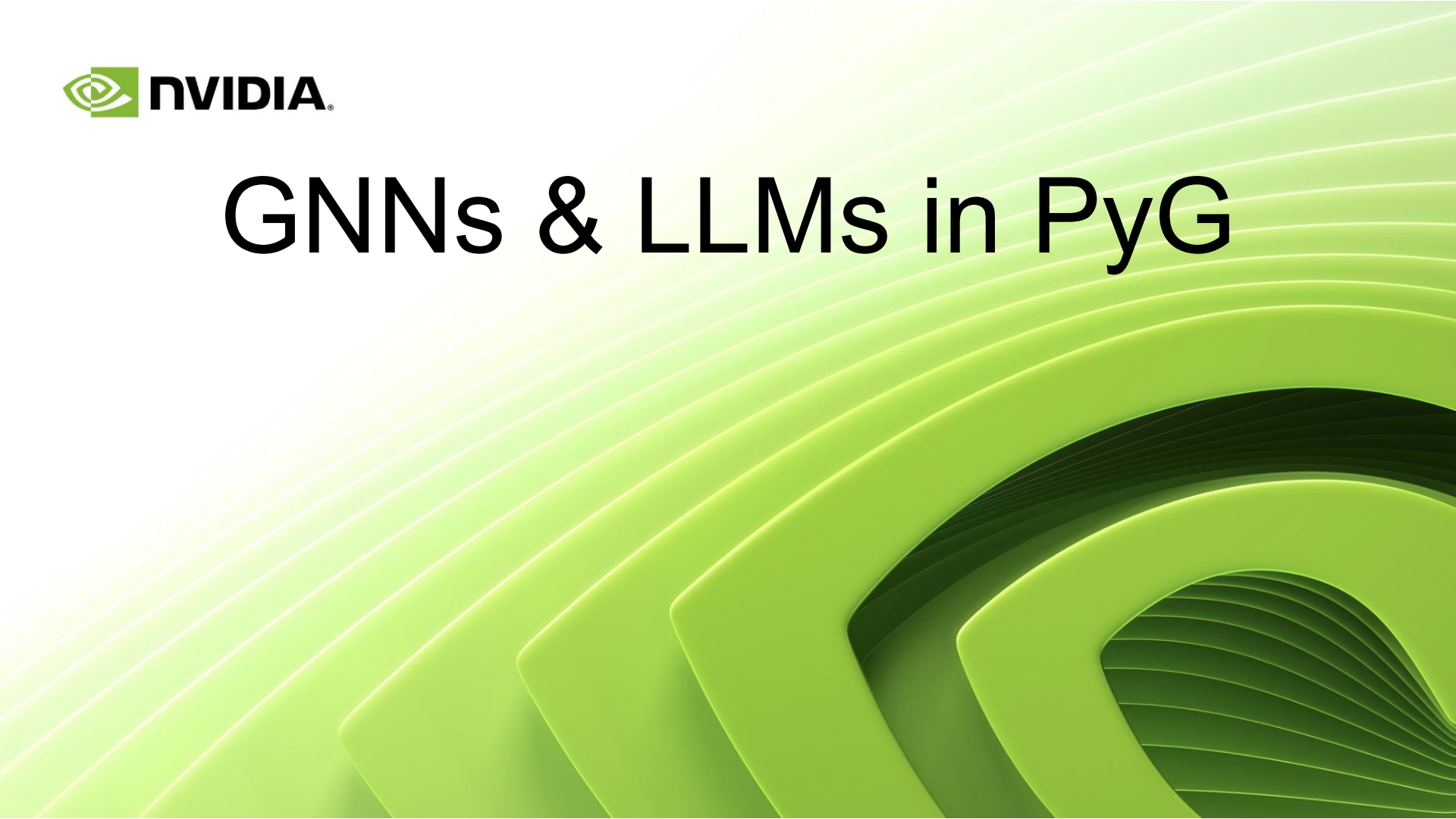
Stanford CS224W: Large Language Models and GNNs

CS224W: Machine Learning with Graphs
Jure Leskovec and Charilaos Kanatsoulis, Stanford University

http://cs224w.stanford.edu

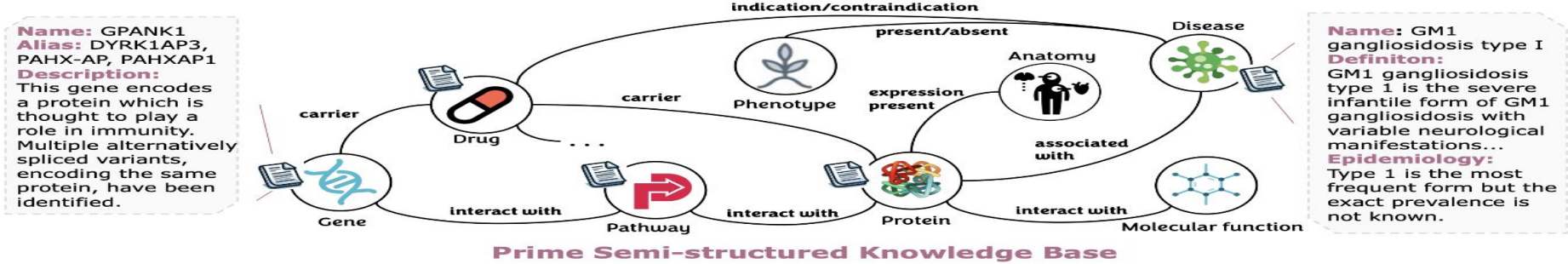
Announcements

- **Exam** is tomorrow! (11/19, 6-8PM)
  - See Ed for your assigned room
  - If taking an OAE or makeup exam, you should have received details by email already
- **Colab 4** due Tuesday 12/2
- **Colab 5** due Thursday 12/4
- **Project Report** due Thursday 12/11


NVIDIA
GNNs & LLMs in PyG

Neo4j Case Study

- Neo4j* Cyphers w/ PyG 2.6 G-retriever on Stark Prime (https://stark.stanford.edu/)
- 2x hit@1!
  - (.16->.32) (7B LLM Agentic GraphRAG vs PyG GNN+LLM)
  - PyG GNN+LLM Solution: LLAMA3.1-8B w/ LoRA + 10M param GAT
  - Even outperforms agentic GraphRAG w/ much larger frontier models:
    - claude-3-opus (.18) and gpt-4-turbo (.2) (Agentic GraphRAGs)
- https://github.com/neo4j-product-examples/neo4j-gnn-llm-example
- https://developer.nvidia.com/blog/boosting-qa-accuracy-with-graphrag-using-pyg-and-graph-databases/



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

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

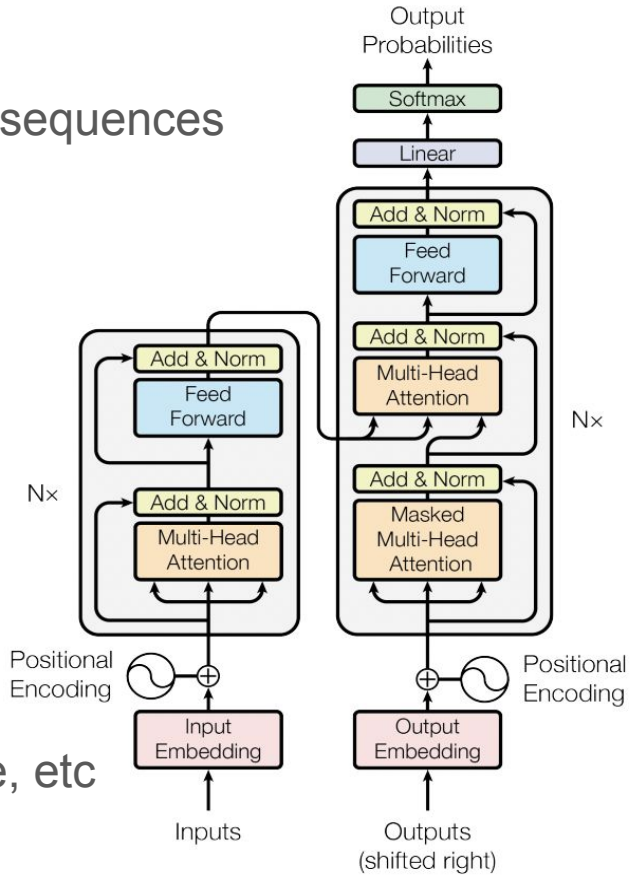


Figure 1: The Transformer - model architecture.

Today's LLMs = Associative Memory

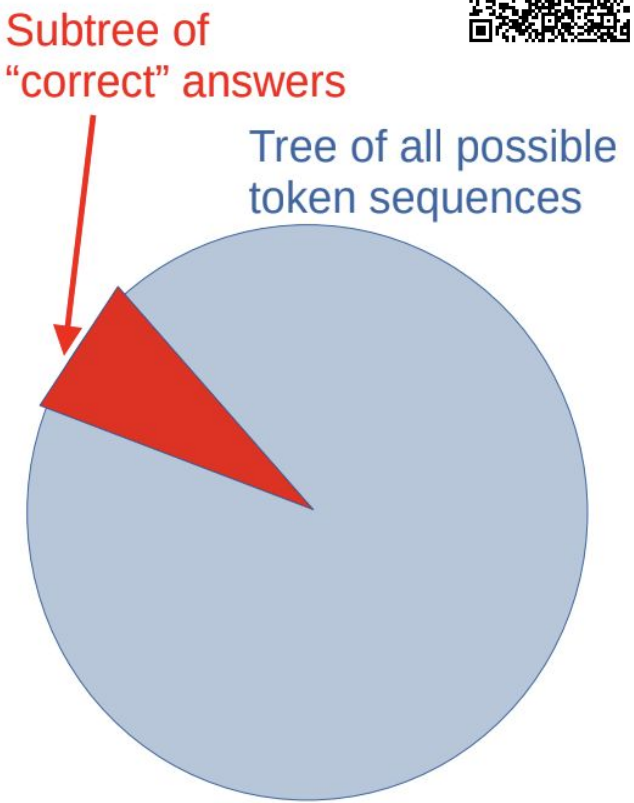
- Imagine: Goal is learn "2, 4, 6, 8, 10, 12, 14, …"
- General Human Intelligence (Won't Fail out of Distribution):
  - Given a number, return that number + 2
- Associative Memory Model (Will Fail out of Distribution)
  - Training: For 500GB worth of numbers, store the next token in the sequence
    - 2->4, 4->6, 6->8, …
  - Testing: passes within distribution, fails out of distribution
    - output undefined -> hallucinate
- LLMs close to associative memory than human intelligence

Auto-Regressive Generative Models Suck!

- Auto-Regressive LLMs are **doomed**.
- They cannot be made factual, non-toxic, etc.
- They are not controllable
- Probability e that any produced token takes us outside of the set of correct answers
- Probability that answer of length n is correct (assuming independence of errors):
  - $P(\text{correct}) = (1-e)^n$
- **This diverges exponentially.**
- **It's not fixable (without a major redesign).**

- See also [Dziri...Choi, ArXiv:2305.18654]

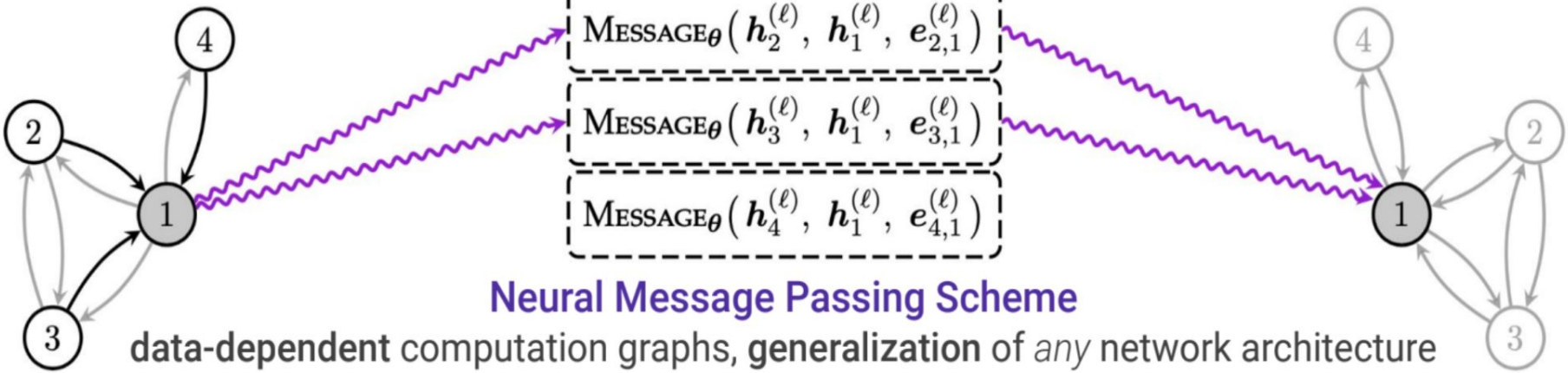
Full Lecture:



GNN Visualizer (GNN 101 University of Minnesota):

GNN Intro

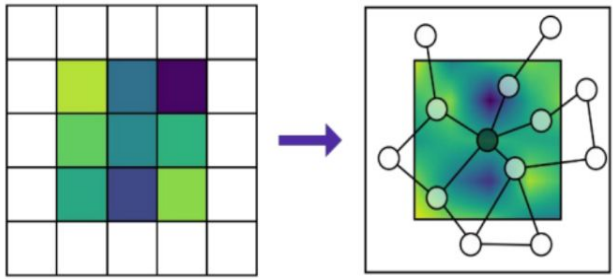
Graph Neural Networks (GNNs) update **node representations** by *repeatedly* **transforming** *and* **aggregating** representations *of* **direct neighbors**

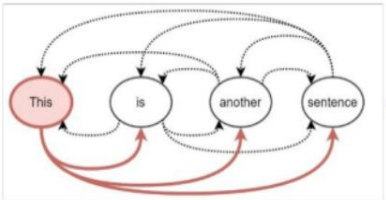


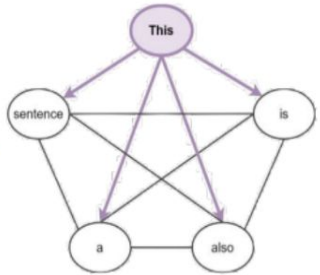
Neural Message Passing Scheme

data-dependent computation graphs, **generalization** of *any* network architecture

From **CNNs** *to* **GNNs**





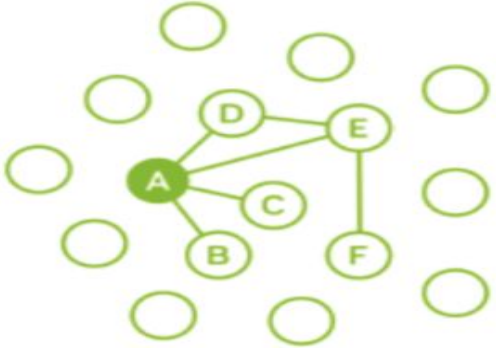


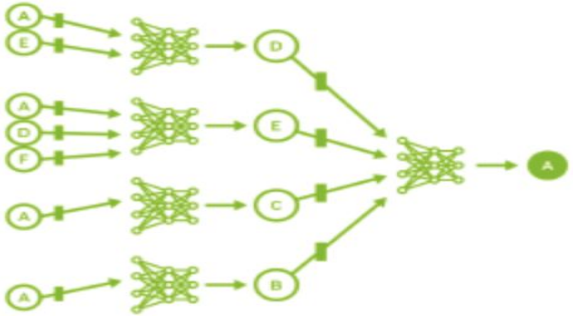
From **Transformers** *to* **GNNs**

Matthias Fey Graph Learning Conference Talk 2023

GNN Intro

- N layer GNN -> N hops of logic
- GNNs can improve LLM accuracy
  - (semi-orthogonal information)
- size(LLM) >> size(GNN)
  - “Transformer layer can be seen as a special GNN that runs on a fully connected “word” graph” - Jure Leskovec
  - -> almost no added params for adding GNN





GNNs are Deterministic, LLMs are Not!

- For any given input, output deterministic for most GNNs
- Not the case for LLMs… They are **Generative** AI
  - Learn the distribution, then randomly **generate** outputs that align with that distribution
  - 1 prompt, 2+ right answers
    - -> high chance of plausible sounding hallucination
  - Toy Example: Train LLM predict next token in 2, 4, 6, 8 and 2, 6, 18, 54
    - What does the LLM predict for 2->?

GNNs are Deterministic, LLMs are Not!

- Real World: Task + Python Solution
- Almost always multiple valid solutions
- Outcome: inference outputs that look valid but have issues
- Why: combining random chunks from correct codebases often doesn’t work
- Problem Caused:
  - Devs using AI spend substantial time debugging
  - Extremely damaging bugs can slip through -> crashes/being hacked/etc

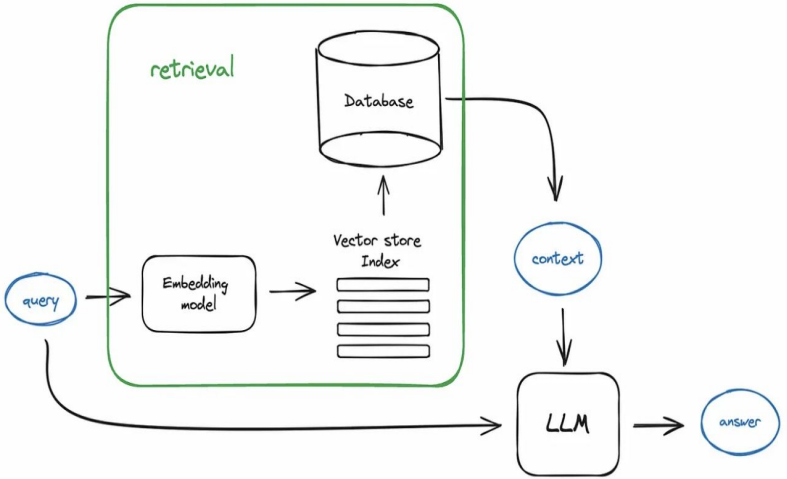
LLMs Strengths and Weaknesses

- Basic LLMs great for creative tasks that don’t need absolute precision:
  - Writing, Art, etc

- Not ideal for precise tasks like coding or making high impact decisions
- Running out of public data for LLM, GNNs enable a whole new world of data
- GNNs won’t solve all problems, but step in right direction.

RAG: VectorRAG vs GraphRAG

- RAG = Retrieval Augmented Generation
- VectorRAG: retrieve top K relevant docs based on their embedding **vector**
  - Good enough when answer requires single doc
- GraphRAG: retrieve relevant sub**graph**
  - Good when answer requires multiple docs with related entities

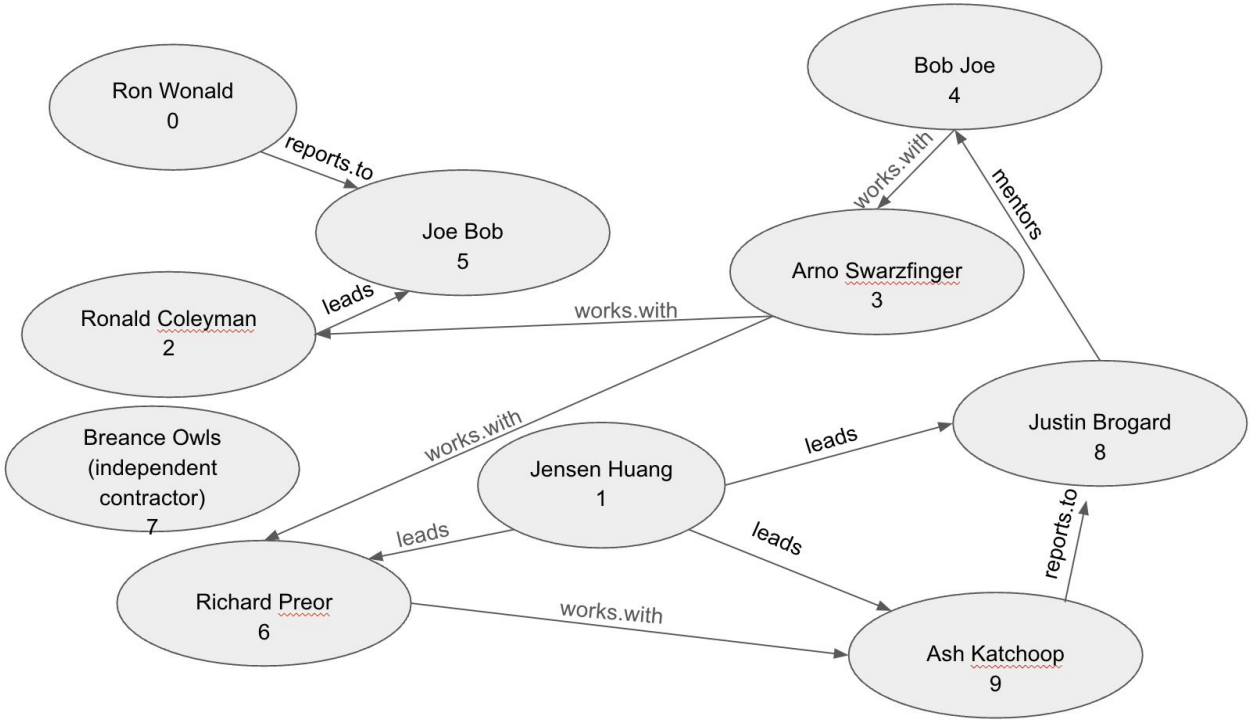


GraphRAG Example: Made Up NVIDIA Org Chart

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]]



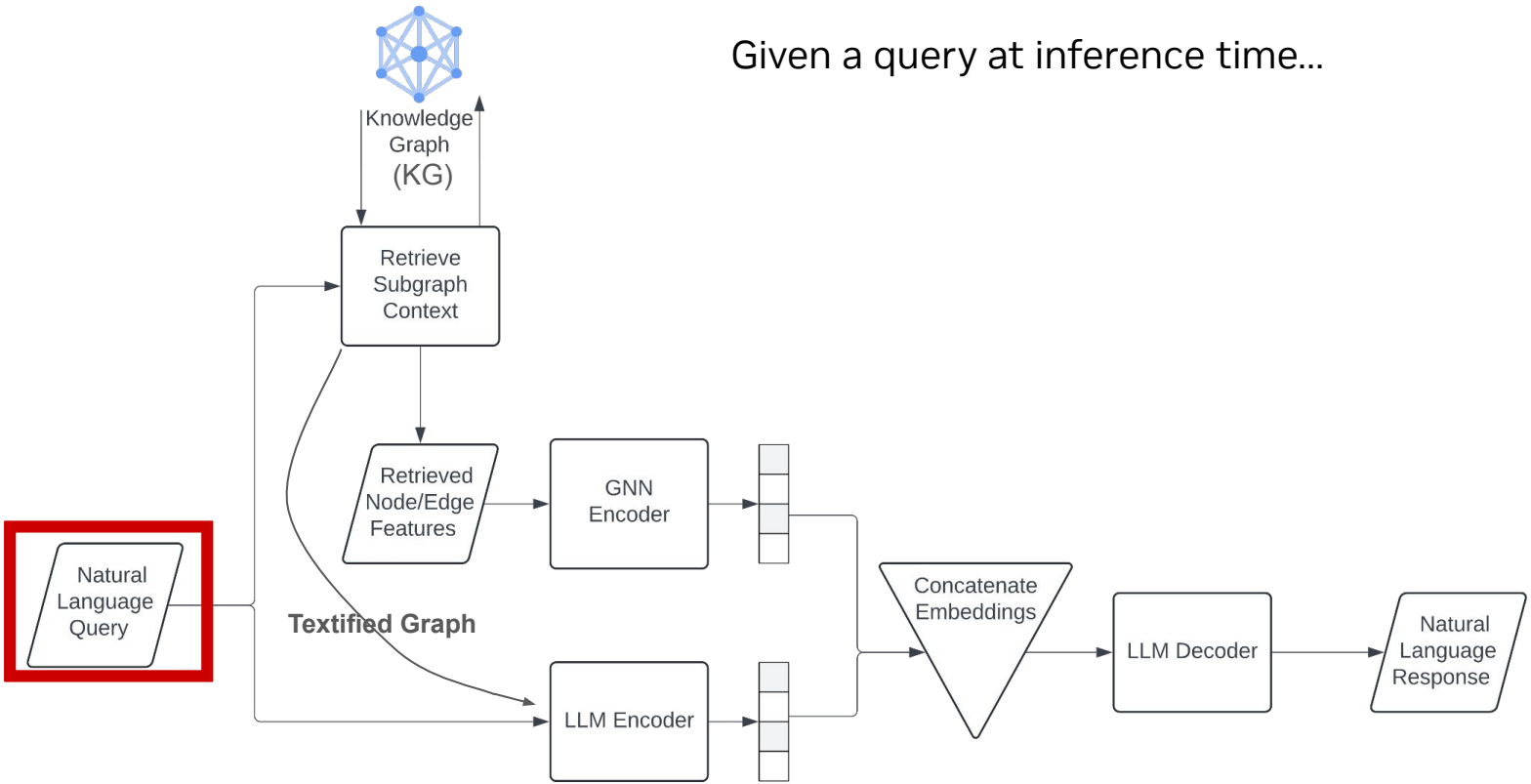
LLM: Wrong ❌
GNN+LLM: Correct ✅

GNNs Meet LLMs (History Lesson)

- “Attention is All You Need” (June 12, 2017 on Arxiv)
- GraphSAGE paper (June 7, 2017 on Arxiv)
- Since 2017, the industry has poured majority of effort into LLMs
- In parallel, small but strong PyG community grows in Academia
  - + Small Group of Companies Capitalizing on GNNs:
    - Pinterest, Kumo, Spotify …
- PyG GNN+LLM aims to combine the learnings of both work streams
  - Goal: Add GNNs to Advance AI systems past the limitations of LLMs

GNN+LLM Graph RAG (GNN Feeds LLM)

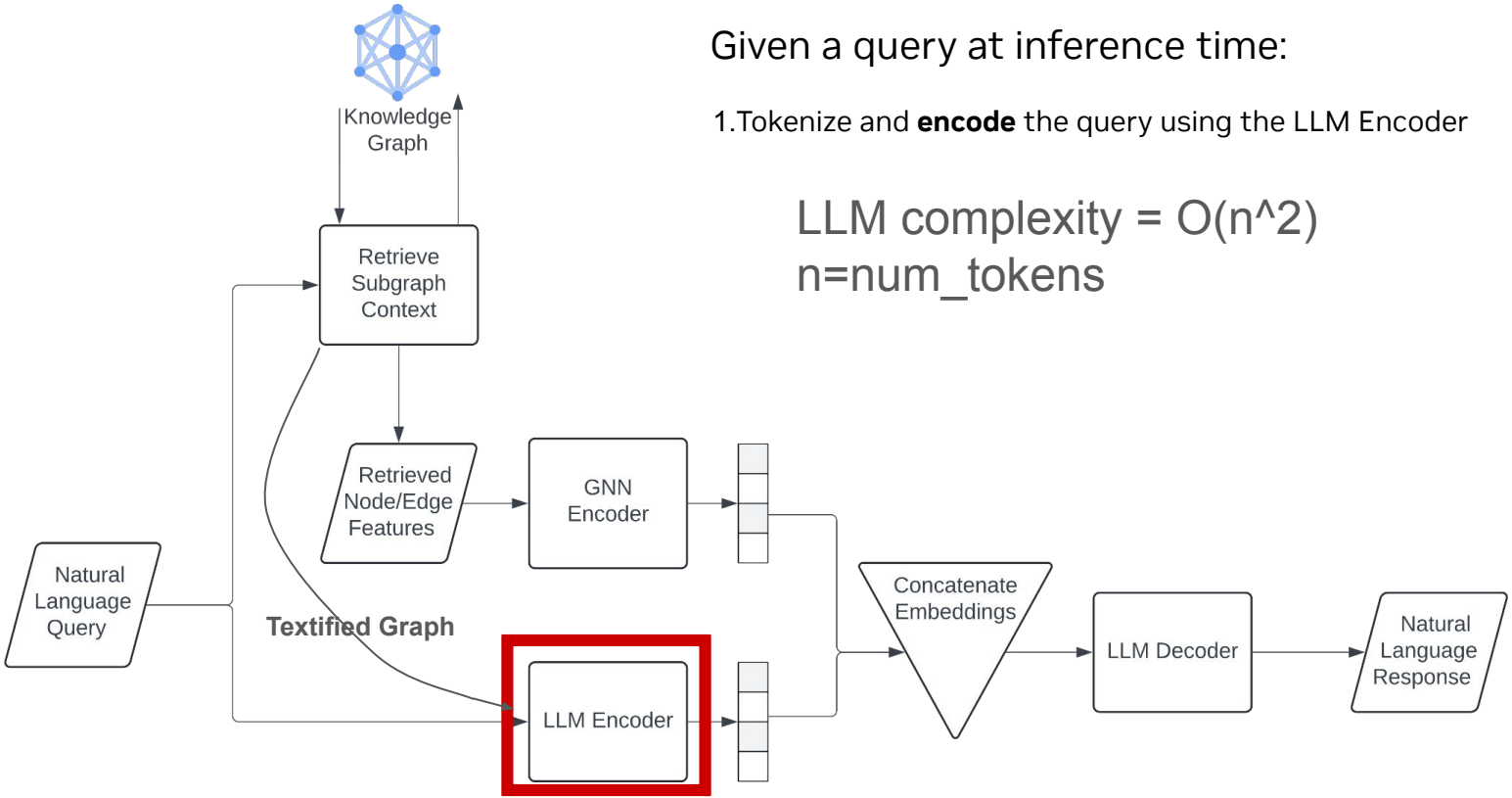
Given a query at inference time…



*RAG = Retrieval Augmented Generation

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

GNN+LLM Graph RAG (GNN Feeds LLM)



Given a query at inference time:

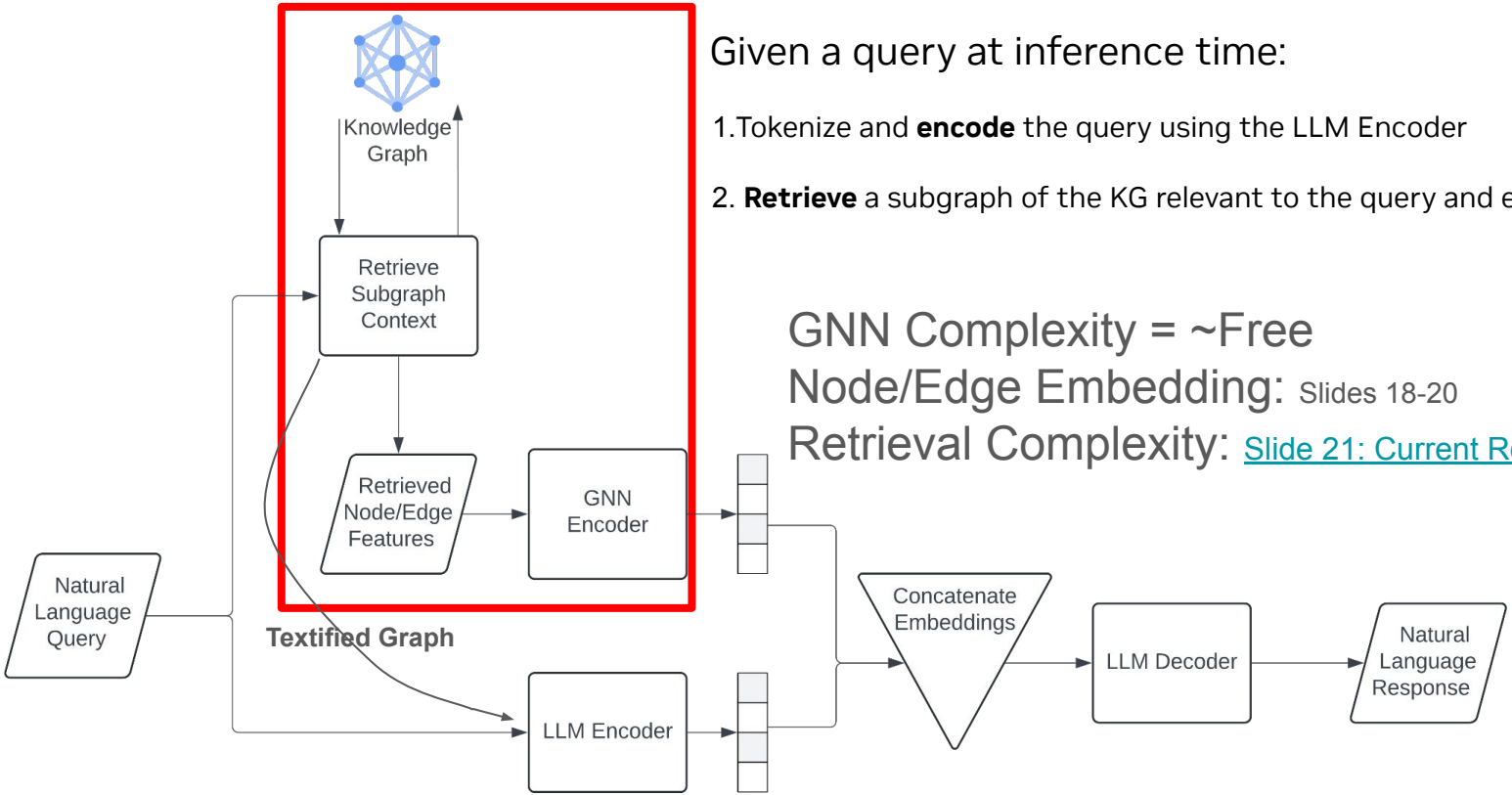
1.Tokenize and **encode** the query using the LLM Encoder

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

*RAG = Retrieval Augmented Generation

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

GNN+LLM Graph RAG (GNN Feeds LLM)



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

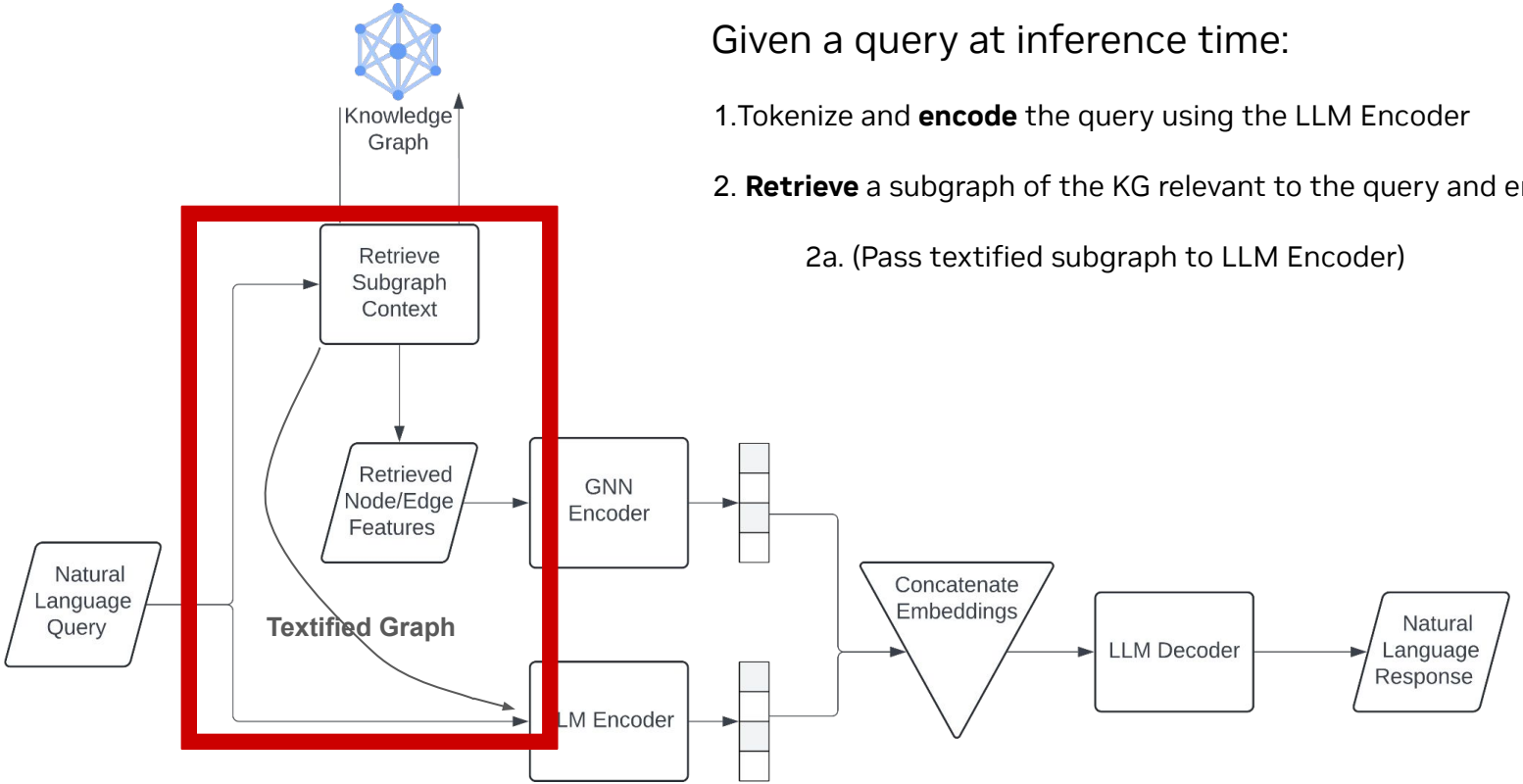
Node/Edge Embedding: Slides 18-20

Retrieval Complexity: Slide 21: Current Retrieval Algorithm

*RAG = Retrieval Augmented Generation

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

GNN+LLM Graph RAG (GNN Feeds LLM)



Given a query at inference time:

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

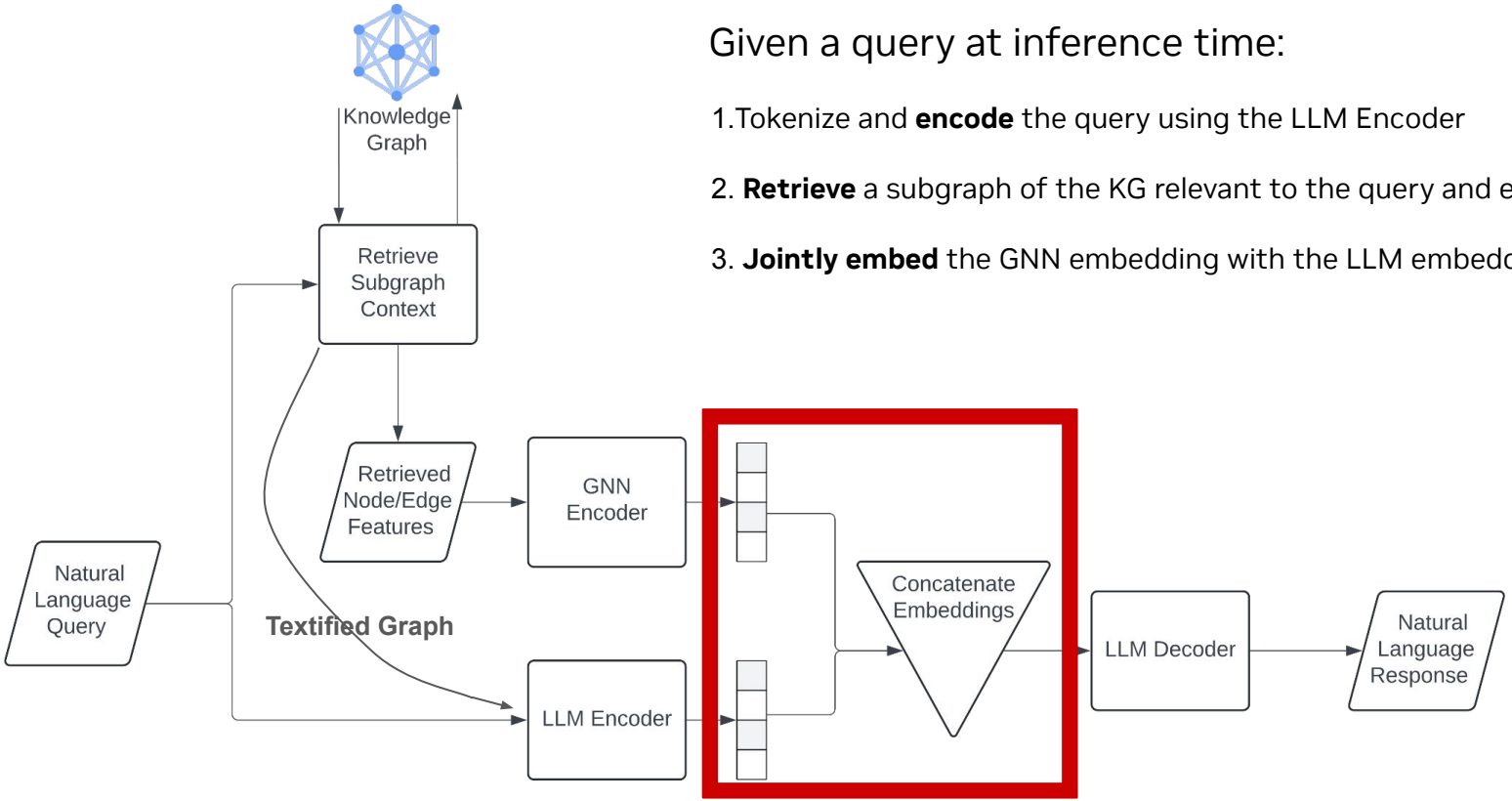
2. **Retrieve** a subgraph of the KG relevant to the query and encode it using a GNN

2a. (Pass textified subgraph to LLM Encoder)

*RAG = Retrieval Augmented Generation

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

GNN+LLM Graph RAG (GNN Feeds LLM)



Given a query at inference time:

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

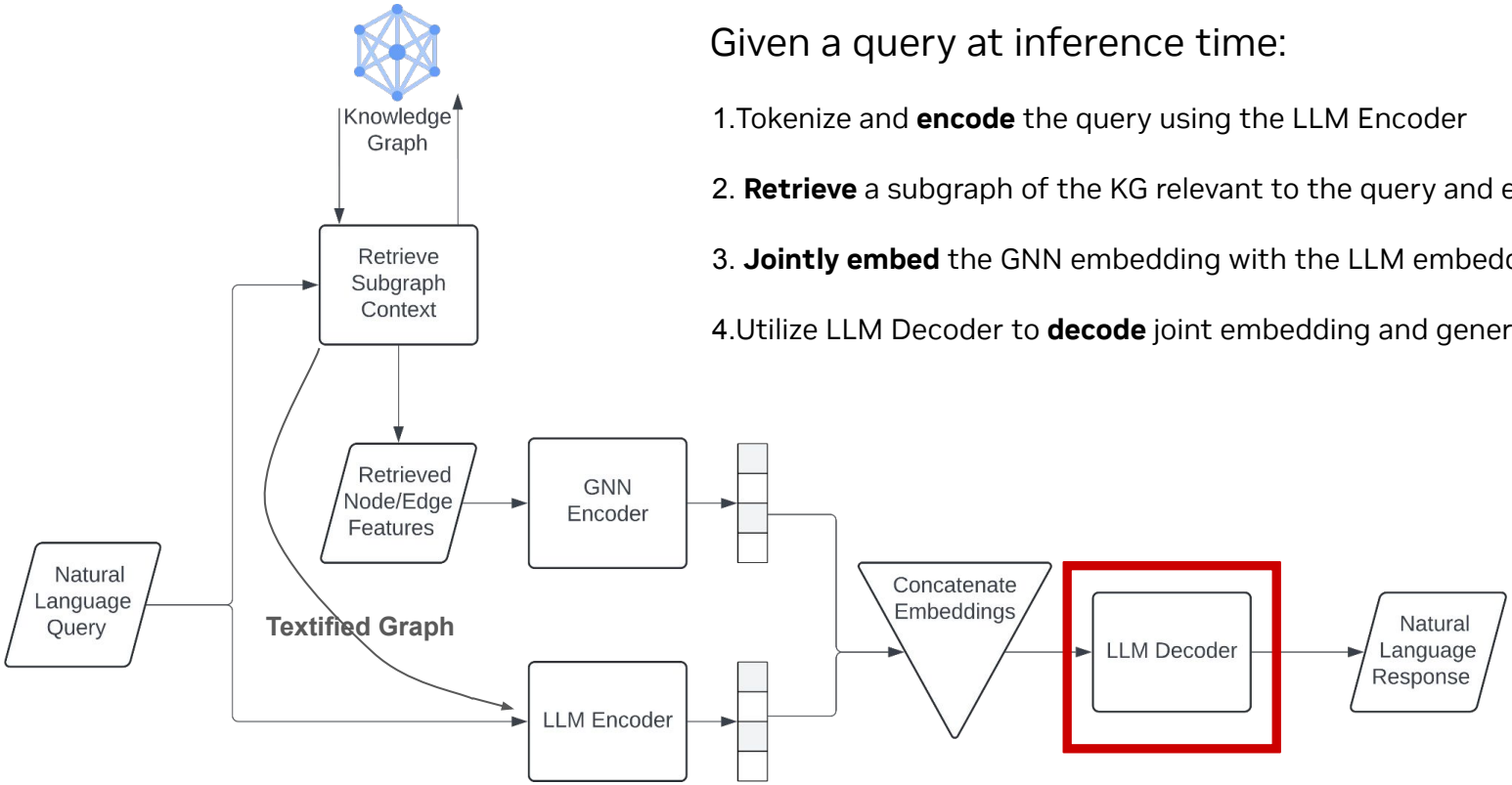
2. **Retrieve** a subgraph of the KG relevant to the query and encode it using a GNN

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

*RAG = Retrieval Augmented Generation

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

GNN+LLM Graph RAG (GNN Feeds LLM)



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

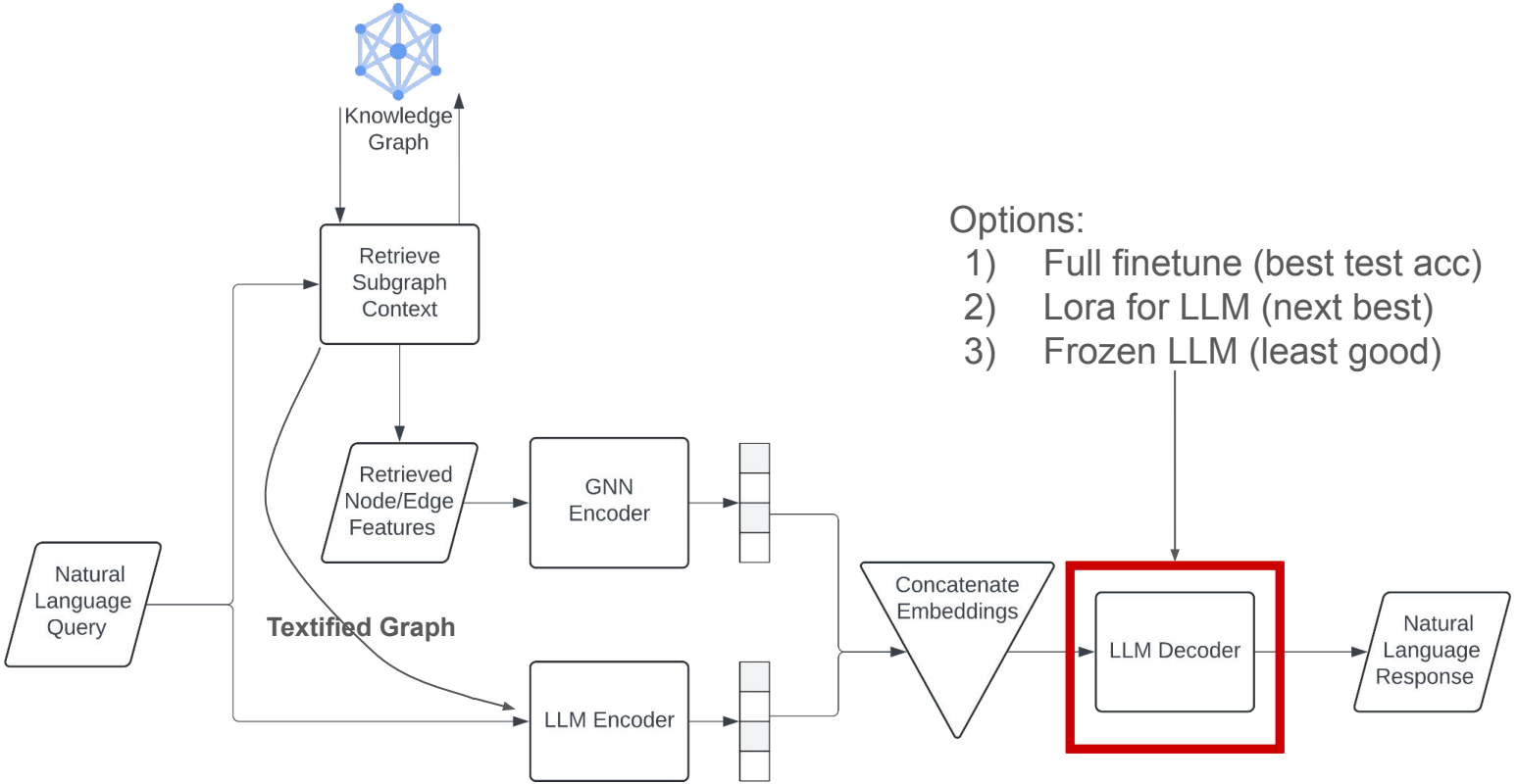
3. **Jointly embed** the GNN embedding with the LLM embedding

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

*RAG = Retrieval Augmented Generation

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

GNN+LLM Graph RAG (GNN Feeds LLM)



*RAG = Retrieval Augmented Generation

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

Text KG -> PyG Graph?

- PyG expects:
  - feature vector for each node
  - Optionally for each edge too
- Entities and relations are short phrases…
  - Example Triple: (cats, eat, dogs)
- Need Model(str) -> feature vector

Sentence Transformer Intro

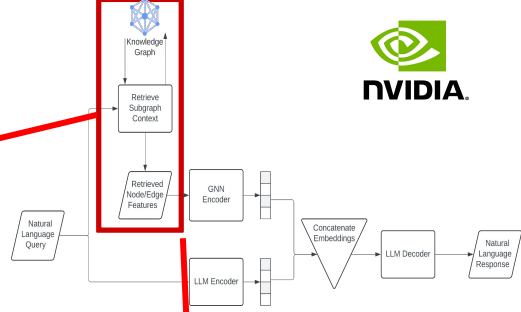
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = SentenceTransformer(
    model_name='sentence-transformers/all-roberta-large-v1').to(device)
```

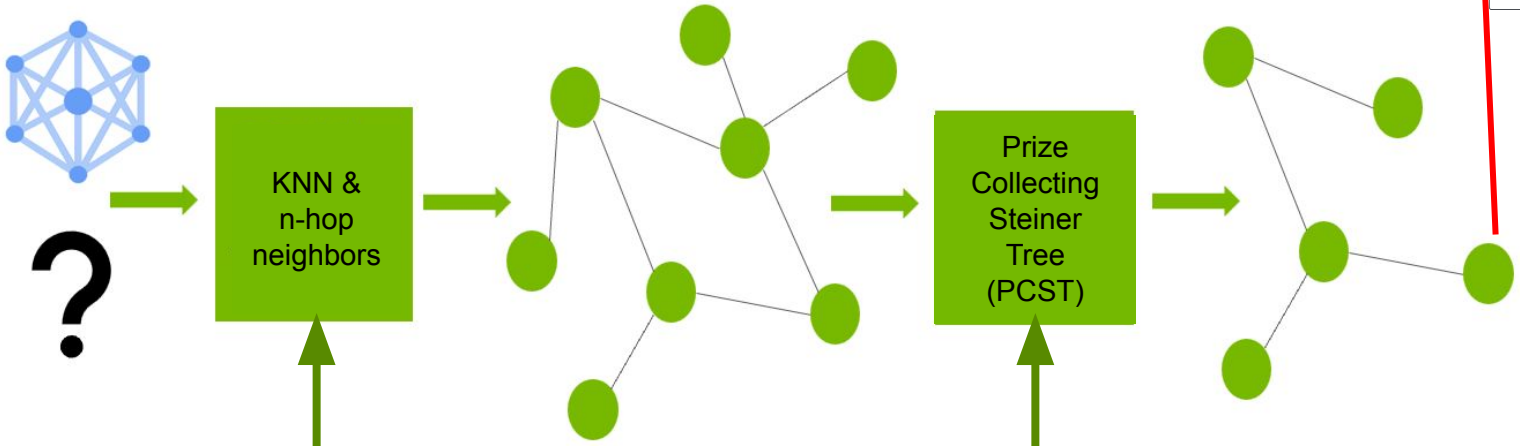
- Sentence Transformer: model(List[str]) -> Embedding Tensor
- Need to call model (3 * num_edges)
  - For each edge, call on both entities and the relation
  - -> Use small LM (SLM) like ModernBert for efficiency

Why Are Small LMs Okay as SentenceTransformers?

- 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
- Latest PyG supports VisionTransformer now!

Basic Retrieval Algorithm (GNN Feeds LLM)





Remote Knowledge Graph & Prompt (millions of nodes)

Untrimmed Subgraph (thousands of nodes)

Pruned Subgraph (dozens of nodes)

X nodes, n hops, fanout=f
-> O(X*K*f^n)

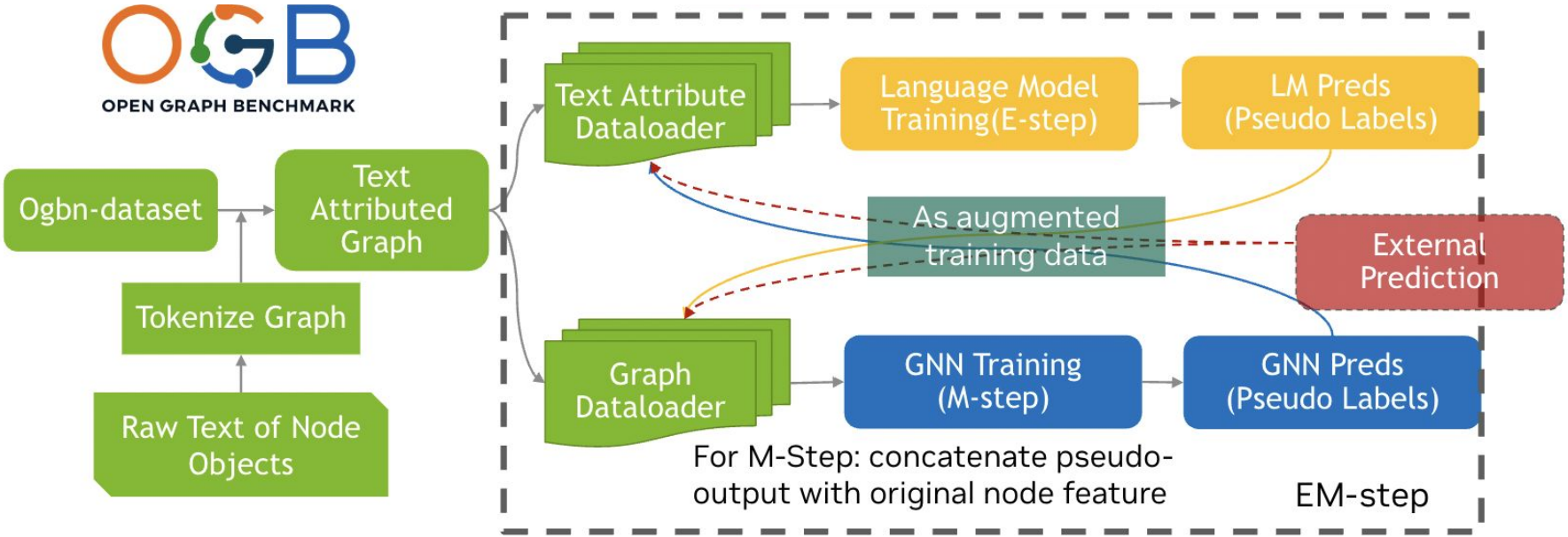
(similarity based graph traversal)
Linear on thousands -> Free

GNN Feeds LLM vs LLM Feeds GNN

- GNN Feeds LLM:
  - Data: prompt conditioned on graph
  - Tasks: Token-level tasks (Ex. Question Answering)
  - Flow:
    - Graph -> GNN = GNN_out
    - (GNN_out + prompt) -> LLM = Answer Tokens
- LLM Feeds GNN:
  - Data: Graph where every node/edge has tokens attributed to it
  - Task: Graph learning (Ex. node classification or link prediction)
  - Flow:
    - feature vector for each node/edge (Sentence/VisionTransformer)
    - Feed this graph to GNN for Graph Learning Task

Node Classification: GLEM (LLM Feeds GNN)

- 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 (LLM Feeds GNN)

- 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)
- See examples/llm/glem.py on PyG GitHub

PEFT = https://pyg.org/

LoRA = https://arxiv.org/abs/2106.09685

PEFT = https://huggingface.co/docs/peft/en/index

TAG Usage (LLM Feeds GNN)

```
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 (LLM Feeds GNN)

```
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)

G-retriever (GNN Feeds LLM) Set Up

```
gnn = GAT(
    in_channels=1024,
    hidden_channels=1024,
    out_channels=1024,
    num_layers=4,
    heads=4,
)

llm = LLM(model_name="meta-llama/Meta-Llama-3.1-8B-Instruct")
model = GRetriever(llm=llm, gnn=gnn)
```

- GRetriever: LLM + GNN
- Auto GPU Set Up!

G-retriever: Get Loss

```
model(
    batch.question, # ["list", "of", "questions", "here"]
    batch.x,  # [num_nodes, num_features]
    batch.edge_index,  # [2, num_edges]
    batch.batch,  # which nodes belong to which question
    batch.label,  # list answers (labels)
    batch.edge_attr,  # [num_edges, num_features]
    batch.desc  # list of text graph descriptions
)
```

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
```

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

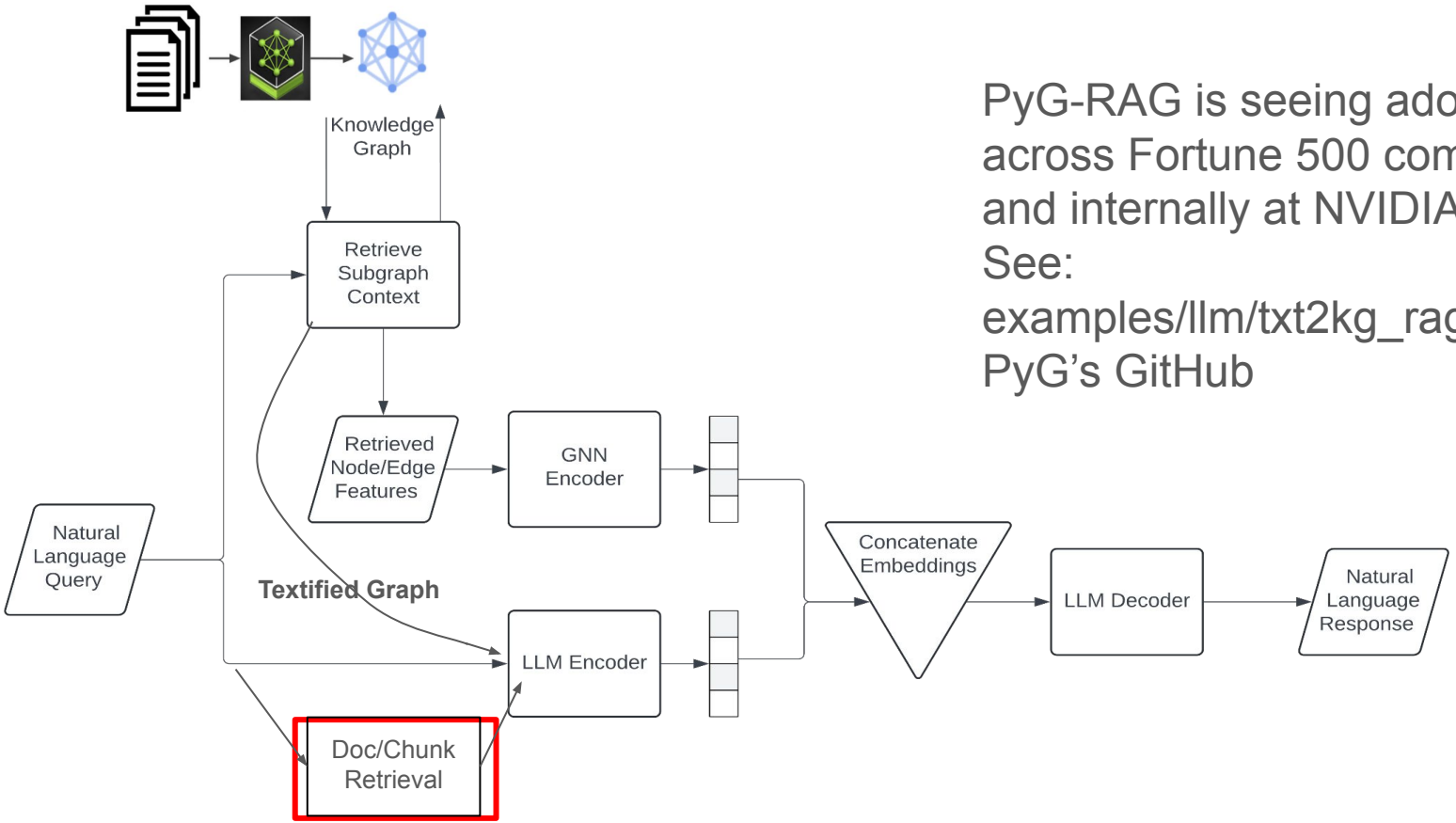
TXT2KG Class in PyG

- Released in PyG 2.7
- Prompt engineering + python filters

```
SYSTEM_PROMPT = "Please convert the above \
text into alist of knowledge triples with the form \
('entity', 'relation', 'entity'). \
Seperate each with a new line. \
Do not output anything else. \
Try to focus on key triples \
that form a connected graph. \
Try to focus on re-using the \
same entities whenever possible.""
```

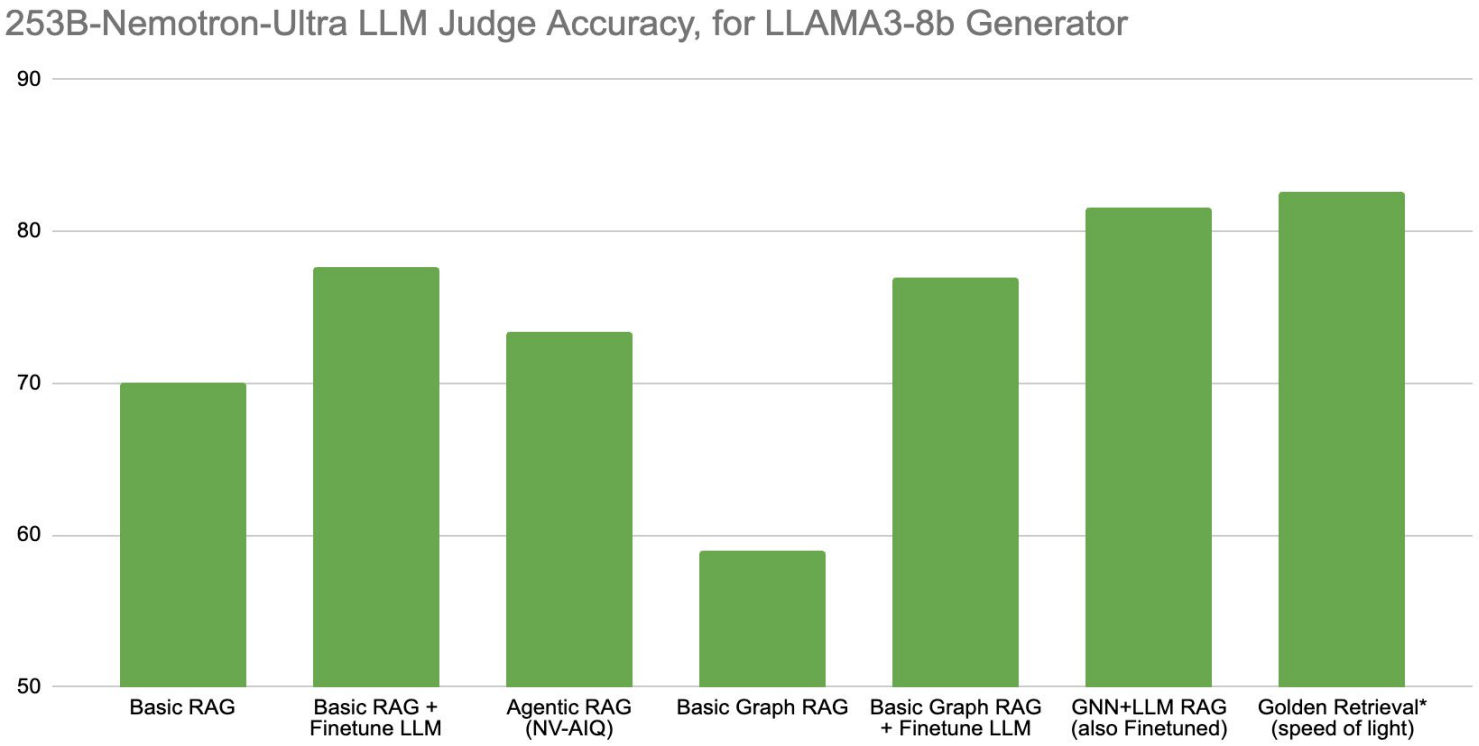
```
for context_doc in tqdm(context_docs, desc="Extracting KG triples"):
    kg_maker.add_doc_2_KG(txt=context_doc)
```

GraphRAG+VectorRAG (PyG-RAG)(torch_geometric.llm)



PyG-RAG is seeing adoption across Fortune 500 companies and internally at NVIDIA!!
See: examples/llm/txt2kg_rag.py on PyG's GitHub

Accuracy on Multi-Hop Synthetic Q & A for Lecture Corpus



*253b Nemotron for TXT2KG.

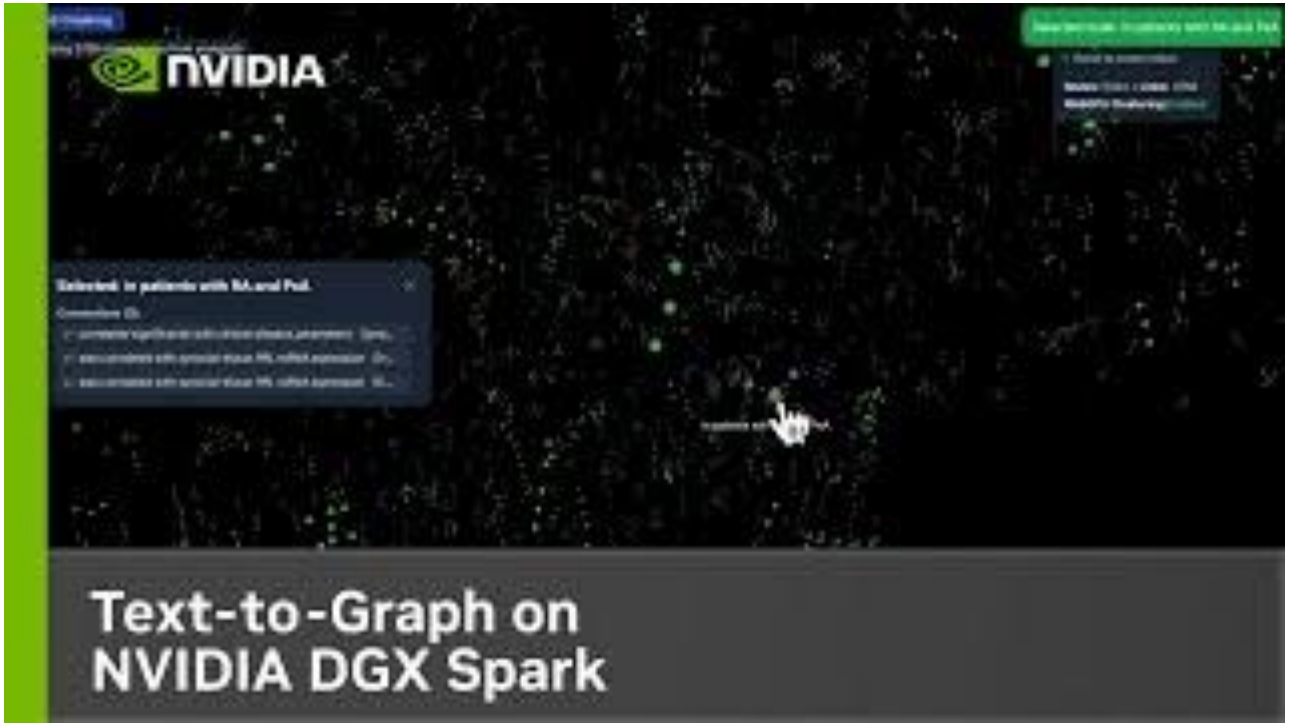
*ModernBert for vector embeddings.

*Feed generator same docs used to generate synthetic Q&A pairs.

Synthetic Q n A made with [NVIDIA TrueQuery]

Vibe Coded GUI

- PyG Codebase is easy to build with
  - Example: Our PM Santosh Bhavani Vibe Coded a GUI using Cursor
- https://build.nvidia.com/spark/txt2kg



Next Steps

- Continued improvements of codebase…
  - Speed, usability, accuracy, understandability, etc
- Accelerate retrieval w/ CuGraph+CuVs
  - Scale up to and beyond trillion edges
- Align TXT2KG w/ KGGen: https://github.com/stair-lab/kg-gen
- More modalities and interesting new problems to solve…
- New retrieval techniques

More Modalities…

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

Scientific GNN+LLM Community Sprint (Biology/Chemistry)

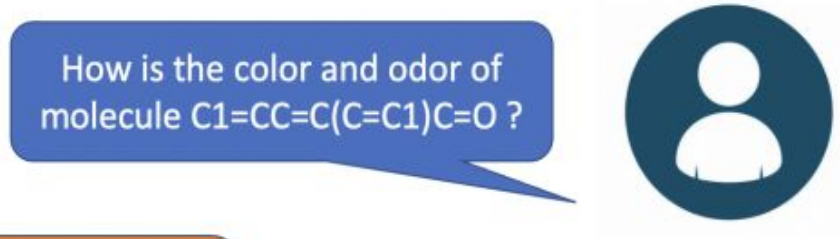
- Goal: Add GNN+LLM support for the sciences like biology and chemistry
- 3 Biology papers & 1 Chemistry paper
- General goal: advance medicine and science
- see examples/llm/README.md on PyG GitHub or NVIDIA Container

Example: MoleculeGPT

- “Talk to your Molecule” (GNN+LLM)

PyG Example link:

MoleculeGPT Paper link:



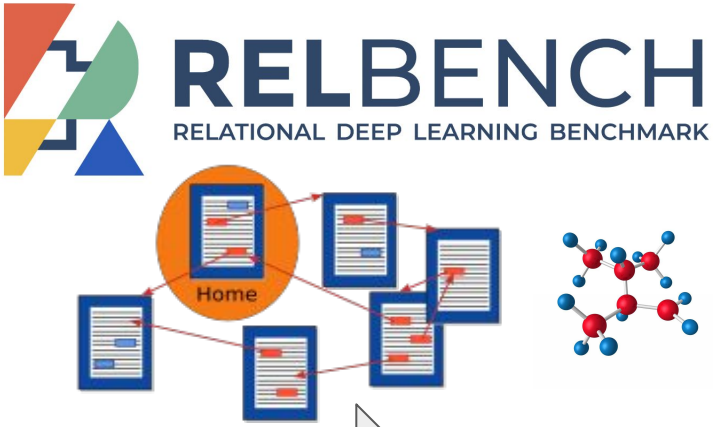
Qformer Paper:

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
  - Ex 3: Graphs = docs w hyperlinks, NLP task=talk to docs
- 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*/Kumo.ai*
  - Node features could be:
    - Text: Natural Language or Code
    - Images
    - Audio
    - Video
    - Molecule/Cell/etc embeddings

* https://relbench.stanford.edu/
* https://kumo.ai/

Unstructured vs Structured Graph Data



- Less Accurate
- They Likely cover different knowledge
- Most enterprises have multiple
- Combine:
  - Separate GNN per Graph Type
- Highly Accurate

Graph Transformers vs Message Passing (Future)

- Today 2 Kinds of GNN:
  - Message Passing (traditional)
  - Graph Transformer (newer)
- More Graph Transformers Coming Soon to PyG
  - Future Goal: See how effects accuracy of previously discussed GNN+LLM systems

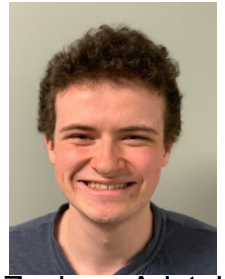
Webinar:

Conclusion

- General AI rule: Data is King!
- Ex: Even the best model can’t learn on randomly labeled data
- Key Takeaway:
  - Structured Data(GNN) + Unstructured Data(LLM) = Better GenAI
- Built & Optimized for NVIDIA PyG container
  - https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pyg
- Stay tuned to LinkedIn for new updates

Acknowledgements


Serge Panev


Zachary Aristei


Junhao Shen


Brian Shi


Vibhor Agrawal


Xiaoyun Wang


Ralph Liu


Rick Ratzel


Fay Wang


Alfred Clemedtson

Zach Blumenfeld


Joseph Nke


Erik Welch

And all the PyG contributors: https://github.com/pyg-team/pytorch_geometric/graphs/contributors

Thank You!