Stanford CS224W: Large Language Models and GNNs

CS224W: Machine Learning with Graphs

Jure Leskovec and Charilaos Kanatsoulis, Stanford University

http://cs224w.stanford.edu

Announcements

- Congratulations on completing the exam!
- **Colab 4** due today (12/3)
- **Colab 5** due Thursday (12/5)
- **Project Report** due next Thursday (12/12)
- Last lecture is this Thursday

NVIDIA

GNNs & LLMs in PyG

By: Rishi Puri, Junhao Shen, & Zack Aristei
NVIDIA, Southern Methodist University, & Georgia Tech

LLM/Transformer Intro

- LLMs (Transformer) excel at predicting the next token in sequences
- “Attention is all you need” is plateauing
- LLM’s pretrained for single hop logic:
    - Given context tokens, predict next tokens
- Single hop logic -> associative memory
    - Fails out of distribution
- Infinite out of distribution tasks in NLP, robotics, medicine, etc

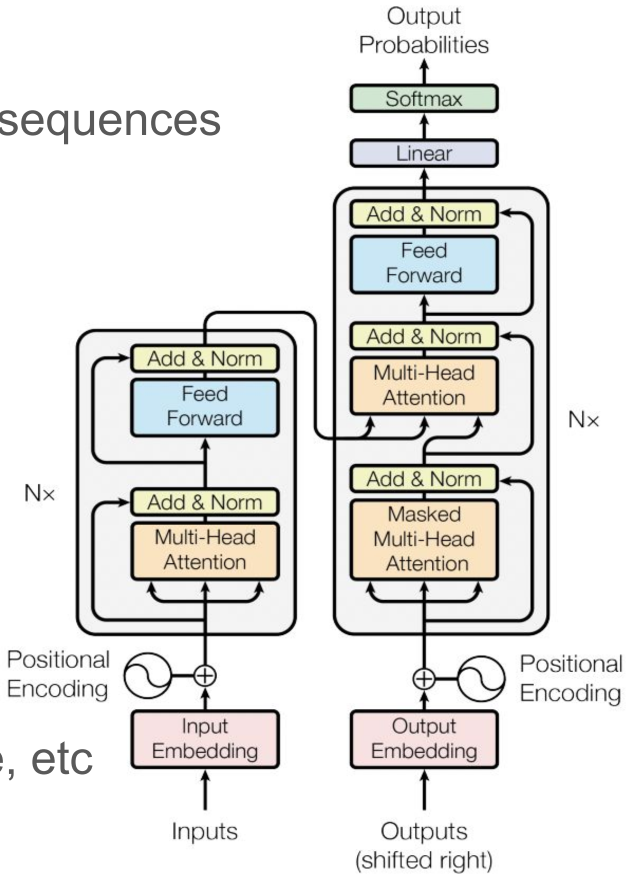
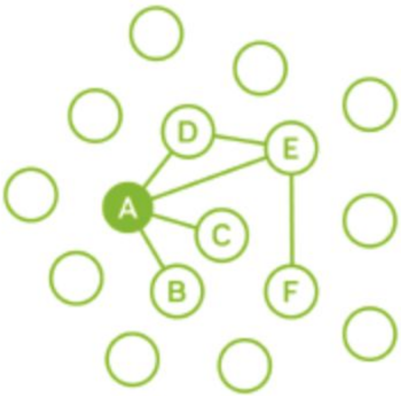
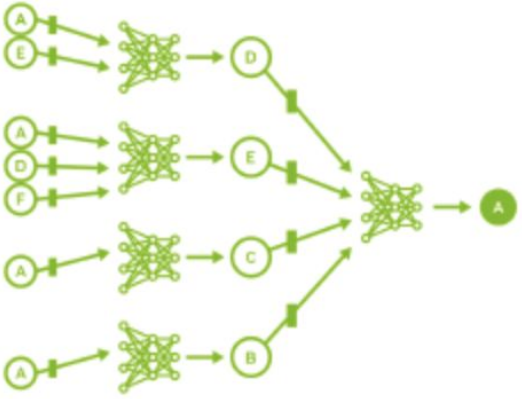


Figure 1: The Transformer - model architecture.

GNN Intro

- The “Transformer layer can be seen as a special GNN that runs on a fully connected ‘word’ graph” - Jure Leskovec CS224W
- GNNs are ideal encoder for graphs
- N layer GNN -> N hops of logic
- GNNs can improve LLM accuracy by adding semi-orthogonal information
- size(LLM) >> size(GNN) -> almost no compute cost for adding GNN to LLM





PyG 2.6: Initial GNN+LLM features

- G-retriever*
- WebQSP* dataset (RAG* Question Answering with Knowledge Graph Context)
    - Source Used: https://huggingface.co/datasets/Youm9602/RoG-webqsp
- Training example for G-retriever on WebQSP
    - https://github.com/pyg-team/pytorch_geometric/blob/master/examples/llm/g_retriever.py
    - Default LLM+GNN: Llama2-7b + Graph Attention Transformer (GAT)
        - TinyLLama-1.1B with --tiny_llama flag
        - Can easily swap in any Huggingface LLM
- GPU speeds up orders of magnitude over CPU!

*RAG = Retrieval Augmented Generation

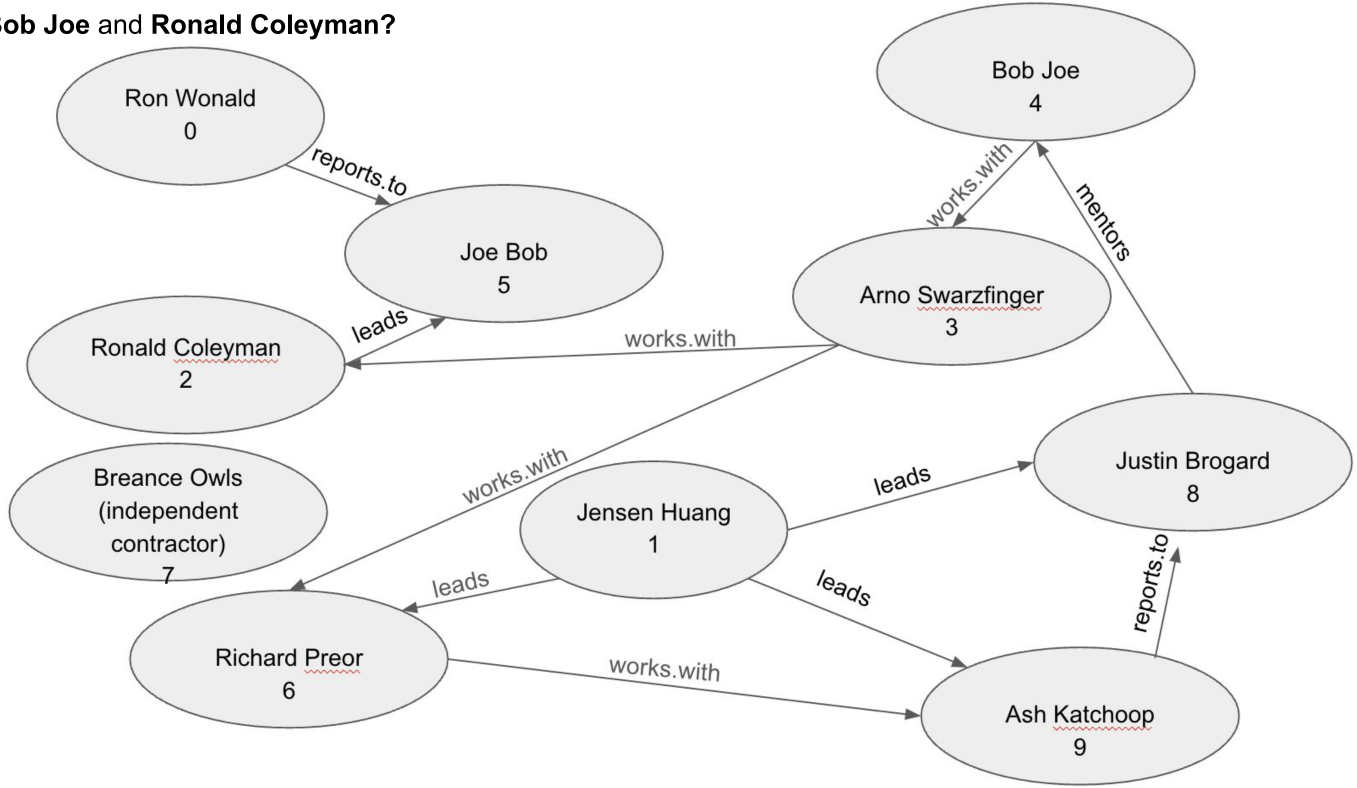
*G-retriever: https://arxiv.org/abs/2402.07630

Example: Made Up NVIDIA Org Chart

Question:
Who is the shared coworker between **Bob Joe** and **Ronald Coleyman?**
Answer:
Arno Swarzfinger

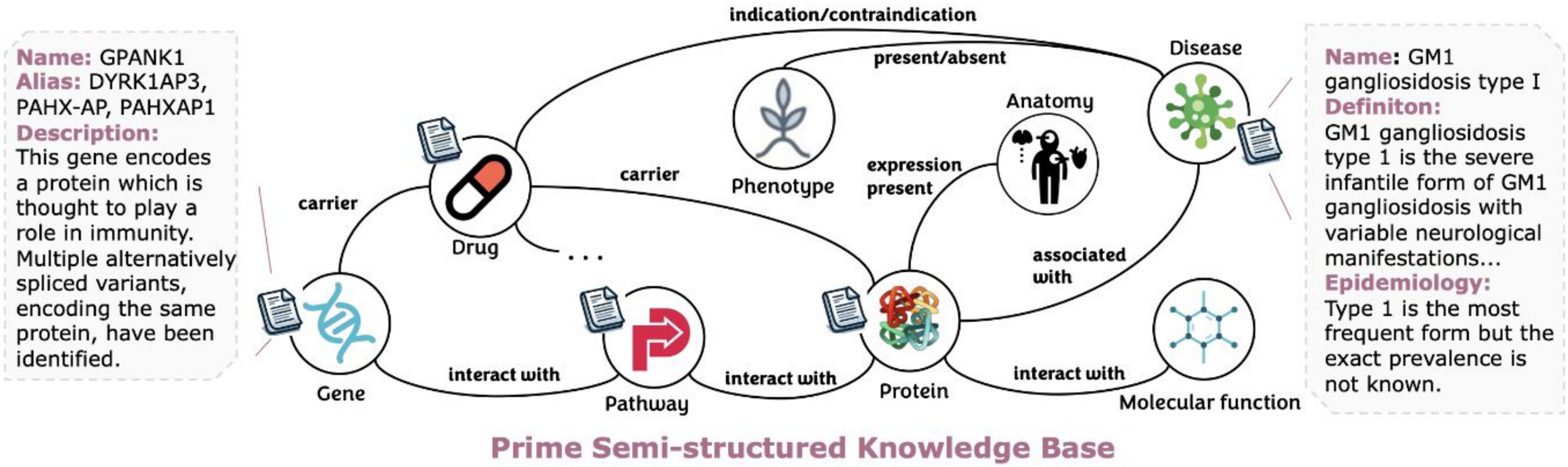
node_id,node_attr
0,Ron Wonald
1,Jensen Huang
2,Ronald Coleyman
3,Arno Swarzfinger
4,Bob Joe
5,Joe Bob
6,Richard Preor
7,Breance Owls
8,Justin Brogard
9,Ash Katchoop
[[stop]]

src,edge_attr,dst
0,reports.to,5
9,reports.to,8
8,mentors,4
2,leads,5
1,leads,6
4,works.with,3
3,works.with,2
1,leads,9
1,leads,8
3,works.with,6
6,works.with,9
[[stop]]



Neo4j Case Study

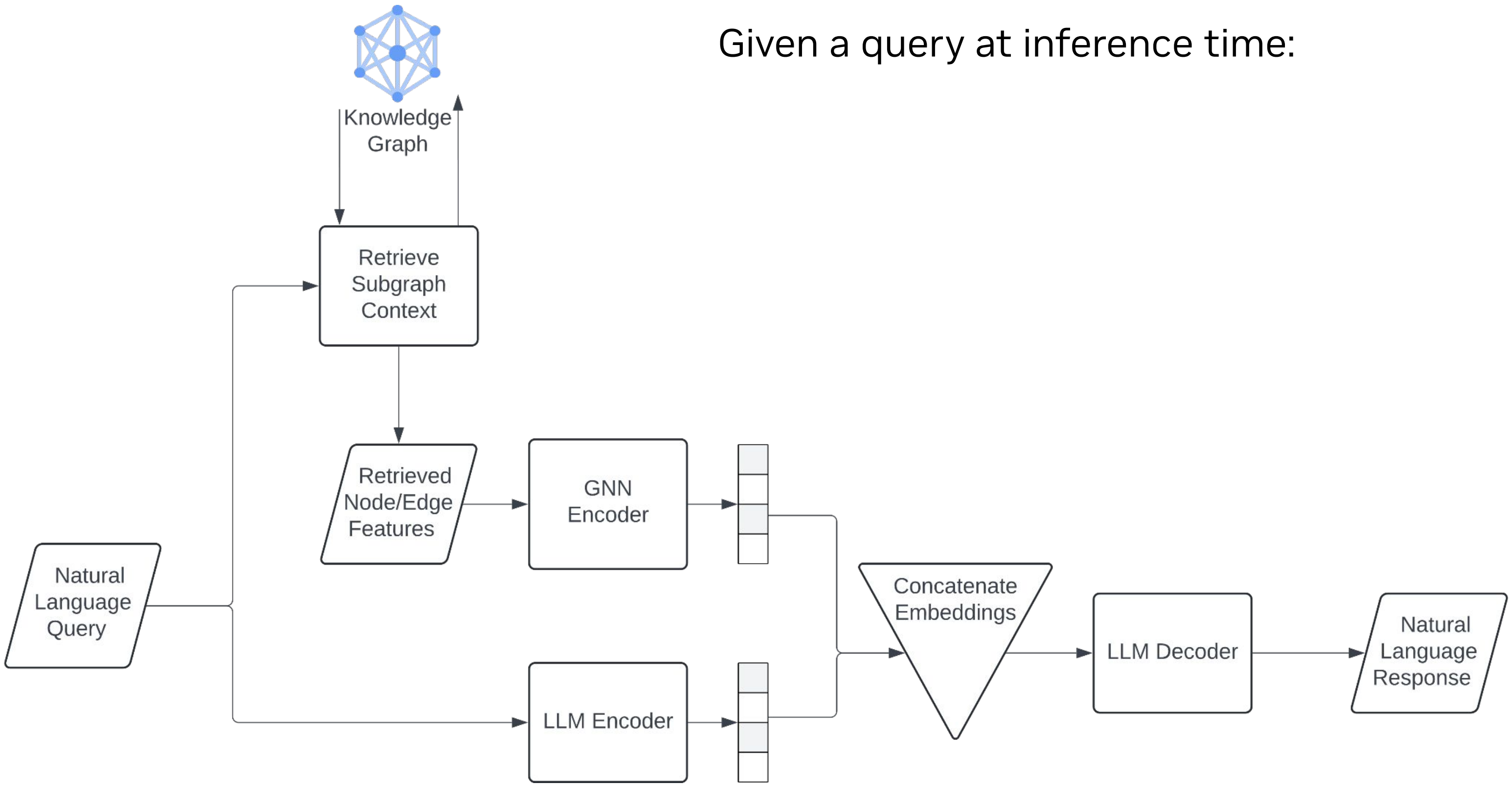
- Neo4j* Cyphers w/ PyG 2.6 G-retriever on Stark Prime (https://stark.stanford.edu/)
- More than 2x the top benchmark hit@1! (.15->.32)
- https://github.com/neo4j-product-examples/neo4j-gnn-llm-example



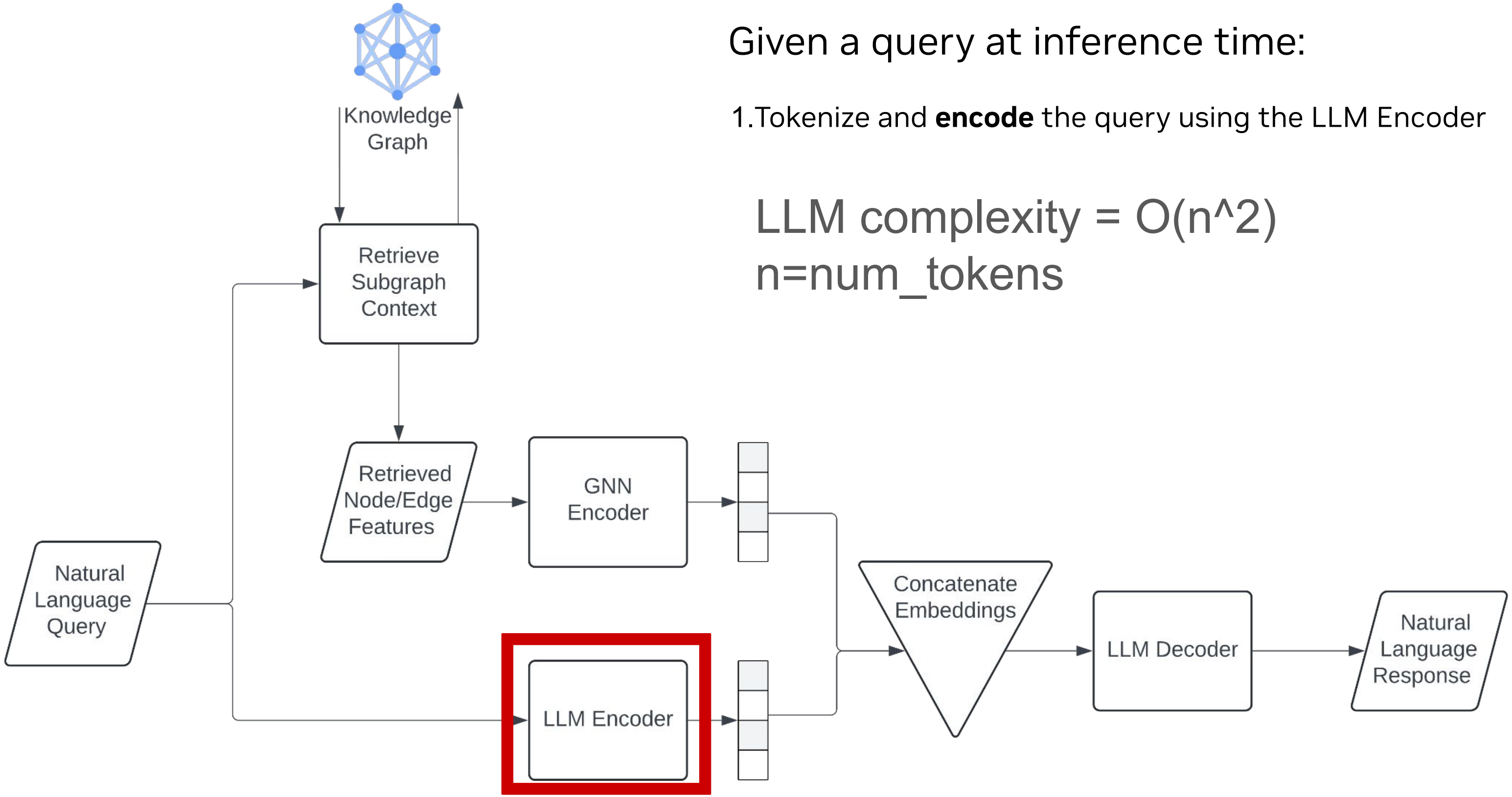
Neo4j (Graph Database and Analytics): https://neo4j.com/

General Graph Based RAG Workflow

Given a query at inference time:



GNN-RAG Workflow

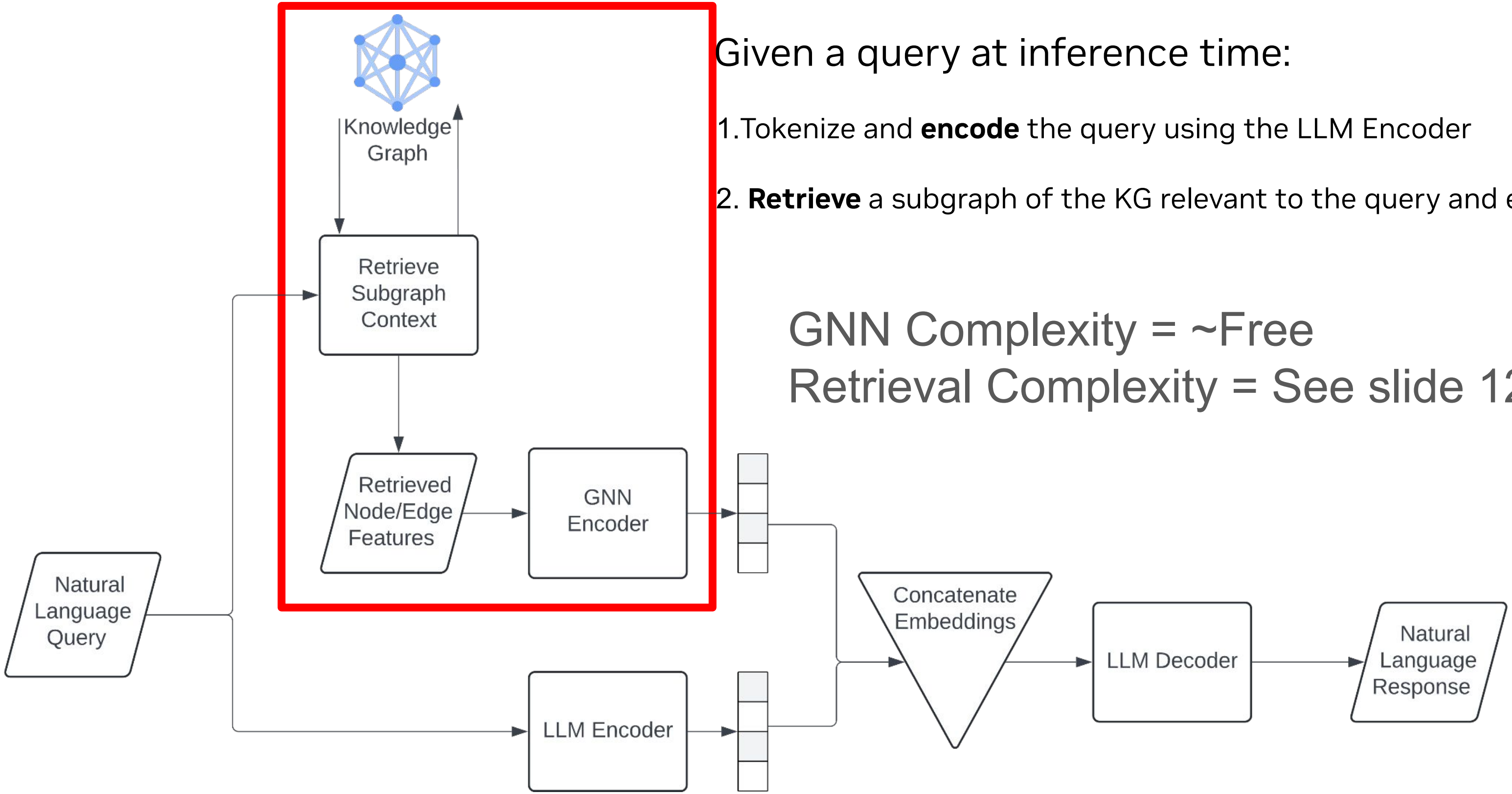


Given a query at inference time:

1. Tokenize and **encode** the query using the LLM Encoder

LLM complexity = O(n^2)
n=num_tokens

GNN-RAG Workflow



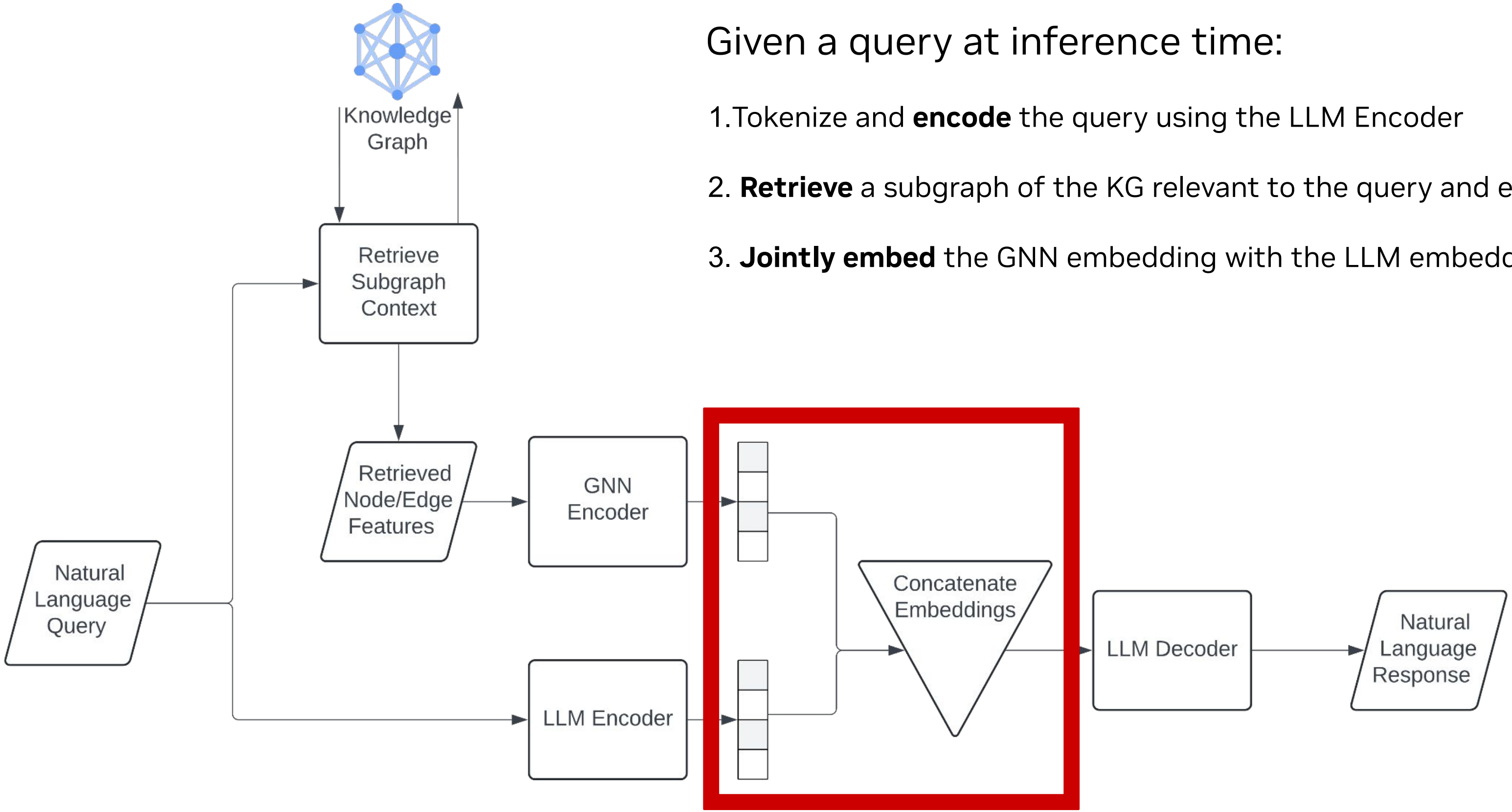
Given a query at inference time:

1. Tokenize and **encode** the query using the LLM Encoder

2. **Retrieve** a subgraph of the KG relevant to the query and encode it using a GNN

GNN Complexity = ~Free
Retrieval Complexity = See slide 12

GNN-RAG Workflow



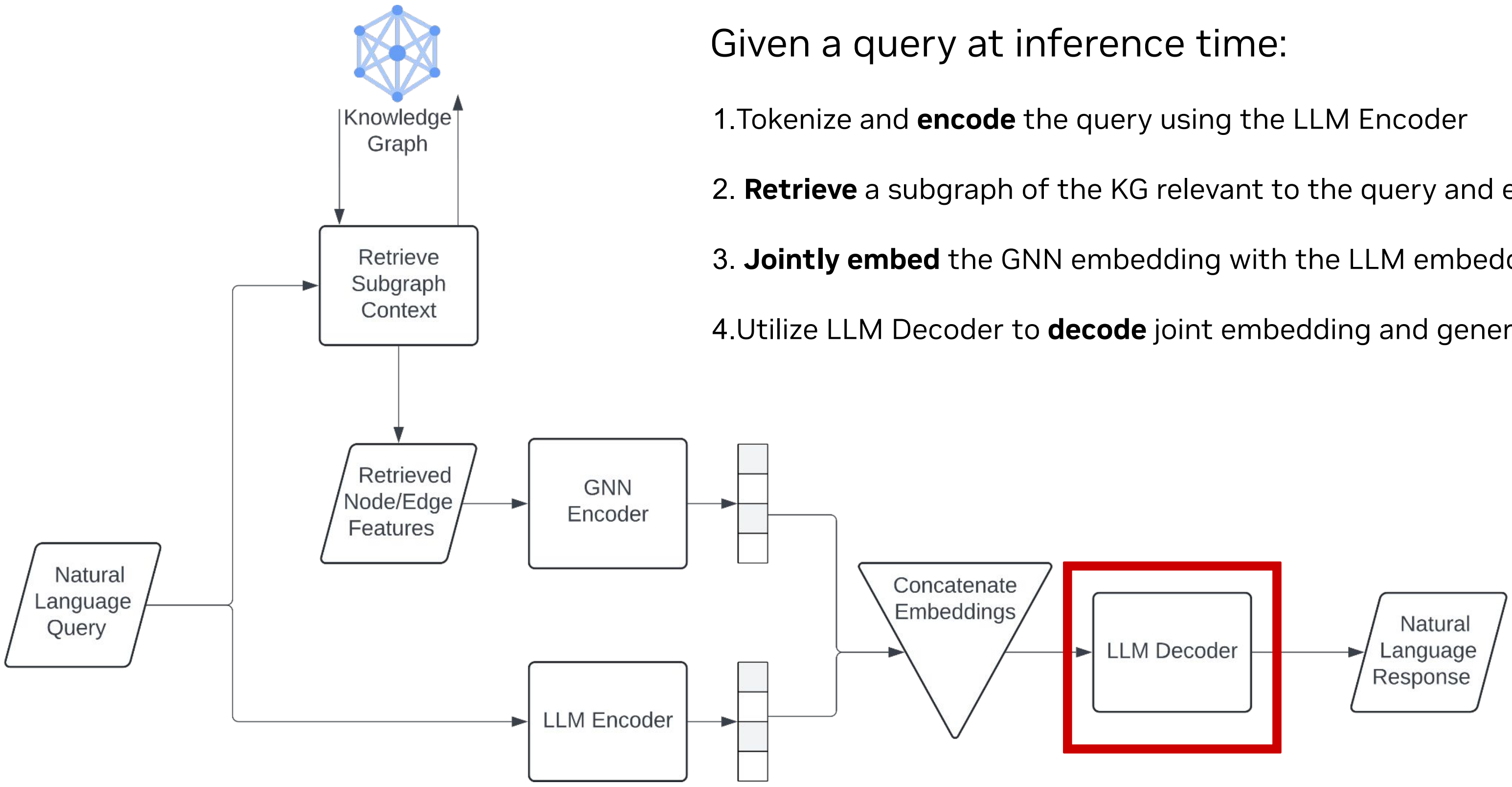
Given a query at inference time:

1. Tokenize and **encode** the query using the LLM Encoder

2. **Retrieve** a subgraph of the KG relevant to the query and encode it using a GNN

3. **Jointly embed** the GNN embedding with the LLM embedding

GNN-RAG Workflow



Given a query at inference time:

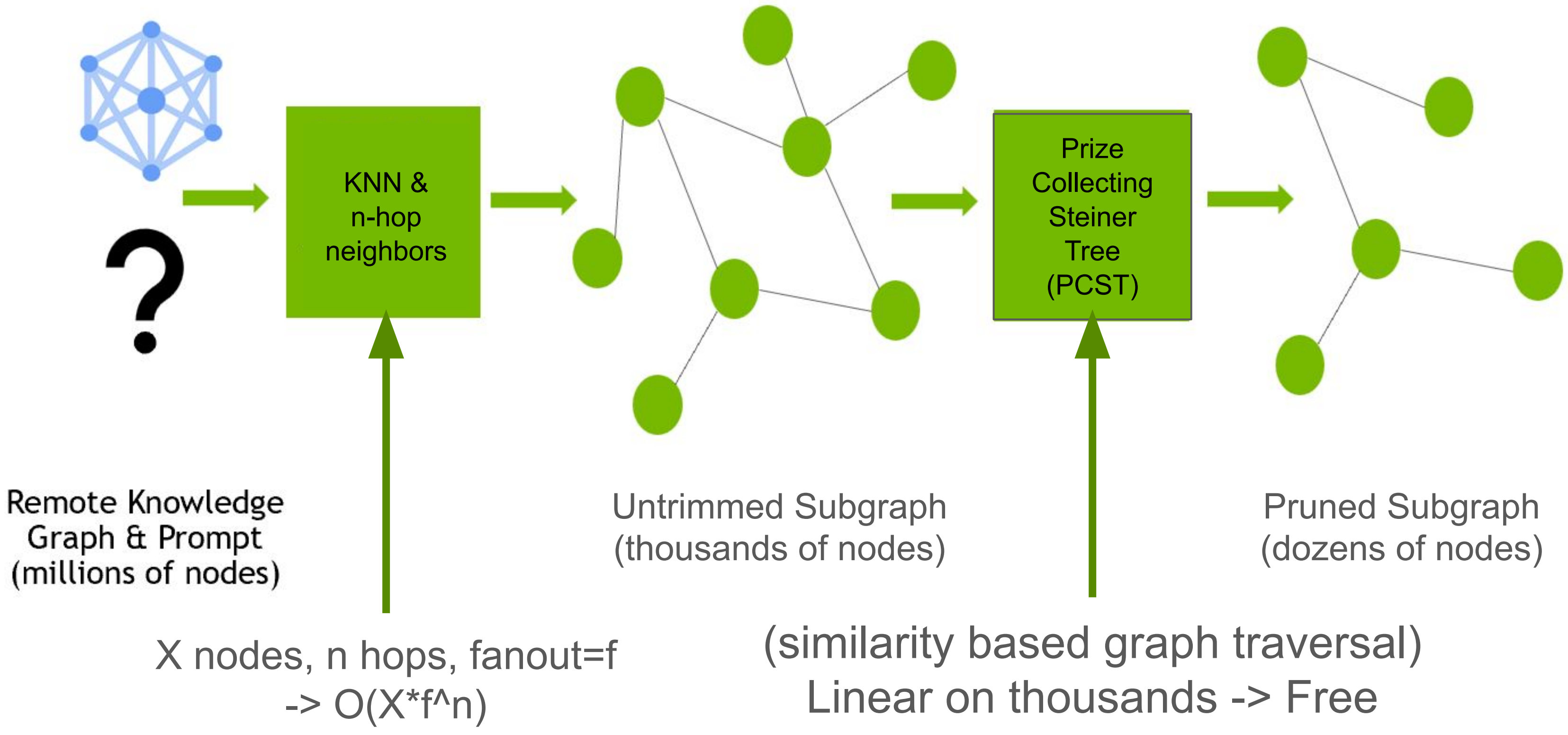
1.Tokenize and **encode** the query using the LLM Encoder

2. **Retrieve** a subgraph of the KG relevant to the query and encode it using a GNN

3. **Jointly embed** the GNN embedding with the LLM embedding

4.Utilize LLM Decoder to **decode** joint embedding and generate a response

Current Retrieval Algorithm



G-retriever Set Up

```
gnn = GAT(in_channels=1024,
          hidden_channels=2048,
          out_channels=1024,
          num_layers=4,
          heads=4,
    )
llm = LLM(model_name='meta-llama/Llama-2-7b-chat-hf', num_params=7)
model = GRetriever(llm=llm, gnn=gnn)
```

- GRetriever: LLM & GNN
- LLM: model name & num_params (in billions)
  - num_params for auto GPU set up
  - Auto GPU Goal: Min # GPUs -> minimize communication overhead

G-retriever: Get Loss

```
# get loss
model(["list", "of", "questions", "here"],
    batch.x # node features,
    batch.edge_index,
    batch.batch, # batch vector, assigns each element to a specific example.
    ["list", "of", "answers", "here"],
    batch.edge_attr, # edge attributes, optional but recommended
    ["list", "of", "textified graphs", "here"]) # optional but recommended
```

G-retriever Inference

```
model.inference(["list", "of", "questions", "here"],
    batch.x # node features,
    batch.edge_index,
    batch.batch, # batch vector, assigns each element to a specific example.
    batch.edge_attr, # edge attributes, optional but recommended
    ["list", "of", "textified graphs", "here"]) # optional but recommended
```

Basic Code WalkThrough

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SentenceTransformer(
    model_name='sentence-transformers/all-roberta-large-v1').to(device)
```

- Sentence Transformer: model(List[str]) -> Embedding Tensor
- Need to call model (3 * num_edges)
  - For each edge, call on both entities and the relation
  - -> Use small LM (SLM) like roberta for efficiency
  - Entities and relations are short phrases. Ex: (cats, eat, dogs)
  - Small LMs have sufficient understanding of short phrases
  - Only need large LM for large/complex bodies of text
  - Future work: Measure tradeoffs for SLM vs LLM

Retrieval Code WalkThrough

```
fs, gs = create_remote_backend_from_triplets(
    triplets=triples, node_embedding_model=model,
    node_method_to_call="encode", path="backend",
    pre_transform=preprocess_triplet, node_method_kwargs={
        "batch_size": min(len(triples), 256)
    }, graph_db=NeighborSamplingRAGGraphStore,
    feature_db=SentenceTransformerFeatureStore).load()
```

- Create PyG Feature Store and Graph Store from Triples
    - Uses SentenceTransformer

Retrieval Code WalkThrough

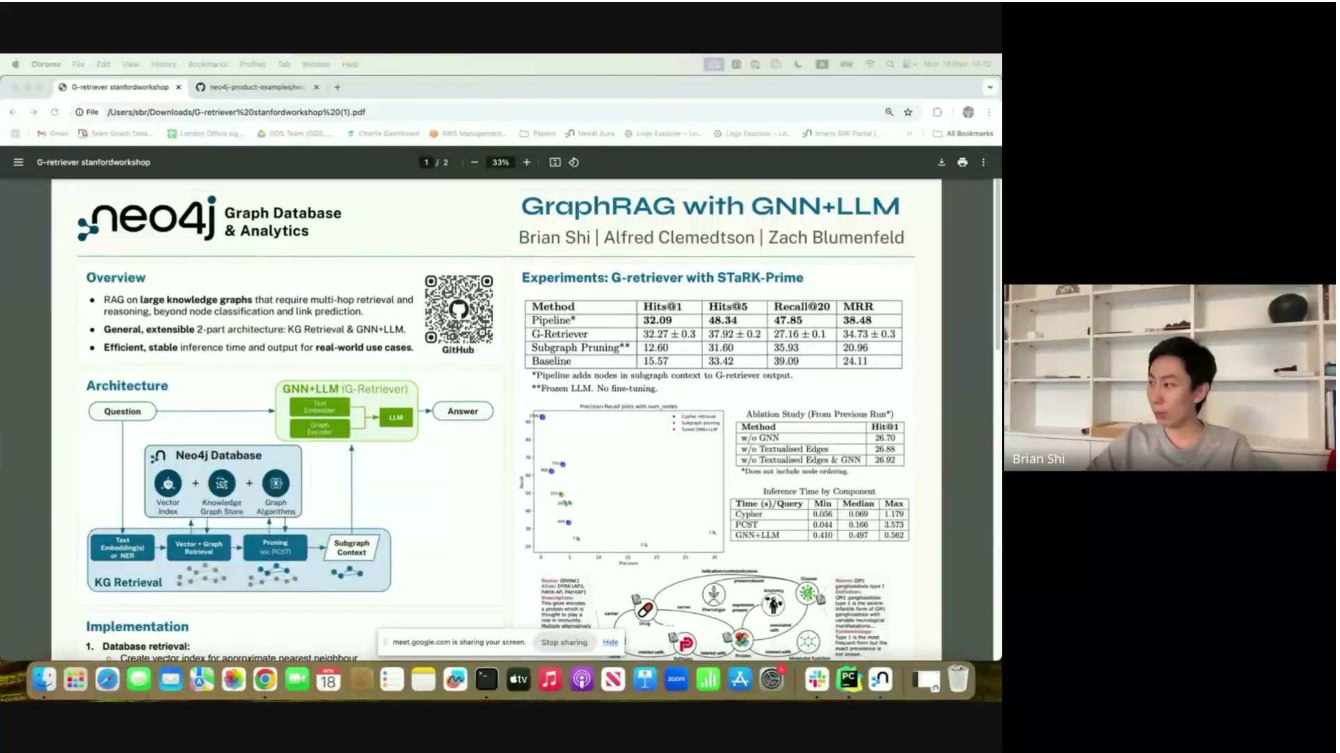
```
query_loader = RAGQueryLoader(
    data=(fs, gs), seed_nodes_kwargs={"k_nodes":
                                       5}, seed_edges_kwargs={"k_edges": 5},
    sampler_kwargs={"num_neighbors": [50] * 2},
    local_filter=make_pcst_filter(triples, model))

retrieved_subgraph = query_loader.query("Query here...")
```

- Set up data loader…
- Takes in Feature/Graph Store for KNN+NeighborSampling
  - “local_filter” applied to output
  - basic “local_filter” = PCST
- Once defined, trivial to query

Neo4j Case Study, Analysis

- LLM = 8B params (LLAMA 3.1) (w/ LoRa)
- GNN = ~10M params (GAT)
- Adding GNN ~0 cost -> 2x hit@1



Knowledge Graph Creation

- Most RAG Datasets only have unstructured text context
- Task: unstructured text -> KG
  - Format: (entity_1, relation, entity_2)
- LLMs specialized for unstructured text -> ideal model for task

TXT2KG Class in PyG

- PR: https://github.com/pyg-team/pytorch_geometric/pull/9728
- KG is source of info for KG RAG -> KG quality essential
- NVIDIA Inference Microservices (NIMS): llama-3_1-nemotron-70b-instruct,
  - NIMs chosen since most PyG users can’t run a 70B LLM
  - Model chosen since on par w/ gpt4o, open source, and smaller
  - https://build.nvidia.com/nvidia/llama-3_1-nemotron-70b-instruct
- Local LM for Dev: 14B param minimum (couldn’t get working with smaller LMs)
  - VAGOsolutions/SauerkrautLM-v2-14b-DPO (best 14B on leaderboard)

```
system_prompt = "Please convert the\
above text into a list of knowledge\
triples with the form\
('entity', 'relation', 'entity').\
Seperate each with a new line.\
Do not output anything else.”"
```

Simple Usage

```python
if local_lm:
    kg_maker = TXT2KG(
        local_LM=True,
        chunk_size=chunk_size,
    )
else:
    kg_maker = TXT2KG(
        NVIDIA_API_KEY=NVIDIA_API_KEY,
        chunk_size=chunk_size,
    )
if os.path.exists("path.pt"):
    kg_maker.load_kg("path.pt")
else:
    for data_point in rag_data_loader:
        q = data_point["question"]
        a = data_point["answer"]
        context_doc = data_point["context_document"]
        kg_maker.add_doc_2_KG(
            txt=context_doc,
            QA_pair=(q, a),
        )
    kg_maker.save_kg("path.pt")
```

Open Source RAG Datasets

❌ = (Bad for LLM vs GNN+LLM eval)

✅ = (Good for “”)

- WebQSP:
  - Toy Data:mostly single-hop ❌
  - Common Knowledge ❌
  - Comes with KG
- HotPotQA: https://huggingface.co/datasets/hotpotqa/hotpot_qa
  - Multihop ✅
  - Common Knowledge ❌
  - Needs TXT2KG (Good for testing)

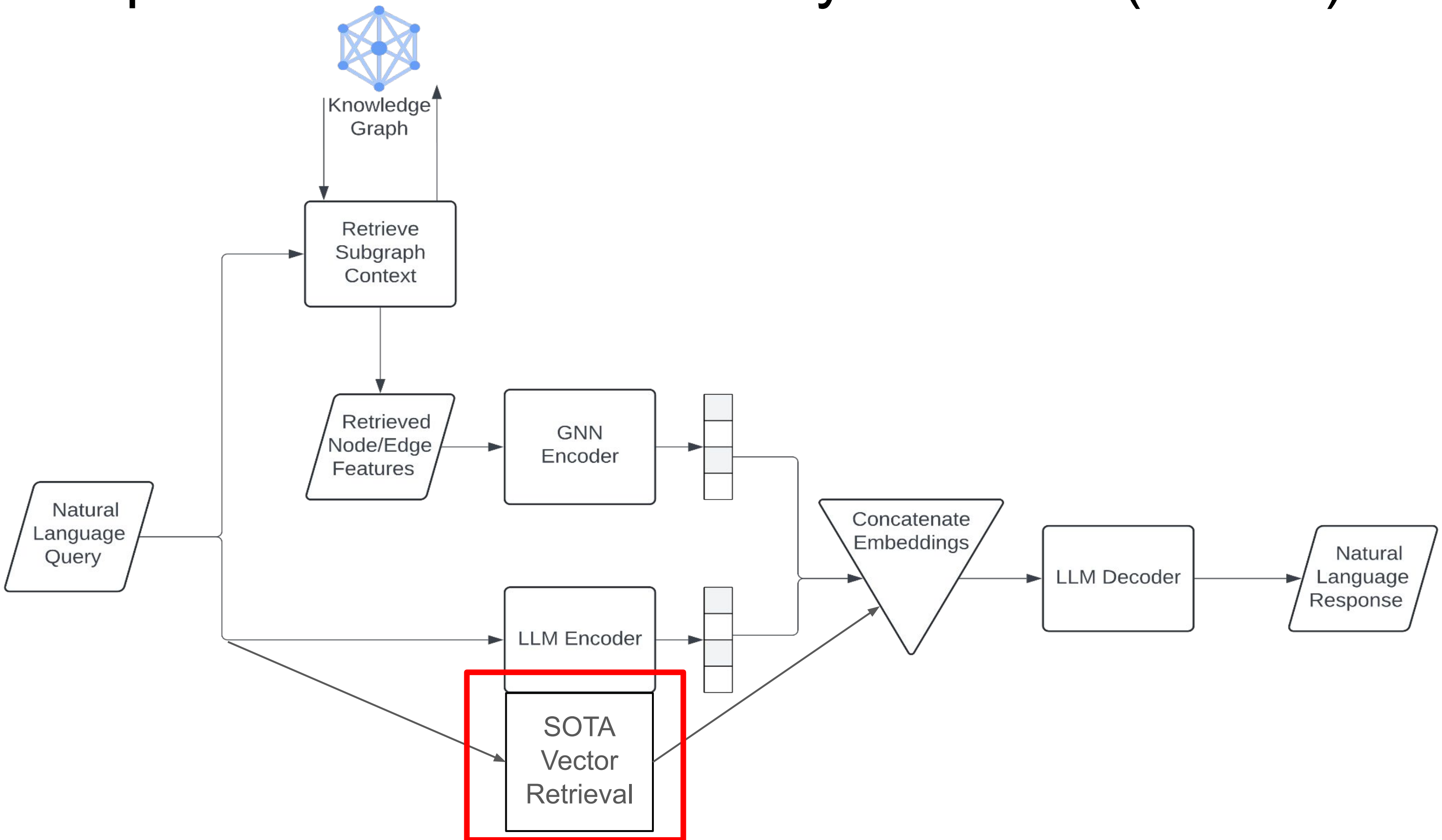
❌ = (Bad for LLM vs GNN+LLM eval)

✅ = (Good for “”)

Open Source RAG Datasets…

- TechQA: https://paperswithcode.com/dataset/techqa
  - IBM Tech Support Q&A dataset for RAG (Multihop ✅)
  - Not Common Knowledge ✅
  - Also needs TXT2KG

GraphRAG+Vector RAG = Hybrid RAG (Future)



More Modalities…

- Idea of GNN embeddings to prefix Transformer/LLM is highly general…

Scientific GNN+LLM Community Sprint (Biology/Chemistry)

- Goal: Add GNN+LLM support for the sciences like biology and chemistry
- Main Issue: https://github.com/pyg-team/pytorch_geometric/issues/9694
- 3 Biology papers & 1 Chemistry paper
- General goal: advance medicine and science
- Not too late to contribute! (2/4 tasks left)

More Modalities…

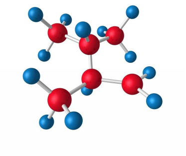
- Idea of GNN embeddings to prefix sequence prompt is highly general
        - Ex 1: Graphs = molecule/cell/etc, NLP task=Bio/Chem/Drug Discovery
        - Ex 2: Graphs = customer data, NLP task=talk to customer data
- Imagine graphs that include multiple modalities.
- Ex:
    - Amazon products, where each node has a text review and a photo
    - Relational Database heterographs as seen in RelBench*
    - Node/edge features could be:
        - Text: Natural Language or Code
        - Images
        - Audio
        - Video
        - Molecule/Cell/etc embeddings

* https://relbench.stanford.edu/

Unstructured vs Structured Graph Data

RELBENCH
RELATIONAL DEEP LEARNING BENCHMARK

or



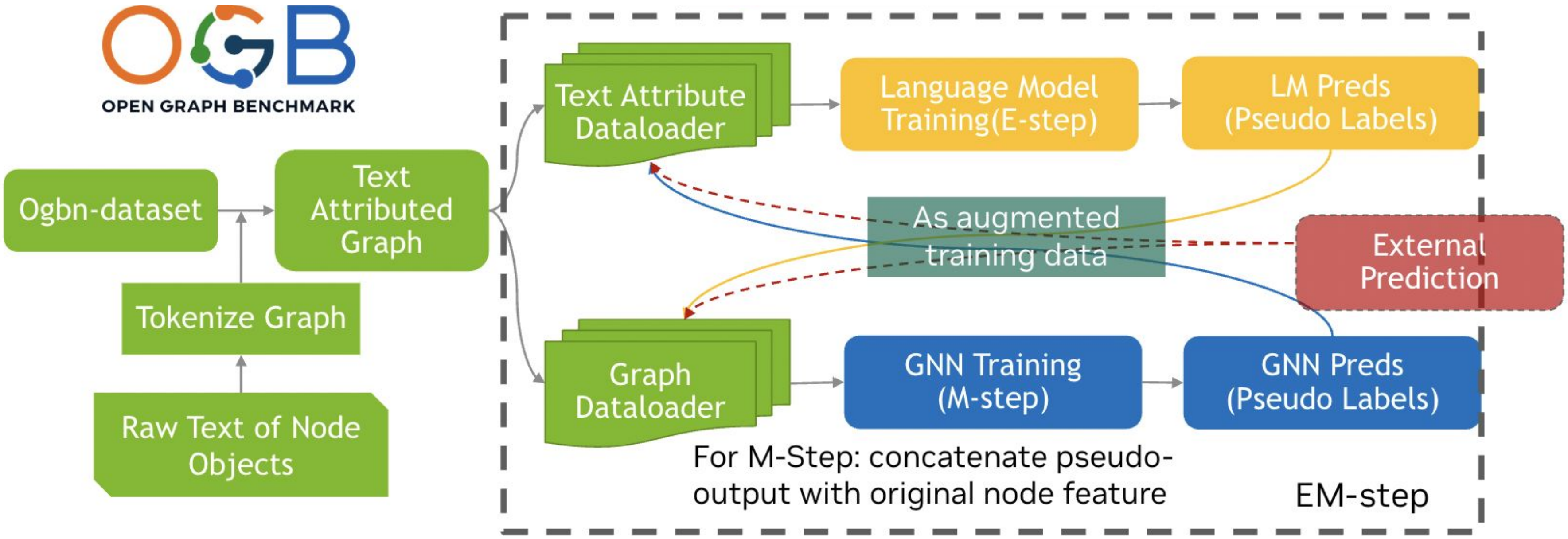
- Less Accurate

- They Likely cover different knowledge
- Most enterprises have both
- Combine? (Future)

- Highly Accurate

Node Classification (GLEM)

- GLEM = SOTA Node Classification for Text Attributed Graphs (TAGs)
  - https://arxiv.org/abs/2210.14709
  - Ex: OGBN-Products (https://ogb.stanford.edu/docs/leader_nodeprop/#ogbn-products)



Node Classification in PyG

- Implementation in PyG 7x faster than original paper's code for OGBN-Products
- New Text Attributed Graph (TAG) Interface
- Also adds optional support for LLM finetuning w/ LoRA* (uses PEFT* library)
- Already available in NVIDIA PyG Container or master branch in PyG GitHub

LoRA = https://arxiv.org/abs/2106.09685 PEFT = https://huggingface.co/docs/peft/en/index

TAG Usage

```
dataset = PygNodePropPredDataset(f'ogbn-products', root=root)
split_idx = dataset.get_idx_split()
tag_dataset = TAGDataset("/root/path", dataset, "prajjwal1/bert-tiny",
                         token_on_disk=token_on_disk)
```

- Takes in path, dataset, and LM of choice
- Optional: save tokens on disk

GLEM Set Up

```
num_classes = dataset.num_classes
gnn = GAT(in_channels=1024,
          hidden_channels=1024,
          num_layers=4,
          out_channels=num_classes,
          heads=4,
        )
model = GLEM(lm_to_use="prajjwal1/bert-tiny", gnn_to_use=gnn, out_channels=num_classes,
             lm_use_lora=lm_use_lora, device=device)
```

- GLEM model: LM of choice, GNN of choice, num_classes
- Optional:
  - Use LoRa
  - Device (default cpu but cpu is **SUPER** slow)

Conclusion

- GNNs and LLMs have complementary strengths
- GNN+LLM is new SOTA in many areas
- Easiest way to start: NVIDIA PyG container (Free!)
  - examples/llm (Container or GitHub)
  - https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pyg

Acknowledgement of Intern Work

Junhao Shen
Southern Methodist University
https://github.com/pyg-team/pytorch_geometric/pull/9662

Zachary Aristei
Georgia Tech
https://github.com/pyg-team/pytorch_geometric/pull/9666

Thank You!