAN inference pseudocode describes the process of using a trained GRALAN model to answer a question Q given a knowledge graph G. Let's walk through the inference procedure step by step:

The INFER procedure takes a knowledge graph G and a question Q as input. It follows a similar pipeline to the training procedure, but without the loss computation and backpropagation steps.

1. **Graph Encoding:** The GRAPHENCODER procedure is called to encode the knowledge graph G. It follows the same process as in the training phase, initializing node embeddings and performing message passing using an RGCN to obtain the updated node embeddings $\mathbf{H}$.

2. **Cross-Modal Alignment:** The CROSSMODALALIGNMENT procedure takes the node embeddings $\mathbf{H}$ and the question Q as input. It aligns the node representations with the question representation using the attention mechanism, as described in the training phase, to obtain the context-enhanced node representations $\mathbf{Z}$.

3. **LLM Input Construction:** The LLMINPUTCONSTRUCTION procedure takes the context-enhanced node representations $\mathbf{Z}$ and the question Q as input. It follows the same process as in the training phase, computing the query embedding, selecting the top-K relevant nodes, and assembling the input sequence $\mathbf{X}$ for the LLM.

4. **LLM Forward Pass:** The LLMFORWARDPASS procedure takes the input sequence $\mathbf{X}$ and forwards it through the pre-trained LLM. It obtains the logits and hidden states from the LLM's output.

5. **Answer Extraction:** The ANSWEREXTRACTION procedure takes the logits and hidden states as input. It extracts the hidden state corresponding to the [ANSWER] token and performs classification based on this hidden state to obtain logits. It then selects the answer with the highest probability as the predicted answer A.

The INFER procedure returns the predicted answer(s) A as the output.

3 Hyperparameter Tuning Procedure

We performed a systematic hyperparameter optimization to maximize model performance. The hyperparameters and their respective tuning ranges are presented in Table 2. Our tuning procedure followed a two-stage approach:

3.1 Stage 1: Grid Search for Critical Parameters

We first conducted a coarse grid search over the most critical hyperparameters that significantly impact model performance: Number of message-passing layers L, Dimension of node and relation embeddings d, Classification weight λ_{cls}. The grid search evaluated $3 \times 5 \times 2 = 30$ different configurations on a randomly selected subset (20%) of the validation data to identify promising regions in the hyperparameter space. We selected the top-5 configurations based on validation accuracy for further fine-tuning.

3.2 Stage 2: Bayesian Optimization

For the remaining hyperparameters and fine-tuning of the critical parameters, we employed Bayesian optimization with Gaussian processes to efficiently explore the hyperparameter space. We used the expected improvement acquisition function with 50 iterations. The following parameters were tuned during this stage Relevance threshold

Parameter	Description	Tuning Interval
θ	Relevance threshold for Question-Focused BFS	$[0.1, 0.9]$
L	Number of message-passing layers in RGCN	$\{2, 3, 4\}$
g	Number of layers in feed-forward network FFN_g	$\{1, 2, 3\}$
K	Number of top-K node embeddings selected	$\{5, 10, 20, 30\}$
τ	Threshold for multiple answer prediction	$[0.3, 0.8]$
λ_{cls}	Weight balancing classification and LLM loss	$[0.1, 0.9]$
α	Weight for regularization loss term	$[10^{-5}, 10^{-2}]$
β	Weight for KL divergence term	$[0.01, 1.0]$
d	Dimension of node and relation embeddings	$\{64, 128, 200, 256, 512\}$
d_{lm}	Embedding dimension of the LLM	Fixed by LLM architecture

Table 2: Hyperparameters of the Graph Language Framework and their suggested tuning intervals.

θ Number of feed-forward layers g, Number of top-K node embeddings K, Multiple answer threshold τ, Regularization weight α (log scale), KL divergence weight β. We evaluated each configuration using 5-fold cross-validation on the full validation set, using accuracy and Mean Reciprocal Rank (MRR) as our primary evaluation metrics.

3.3 Early Stopping and Model Selection

During training, we employed early stopping with a patience of 10 epochs, monitoring the validation MRR. The final model was selected based on the best validation performance across all evaluated configurations. We then conducted an ablation study on the selected hyperparameters to analyze their individual impact on the model's performance. The LLM embedding dimension d_{lm} remained fixed according to the pre-trained language model architecture used in the experiments.