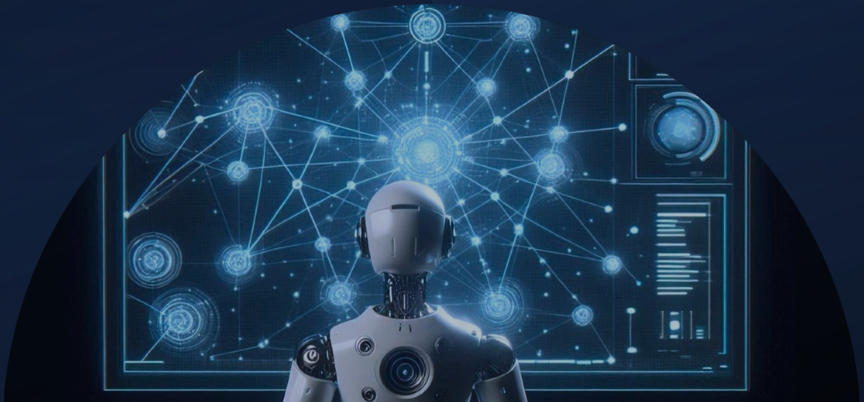
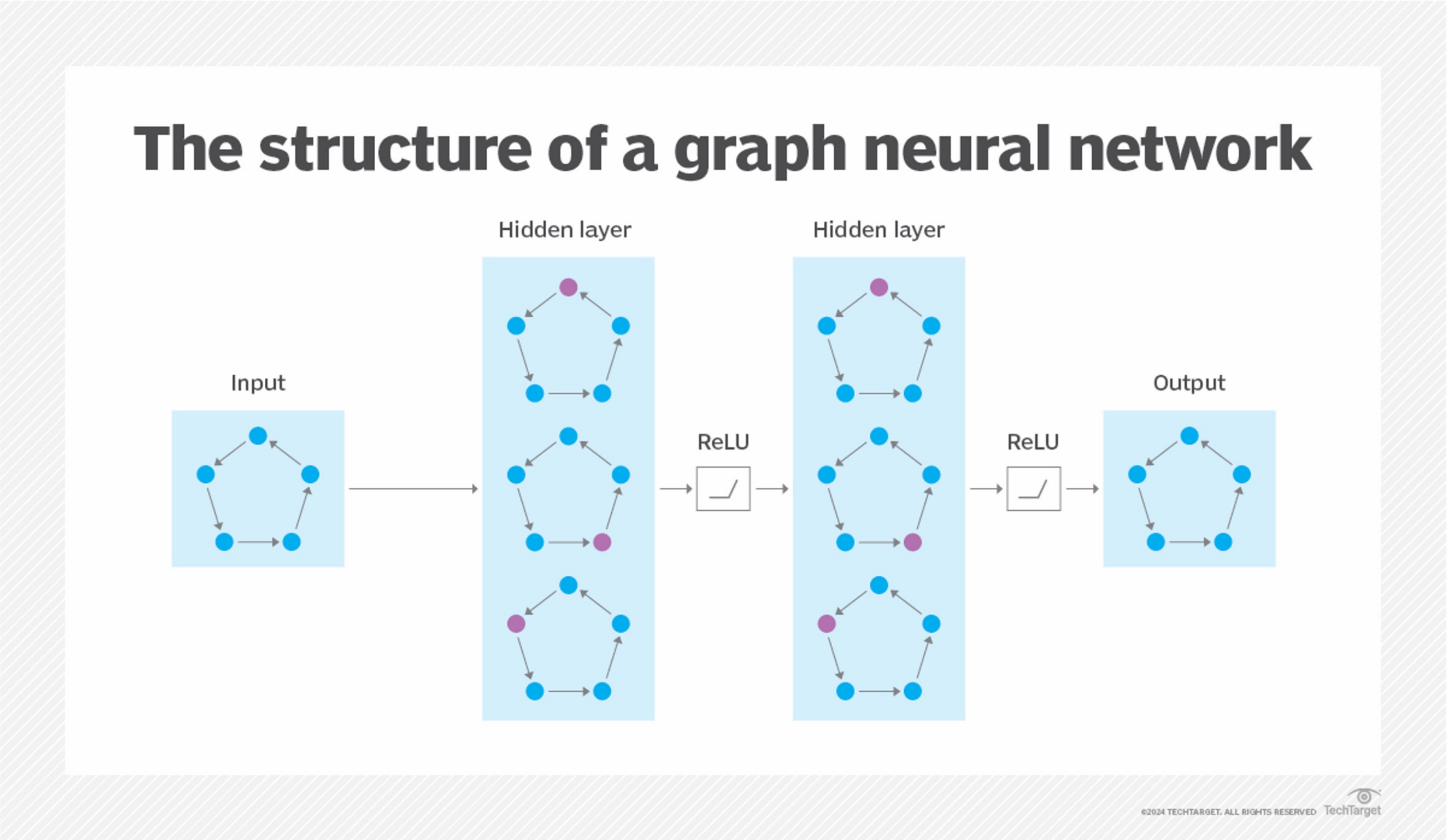
Language Models and Knowledge Graphs

Tong Li, Sunny Yuan, Yiqun Du



Graph neural network basics



Content:

- GNN-LM: language modeling based on global contexts via gnn
- G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering
- KG-BART: Knowledge Graph-Augmented BART for Generative Commonsense Reasoning
- Head-to-Tail: How Knowledgeable are Large Language Models (LLMs)?

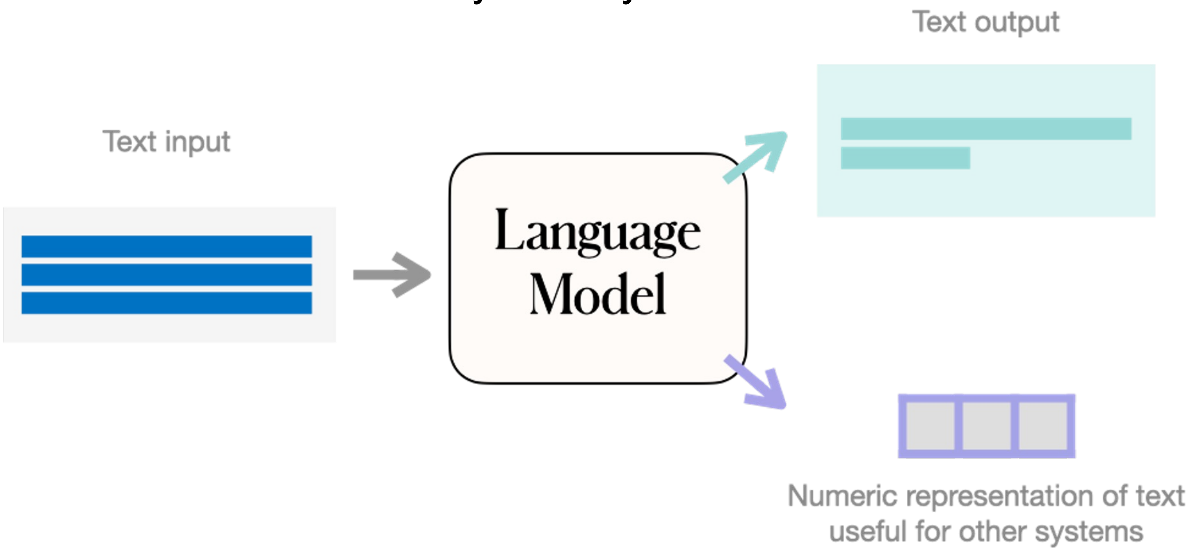
GNN-LM: LANGUAGE MODELING BASED ON GLOBAL CONTEXTS VIA GNN

Meng, Y., Zong, S., Li, X., Sun, X., Zhang, T., Wu, F., & Li, J. (2021)

gnn-lm: language modeling - arXiv

Language Modeling Background：

- Traditional training of LMs can be viewed as a **closed-book examination**
- At inference time, the model does not have access to the training data and must rely entirely on **memorization**.



Limitation： Long-tail cases ； Memory limitations

https://radical.vc/wp-content/uploads/2024/01/Language-Model-Cohere.png

Copying vs. Memorizing：

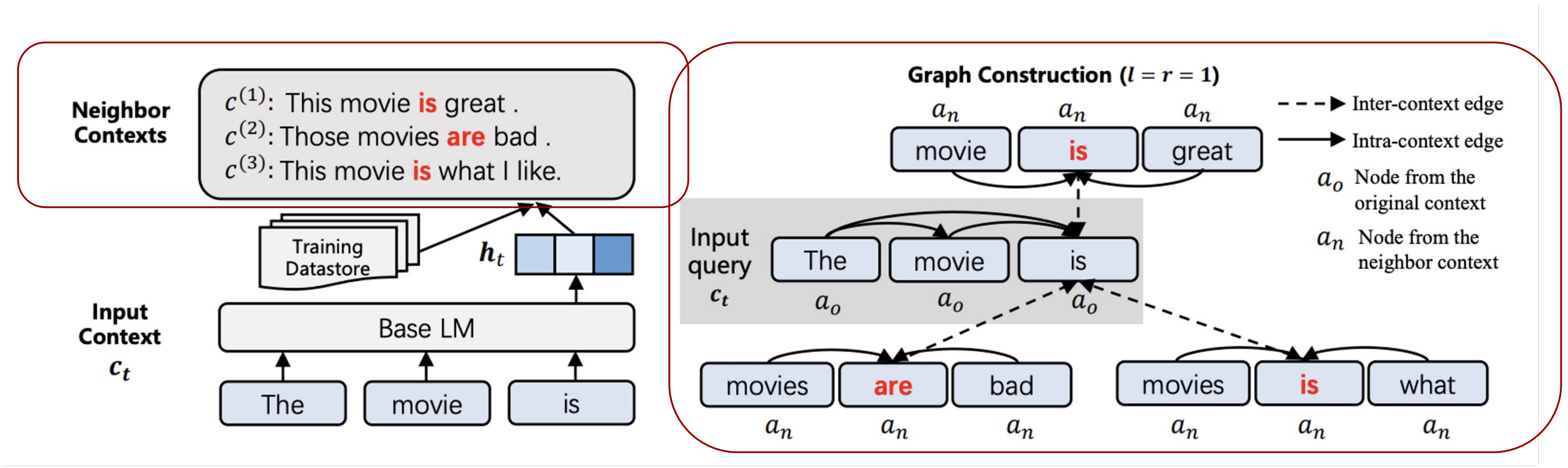
Open-book examination: LMs should have the ability to "**copy**" or reference similar contexts from the training set, even during inference.

Copying is easier than memorizing!

- Introduces the **GNN-LM** model, which allows the LM to reference contexts from the entire training corpus by constructing a **graph** of related contexts.

- A **GNN** is then applied to this graph, allowing the LM to aggregate information from both the input context and retrieved neighbor contexts.

Overview of the proposed GNN-LM model pipeline：



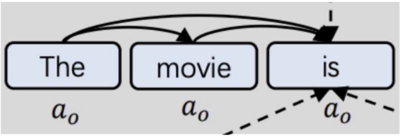
General LMs:

- LM encodes a sequence of context tokens $c_t = (w_1, w_2, ..., w_{t-1})$ -> high dimensional representation h_t
- a transformation matrix/function to estimate the probability of the t-th token: $p(w_t|c_t) = \text{softmax}(\boldsymbol{W}\boldsymbol{h}_t)$

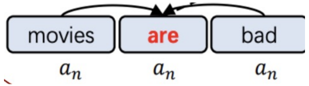
Step1: Graph Construction

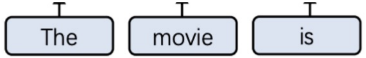
Define a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R}, \tau, \phi)$

$\mathcal{V}$ is a collection of nodes v,
$\mathcal{E}$ is a collection of edges e.



$\mathcal{A} = \{a_o, a_n\}$ is **two types of nodes**, a_o is the node is within the input c_t, a_n means the node is in N(c_t).





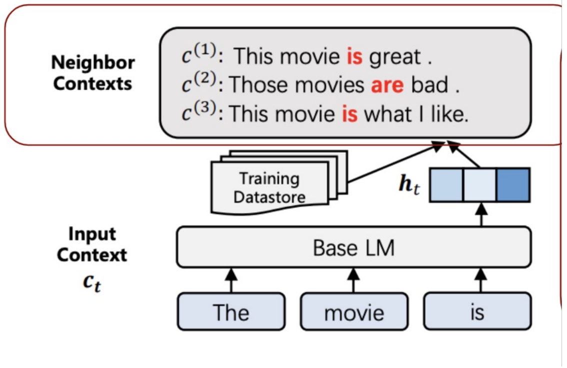
$\mathcal{R} = \{r_{\text{inter}}, r_{\text{intra}}\}$ is **two types of edges**, where r_inter means **inter-context** connection (from a_n nodes to a_o nodes);
r_intra means **intra-context** connection (between two nodes of same type).

Step2: Retrieve the neighbors

For an input context c_t, we retrieve k nearest neighbors $\mathcal{N}(c_t) = \{c_t^{(1)}, ..., c_t^{(k)}\}$ of c_t from the training set:

- Use h_t (high-dimensional representation of c_t) to query the cached representations (obtained by a pretrained LM) of all tokens for training samples
- Distance is measured by the cosine similarity
- retrieve the top K tokens $\{w_j^{(i)}\}$ on i -th training sample, and j-th time step Then, it is expanded with the left and right window:

$$\boldsymbol{c}_j^{(i)} = \{w_{j+p}^{(i)}\}_{p=-l}^{r}$$



Step3: GNN on the constructed graph

Use graph neural networks (GNNs) to aggregate and percolate the token information based on the graph constructed.

Attention:

$$K(s) = \boldsymbol{W}^k_{\tau(s)} \boldsymbol{h}^{[l-1]}_s, \quad Q(n) = \boldsymbol{W}^q_{\tau(n)} \boldsymbol{h}^{[l-1]}_n,$$

$$\text{Attention}(s,e,n) = \frac{1}{Z} \exp\left(K(s) \boldsymbol{W}^{\text{ATT}}_{\phi(e)} Q(n)^\top \cdot \frac{\boldsymbol{\mu}_{\langle \tau(s), \phi(e), \tau(n) \rangle}}{\sqrt{d}} \right),$$

$$Z = \sum_{s' \in \mathcal{N}(n), e' \in \phi(e)} \text{Attention}(s', e', n),$$

Transformer attention feature learn

Feature:

$$\text{Feature}(s,e,n) = \boldsymbol{W}^v_{\tau(s)} \boldsymbol{h}^{[l-1]}_s \boldsymbol{W}^{\text{FEA}}_{\phi(e)},$$

Aggregate:

$$\text{Aggregate}(\cdot) = \boldsymbol{W}^o_{\tau(n)} \left(\bigoplus_{\forall s \in \mathcal{N}(n)} (\text{Attention}(s,e,n) \cdot \text{Feature}(s,e,n)) \right)$$

GNN feature propagation and aggregation

Step4: kNN based probability for next token

Retrieve the k nearest neighbors $\mathcal{N}(\boldsymbol{c}_t) = \{\boldsymbol{c}_{t_1}^{(1)}, ..., \boldsymbol{c}_{t_k}^{(k)}\}$

Compute the kNN based probability for the next token:

$$p(w_t|\boldsymbol{c}_t) = \lambda p_{\text{kNN}}(w_t|\boldsymbol{c}_t) + (1-\lambda)p_{\text{LM}}(w_t|\boldsymbol{c}_t),$$

$$p_{\text{kNN}}(w_t|\boldsymbol{c}_t) = \frac{1}{Z}\sum_{i=1}^{k} \mathbb{1}_{w_t=w_{t_i}^{(i)}} \exp\left(\cos(f(\boldsymbol{c}_t), f(\boldsymbol{c}_{t_i}^{(i)}))/T\right),$$

It extends a vanilla LM by linearly interpolating it with a k-nearest neighbors (kNN) model

Experiments:

For all experiments, we add a 3-layer self-attention augmented GNN on top of the pretrained base LM, and use the same hidden dimension and number of heads as our base LM. We retrieve k = 1, 024 nearest neighbors for each source token, among them the top 128 neighbors are used in graph, and all of them are used in computing the kNN-based probability.

Datasets:
WikiText-103, One Billion Word, Enwik8

Main Results:

Model	# Param	Test ppl (↓)
Hebbian + Cache (Rae et al., 2018)	151M	29.9
Transformer-XL (Dai et al., 2019)	257M	18.3
Transformer-XL + Dynamic Eval (Krause et al., 2019)	257M	16.4
Compressive Transformer (Rae et al., 2019)	-	17.1
KNN-LM + Cache (Khandelwal et al., 2019)	257M	15.8
Sandwich Transformer (Press et al., 2020a)	247M	18.0
Shortformer (Press et al., 2020b)	247M	18.2
SegaTransformer-XL (Bai et al., 2021)	257M	17.1
Routing Transformer (Roy et al., 2021)	-	15.8
base LM (Baevski & Auli, 2018)	247M	18.7
+GNN	274M	16.8
+GNN+*k*NN	274M	**14.8**

Table 1: Test perplexity on WikiText-103 dataset.

Model	# Param	Test ppl (↓)
LSTM+CNN (Jozefowicz et al., 2016)	1.04B	30.0
High-Budget MoE (Shazeer et al., 2016)	5B	28.0
DynamicConv (Wu et al., 2018)	0.34B	26.7
Mesh-Tensorflow (Shazeer et al., 2018)	4.9B	24.0
Evolved Transformer (Shazeer et al., 2018)	-	28.6
Transformer-XL (Dai et al., 2019)	0.8B	21.8
Adaptive inputs (base) (Baevski & Auli, 2018)	0.36B	25.2
Adaptive inputs (large) (Baevski & Auli, 2018)	0.46B	23.9
base LM (Baevski & Auli, 2018)	1.03B	23.0
+*k*NN	1.02B	22.8
+GNN	1.05B	22.7
+GNN+*k*NN	1.05B	22.5

Table 2: Test perplexity on One Billion Word dataset.

GNN-LM reduces the base LM perplexity from 18.7 to 16.8.
The combination of GNN and kNN further boosts the performance to 14.8, a new state-of-the-art result on WikiText-103.

GNN-kNN-LM helps base LM reduce 0.5 perplexity with only 27M additional parameters. For comparison, Baevski & Auli (2018) use 560M additional parameters to reduce perplexity from 23.9 to 23.0.

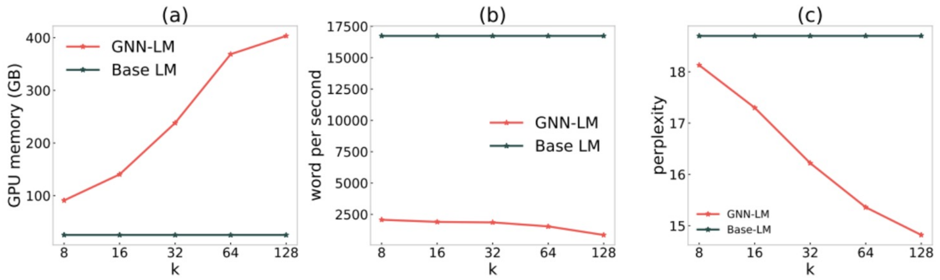
Model	# Param	BPC (↓)
64L Transformer (Al-Rfou et al., 2019)	235M	1.06
18L Transformer-XL (Dai et al., 2019)	88M	1.03
24L Transformer-XL (Dai et al., 2019)	277M	0.99
24L Transformer-XL + Dynamic Eval (Krause et al., 2019)	277M	0.94
Longformer (Beltagy et al., 2020)	102M	0.99
Adaptive Transformer (Sukhbaatar et al., 2019)	209M	0.98
Compressive Transformer (Rae et al., 2019)	277M	0.97
Sandwich Transformer (Press et al., 2020a)	209M	0.97
12L Transformer-XL (Dai et al., 2019)	41M	1.06
+kNN	41M	1.04
+GNN	48M	1.04
+GNN+kNN	48M	1.03

Table 3: Bit per Character on the Enwik8 dataset.

GNN-kNN-LM outperforms base LM by 0.03 Bit per Character (BPC), achieving 1.03 BPC with only 48M parameters, comparable to 18L Transformer-XL with 88M parameters.

Space Complexity:

- **Key Factor**: GNN-LM requires significantly more memory than a vanilla language model (LM) because it constructs a graph with k nearest neighbors for each token c_i, making the number of nodes in the graph k times larger.
- **Memory Mitigation Strategies**: Smaller k during initial training; Context truncation



Time Complexity:

- GNN-LM is slower than the base LM, with complexity increasing as k grows. The overhead comes from kNN retrieval and graph construction, but preprocessing these can mitigate some of the time cost.

Conclusion:

The paper proposes a new language modeling approach called GNN-LM, which extends the vanilla neural language model by allowing it to reference similar contexts from the entire training corpus.

Experiments show that GNN-LM outperforms strong baselines on standard language modeling benchmarks, and when combined with kNN-LM, it achieves a new state-of-the-art perplexity on the WikiText-103 dataset.

Improve the efficiency of building the graph and retrieving nearest neighbors in future work?

G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V. Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, Bryan Hooi

https://arxiv.org/abs/2402.07630
2024

Introduction

Context:

- Large Language Models (LLMs) have reshaped AI, especially in handling complex tasks.
- Many real-world datasets have **graph structures**, such as the Web, e-commerce platforms, and knowledge graphs.

Challenge:

- Previous approaches combining LLMs and Graph Neural Networks (GNNs) mainly focus on:
  - **Conventional graph tasks** (node/edge classification).
  - Simple questions on small, synthetic graphs.
- Need for handling **real-world textual graphs** with both graph structure and textual data.

Related Work

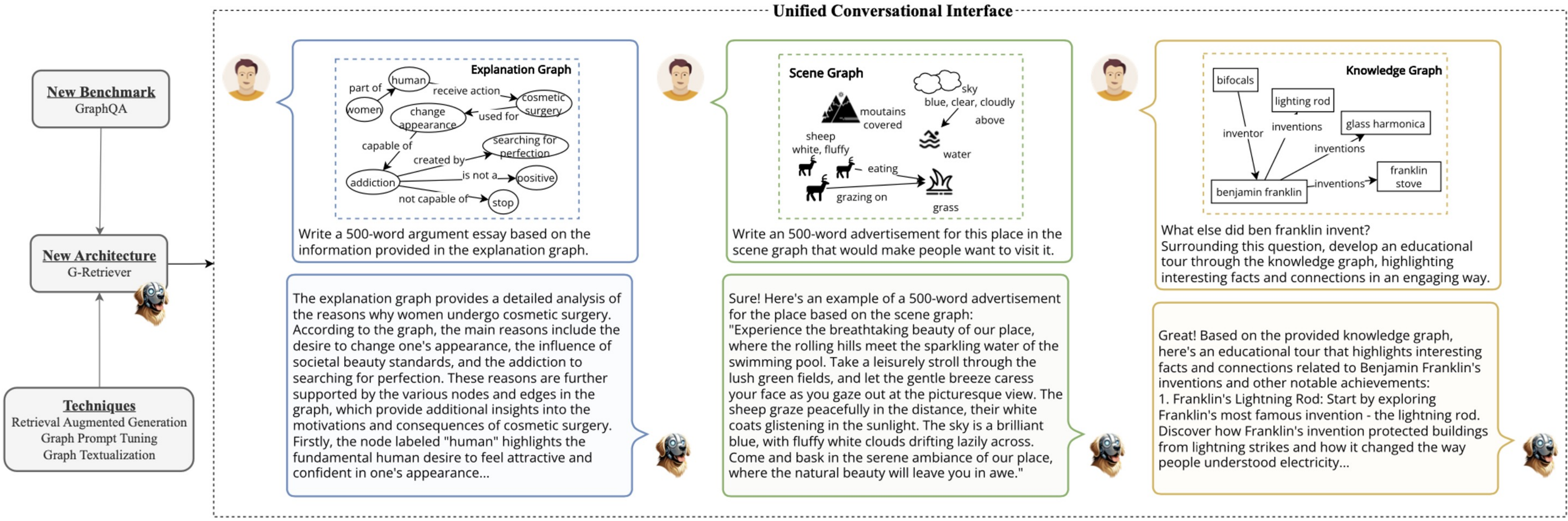
- Graphs and Large Language Models
  - Research spans from designing general graph models to multi-modal architectures.
- Retrieval-Augmented Generation (RAG)
  - Proposed by Lewis et al. [21], RAG helps mitigate hallucination issues in LLMs and enhances trustworthiness and explainability of model outputs.
- Parameter-Efficient Fine Tuning
  - Help optimize LLM performance with minimum parameter training
  - Important methods
    - Prompt tuning
    - Prefix tuning
    - LoRA

G-Retriever: Key Contributions

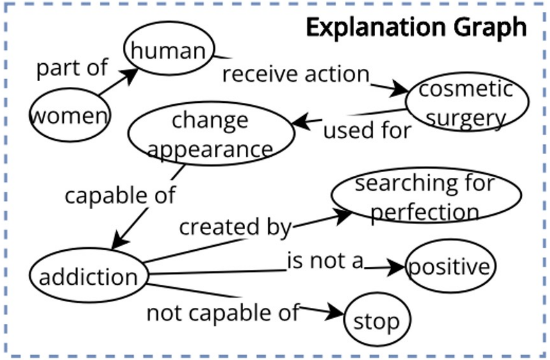
- Enabling ‘Chat with Your Graph’.
- Introducing A Novel GraphQA Benchmark with data collected from different tasks.
- Pioneering the integration of Graph RAG by introducing G-Retriever
- Empirical Findings.

“Chat with your Graph”

We develop a flexible question-answering framework targeting real-world textual graph applications via a unified conversational interface



Example of Real-world Textual Graph

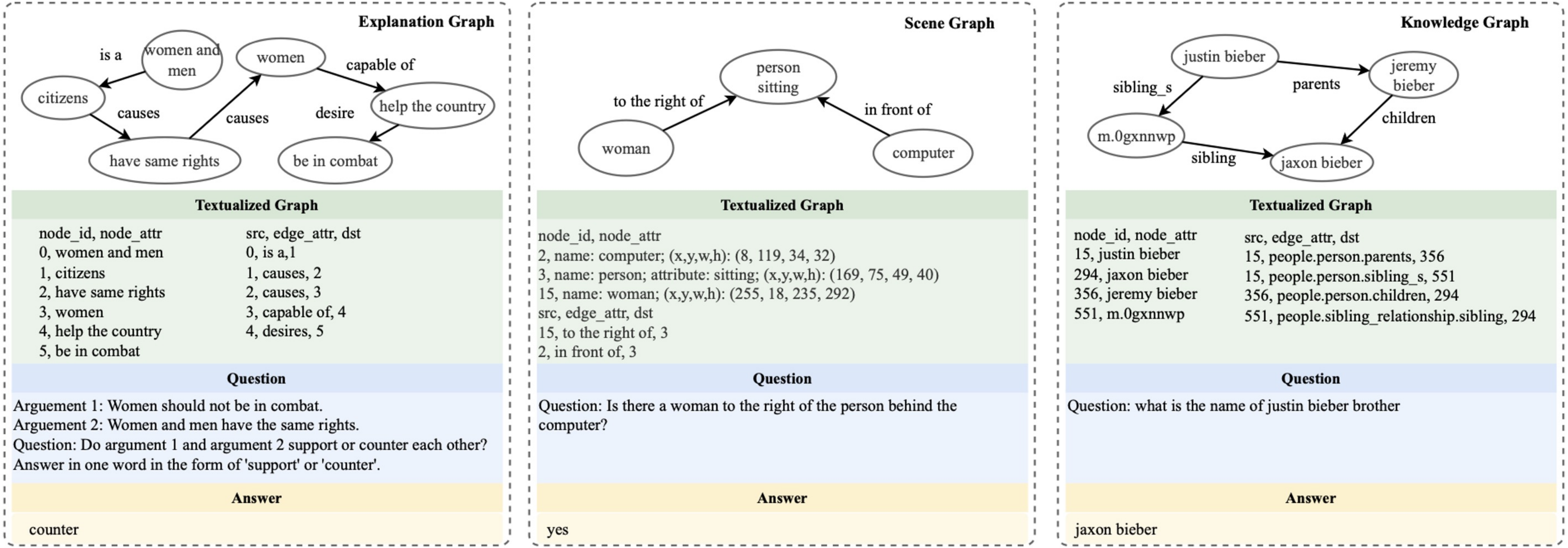


Write a 500-word argument essay based on the information provided in the explanation graph.

The explanation graph provides a detailed analysis of the reasons why women undergo cosmetic surgery. According to the graph, the main reasons include the desire to change one's appearance, the influence of societal beauty standards, and the addiction to searching for perfection. These reasons are further supported by the various nodes and edges in the graph, which provide additional insights into the motivations and consequences of cosmetic surgery. Firstly, the node labeled "human" highlights the fundamental human desire to feel attractive and confident in one's appearance...

GraphQA Benchmark

- Data format
  - A textual graph, a question related to the graph, and one or more corresponding answers

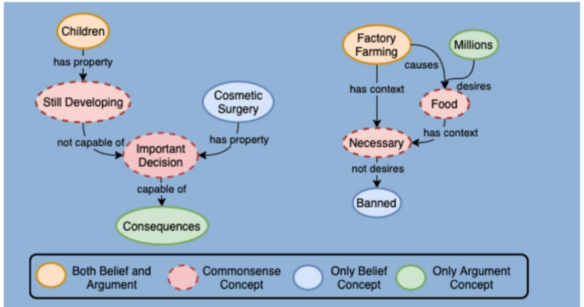


GraphQA Benchmark

- Overview of datasets

Table 2: Summary of datasets used in GraphQA benchmark.

Dataset	ExplaGraphs	SceneGraphs	WebQSP
#Graphs	2,766	100,000	4,737
Avg. #Nodes	5.17	19.13	1370.89
Avg. #Edges	4.25	68.44	4252.37
Node Attribute	Commonsense concepts	Object attributes (*e.g.*, color, shape)	Entities in Freebase
Edge Attribute	Commonsense relations	Relations (*e.g.*, actions, spatial relations)	Relations in Freebase
Task	Common sense reasoning	Scene graph question answering	Knowledge based question answering
Evaluation Matrix	Accuracy	Accuracy	Hit@1



An Explanation Graph Generation Task for Structured Commonsense Reasoning

[[’FedEx Cup’, ’sports.sports_award_type.winners’, ’m.0n1v8cy’],
[’Brandt Snedeker’, ’sports.sports_award_winner.awards’, ’m.0n1v8cy’],
[’FedEx Cup’, ’common.topic.article’, ’m.08q5wy’],
[’FedEx Cup’, ’common.topic.notable_for’, ’g.12559n8g_’],
[’Sports League Award Type’, ’freebase.type_profile.published’, ’Published’],
[’FedEx Cup’, ’common.topic.notable_types’, ’Sports League Award Type’],
[’m.0n1v8cy’, ’sports.sports_award.award_winner’, ’Brandt Snedeker’],
[’Sports League Award Type’, ’type.type.expected_by’, ’Award’],
[’Sports League Award Type’, ’common.topic.article’, ’m.06zxtxj’],
[’2012 PGA Tour’, ’sports.sports_league_season.awards’, ’m.0n1v8cy’],
[’Sports League Award Type’, ’freebase.type_hints.included_types’, ’Topic’],
[’Sports League Award Type’, ’type.type.domain’, ’Sports’],
[’m.0n1v8cy’, ’sports.sports_award.award’, ’FedEx Cup’],
[’Sports League Award Type’, ’freebase.type_profile.strict_included_types’, ’Topic’],
[’Sports League Award Type’, ’freebase.type_profile.kind’, ’Classification’],

G-Retriever Architecture

Overview:

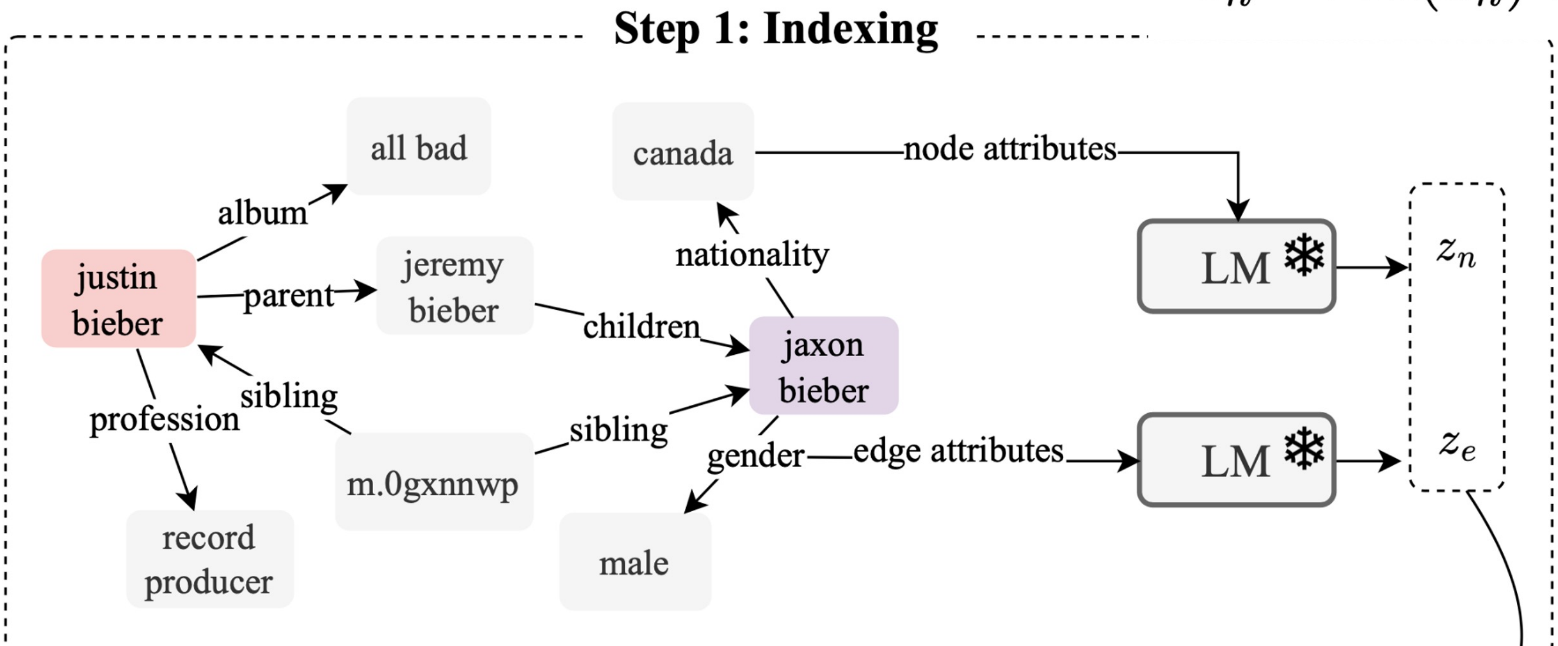
- **Indexing**: Initial storage and embedding of graphs.
- **Retrieval**: Use of RAG to select relevant subgraphs.
- **Subgraph Construction**: Prize-Collecting Steiner Tree for effective retrieval.
- **Answer Generation**: Using LLMs to generate responses.

Indexing

Graphs are indexed for efficient query processing.

We generate node and graph embeddings using a pre-trained LM, and then store these embeddings in a nearest neighbor data structure.

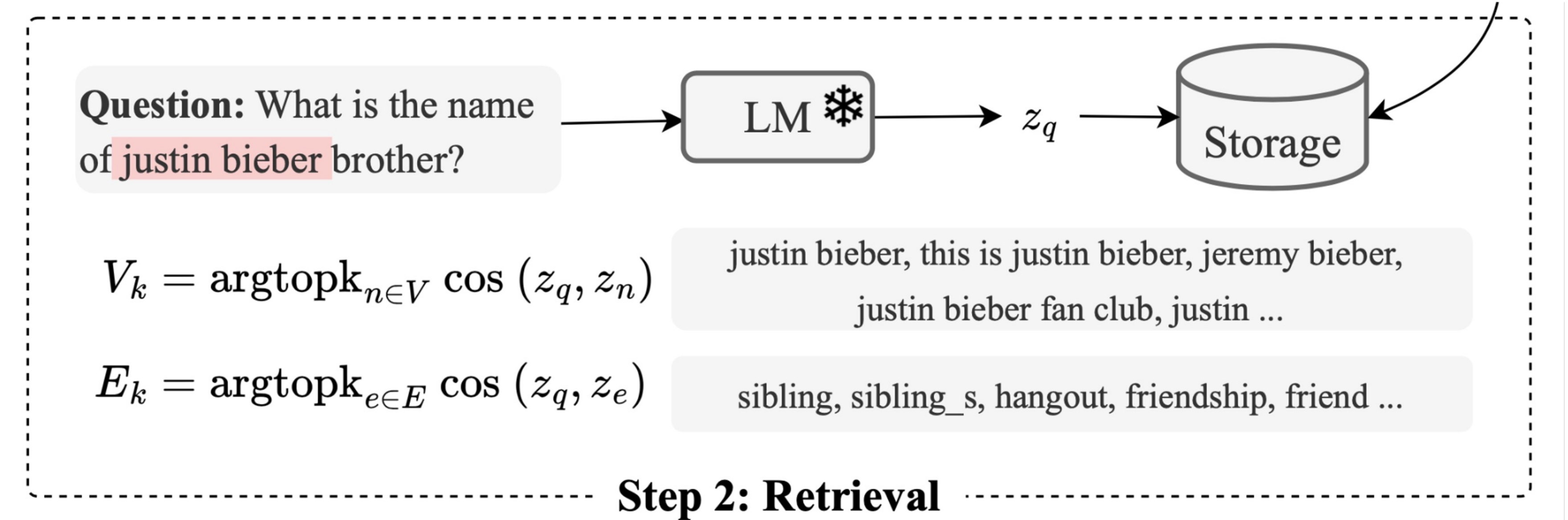
$$z_n = \mathbf{LM}(x_n) \in \mathbb{R}^d,$$



Retrieval

The most semantically relevant nodes and edges are retrieved, conditioned on the query.

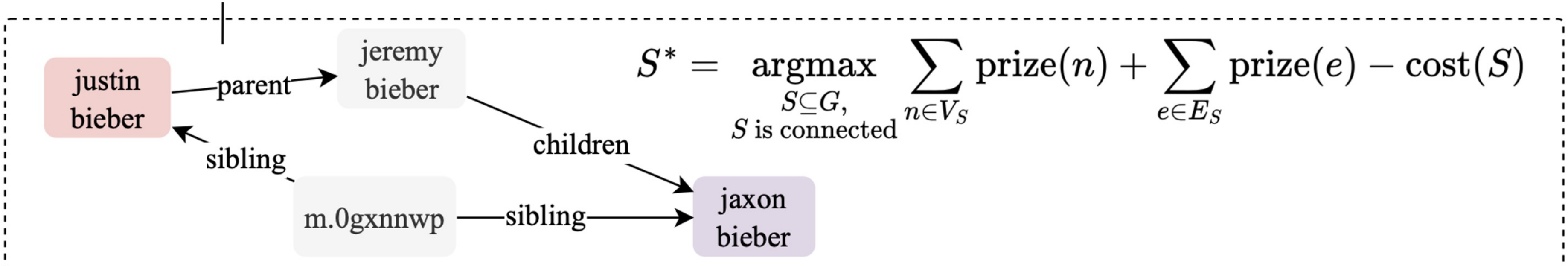
$$V_k = \mathrm{argtopk}_{n \in V} \cos(z_q, z_n)$$
$$E_k = \mathrm{argtopk}_{e \in E} \cos(z_q, z_e),$$



Subgraph Construction

This step aims to construct a subgraph that encompasses as many relevant nodes and edges as possible while keeping the graph size manageable.

Used Prize-Collecting Steiner Tree algorithm to identify optimal size and subgraph.



Step 3: Subgraph Construction

Prize-Collecting Steiner Tree algorithm (PCST)

Aims to find a connected subgraph that maximizes the total prize values of its nodes while minimizing the total costs of its edges.

$$\text{prize}(n) = \begin{cases} k - i, & \text{if } n \in V_k \text{ and } n \text{ is the top } i \text{ node,} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

$$S^* = \underset{\substack{S \subseteq G, \\ S \text{ is connected}}}{\text{argmax}} \sum_{n \in V_S} \text{prize}(n) + \sum_{e \in E_S} \text{prize}(e) - \text{cost}(S), \tag{7}$$

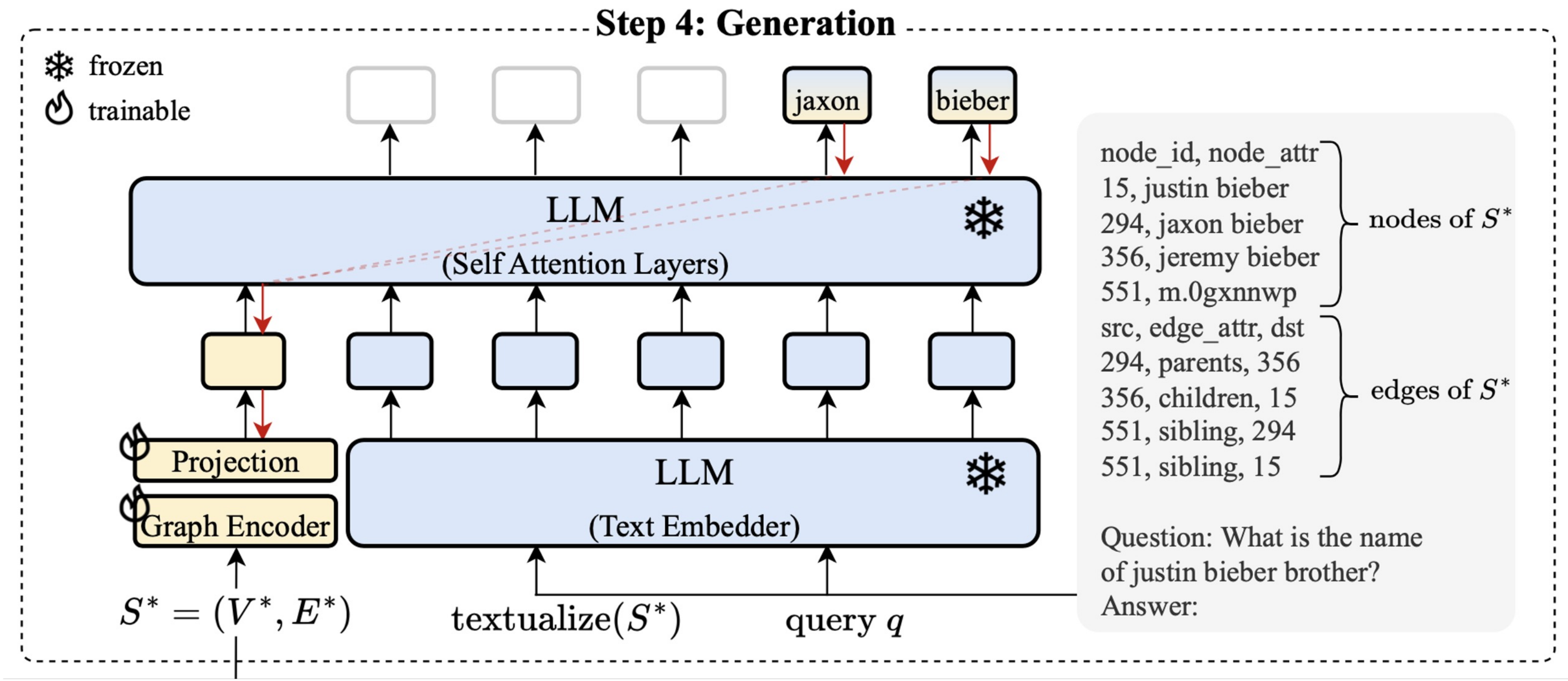
where

$$\text{cost}(S) = |E_S| \times C_e, \tag{8}$$

and C_e denotes a predefined cost per edge, which is adjustable to control the subgraph size.

Answer generation

Graph Encoder, Projection Layer, Text Embedder, LLM Generation with Graph Prompt Tuning



Experimental Results

Effectiveness Evaluation

Setting	Method	ExplaGraphs	SceneGraphs	WebQSP
Inference-only	Zero-shot	0.5650	0.3974	41.06
	Zero-CoT [18]	0.5704	0.5260	51.30
	CoT-BAG [44]	0.5794	0.5680	39.60
	KAPING [1]	0.6227	0.4375	52.64
Frozen LLM w/ PT	Prompt tuning	0.5763 ± 0.0243	0.6341 ± 0.0024	48.34 ± 0.64
	GraphToken [31]	0.8508 ± 0.0551	0.4903 ± 0.0105	57.05 ± 0.74
	G-Retriever	0.8516 ± 0.0092	<u>0.8131 ± 0.0162</u>	<u>70.49 ± 1.21</u>
	$\Delta_{\text{Prompt tuning}}$	$\uparrow 47.77\%$	$\uparrow 28.23\%$	$\uparrow 45.81\%$
Tuned LLM	LoRA	<u>0.8538 ± 0.0353</u>	0.7862 ± 0.0031	66.03 ± 0.47
	G-Retriever w/ LoRA	**0.8705 ± 0.0329**	**0.8683 ± 0.0072**	**73.79 ± 0.70**
	Δ_{LoRA}	$\uparrow 1.95\%$	$\uparrow 11.74\%$	$\uparrow 10.44\%$

Experiment results

Efficiency Evaluation

Dataset	Before Retrieval (Avg.)			After Retrieval (Avg.)		
	# Tokens	# Nodes	Min/Epoch	# Tokens	# Nodes	Min/Epoch
SceneGraphs	1,396	19	123.1	235 (↓83%)	5 (↓74%)	86.8 (↓29%)
WebQSP	100,627	1,371	18.7	610 (↓99%)	18 (↓99%)	6.2(↓67%)

Ablation study

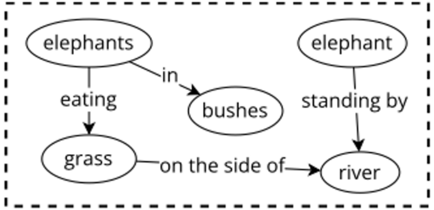
Table 6: Ablation study on the WebQSP dataset.

Method	Hit@1	$\Delta_{G\text{-}Retriever}$
w/o Graph Encoder	54.62 ± 0.78	↓ 22.51%
w/o Projection Layer	69.70 ± 0.68	↓ 1.11%
w/o Textualized Graph	56.96 ± 1.83	↓ 19.19%
w/o Retrieval	63.84 ± 0.41	↓ 9.43%

Mitigate Hallucination

Table 5: Quantitative comparison of hallucination on the `SceneGraphs` dataset.

	Baseline	*G-Retriever*
Valid Nodes	31%	77%
Valid Edges	12%	76%
Fully Valid Graphs	8%	62%



User	What kind of animal is in the bushes? Answer the question and list the nodes or edges from the graph that support your answer.
LLM w/ Graph Prompt Tuning	[Incorrect response with hallucinated nodes and edges] The animal in the bushes is a deer. Nodes: * Deer (node 1) * Bushes (node 2) Edges: * Deer $\rightarrow$ Bushes (edge 1) * Deer $\rightarrow$ Grass (edge 2) * Bushes $\rightarrow$ Grass (edge 3)
G-Retriever (Ours)	[Correct response with accurate node and edge references] Based on the information provided in the graph, the animal in the bushes is an elephant. The nodes that support this answer are: * Node 1: The elephant is mentioned in the first line of the graph. The edges that support this answer are: * The edge connects the elephant to the bushes, indicating that the elephant is in the bushes. Therefore, the answer to the question is: The animal in the bushes is an elephant.

KG-BART: Knowledge Graph-Augmented BART for Generative Commonsense Reasoning

Ye Liu, Yao Wan, Lifang He, Hao Peng, Philip S. Yu

https://arxiv.org/abs/2009.12677

Pain points of state-of-the-art pre-trained language generation models

- Ignoring knowledge information
- Failing to generate output towards capturing the human commonsense

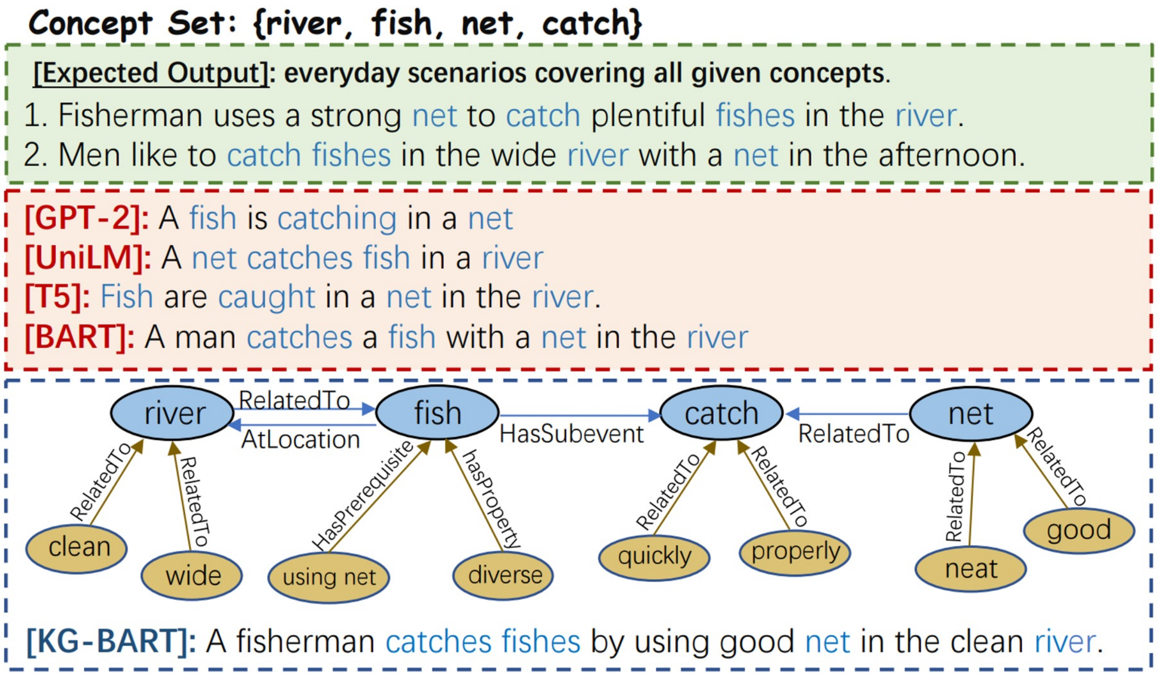


Figure 1: An example of the generation outputs of our KGBART model (blue dotted box) and the existing models without knowledge graph augmentation (red dotted box)

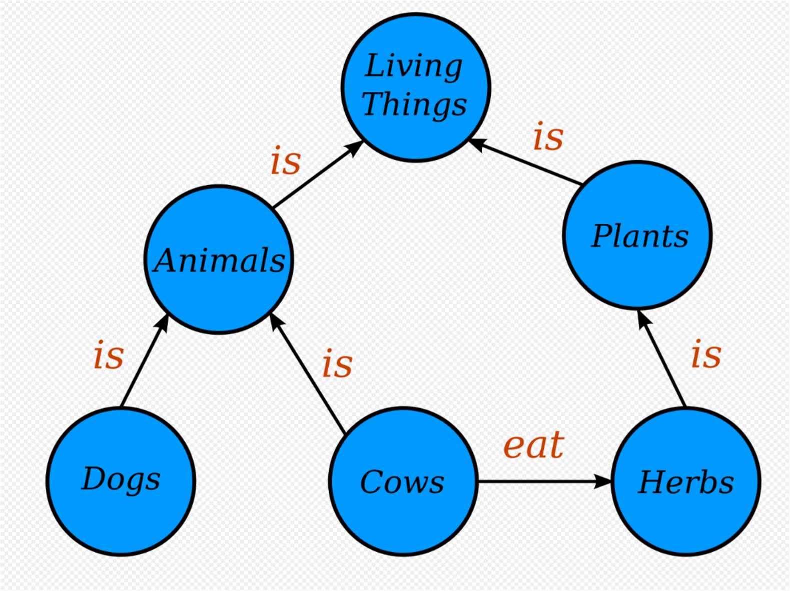
The solution

- Use knowledge graph to augment pre-trained language generation model :BART
  - To encompass the complex relations of concepts
  - To produce more logical and natural sentences
  - To enhance the model generalization on unseen concept sets
- Effectiveness:
  - Outperforms BART by 5.80, 4.60, in terms of BLEU-3, 4
  - Can work as background scenarios to benefit downstream commonsense QA tasks

The methodology

- Two steps
  - Knowledge graph grounding
    - Constructing two KGs: concept-reasoning graph (referred to as GR) and concept-expanding graph (referred to as GE)
  - Graph-based encoder-decoder modeling
    - Incorporating the grounded KGs into the state-of-the-art pre-trained language generation model BART

Knowledge graph notations



- V: the set of entities
  - $v_i \in V$ (subject)
  - $v_j \in V$ (object)
- $r_{ij} \in R$: relation between concepts
- edge $e_{ij} \in E$: (v_i , r_{ij} , v_j)

Concept reasoning graph(GR)

- Consisting of all concept triplets (v_i^R, r_{ij}^R, v_j^R)
- Used in the the encoding phase

Concept expanding graph(GE)

- $\mathcal{G}^R \cup \mathcal{N}(v^R)$
- N (v^R) characterizes the neighborhood relationship between concept (v^R) and its adjacencies in the KG database
- Enrich the graph with adjunct information

(climb) --requires--> (gear)
(climb) --associated with-->(outdoor activity)
(mountain) --isA--> (natural formation)

(climb) --requires--> (gear)
(climb) --associated with--> (outdoor activity)
(climb) --similar to--> (hiking)
(mountain) --isA--> (natural formation)
(mountain) --related to--> (snowy)
(mountain) --similar to--> (hill)

Grounding: constructing GR and GE

1. Match the concept set to the entities from KG to generate GR
2. Couple GR with the association of selected neighboring nodes with each concept in KG to from GE
    a. Ranking the neighboring nodes of each concept according to the word similarity scores
    b. Selecting their potential top-k neighboring nodes adding to GR
    c. Pre-trained GloVe embedding
3. TranSE: Translation Embedding
    a. A popular model for knowledge graph embeddings
    b. For each triplet, TranSE assumes that the relation r is a translation vector between vi and in vj in the embedding space, so that:
        i. $Vi + r \approx Vj$

An Overview of KG-BART

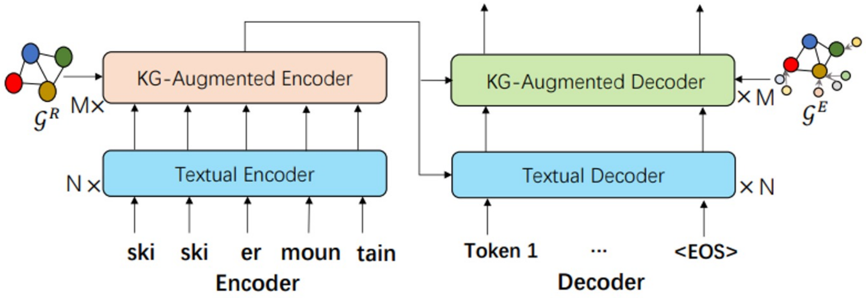


Figure 2: The proposed KG-BART model.

- Traditional textual Transformer encoder module
  - to represent the contextual information of each token
- KG-augmented Transformer module based on graph attention mechanism
  - to integrate the entity-oriented knowledge information into token representation.
- Textual Transformer decoder module
- KG-augmented Transformer decoder module

Encoder

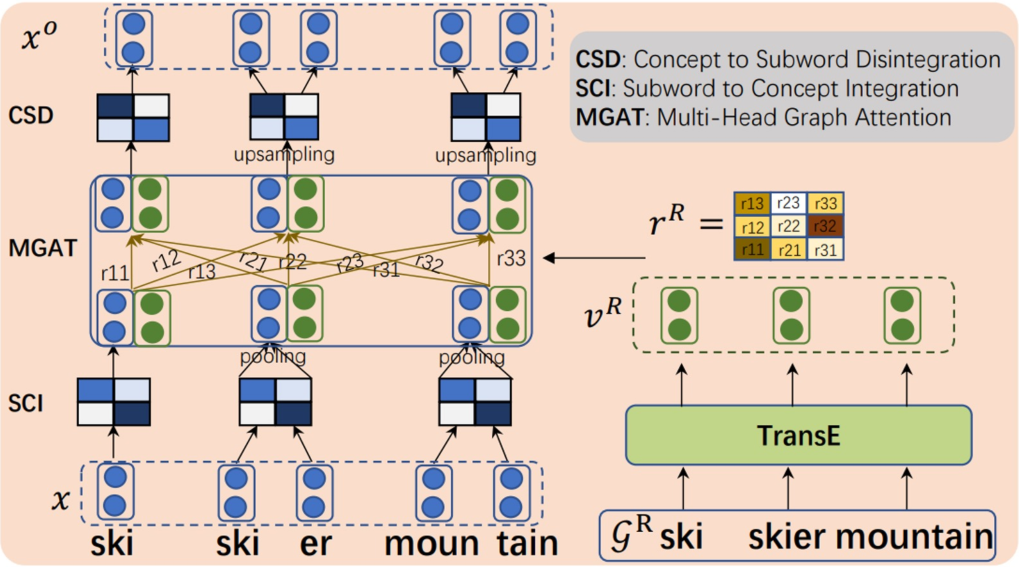


Figure 3: The KG-augmented encoder.

- Integrates the input token embeddings {x1, . . . , xn}, which is the output of the textual encoders, as well as the embedding of GR to update the token representation as $\{\mathbf{x}_1^o, \ldots, \mathbf{x}_n^o\}$
- Incorporates graph representations into the neural encoding process via a graph-informed attention mechanism

Encoder:Subword to Concept Integration (SCI)

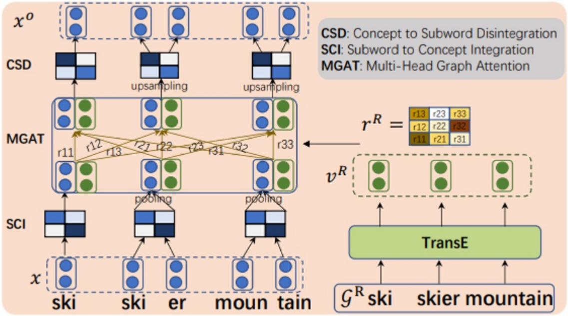


Figure 3: The KG-augmented encoder.

1. Grouping the subwords for each concept
   a. **BART** models each token as subwords, unlike GPT
   b. because concepts in the KG are at word-level
2. Applying convolutional neural network (CNN) (Kim 2014) with a max-pooling layer
   a. To efficiently obtain the representation in word-level
   b. Concept Ci is made up of a sequence of subwords {x1, x2, . . . , xm},
   c. Conv1D layer: ('L' here is the sequence length of a window)

$$\mathbf{Z}\left(\mathbf{x}_t, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_{t+l-1}\right)^T, t \in [1, m-l+1]$$

   a. Max-pooling layer:

$$\mathbf{e}\left(c_i\right) = \operatorname{MaxPooling}\left(\mathbf{x}_1', \ldots, \mathbf{x}_{m-l+1}'\right)$$

   a. Final word-level textual embedding of concept:

$$\mathbf{e}^w = \{\mathbf{e}(c_1), \ldots, \mathbf{e}(c_l)\}$$

Multi-Head Graph Attention (MGAT)

- Iteratively update the representations for each concept V^R_i through its neighbors N^R_i
- Notations:
  - $\mathbf{H} = [\mathbf{e}^w;\ \mathbf{W}_e \mathbf{v}^R]$ Word-level hidden state H: concatenation of the word embedding e^w and the graph-based embedding transformed by a weight matrix W
    aij -> attention weights:

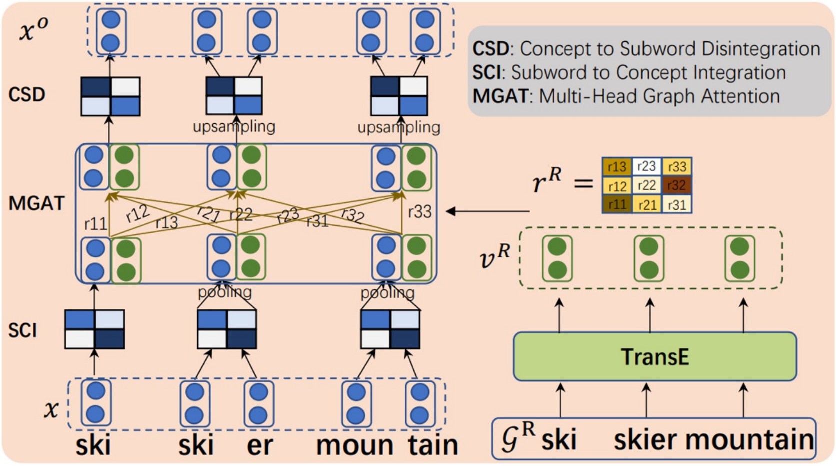


Figure 3: The KG-augmented encoder.

$$z_{ij} = \mathrm{LeakyReLU}\left(\mathbf{W}_a \left[\mathbf{W}_q \mathbf{h}_i; \mathbf{W}_k \mathbf{h}_j; \mathbf{W}_r \mathbf{r}_{ij}^R\right]\right),$$

$$\alpha_{ij} = \frac{\exp(z_{ij})}{\sum_{l=1}^{|\mathcal{N}_i^R|} \exp(z_{il})},$$

$$\mathbf{h}_i' = \|_{k=1}^{K} \sigma \left(\sum_{j=1}^{|\mathcal{N}_i^R|} \alpha_{ij}^k \mathbf{W}_v^k \mathbf{h}_i \right)$$

  - Updating hidden state of node i using message passing
  - Concatenating the attention embeddings from different heads
  - Wa,We,Wr,Wq,Wk and Wv are trainable weights

Concept to Subword Disintegration (CSD)

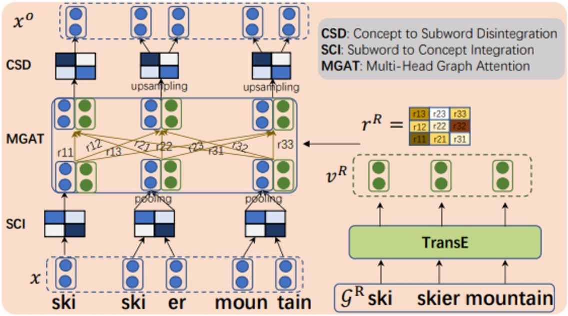


Figure 3: The KG-augmented encoder.

- Disintegrating the concept to the subword-level
- Upsampling word-level hidden state $\mathbf{h}'_i$ with (m−l+1) times (the length before MaxPooling)
- Utilizing a Deconv1D layer with vector Z = [z0, . . . , zl] ∈ R^(1×l) used in Conv1D to form the Deconv1D matrix ZD ∈ R^(m×(m−l+1)) to get the subword-level hidden state ui
- A two-layer feed-forward network with GeLU activation function and a residual layer normalization to get final output $\mathbf{x}^o_i$

$$[\mathbf{u}_i^1, \ldots, \mathbf{u}_i^m]^T = \begin{pmatrix} z_0 & & & \\ \cdots & z_0 & & \\ z_l & \cdots & \cdots & \\ & z_l & & z_0 \\ & & & \cdots \\ & & & z_l \end{pmatrix} * \begin{pmatrix} \mathbf{h}_i'^1 \\ \mathbf{h}_i'^2 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{h}_i'^{m-l+1} \end{pmatrix}$$

Decoder

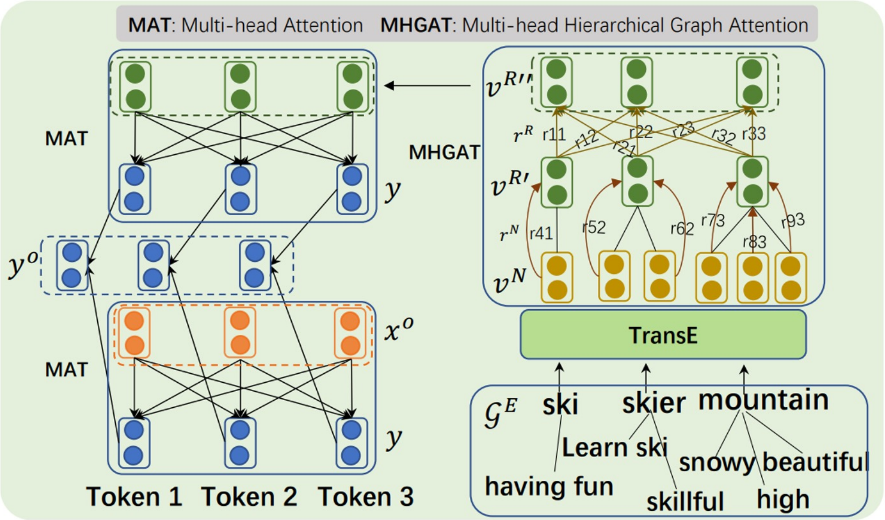


Figure 4: The KG-augmented decoder.

- Incorporating hierarchical graph structure into the decoding process to capture the relations between concepts and their neighboring nodes
- Using **Multi-Head Hierarchical Graph Attention (MHGAT)** to obtain the updated concept embeddings
- The final decoder output is
  - The attention between the encoder hidden state x^o and the previously generated token hidden state y
  - The attention between the updated concept embeddings v^(R”) and the previously generated token hidden state y

$$\mathrm{AT}^{\mathrm{KG}} = \mathrm{MAT}(\mathbf{y}, \mathbf{v}^{R\prime\prime}, \mathbf{v}^{R\prime\prime}), \quad \mathrm{AT}^{\mathrm{TX}} = \mathrm{MAT}(\mathbf{y}, \mathbf{x}^{o}, \mathbf{x}^{o})$$

  - Concatenating of the two attention with a residual connection

$$\mathbf{y}^{o} = \mathbf{W}_{att}[\mathrm{AT}^{\mathrm{KG}};\ \mathrm{AT}^{\mathrm{TX}}] + \mathbf{y}$$

  - y^o is used to predict the token sequence: $P_{\text{vocab}} = \mathrm{softmax}(W_{\text{out}} y_o + b_{\text{out}})$

Multi-Head Hierarchical Graph Attention (MHGAT)

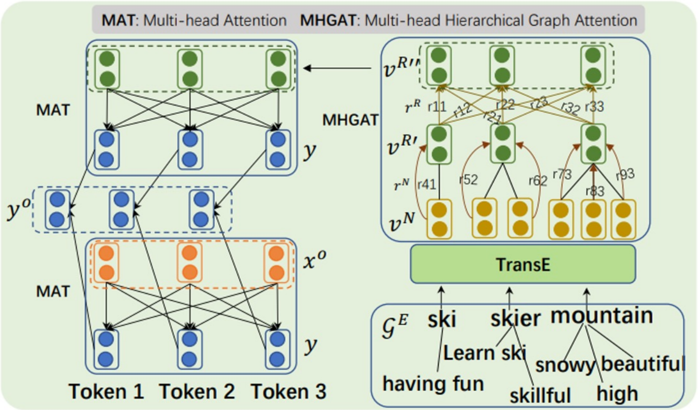


Figure 4: The KG-augmented decoder.

- The first layer of hierarchical graph attention is to update the concept node $\mathbf{v}_i^R \in \mathbb{R}^{de}$ through its inter-concept neighboring nodes $\mathcal{N}_i^N$ with relation embedding

$$z_{ij} = \mathrm{LeakyReLU}\left(\mathbf{W}_a\left[\mathbf{W}_q\mathbf{v}_i^R; \mathbf{W}_k\mathbf{v}_j^N; \mathbf{W}_r\mathbf{r}_{ij}^N\right]\right),$$

$$\alpha_{ij} = \frac{\exp(z_{ij})}{\sum_{l=1}^{|\mathcal{N}_i^N|}\exp(z_{il})}, \quad \mathbf{v}_i^{R\prime} = \|_{k=1}^{K}\sigma\left(\sum_{j=1}^{|\mathcal{N}_i^N|}\alpha_{ij}^k\mathbf{W}_v^k\mathbf{v}_j^R\right). \tag{5}$$

- The second graph attention layer updates the concept representation considering the intra-concept relations

KG-BART Model Pre-Training

- Similar to BART
- Corrupting texts and then optimizing a reconstruction loss, the cross-entropy, between the decoder’s output and the original texts
- For example, “[mask] wound [mask] teach soldier” in the encoder and “student wound treat teach soldier” in the decoder

The experiment

- Dataset:
    - CommonGen
    - Testing the ability of machines on commonsense reasoning when generating a text
    - 77k commonsense descriptions over 35k unique concept sets

	Train	Dev	Test
# **Concept sets**	32,651	993	1,497
# **Sentences**	67,389	4,018	6,042
% **Unseen Concepts**	-	6.53%	8.97%
% **Unseen Concept-Paris**	-	96.31%	100.00%
% **Unseen Concept-Triples**	-	99.60%	100.00%

Table 1: The basic statistics of the CommonGen dataset.

Results

Table 2: Experimental results of different baseline methods on the CommonGen test dataset. We show the best results in boldface, and those with the second best performance are underlined

Model\Metrics	BLEU-3/4		ROUGE-2/L		METEOR	CIDEr	SPICE	Coverage
GPT-2 (Radford et al. 2019)	30.70	21.10	17.18	39.28	26.20	12.15	25.90	79.09
BERT-Gen (Bao et al. 2020)	30.40	21.10	18.05	40.49	27.30	12.49	27.30	86.06
UniLM (Dong et al. 2019)	38.30	27.70	21.48	<u>43.87</u>	29.70	14.85	30.20	89.19
UniLM-v2 (Bao et al. 2020)	31.30	22.10	18.24	40.62	28.10	13.10	28.10	89.13
T5-Base (Raffel et al. 2020)	26.00	16.40	14.57	34.55	23.00	9.16	22.00	76.67
T5-Large (Raffel et al. 2020)	<u>39.00</u>	<u>28.60</u>	22.01	42.97	30.10	<u>14.96</u>	<u>31.60</u>	95.29
BART (Lewis et al. 2020)	36.30	26.30	<u>22.23</u>	41.98	<u>30.90</u>	13.92	30.60	<u>97.35</u>
Human Performance	48.20	44.90	48.88	63.79	36.20	43.53	63.50	99.31
KG-BART	**42.10**	**30.90**	**23.38**	**44.54**	**32.40**	**16.83**	**32.70**	**98.68**

- Automatic metrics to automatically assess the performance
- BLEU, ROUGE and METEOR mainly focus on measuring ngram similarities between model output and reference descriptions
- CIDEr and SPICE focus on evaluating the associations between mentioned concepts instead of n-gram overlap: content relevance and accuracy with respect to a structured knowledge source
- Coverage of concepts

Human evaluations

Model	1	2	3	4	5	Rating
GPT-2	22%	16%	23%	20%	19%	2.98
UniLM	5%	17%	22%	24%	32%	3.61
T5-large	2%	15%	12%	32%	39%	3.91
BART	1%	10%	17%	30%	42%	4.02
KG-BART	0 %	8%	12%	25%	55%	**4.27**

Table 3: Ranking results of system outputs by human evaluation. 1 is the worst and 5 is the best. The larger rating denotes a better summary quality.

(1) Rationality: is the sentence the reasonable commonsense scenario? (2) Fluency: is the sentence fluent and grammatical? (3) Succinctness: does the sentence avoid repeating information? (4) Naturalness: does the sentence use adjunct words?

KG-BART encoder can capture the better relationship between concepts

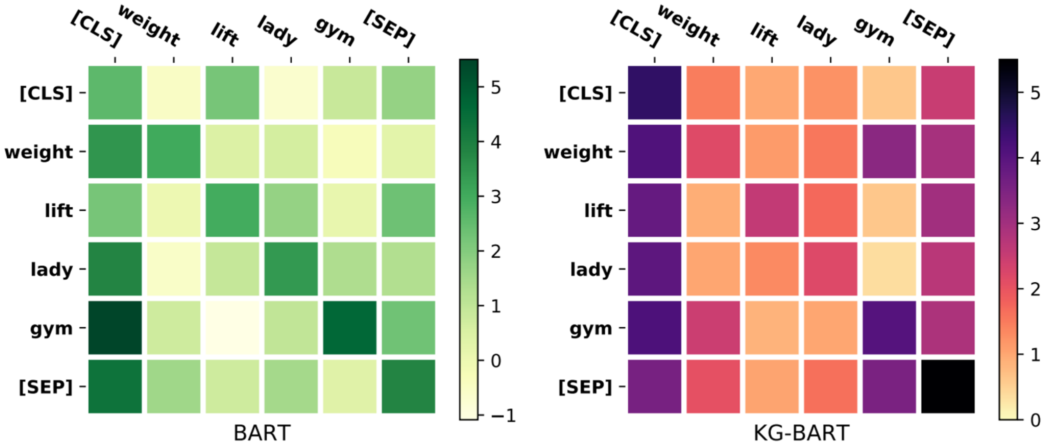


Figure 6: Attention weights of the last layers of BART and KG-BART encoder.

- Related concept pairs in KG-BART attend much more attention

Transfer KG-BART to Commonsense QA

- Extracting the nouns and verbs in questions and five choices, and combine the concepts of question q and each choice ci to build concept sets
- For example, q="What would you do if you want to be able to earn money?", ci="apply for job" (correct) with gi="applying for a job so i would earn money."; cj="stand in line" (wrong) gj="i would want to earn money standing in line to get a deal on a product."
- More reasonable and natural sentences for correct choices while noisy sentences for wrong choices

Conclusion

- KG-augmented approach KG-BART is based on pre-trained BART for generative commonsense reasoning
- KG-BART can generate high-quality sentences even in the unseen concept sets
- KG-BART has better abilities of both commonsense reasoning and text generalization
- Paper published in 2021, at the time data-driven conversational agents like Apple’s Siri, Google Assistant and Amazon’s Alexa are struggling at achieving the ability of commonsense reasoning on generating the human-like responses

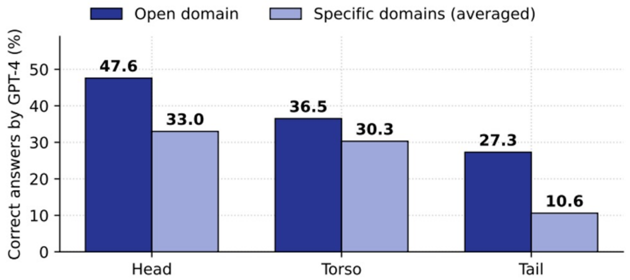
Head-to-Tail: How Knowledgeable are Large Language Models (LLMs)?

Kai Sun, Yifan Ethan Xu, Hanwen Zha, Yue Liu, Xin Luna Dong

https://arxiv.org/abs/2308.10168

Overview

- Head-to-Tail: a benchmark that consists of 18K question-answer (QA) pairs regarding head, torso, and tail facts in terms of popularity
- Context: the rise of LLMs has sparked debates on whether Knowledge Graphs (KGs), which store real-world factual knowledge in triplet form (subject, predicate, object), will be replaced with LLM
- Trying to answer How knowledgeable are LLMs



Example questions where GPT-4 gives incorrect answers

Movie

Question: What profession does Tj Singh (known for John Carter (2012)) have?
Ground Truth: Visual effects
GPT-4: Actor

Book

Question: Who authored Choke (published in 1996)?
Ground Truth: Stuart Woods
GPT-4: Chuck Palahniuk

Academics

Question: Where did Josef Kittler receive the Ph.D. (thesis: Development and application of pattern recognition techniques.)?
Ground Truth: University of Cambridge, UK
GPT-4: University of Surrey

Open

Question: What college is the sister college of Trinity College, Oxford?
Ground Truth: Churchill College, Cambridge
GPT-4: Balliol College

Figure 1: The question-answering accuracy of GPT-4 decreases in the order of head, torso, and tail entities on the Head-to-Tail benchmark, and is only 31% on

- How reliable are LLMs in answering factual questions?
- Do LLMs perform equally well on head, torso, and tail facts?
- Does normal methods that improve LLMs increase the factuality?
- Motivation
  - hard to directly “query” the knowledge embedded in an LLM
  - No ready to use benchmark that well represents user interest and uniform distribution of world knowledge
- Contribution
  - A benchmark to test knowledge
  - Evaluation method and metrics
  - Comprehensive evaluation of 16 LLMs

Domains

Selected three domains along with a KG where public data are easily accessible.

- DBpedia knowledge graph, where the knowledge originates from Wikipedia (English snapshot from December 1, 2022.2).
- Movie: We used a snapshot of IMDb3 from May 21, 2023.
- Book: We used the data of Goodreads scraped in 2017 released by Wan and McAuley (2018).
- Academics: We used a snapshot of MAG (Sinha et al., 2015) from September 13, 2021 and DBLP4 from May 10, 2023.

Head, torso and tail entities

- Decided by the popularity of the entities
- Two ways to approximate popularity: traffic and density
    - Traffic: views and votes
    - Density: the number of facts or authored works about the entity
    - When there is traffic information, we conveniently use traffic to measure the popularity
    - Head entities comprising entities whose cumulative popularity score is up to 1/3 of that of all entities, torso entities comprising entities with cumulative scores ranging from 1/3 to 2/3, and tail entities from 2/3 to 1
    - A specific movie vs more niche topics

Questions generation

- Generating questions using a template-based approach, where each generated question asks for an attribute of an entity
- Filtering out the following types of attributes: (i) unspecific (ii) dynamic (e.g., lastLaunchRocket in DBpedia), (iii) data source specific (e.g., averageRating in IMDb), and (iv) non-textual (e.g., picture in DBpedia)

```
You are given a few samples of a relation in the format of <X, relation, Y>.
You need to write a question *template* about the relation, which can be
used to generate questions. The template needs to have one blank such that
a question about Y can be generated by filling the blank with X.

#Example 1
Samples: <!Hero, musicBy, Eddie DeGarmo>, <9 to 5 (musical), musicBy,
Dolly Parton>, <All About Us (musical), musicBy, John Kander>
Template: The music of _ is by whom?

#Example 2
Samples: <10,000 Maniacs, bandMember, Dennis Drew>, <16bit (band),
bandMember, Eddie Jefferys>, <1TYM, bandMember, Teddy Park>
Template: Name a band member of _?

#Example 3
Samples: {SAMPLES}
Template:
```

Question generation(cont'd)

Domain	Sources	# Templates	# Questions
Movie	IMDb	13	3,093
Book	Goodreads	4	3,000
Academics	MAG, DBLP	13	2,946
Open	DBpedia	393	9,132
Total		423	18,171

Table 2: The overall statistics of Head-to-Tail.

- For each specific domain: ∼1K questions for each of the head, torso, and tail buckets.
- DBPedia: ∼3K questions for each bucket. T

Metrics

- Accuracy (A), hallucination rate (H), and missing rate (M)
- Models are prompted to reply with “unsure” for uncertain answers
- Models are prompted to reply with concise answers
- A + H + M = 100%.
- Judging tools:
  - LLM-based
  - Rule-based(i.e. Exact match/average normalized longest common subsequence)
  - Variants given
    - (e.g., “W Shakespeare” is a variant of “William Shakespeare”)

Experiment Analysis

Model	All		Open		Movie		Book		Academics	
	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}
GPT-4	30.9	19.7	37.1	25.3	41.7	15.5	21.3	19.4	10.0	6.8
ChatGPT	20.3	14.1	22.1	14.8	34.7	13.3	16.9	24.9	3.0	1.9
Llama 2 (70B)	11.8	34.0	7.5	24.8	27.9	34.3	10.3	54.5	9.8	41.0
LLaMA (33B)	18.2	80.0	19.0	79.1	28.7	70.1	15.8	82.9	7.1	90.3

Table 3: The best overall accuracy is only $\sim$31% on Head-to-Tail. All numbers are in percentage (%).

RQ1: How reliable are LLMs in answering factual questions?

- GPT-4 and ChatGPT give unsure or empty answers for the majority of them, and the hallucination rate is <20% (still non-negligible)
- LLaMA-33B mostly provides hallucinated answers
- The overall performance varies substantially across different specific domains

Model	All		Open		Movie		Book		Academics	
	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}
GPT-4	30.9	19.7	37.1	25.3	41.7	15.5	21.3	19.4	10.0	6.8
ChatGPT	20.3	14.1	22.1	14.8	34.7	13.3	16.9	24.9	3.0	1.9
Llama 2 (70B)	11.8	34.0	7.5	24.8	27.9	34.3	10.3	54.5	9.8	41.0
LLaMA (33B)	18.2	80.0	19.0	79.1	28.7	70.1	15.8	82.9	7.1	90.3

Table 3: The best overall accuracy is only ~31% on Head-to-Tail. All numbers are in percentage (%).

RQ2: Do LLMs perform equally well on head, torso, and tail facts?

Domain	Head		Torso		Tail	
	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}
Movie	59.3	14.8	55.0	16.9	10.9	14.7
Book	22.8	24.4	24.3	21.8	16.9	12.0
Academics	15.8	9.9	10.5	6.8	3.9	3.7
Open	47.6	30.2	36.5	24.1	27.3	21.6
All	40.3	23.3	33.4	19.7	19.0	15.9

(a) GPT-4.

Domain	Head		Torso		Tail	
	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}
Movie	39.2	28.2	33.9	29.8	10.7	44.9
Book	15.0	52.2	12.9	54.4	3.1	56.9
Academics	12.9	35.2	11.1	38.2	5.3	49.7
Open	9.9	22.3	6.9	25.4	5.7	26.8
All	16.2	30.3	13.2	33.0	6.1	38.6

(b) Llama 2-70B.

Table 4: LLMs' factuality, measured by A_{LM} (%), decreases in the order of head, torso, and tail entities from Head-to-Tail.

- The overall accuracy of GPT-4 and Llama 2-70B (ALM) declines in the order of head, torso, and tail entities
- The same pattern is seen in other LLMs

Model	$\mathbf{A_{LM}}$	$\mathbf{H_{LM}}$	**M**
GPT-4	46.0 (↑5.7)	21.4 (↓1.9)	32.6 (↓3.7)
Llama 2 (70B)	18.7 (↑2.5)	29.7 (↓0.6)	51.6 (↓1.9)

Table 5: Accuracy on the top-10% popular questions in the head bucket is only slightly better than overall head entities. (↑/↓: increased/decreased % compared with using all head instances.)

RQ3: Does normal methods that improve LLMs increase the factuality

- Increased model size does not automatically translate to a better grasp of factual knowledge
- The instruction-tuned counterparts (i.e., Vicuna and Falcon-Instruct) have lower accuracy
    - More conservative in providing factual

Model	Head-to-Tail			Head		Torso		Tail	
	A_{LM}	H_{LM}	M	A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}
LLaMA (7B)	12.1	80.0	7.9	19.0	74.4	11.7	81.0	5.4	84.8
LLaMA (13B)	14.4	84.3	1.3	22.0	77.2	14.8	83.8	6.3	91.9
LLaMA (33B)	18.2	80.0	1.8	26.0	72.8	19.8	78.7	8.8	88.6
LLaMA (65B)	17.8	81.9	0.3	25.9	73.8	18.7	81.0	8.7	90.9
Vicuna (7B)	10.1	79.2	10.8	16.2	72.7	9.6	79.8	4.3	85.0
Vicuna (13B)	9.2	62.6	28.2	14.0	55.0	8.8	62.8	4.7	70.0
Flan-T5 (3B)	2.3	17.4	80.3	3.9	19.7	1.5	17.1	1.3	15.5
Flan-T5 (11B)	4.2	20.0	75.7	7.6	23.7	3.2	19.9	2.0	16.5
Falcon (7B)	9.5	57.9	32.6	14.5	53.8	9.2	57.9	4.8	62.0
Falcon (40B)	10.8	41.0	48.2	16.2	36.4	11.2	40.0	4.9	46.6
Falcon-Instruct (7B)	6.8	56.7	36.5	11.5	56.0	5.6	57.2	3.4	56.7
Falcon-Instruct (40B)	10.8	32.2	57.0	16.7	30.5	11.5	31.1	4.3	34.8

Table 7: Comparison of different LLMs with different sizes. All numbers are in percentage (%).

Robustness of the evaluation methodology

- Correlations between rule- and LLM-based metrics are high, indicating the rule based metrics are good alternative
- Removing "unsure" and "brief" increases hallucination rate
- In-domain example prompt help get more correct answers, compared to zero shot and few shot

LLM-Based	Rule-Based	ρ Min.	ρ Mean	r Min.	r Mean
A_{LM}	A_{EM}	0.721	0.915	0.921	0.966
	A_{F1}	0.775	0.951	0.781	0.969
	A_{RL}	0.730	0.947	0.775	0.969
H_{LM}	H_{EM}	0.968	0.991	0.993	0.998
	H_{F1}	0.976	0.995	0.998	0.999
	H_{RL}	0.976	0.995	0.998	0.999

Table 8: The minimum and mean Spearman's rank correlation coefficients (ρ) and Pearson correlation coefficients (r) show high correlation between LM- and rule-based metrics.

	Domain	Few-shot		Zero-shot		In-domain	
		A_{LM}	H_{LM}	A_{LM}	H_{LM}	A_{LM}	H_{LM}
Head	Open	32.7	**20.8**	32.6	24.7	**45.0**	27.8
	All	29.4	**17.2**	29.2	18.6	**38.3**	24.7
Torso	Open	19.7	**13.3**	21.6	17.9	**30.1**	23.0
	All	21.9	**14.6**	22.8	16.7	**29.8**	22.8
Tail	Open	13.8	**10.2**	14.9	14.5	**23.0**	19.5
	All	9.5	**10.5**	10.3	12.7	**15.4**	20.2

Table 9: Performance of ChatGPT with different prompts on Head-to-Tail. All numbers are in percentage (%).

Conclusion

- The amount of this encoded knowledge in LLMs remains limited
- Mediocre QA accuracy for popular entities
- Torso-to-tail and recent knowledge is best to be represented by KG
- New research areas need to seamlessly blend knowledge in the symbolic form and neural form

Future Work

Future Work for LLMs in Graph Processings

1. **LLMs for Multi-modal Graphs**: Integrate multi-modal data (text, images, audio) with graph models for comprehensive understanding and reasoning.
2. **Efficiency and Computational Cost**: Develop efficient strategies to reduce training and inference costs for large-scale graph data, especially when combining LLMs with GNNs.
3. **Tackling Complex Graph Tasks**: Expand LLM applications beyond traditional tasks like link prediction and node classification to more generative and understanding tasks such as graph generation and graph-based QA.
4. **User-Centric Agents on Graphs**: Design interactive, adaptive LLM agents for multi-run tasks, capable of handling a variety of user inputs with iterative feedback and dynamic adjustments.