TALK LIKE A GRAPH: ENCODING GRAPHS FOR LARGE LANGUAGE MODELS

BY: BAHARE FATEMI, JONATHAN HALCROW, BRYAN PEROZZI

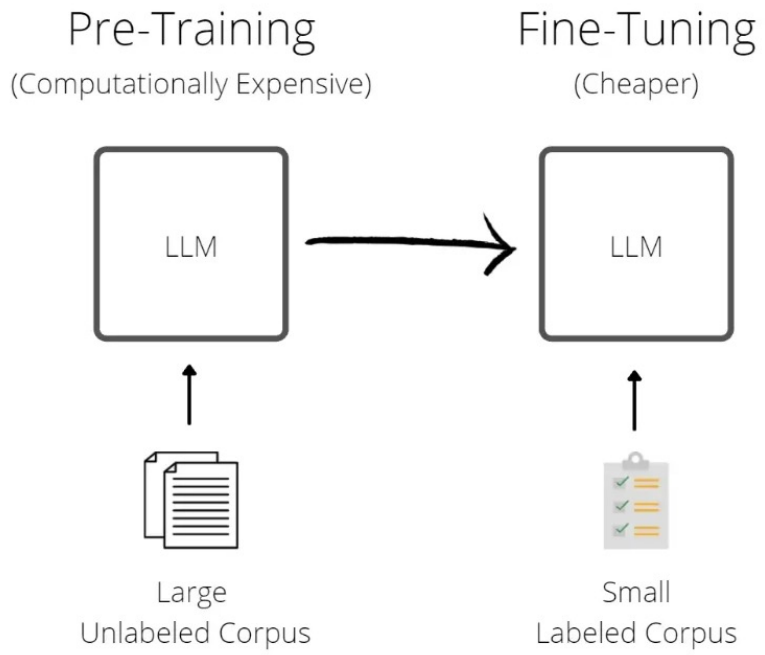
3/20/25

Presenter: Gurjot Singh

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS

Training of Large Language Models (LLMs)

- **Massive Data & Self-supervised Learning**:
  - LLMs are trained on huge text corpora using self-supervised objectives (e. g. , predicting next tokens).
  - They leverage the transformer architecture to capture long-range dependencies.
- **Pre-training & Fine-tuning**:
  - Pre-training establishes a broad understanding of language.
  - Fine-tuning or in-context learning adapts models to specific tasks.



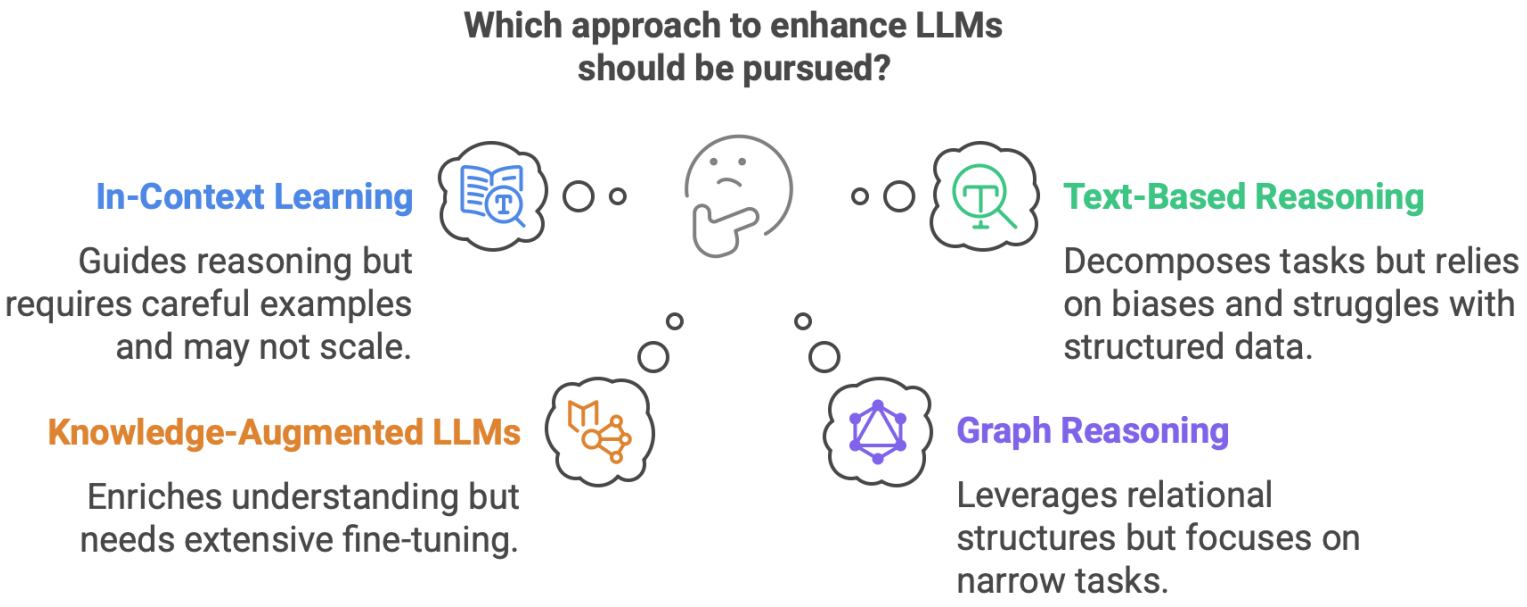
Inference in LLMs

- **Black-Box Inference**: Once trained, LLMs operate in a "black-box" manner: input text → output text.
- **Prompting Techniques**: Methods such as zero-shot, few-shot, and chain-of-thought prompting guide the model's reasoning.
- **Effectiveness in Natural Language Tasks**: LLMs excel at tasks like translation, summarization, and question-answering when the input is well-formulated text.

LLMs and Graph–Structured Data

- **Challenge with Graphs**:
  - Graphs represent structured relationships (nodes and edges) that are not inherently sequential.
  - LLMs are primarily trained on unstructured, natural language text.
- **Why Direct Graph Reasoning Fails**:
  - Graph–specific tasks (e. g. , edge existence, node degree, cycle detection) require an understanding of topology that LLMs lack.
  - Examples:
    - **Edge Existence**: LLMs might wrongly infer the presence or absence of an edge.
    - **Cycle Detection**: Graphs with specific topologies (e. g. , path graphs) may confuse the model.

Previous Work



Outline

- Introduction & Background
- Contributions
- Methodology
- Results & Analysis
- Limitations
- Conclusion

Key Contributions

- **Comprehensive Graph Encoding Study**:
  Explored multiple methods for representing nodes and edges as text, revealing significant impacts on LLM reasoning accuracy.
- **Effective Prompting Strategies**:
  Analyzed various prompting techniques (zero-shot, few-shot, chain-of-thought) to develop best practices for querying LLMs with graph data.
- **GraphQA Benchmark Introduction**:
  Established a new benchmark with diverse graph tasks to systematically evaluate and compare LLM performance on structured graph problems.

Methodology

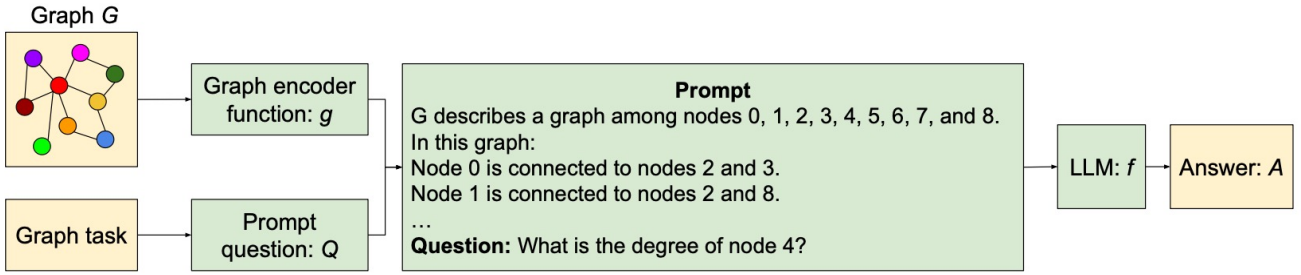
The objective is to enhance the reasoning capabilities of Large Language Models (LLMs) over graph-structured data by encoding the graph into text. The key components of the methodology are:

- **Graph Encoding Function (g):** Converts a graph $G = (V, E)$ into a text sequence.
- **Question Rephrasing Function (q):** Reformulates the task questions into prompts that are easier for LLMs to interpret.

The final prompt is constructed by concatenating the encoded graph and the rephrased question:

$$A = f(g(G), q(Q))$$

This black-box approach does not require any modification of the internal parameters of the LLM.



Training Loss Function

The methodology includes optimizing a loss function over a training dataset. Specifically, given a dataset D of triples (G, Q, S), where:

- G is a graph,
- Q is a question, and
- S is the correct solution,

the aim is to find the functions g and q that maximize the expected score of the LLM:

$$\max_{g,q} E_{(G,Q,S)\sim D}\left[\text{score}_f\left(g(G), q(Q), S\right)\right]$$

This loss optimization is crucial as it guides the selection of encoding and prompt formulations that most effectively enable the LLM to reason about graph-structured data.

Prompting Heuristics

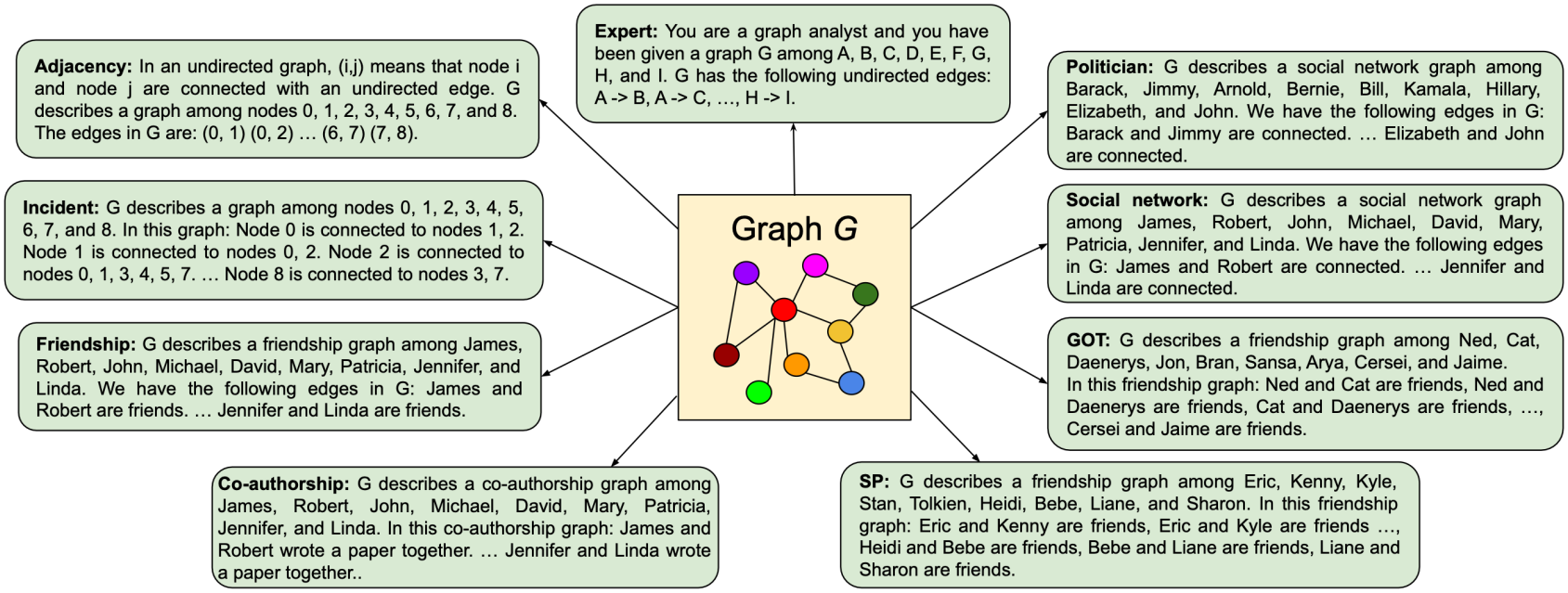
The authors discuss several prompting strategies designed to optimize the text prompt Q provided to a fixed Large Language Model (LLM). These heuristics are intended to better expose the graph-structured information encoded by the function $g(G)$ and improve performance on graph reasoning tasks. The primary methods are:

- **Zero-shot Prompting (ZERO-SHOT):**
  - The model is given only a task description and is asked to produce the desired output without any training examples.
- **Few-shot In-Context Learning (FEW-SHOT):**
  - The model is provided with a small set of input-output examples that demonstrate the task. These examples help the model adapt its output to new, similar inputs.
- **Chain-of-Thought (CoT) Prompting:**
  - A series of examples is provided, each detailing a step-by-step reasoning process. This encourages the model to generate a chain of thought when addressing new problems.

Prompting Heuristics

- **Zero-shot CoT Prompting (ZERO-COT):**
  - Similar to CoT prompting, but without providing explicit examples. Instead, the prompt includes a simple cue (e.g., "Let's think step by step") that nudges the model to generate its own reasoning chain.
- **Bag Prompting (COT-BAG):**
  - This technique appends an instruction such as "Let's construct a graph with the nodes and edges first" to the graph description. The aim is to guide the model in organizing and processing the graph information before answering the main query.

Graph Encoding Function



Graph Encoding Function

The graph encoding function is responsible for translating the structured information of a graph into a textual format. This process is broken down into two main parts:

a. **Node Encoding:** Options include using:

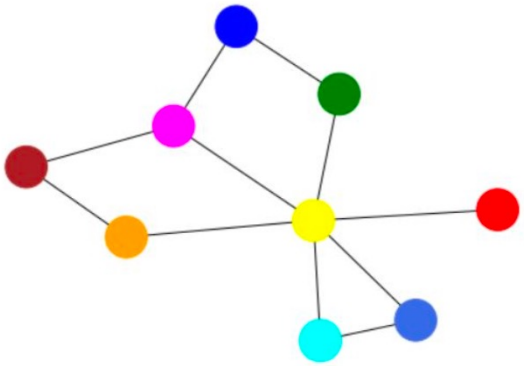
- Integers (e.g., "Node 0")
- Common names
- Fictional character names
- Alphabet letters

b. **Edge Encoding:** Different techniques are explored, such as:

- Adjacency notation (e.g., "(0,1)")
- Descriptive relations (e.g., "is friends with", "collaborates with")

Different encoding methods capture various aspects of the graph, and empirical results indicate that these choices have a significant impact on the performance of LLMs on basic graph tasks.

Graph Encoding Function

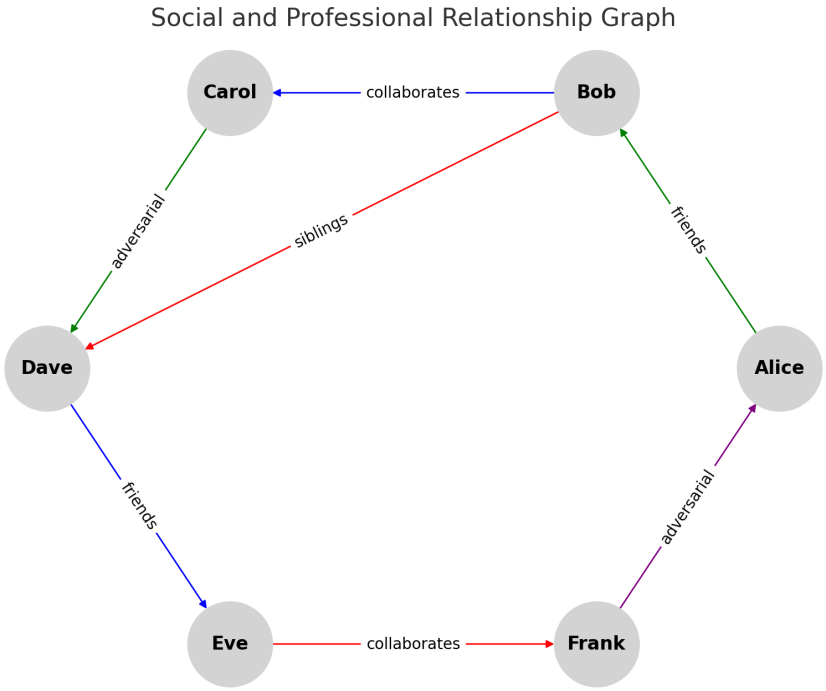


Incident: G describes a graph among 0, 1, 2, 3, 4, 5, 6, 7, and 8.
In this graph:
Node 0 is connected to nodes 1, 2.
Node 1 is connected to nodes 0, 2.
Node 2 is connected to nodes 0, 1, 3, 4, 5, 7.
Node 3 is connected to nodes 2, 8.
Node 4 is connected to node 2.
Node 5 is connected to nodes 2, 6.
Node 6 is connected to nodes 7, 5.
Node 7 is connected to nodes 2, 8, 6.
Node 8 is connected to nodes 3, 7.

Co-authorship: G describes a co-authorship graph among James, Robert, John, Michael, David, Mary, Patricia, Jennifer, and Linda.
In this co-authorship graph:
James and Robert wrote a paper together.
James and John wrote a paper together.
Robert and John wrote a paper together.
John and Michael wrote a paper together.
John and David wrote a paper together.
John and Mary wrote a paper together.
John and Jennifer wrote a paper together.
Michael and Linda wrote a paper together.
Mary and Patricia wrote a paper together.
Patricia and Jennifer wrote a paper together.
Jennifer and Linda wrote a paper together.

Example

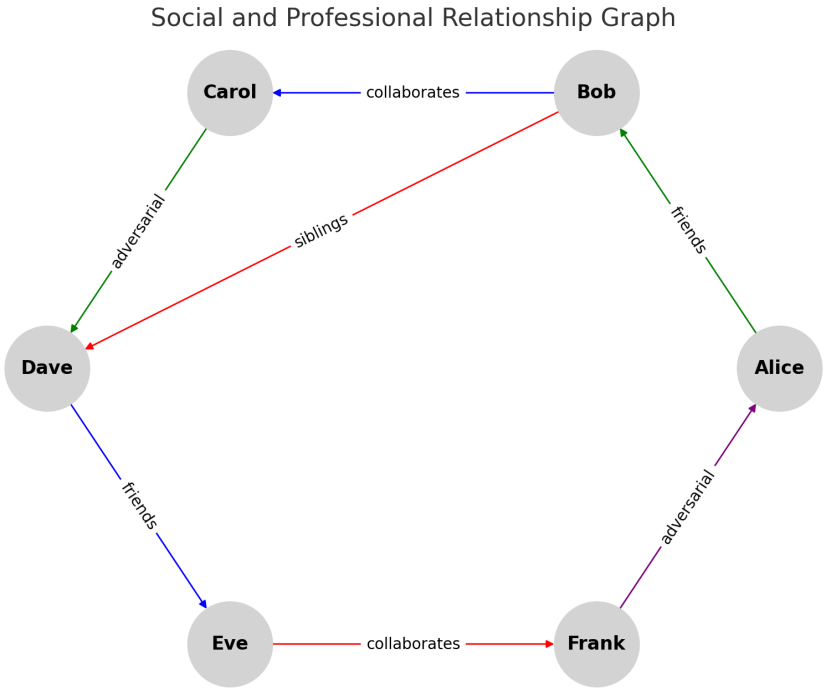
- "Given the following graph: Alice is friends with Bob. Bob collaborates professionally with Carol. Carol is adversarial towards Dave. Dave is friends with Eve. Eve collaborates professionally with Frank. Frank is adversarial towards Alice. Additionally, Bob and Dave are siblings. Determine whether there exists a cycle that starts and ends at Alice, includes at least one friendship edge, one professional collaboration edge, and one adversarial edge, and does not include any sibling relationships. Explain your reasoning and list the nodes in the cycle if it exists. "



Example

Since the cycle

Alice → Bob → Carol → Dave → Eve → Frank → AliceAlice → Bob → Carol → Dave → Eve → Frank → Alice satisfies all constraints, such a cycle **exists**.



Results

`GraphQA` consists of a diverse set of basic graph problems, including:

- **Edge existence.** Determine whether a given edge exists in a graph.
- **Node degree.** Calculate the degree of a given node in a graph.
- **Node count.** Count the number of nodes in a graph.
- **Edge count.** Count the number of edges in a graph.
- **Connected nodes.** Find all the nodes that are connected to a given node in a graph.
- **Cycle check.** Determine whether a graph contains a cycle.
- **Disconnected nodes.** Find all the nodes that are not connected to a given node in a graph.

Results

Method	Encoding	Edge Existence	Node degree	Node count	Edge count	Connected nodes	Cycle check
ZERO-SHOT	Overall (μ/δ)	44.5 / 9.4	14.0/16.0	21.73 / 8.6	12.4 / 4.8	14.7 / 11.0	76.0 / 13.2
	Adjacency	45.8	12.4	18.8	14.0	19.8	71.6
	Incident	39.6	25.0	15.6	10.6	53.8	68.8
	Co-authorship	44.0	13.8	22.0	11.4	7.6	70.8
	Friendship	46.6	11.2	23.0	10.2	4.0	**82.0**
	SP	46.4	9.0	22.4	15.0	6.2	80.4
	GOT	**49.0**	13.6	22.8	13.2	7.6	79.0
	Social network	43.2	16.0	22.8	10.8	8.2	81.2
	Politician	44.6	15.2	24.2	11.6	8.8	81.0
	Expert	41.2	10.0	24.0	14.8	16.4	69.6
ZERO-COT	Overall (μ/δ)	33.5 / 11.6	10.4 / 22.4	14.6 / 9.4	9.4 / 4.8	8.8 / 9.2	32.3 / 23.2
	Adjacency	34.2	15.4	11.0	12.2	6.0	46.2
	Incident	41.4	26.6	10.0	12.2	35.2	39.0
	Co-authorship	29.8	9.8	15.6	8.2	3.0	28.2
	Friendship	28.4	7.0	19.4	7.4	3.0	31.2
	SP	32.6	9.2	15.6	8.4	5.0	34.8
	GOT	34.6	8.4	16.2	8.4	5.4	33.4
	Social network	30.8	6.6	14.0	9.2	3.8	26.0
	Politician	38.0	4.2	14.6	8.6	3.2	23.0
	Expert	31.6	6.0	14.8	10.0	14.2	28.8
FEW-SHOT	Overall (μ/δ)	36.8 / 13.8	17.4 / 23.4	25.3 / 35.6	12.0 / 9.0	12.4 / 15.2	37.4 / 24.0
	Adjacency	42.8	15.4	47.2	18.6	22.2	47.8
	Incident	38.8	33.6	51.2	14.6	36.6	45.0
	Co-authorship	29.4	15.6	15.6	10.2	9.0	46.8
	Friendship	40.6	12.2	18.4	9.8	6.4	41.4
	SP	34.6	18.0	18.0	12.0	6.8	38.2
	GOT	40.6	17.2	14.2	12.0	3.4	28.6
	Social network	37.4	15.0	21.2	10.2	7.8	34.2
	Politician	38.0	13.4	21.4	9.6	7.8	30.8
	Expert	29.0	16.6	20.4	11.2	11.8	23.8

Results

COT	Overall (μ/δ)	42.8 / 7.0	29.2 / 60.4	27.6 / 42.4	12.8 / 17.4	13.1 / 18.0	58.0 / 16.4
	Adjacency	42.8	71.2	57.0	**25.2**	22.4	56.6
	Incident	41.6	**75.0**	**57.6**	21.4	30.2	62.6
	Co-authorship	43.2	16.4	15.2	8.8	8.4	54.8
	Friendship	46.6	14.6	23.0	7.8	9.6	61.8
	SP	42.6	17.4	17.0	10.6	8.2	59.4
	GOT	44.0	17.8	16.2	11.8	7.2	60.4
	Social network	42.6	16.4	21.6	8.4	8.0	60.6
	Politician	42.2	16.6	22.6	9.2	9.4	59.4
	Expert	39.6	17.4	18.0	12.4	14.4	46.2
COT-BAG	Overall (μ/δ)	37.3 / 16.6	28.0 / 61.8	26.9 / 33.8	12.5 / 17.8	15.8 / 31.8	52.1 / 26.0
	Adjacency	45.8	66.8	48.6	25.0	20.6	56.8
	Incident	45.6	75.2	51.2	21.8	**41.0**	63.0
	Co-authorship	25.0	14.6	17.4	7.2	9.2	37.0
	Friendship	39.0	16.2	21.8	7.4	9.8	52.0
	SP	33.6	17.0	21.6	11.4	11.4	52.2
	GOT	32.6	15.6	18.0	11.0	10.0	54.6
	Social network	44.8	13.4	19.6	9.0	10.0	51.2
	Politician	40.4	17.6	22.8	8.2	10.2	57.2
	Expert	29.2	15.8	20.8	11.6	20.4	45.0

Table 1: Comparison of various graph encoder functions based on their accuracy on different graph tasks using PaLM 62B. The most effective prompting heuristic is highlighted with an underline, and the top-performing graph encoder function for it is highlighted in bold. The overall result is represented its average (μ) and an absolute difference (δ) of its best and worst graph encoder.

Results

- **Overall LLM Performance:**
  - LLMs performed poorly on almost all basic graph tasks, often underperforming the majority baseline.
  - Notably, tasks such as edge existence and cycle check reveal significant weaknesses.
- **Prompting Heuristics:**
  - *Simple Prompts:* Zero-shot prompting outperforms Zero-shot CoT for simple tasks, as these tasks do not require multi-hop reasoning.
  - *Enhanced Prompts:* For more complex tasks, adding few-shot examples or using Chain-of-Thought (CoT) prompting generally improves performance.

Results

- **Impact of Graph Encoding Functions:**
  - The choice of graph encoder has a significant effect on LLM performance.
  - For example, **incident encoding** achieves notably higher accuracy (e.g., 53.8% for connected nodes) compared to **adjacency encoding** (e.g., 19.8%).
- **Node Encoding Schemes:**
  - Integer encoding (e.g., using "Node 0") improves performance on arithmetic tasks like node degree, node count, and edge count.
  - Encoder functions using specific names (e.g., GOT, Friendship) can be beneficial for non-integer tasks.

Results

graph encoding	ZERO-SHOT	ZERO-COT	FEW-SHOT	COT	COT-BAG
Adjacency	4.83	3.25	2.16	3.00	1.83
Incident	6.16	**2.58**	**2.00**	**2.33**	**1.33**
Co-authorship	6.08	6.33	5.58	6.75	8.83
Friendship	5.16	6.41	6.25	4.66	6.00
SP	5.16	4.50	5.25	5.75	4.66
GOT	4.33	4.08	5.83	5.00	6.25
Social Network	4.58	6.50	5.83	6.16	6.41
Politician	**3.50**	6.33	6.25	5.58	4.00
Expert	5.16	5.00	5.83	5.75	5.66

Table 5: Ranking of graph encodings from experiment in Section 3.1 (lower better).

Varying Prompt Questions

- **Objective:**
  - Compare two distinct question encoder functions while keeping the graph encoding function constant (friendship-based).
- **Question Encoders Compared:**
  - *Graph Question Encoder:* Directly formulates graph-related questions (e.g., "What is the degree of Node X?").
  - *Application Question Encoder:* Reframes the query in a more practical, real-world context (e.g., "How many friends does Node X have?").

Varying Prompt Questions

Method	Question encoder	LLM	Edge Existence	Node degree	Node count	Edge count	Connected nodes
ZERO-SHOT	Graph	PaLM 2-XXS	42.8	10.8	5.4	**5.6**	1.6
	Application	PaLM 2-XXS	**60.8**	**14.0**	**9.4**	4.4	**11.4**
	Graph	PaLM 62B	46.6	11.2	**23.0**	10.2	4.0
	Application	PaLM 62B	**47.8**	**16.6**	17.8	**13.2**	**6.0**
COT	Graph	PaLM2 XXS	50.4	8.8	8.4	4.2	10.2
	Application	PaLM2 XXS	**56.4**	**12.2**	**8.6**	**5.4**	**11.0**
	Graph	PaLM 62B	**46.6**	14.6	**23.0**	7.8	9.6
	Application	PaLM 62B	38.6	**16.6**	16.0	**12.2**	**10.0**

Table 2: Comparing two question encoders based on their accuracy for PaLM 2 XXS and PaLM 62B. The top-performing question encoder for the respective LLM is highlighted in bold.

Varying Prompt Questions

- **Key Findings:**
  - The Application Question Encoder generally outperforms the Graph Question Encoder across various tasks.
  - For example, on the ZERO-SHOT edge existence task using a PaLM 2 XXS model, the application encoder achieved significantly higher accuracy (e.g., 60.8% vs. 42.8%).
  - Improved performance suggests that contextualizing graph questions in application-relevant language aids LLM reasoning.

Multiple Relation Encoding

- **Objective:**
  - Modify the friendship graph encoding to incorporate multiple relation types.
- **Method:**
  - Instead of using a single token (e.g., "friends") for edge representation, the relation is randomly selected from a set including:
    - Friends, colleagues, spouses, siblings, neighbors, acquaintances, teammates, classmates, coworkers, or roommates.
- **Motivation:**
  - Provide richer, more natural textual cues.
  - Align the encoded graph description closer to the distribution of text seen during LLM pre-training.

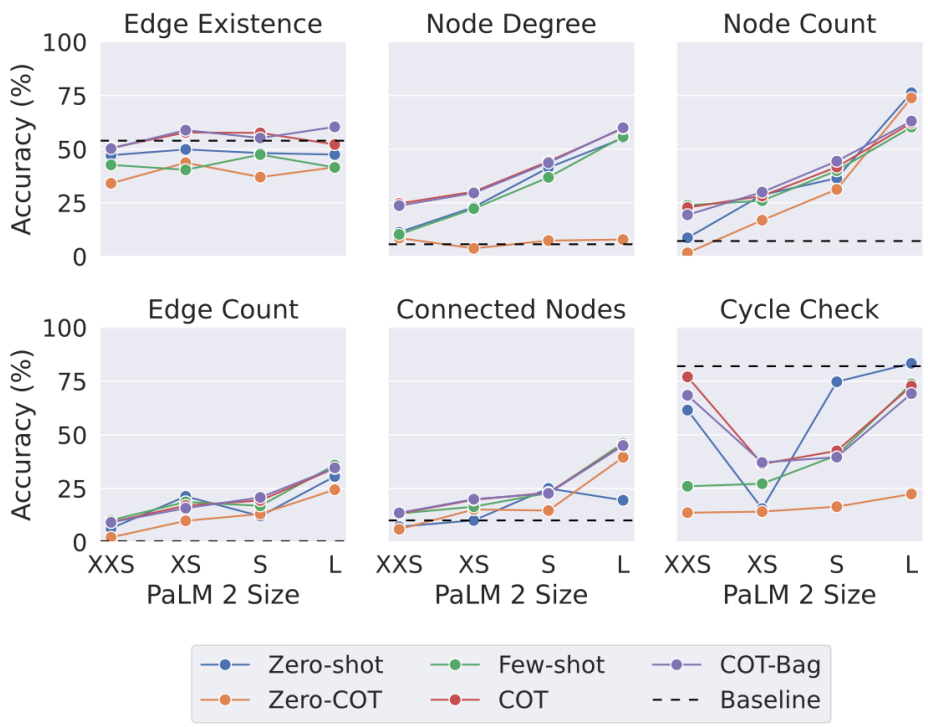
Multiple Relation Encoding

- **Results:**
  - Using multiple relation words does not hurt performance.
  - In some tasks, it even improves accuracy by offering more descriptive context.

	Task	Same relation	Multiple relations
ZERO-SHOT	Edge Existence	**42.8**	39.8
	Node degree	10.8	**11.6**
	Node count	5.4	**6.6**
	Edge count	**5.6**	5.4
	Connected nodes	1.6	**3.4**
	Cycle Check	65.2	**84.4**
COT	Edge Existence	50.4	**50.8**
	Node degree	8.8	**10.0**
	Node count	**8.4**	5.8
	Edge count	4.2	**5.0**
	Connected nodes	**10.2**	7.2
	Cycle Check	**77.4**	74.4

Model Capacity

- Model Capacity:
  - Larger models yield improved graph reasoning performance.
  - Some tasks (e.g., edge existence) are less sensitive to model size.



Disconnected Nodes

- **Task Overview:**
  - Evaluate LLMs on reasoning about nodes that are *not* connected in the graph.
  - Unlike connected nodes tasks, this requires understanding the absence of relationships.
- **Results:**
  - Zero-shot prompting achieved only around 0.5% accuracy.
  - Other methods (ZERO-COT, FEW-SHOT, CoT, and COT-BAG) resulted in near-zero performance.
- **Analysis:**
  - Graph encoding functions are primarily designed to represent existing edges.
  - They do not explicitly capture information about missing or disconnected nodes.

Impact of Graph Structure on LLM Reasoning

Random Graph Generation

- **Objective:**
  - Generate graphs with diverse structural properties to test LLM performance.
- **Approach:**
  - Use various graph generators:
    - **Erdős-Rényi (ER):** Produces sparse graphs with a low average degree.
    - **Barabási-Albert (BA):** Yields denser graphs with a power-law degree distribution.
    - **Stochastic Block Model (SBM):** Generates graphs with community structures.
    - Specific topologies such as **star**, **path**, and **complete** graphs.

Figure 4: Samples of graphs generated with different graph generators in our framework.

Impact of Graph Structure on LLM Reasoning

Method	Graph generator	Edge Existence	Node degree	Node count	Edge count	Connected nodes	Cycle check
ZERO-SHOT	Overall	49.1	17.6	23.0	12.1	23.3	75.2
	ER	45.1	13.6	22.1	11.7	14.9	76.3
	BA	50.2	18.0	24.9	13.6	20.1	72.0
	SBM	45.0	13.8	21.9	9.2	13.8	86.5
	Star	58.0	34.0	32.8	31.7	61.7	8.1
	SFN	57.6	23.1	19.9	8.0	38.1	90.0
	Path	**60.9**	14.8	31.9	28.8	26.6	5.9
	Complete	19.8	12.6	20.7	6.2	13.3	**91.7**
COT	Overall	40.4	29.6	31.7	12.2	24.3	59.5
	ER	41.2	28.4	28.8	12.6	12.8	61.2
	BA	40.0	30.0	35.0	14.3	20.8	58.5
	SBM	40.3	26.5	30.2	8.7	13.0	65.8
	Star	40.3	**38.0**	**41.8**	**31.6**	**68.6**	21.3
	SFN	40.2	32.2	30.8	7.1	43.2	66.0
	Path	42.0	35.1	35.3	31.1	27.6	19.7
	Complete	39.6	21.9	28.9	3.9	14.6	69.3

Table 4: Comparing different graph generators on different graph tasks on PaLM 62B. The most effective prompting heuristic is highlighted with an underline, and the top-performing graph generator algorithm for the respective heuristic is highlighted in bold.

Impact of Graph Structure on LLM Reasoning

- **Performance Variation Across Graph Types:**
  - **Cycle Check Task:**
    - *Complete Graphs:* Always contain cycles, leading to very high accuracy (e.g., over 90%).
    - *Path Graphs:* Never have cycles, resulting in very low accuracy (e.g., below 10%).
  - **Edge Existence and Node Counting Tasks:**
    - Dense graphs (such as complete graphs) may introduce distracting information due to the high number of edges.
    - Sparse graphs (such as ER graphs) might better reflect the task conditions but can also challenge the model if the graph structure is too minimal.

Impact of Graph Structure on LLM Reasoning

- **Use of Out-of-Distribution Few-Shot Examples:**
  - The authors also experimented with few-shot examples drawn from different graph generators.
  - These out-of-distribution examples help the LLM understand the task better by showing diverse instances, but they do not need to come from the same graph generator as the test examples.
  - This suggests that the primary benefit of few-shot prompting is to clarify the task, rather than to provide a template of the graph structure.

Limitation

- The black-box approach uses fixed pre-trained LLMs without fine-tuning, limiting its adaptability for specialized or more complex graph reasoning tasks.
- The current encoding methods primarily capture existing connections, resulting in poor performance on tasks that require reasoning about absent or disconnected relationships.

Conclusion

- **Graph Encoding Matters**:
  Tailored encoding of graph data into text significantly improves LLM reasoning performance.

- **Effective Prompting is Key**:
  Optimizing the prompt via various heuristics (zero-shot, few-shot, CoT, bag prompting) boosts task accuracy.

- **Graph Structure Impacts Performance**:
  LLMs show varied performance based on graph topology, underscoring the need for structure-aware encoding.

- **Black-Box Approach Benefits**:
  The method leverages pre-trained LLMs without modifying their internals, making it practical for diverse applications.

UNIVERSITY OF
WATERLOO

FACULTY OF MATHEMATICS

Thank you!