# GRAPHGEM: Improving Graph Reasoning in Language Models with Synthetic Data

Stanford CS224N Custom Project

**Tony Sun**
Department of Computer Science
Stanford University
`tony.sun@cs.stanford.edu`

## Abstract

Reasoning over graph problems is challenging for language models. By adapting code from Fatemi et al. (2023), we generate and publically release `GraphQA (Easy)` and `GraphQA (Hard)`, two synthetic datasets composed of 432,000 question-answer-explanation triplets over 12 graph-related tasks in total. We fine-tune Gemma-2B-it on these datasets and find that the resulting model family, GRAPHGEM, improves on average by 45.7% (Easy) and 42.4% (Hard). Additionally, we find that find that fine-tuning Gemma-2B-it on `GraphQA (Hard)` improves its 0-shot GSM8K performance by 17.5%, despite never seeing any GSM8K problems during training. [1]

## 1 Introduction

In order to accurately answer questions such as "What are the indicators of collusive bidding patterns in some given data from public procurement auctions?" or "What is most efficient pathway for a given drug to reach a specific target cell in the human body?", language models should be able to implicitly reason over graphs. For instance, the first question may be mapped to a triangle counting problem, and the second question may be mapped to a shortest path problem.

However, existing language models do not exhibit strong graph reasoning ability. In Table 1, Gemma-2B-it and GPT-4 are asked about a triangle counting problem in a social network graph. Gemma-2B-it thinks there are 9 nodes and 13 edges, when in fact there are 10 nodes and 16 edges. Additionally, Gemma-2B-it implicitly reasons that there is a direct relationship between the number of nodes and edges to the number of triangles, which is false. GPT-4 almost answers correctly, but mistakenly counts a triangle that does not exist. [2]

In this work, we make the following contributions:

$(i)$ We train a family of models, GRAPHGEM, by fine-tuning Gemma-2B-it with QLoRA on completely synthetic data to improve its graph reasoning capabilities.

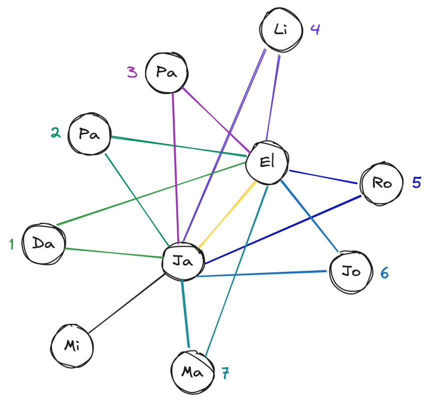
$(ii)$ We experiment with training with explanations and investigate the effect of scale and curriculum learning on training. Additionally, we examine whether or not training on one set of graph problems transfers to other graph problems and GSM8K.

$(iii)$ We release `GraphQA (Easy)` and `GraphQA (Hard)`. In total, there are 432,000 question-answer-explanation triplets across 12 graph-related tasks. The dataset is synthetically generated using code adapted from Fatemi et al. (2023). [3]

[1] Mentor: Rohan Taori; External collaborators (if you have any): N/A; External mentor (if you have any): Ian Covert; Sharing project: N/A

[2] `https://chat.openai.com/share/5d28c0e1-bd6c-42c7-ba47-0c6a5178660a`

[3] `https://huggingface.co/datasets/tonysun9/GraphQA_Easy`
`https://huggingface.co/datasets/tonysun9/GraphQA_Hard`

| *Question* | *For Visualization Purposes* |
|---|---|
| G describes a social network graph among nodes James, Robert, John, Michael, David, Mary, Patricia, Jennifer, Linda, and Elizabeth.<br><br>We have the following edges in G:<br>James and Robert are connected.<br>James and John are connected.<br>...<br>Robert and Elizabeth are connected.<br>John and Elizabeth are connected.<br>...<br>**Q: How many triangles are in this graph?**<br>**A:**<br><br>*Target*<br>**7.** |  Li<br>4<br>3<br>Pa<br>2<br>Pa<br>El<br>Ro<br>5<br>1<br>Da<br>Ja<br>Jo<br>6<br>Mi<br>Ma<br>7 |

| *GraphGem (correct)* | *Gemma-2B-it (incorrect)* | *GPT-4 (incorrect)* |
|---|---|---|
| **7.**<br><br>There are 7 triangles including node James as a vertex.<br>There is 1 triangle including node Robert as a vertex.<br>There is 1 triangle including node John as a vertex.<br>...<br>There are 7 triangles including node Elizabeth as a vertex.<br>Summing the number of triangles for all nodes and dividing them by three gives us **7 triangles in total**.<br>The triangles are: [(0, 1, 9), (0, 2, 9), (0, 4, 9), (0, 5, 9), (0, 6, 9), (0, 7, 9), (0, 8, 9)]. | A graph with 9 nodes and 13 edges has **5 triangles.** | To determine the number of triangles in the graph described, let's first understand the structure based on the connections (edges) provided. A triangle in a graph is a set of three nodes where each node is connected to the other two nodes directly, forming a closed loop.<br>...<br>1. James - Elizabeth - Robert<br>2. James - Elizabeth - John<br>3. James - Elizabeth - Michael<br>4. James - Elizabeth - David<br>5. James - Elizabeth - Mary<br>6. James - Elizabeth - Patricia<br>7. James - Elizabeth - Jennifer<br>8. James - Elizabeth - Linda<br>There are **8 unique triangles** involving James and Elizabeth with each of the other nodes, based on the connections provided. |

Table 1: A question about counting the number of triangles in a social network graph. James is connected to everyone, and Elizabeth is connected to everyone except Michael. GRAPHGEM responds with the correct answer, 7 triangles. Gemma-2B-it and GPT-4 answer incorrectly, responding with 5 and 8 triangles, respectively. The key result of each answer is bolded. Mistakes in reasoning are underlined.

## 2 Related Work

### 2.1 Large Language Models

Large language models (LLMs) are pre-trained on vast corpora of text data and are suitable for fine-tuning on a target task (Devlin et al., 2018; Brown et al., 2020). Parameter-efficient fine-tuning (PEFT) is a class of methods that adapt LLMs while making minimal adjustments to the model's parameters (Houlsby et al., 2019; Li and Liang, 2021).

LoRA is a PEFT method that keeps the pre-trained language model weights frozen and only updates a small, low-rank projection of the weight matrices (Hu et al., 2021). QLoRA leverages the 4-bit NormalFloat (NF4) type for quantization among other optimizations to be even more memory-efficient while maintaining 16-bit fine-tuning performance Dettmers et al. (2023).

### 2.2 Machine Learning for Graphs

Graph neural networks (GNNs) are a class of machine learning models designed to capture the complex relationships and structures inherent in graph data. They extend traditional neural network approaches to directly operate on graphs, enabling effective representation and processing of non-Euclidean data for tasks such as node classification, link prediction, and graph classification (Kipf and Welling, 2017; Hamilton et al., 2018).

However, in comparison to LLMs, GNNs struggle with generalization across diverse domains due to the specialized nature of graph data and lack a unified pre-training approach that leverages vast, varied datasets for broad applicability.

### 2.3 LLMs and Graph Reasoning

Tasks such as multi-hop question answering and structured commonsense reasoning may ask LLMs to maintain an implicit graph structure in order to effectively deduce logical conclusions. Recent work has laid the foundation for understanding the ability of LLMs to reason over graphs.

`GraphQA` is a thorough benchmark designed to test language models' performance on various graph-related tasks (Fatemi et al., 2023). Questions are constructed across 7 graph generation algorithms and 9 diverse graph encoding methods. NLGraph is a similar benchmark, composed of graph-based problem solving designed in natural language Wang et al. (2024). Several tasks in `GraphQA` overlap with NLGraph.

Both works perform preliminary investigation into the graph reasoning abilities of language models and find that while large language models (LLMs) are capable of solving some basic graph problems, they consistently struggle with more complex ones, even with various prompting methods.

GraphToken is a PEFT method that learns to encode the graph data in the prompt (Perozzi et al., 2024). Without prior knowledge, however, it may be trivial to identify which parts of a given prompt constitute a graph.

## 3 Methodology

### 3.1 Dataset

We choose to use the `GraphQA` benchmark since there is code to generate synthetic question-answer pairs in the authors' repository Fatemi et al. (2023). We divide the tasks into `GraphQA (Easy)` and `GraphQA (Hard)`. Tasks are divided such that those present in Fatemi et al. (2023) as of March 2024 are categorized as "easy" and tasks not yet in the paper but present in `https://github.com/google-research/google-research/tree/master/graphqa`, commit hash `eb560d0`, are categorized as "hard." Explanation of each of the tasks can be found in Table 2.

While the authors of `GraphQA` release the code to generate graph reasoning problems, the corresponding dataset is not public. Therefore, we adapt their code to create `GraphQA (Easy)`and `GraphQA (Hard)`, which are solely composed of synthetic data. In total, there are 432,000 question-answer-explanation triplets across 80-10-10 train-validation-test splits. Each sample corresponds to a task-specific problem on a given graph, which is represented by its graph generation algorithm, text encoding (e.g. representing an edge as (0, 1) vs "John wrote a paper with Amy"), and a random seed. Additionally, we leverage existing code to create a new column, "explanation," which provides reasoning for the answer. For instance, for the cycle check task, the explanation would also provide a sample cycle through the graph in addition to the yes / no answer.

| `GraphQA` | **Task** | **Description** |
|---|---|---|
| Easy | 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. |
| Hard | Disconnected Nodes | Find all the nodes that are not connected to a given node in a graph. |
| | Reachability | Determine whether or not there exists a path between two given nodes. |
| | Shortest Path | Calculate the length of the shortest path between two given nodes. |
| | Maximum Flow | Calculate the maximum capacity of the flow between a given source and sink node. |
| | Triangle Counting | Count the number of triangles in a graph. |
| | Node Classification | Classify the category of a given node in a graph. |

Table 2: We split `GraphQA` into `GraphQA (Easy)` and `GraphQA (Hard)`. Each split contains 6 tasks.

We use 4 graph generation algorithms and 9 text encoding methods [4]. Notably, we use 4 types of graph generators rather than 7 as done in Fatemi et al. (2023) because we find that some types of graphs are limited in their diversity, leading to train-test contamination. The 3 types of graphs we exclude are star, path, and complete graphs. Other combinations of graph generation algorithms and text encodings also lead to some duplicate questions. Therefore, we remove duplicates while ensuring an even split of questions across graph types and text encodings. Finally, we check that each question in the final dataset is unique. Additional information about the dataset can be found in Table 3.

| | **Graphs Per Generator** | **Graphs Generated** | **QA-Pairs (Single Task)** | **QA-Pairs (Easy / Hard)** | **# Tokens (Questions)** | **# Tokens (Answers)** | **# Tokens (Explanations)** |
|---|---|---|---|---|---|---|---|
| Multiplying Factor | N/A | 4 Graph Generators | 9 Encoding Methods | 6 Tasks per Split | 290.5 Avg Tokens per Question | 5.6 Avg Tokens per Answer | 100.9 Avg Tokens per Explanation |
| Train | 800 | 3,200 | 28,800 | 172,800 | 49.6 / 49.4M | 1.0 / 0.9M | 8.3 / 25.9M |
| Validation | 100 | 400 | 3,600 | 21,600 | 7.0 / 6.7M | .13 / .12M | 1.2 / 3.7M |
| Test | 100 | 400 | 3,600 | 21,600 | 6.4 / 6.4M | .13 / .12M | 1.1 / 3.4M |
| Easy / Hard | 1,000 | 4,000 | 36,000 | 216,000 | 63 / 62.5M | 1.3 /1.1M | 10.6 / 33M |
| Total | 2,000 | 8,000 | 72,000 | 432,000 | 125.5M | 2.4M | 43.6M |

Table 3: Number of graphs generated, question-answer pairs, and number of tokens in `GraphQA (Easy)` and `GraphQA (Hard)`. We adapt the graph generation code from Fatemi et al. (2023) to create these datasets. The number of graphs generated depends on the number of graph generators (4) and a choice of number of graphs per graph generation algorithm. Then, for each task, (9) question-answer pairs are generated for each graph, one for each type of graph encoding methods. The number of question-answer pairs is multiplied by the number of tasks (6). We use the Gemma tokenizer to count the number of tokens.

## 3.2 Prompting

We prompt the language model as follows:

```
{<bos><start_of_turn>user\n{question}<end_of_turn>\n<start_of_turn>model\n}
```

where `{question}` contains both the graph representation in text and the task-specific question. This follows the standard instruction-prompting method for Gemma-2B-it [5].

On an earlier version of the dataset, we experimented with variations in the prompting method, such as removing the `<bos>` token and also only using `{question}` as the prompt, each in a zero-shot

[4]Four graph generation algorithms: Erdős–Rényi, scale-free networks, Barabási–Albert, stochastic block model. Nine text encoding methods: adjacency, incident, friendship, co-authorship, South Park, Game of Thrones, social network, politician, expert

[5]`https://ai.google.dev/gemma/docs/formatting`

and chain-of-thought setting. Although the presence of the `<bos>` token seemed to have a negligible impact on performance, we decide to include it for best practices [6]. We found that stripping the entire Gemma instructions degraded performance in both the zero-shot and chain-of-thought settings, the two heuristics identified as most effective in Fatemi et al. (2023). On the other hand, the inclusion of the Gemma instructions in a zero-shot setting led to comparable performance compared to chain-of-thought prompting, with and without instructions. The full results of this experiment are in Appendix A.

### 3.3 Fine-tuning Gemma

We fine-tune Gemma with the language modeling objective separately on question-answer pairs from `GraphQA (Easy)` and `GraphQA (Hard)` to produce GraphGem (E) and GraphGem (H), respectively. We also experiment with concatenating the explanation to the answer. GraphGem models trained with explanations in addition to the answer are denoted with the suffix "-R." for reasoning.

We use Unsloth [7], bitsandbytes[8], and the Huggingface Transformers, TRL, and PEFT libraries (Wolf et al., 2020; Mangrulkar et al., 2022; von Werra et al., 2020) for training. In the style of QLoRA (Dettmers et al., 2023), we load Gemma-2B-it, quantized to 4-bit precision using the NF4 data type, and train a LoRA module in bfloat16 applied to all parameters to reduce memory usage. We use LoRA rank=16, alpha=16. For hyperparameters, we use batch size=8, learning rate=1e-5, cosine learning rate scheduler, 500 warm-up steps, and the 8-bit AdamW optimizer. Training one model on a single A100 took less than 12 hours on average.

### 3.4 Curriculum Fine-tuning

We want to take advantage of the fact that graph problems are easy to generate and answers are quick to verify. Our idea is simple: at some interval, evaluate the model's performance, and generate more problems for areas where the model struggles, and generate less problems for areas where the model does well. More formally, we evaluate the model performance on the validation set at every interval $i$, starting from a base model, and create a dataset of size $|S|$ with the following distribution over tasks $T$, graph generation algorithms $G$, and text encodings $E$:

$$\alpha_{t,g,e,i+1} = \frac{1 - A_{t,g,e,i}}{\sum_{t=1}^{6} \sum_{g=1}^{4} \sum_{e=1}^{9} 1 - A_{t,g,e,i}}, \text{ where } (t, g, e) \in (T \times G \times E)$$

and $\alpha_{t,g,e,i+1} \cdot |S|$ represents the number of examples with task $t$, graph generator $g$, and encoding $e$ in the $i + 1$ training set, and $1 - A_{t,g,e}$ represents the model's accuracy at interval $i$ on the ($t$, $g$, $e$) subset of the validation set.

In practice, we set $i$ to be 2,160, which is one-tenth of the steps for training on `GraphQA (Hard)`, and $|S|$ to be 17,280, which is one-tenth of the size of `GraphQA (Hard)`.

## 4 Results

### 4.1 `GraphQA (Easy)`

We evaluate Gemma-2B-it and GraphGem (E) on `GraphQA (Easy)`. We use the reported results from Fatemi et al. (2023); Perozzi et al. (2024) for PaLM and Flan-it with GraphToken. However, the test set may slightly differ due to our concerns around train-test contamination and consequent exclusion of the star, path, and complete graph types. We report our results in Table 9.

In the zero-shot setting, GraphGem-R (E) improves on average by 45.7% compared to Gemma. The greatest gains in accuracy are in the edge existence, node degree, and connected nodes tasks, while the smallest gains are in the edge count and cycle check tasks. We find that Gemma-2B-it, surprisingly, already demonstrates high accuracy in the cycle check task, leaving little room for

[6]`https://unsloth.ai/blog/gemma-bugs`

[7]`https://github.com/unslothai/unsloth`

[8]`https://github.com/TimDettmers/bitsandbytes?tab=readme-ov-file`

| Model | Method | Edge Exist. | Node Degree | Node Count | Edge Count | Conn. Nodes | Cycle Check | Avg. |
|---|---|---|---|---|---|---|---|---|
| PaLM 2-XXS* | 0-shot | 47.2 | 11.3 | 8.7 | 6.4 | 7.2 | 61.5 | 23.7 |
| PaLM 2-XXS* | CoT | 50.6 | 24.7 | 22.8 | 9.3 | 13.3 | 77.0 | 32.9 |
| PaLM 62B* | 0-shot | 44.5 | 14.0 | 21.7 | 12.4 | 14.7 | 76.0 | 30.6 |
| PaLM 62B* | CoT | 42.8 | 29.2 | 27.6 | 12.8 | 13.1 | 58.0 | 30.6 |
| Gemma-2B-it | 0-shot | 42.1 | 28.7 | 16.0 | 6.4 | 11.1 | 91.0 | 32.6 |
| Gemma-2B-it | CoT | 50.8 | 25.7 | 18.7 | 9.2 | 11.1 | 75.6 | 31.8 |
| Flan-it (PaLM 2 S)* | GraphToken | 73.8 | **96.2** | **99.6** | **42.6** | 26.4 | 95.6 | 72.4 |
| GraphGem (E) | 0-shot | 99.1 | 87.8 | 57.7 | 23.7 | 86.4 | 98.9 | 75.6 |
| GraphGem-R (E) | 0-shot | **99.4** | 90.2 | 67.8 | 24.9 | **88.7** | **99.0** | **78.3** |

Table 4: Performance comparison of PaLM, Flan-it, Gemma-2B-it, and GraphGem based on accuracy on `GraphQA (Easy)`. Results for models marked with an asterisk (*) were reported in a previous study. The highest accuracies on each task are bolded / underlined (with / without training on `GraphQA (Easy)`).

improvement. However, upon closer inspection, the cycle check task is heavily skewed towards graphs with cycles. In fact, 91% of the questions for the cycle check task contain a cycle, and Gemma always responds that there is a cycle in the graph, leading to exactly 91% accuracy.

## 4.2 `GraphQA (Hard)`

Similarly, we evaluate Gemma-2B-it and GraphGem (H) on `GraphQA (Hard)` and use the reported results from Perozzi et al. (2024) for Flan-it with GraphToken.

| Model | Method | Disconn. Nodes | Max. Flow | Node Class. | Reach-ability | Shortest Path | Triangle Count. | Avg. |
|---|---|---|---|---|---|---|---|---|
| Gemma-2B-it | 0-shot | 5.2 | 7.7 | 53.4 | 86.8 | 11.8 | 4.6 | 28.2 |
| Gemma-2B-it | CoT | 5.0 | 11.8 | 60.7 | 37.3 | 33.2 | 14.0 | 27.0 |
| Flan-it (PaLM 2 S)* | GraphToken | - | - | - | 93.2 | 63.8 | 34.8 | 63.9 |
| GraphGem (H) | 0-shot | 64.8 | 14.5 | **99.8** | **99.7** | 91.2 | 33.8 | 67.3 |
| GraphGem-R (H) | 0-shot | **74.8** | **15.6** | **99.8** | **99.7** | **96.1** | **37.9** | **70.6** |

Table 5: Performance comparison of Flan-it, Gemma-2B-it, and GraphGem based on accuracy on `GraphQA (Hard)`. Results for models marked with an asterisk (*) were reported in a previous study. The highest accuracies on each task are bolded / underlined (with / without training on `GraphQA (Hard)`).

In the zero-shot setting, GraphGem-R (H) improves on average by 42.4% compared to Gemma. The greatest gains in accuracy are in the disconnected nodes, node classification, and shortest path tasks, with a 84.3% improvement in the shortest path task. Maximum flow and triangle counting are the most difficult tasks. Similar to the connected nodes task in `GraphQA (Easy)`, the reachability task in `GraphQA (Hard)`is heavily skewed towards questions in which there does exist a path.

We find that fine-tuning with reasoning improves performance across all tasks.

Additionally, in Figure 1, we explore the effect of the scale of training data on GraphGem-R (H) performance over each of the 6 tasks in `GraphQA (Hard)`. The sharpness of the curve may indicate how difficult a task is to learn or how much knowledge the base model, Gemma-2B-it, already has about a given task.

For example, since performance increases dramatically on the disconnected nodes task as training data increases, it may indicate that the model may be gaining new knowledge about this task in the training process. Conversely, for the shortest path task, given that performance on the shortest path

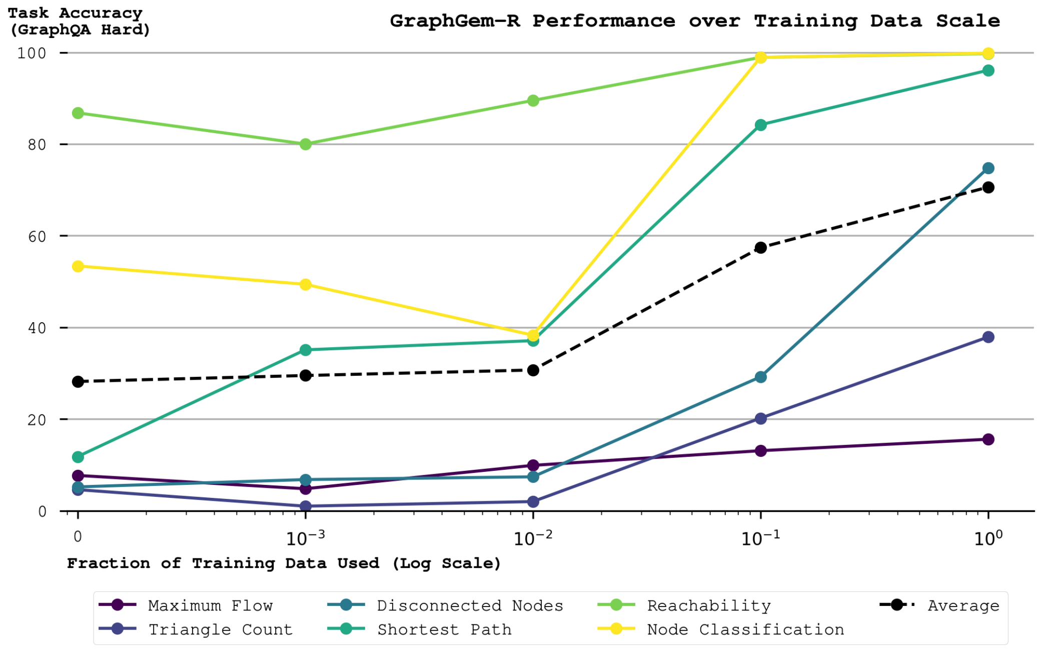


Figure 1: Fraction of training data used vs GRAPHGEM-R (H) accuracy.

task increases greatly with just 10% of the data and plateaus after, fine-tuning may serve to help the model understand the style of the questions in the dataset rather than teach it how to find the shortest path through a graph. For the maximum flow task, where the performance is stagnant even as the amount of training data increases, it may indicate that the task is difficult or that the explanation may not be written in a way that induces learning.

## 4.3 Curriculum Fine-tuning

We apply curriculum fine-tuning to GRAPHGEM-R (H) over 3 iterations: 10%, 20%, and 30%. We report our results in Table 6.

We find that curriculum fine-tuning has a negligible impact on model performance over 3 iterations. We posit that this could be due to applying curriculum fine-tuning early in the training process. Since the model has room to improve on all tasks at the beginning, curriculum fine-tuning does little to change the distribution of graph problems in the training set. However, had we applied curriculum fine-tuning towards the end of training, we might have expected the training data to be more skewed towards harder tasks.

| % of Training Data | Curriculum FT | FT |
|---|---|---|
| 10% | 55.3 | **57.4** |
| 20% | **61.9** | 61.8 |
| 30% | 65.1 | **66.3** |

Table 6: GRAPHGEM-R (H) average performance on `GraphQA (Hard)` with various amounts of training data with and without curriculum fine-tuning (FT).

## 4.4 Transfer Learning

Finally, we explore whether training on one split of `GraphQA (Easy / Hard)` transfers to improved performance on the other split and if training on graph-related tasks improves model performance on GSM8K. We report our findings in Table 7.

We find that training on `GraphQA (Hard)` indeed improves performance on `GraphQA (Easy)`. Notably, training with reasoning leads to a much greater gain in performance than training without. On the other hand, training on `GraphQA (Easy)` surprisingly degrades performance on `GraphQA (Hard)`, even with reasoning.

| **Model** | `GraphQA (Easy)` | `GraphQA (Hard)` | **GSM8K** |
|---|---|---|---|
| Gemma-2B-it | 32.6 | **28.2** | 10.0 |
| GRAPHGEM (E) | - | 16.0 | 24.1 |
| GRAPHGEM-R (E) | - | 20.0 | 11.2 |
| GRAPHGEM (H) | 33.8 | - | **27.5** |
| GRAPHGEM-R (H) | **40.0** | - | 18.4 |

Table 7: Effect of training on `GraphQA (Easy / Hard)` on unseen graph problems and GSM8K. Evaluation is done in a 0-shot setting, and we take the average performance across tasks for `GraphQA (Easy)` and `GraphQA (Hard)`. Evaluation on test sets that are in-distribution are omitted for clarity.

Training on either `GraphQA (Easy)` or `GraphQA (Hard)` improves 0-shot performance on GSM8K, despite never seeing any problems in GSM8K. The biggest gain is training on `GraphQA (Hard)`, which improves task performance by 17.5%. Training with reasoning actually worsens performance compared to training with just the answer, although still improves performance compared to Gemma-2B-it. This may be because learning to reason over graph problems may not directly transfer to answering more accurately on math problems.

## 5 Future Work

We explored fine-tuning Gemma-2B-it on synthetic graph data to improve its graph reasoning capabilities. We examined the effect of scale on model performance and whether training on one set of tasks led to better performance on unseen graph and math problems.

For future work, promising directions include balancing the positive and negative class for the cycle check and reachability tasks, applying curriculum fine-tuning for more iterations, and examining GRAPHGEM out-of-distribution performance on more datasets.

## References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Inductive representation learning on large graphs.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for llms.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2024. Can language models solve graph problems in natural language?

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

## A Prompting

| Eval Mode | Prompt | Edge Existence | Node Degree | Node Count | Edge Count | Connected Nodes | Cycle Check |
|---|---|---|---|---|---|---|---|
| ZERO-SHOT | Question Only | 39.3 | 32.1 | 34.4 | 21.0 | 20.7 | 45.1 |
| | Conversation Format | 46.0 | 31.8 | 21.4 | 18.9 | 16.5 | 65.9 |
| | <bos> + Conversation Format | 45.2 | 34.9 | 15.8 | 14.8 | 16.9 | 65.9 |
| COT | Question Only | 50.8 | 29.1 | 17.8 | 13.9 | 15.0 | 61.8 |
| | Conversation Format | 49.5 | 28.5 | 12.6 | 7.4 | 14.5 | 65.2 |
| | <bos> + Conversation Format | 50.1 | 29.6 | 11.1 | 8.9 | 11.8 | 65.2 |

Table 8: Prompting on `GraphQA (Easy)`

## B Ablations

| SFT Method | Edge Existence | Node Degree | Node Count | Edge Count | Connected Nodes | Cycle Check | Average |
|---|---|---|---|---|---|---|---|
| Base SFT | 0.952 | 1.0 | 0.429 | 0.825 | 1.0 | 0.73 | 0.823 |
| No gradient accumulation | 1.0 | 1.0 | 0.429 | 0.905 | 1.0 | 0.714 | 0.841 |
| Query-key LoRA | 1.0 | 1.0 | 0.429 | 0.794 | 1.0 | 0.714 | 0.823 |
| 4e-4 lr | 0.984 | 1.0 | 0.429 | 0.825 | 1.0 | 0.635 | 0.812 |
| 1e-4 lr | 1.0 | 1.0 | 0.429 | 0.857 | 1.0 | 0.698 | 0.831 |
| 2e-5 lr | 0.968 | 1.0 | 0.429 | 0.921 | 1.0 | 0.762 | 0.847 |
| 1e-5 lr | 0.984 | 1.0 | 0.429 | 0.889 | 1.0 | 0.73 | 0.839 |
| 1e-5 lr, 2 epochs | 0.984 | 1.0 | 0.762 | 0.825 | 1.0 | 0.841 | 0.902 |
| 1e-5 lr, 3 epochs | 0.984 | 1.0 | 0.937 | 0.873 | 1.0 | 0.873 | 0.945 |

Table 9: Ablations on a base SFT run on `GraphQA (Easy)` using a training dataset with 30,420 QA pairs. The results are calculated on a small validation set of 378 examples to save time. The base SFT run uses learning rate (lr) = 2e-4, linear learning rate schedule, gradient accumulation steps = 4, per device train batch size = 2 (effective batch size of 8), max sequence length = 2048, LoRA for all modules, LoRA r = 16, LoRA alpha = 16, and LoRA gradient checkpointing. Additionally, the model is loaded in 4-bit and uses an 8-bit AdamW optimizer and the bfloat16 data type for LoRA.

## C Stratified Performance

| | Method | Disconn. Nodes | Max. Flow | Node Class. | Reach- ability | Shortest Path | Triangle Count. |
|---|---|---|---|---|---|---|---|
| | Overall | 99.4 | 90.2 | 67.8 | 24.9 | 88.7 | 99.0 |
| Graph Generator | Adjacency | 99.0 | 94.5 | 95.8 | 36.5 | 96.5 | 99.0 |
| | Incident | 99.5 | 93.5 | 99.8 | 17.2 | 99.5 | 99.0 |
| | Co-authorship | 99.8 | 89.2 | 70.8 | 24.2 | 89.5 | 99.2 |
| | Friendship | 100.0 | 90.0 | 73.2 | 24.8 | 88.8 | 99.2 |
| | SP | 99.8 | 89.0 | 47.0 | 26.8 | 83.0 | 99.0 |
| | GOT | 99.0 | 91.0 | 52.0 | 27.5 | 82.8 | 99.0 |
| | Social Network | 99.5 | 86.8 | 65.5 | 23.8 | 88.0 | 99.0 |
| | Politician | 99.5 | 94.0 | 53.0 | 25.2 | 78.5 | 98.5 |
| | Expert | 99.0 | 83.8 | 53.0 | 18.2 | 91.8 | 99.2 |
| Encoding | ER | 99.4 | 89.2 | 70.1 | 24.6 | 86.6 | 97.6 |
| | BA | 99.4 | 88.4 | 63.7 | 15.6 | 90.2 | 100.0 |
| | SBM | 99.7 | 87.2 | 72.0 | 25.7 | 83.3 | 98.6 |
| | SFN | 99.2 | 95.9 | 65.3 | 33.9 | 94.7 | 100.0 |

Table 10: GRAPHGEM-R (E) performance across graph generators and encodings.

| | Method | Disconn. Nodes | Max. Flow | Node Class. | Reach- ability | Shortest Path | Triangle Count. |
|---|---|---|---|---|---|---|---|
| | Overall | 74.8 | 15.6 | 99.8 | 99.7 | 96.1 | 37.9 |
| Graph Generator | Adjacency | 73.5 | 17.5 | 99.5 | 99.8 | 96.2 | 37.0 |
| | Incident | 81.2 | 15.2 | 99.8 | 99.8 | 96.2 | 32.8 |
| | Co-authorship | 76.2 | 14.0 | 100.0 | 99.5 | 96.8 | 41.8 |
| | Friendship | 77.2 | 15.0 | 99.8 | 99.5 | 96.0 | 39.5 |
| | SP | 71.0 | 14.5 | 99.8 | 99.8 | 96.8 | 38.8 |
| | GOT | 70.0 | 16.2 | 100.0 | 99.8 | 97.2 | 40.0 |
| | Social Network | 76.2 | 14.5 | 99.8 | 99.8 | 96.2 | 38.0 |
| | Politician | 71.5 | 15.8 | 99.8 | 99.5 | 95.2 | 38.2 |
| | Expert | 76.5 | 15.0 | 99.8 | 99.8 | 94.0 | 34.5 |
| Encoding | ER | 67.1 | 22.1 | 99.7 | 99.1 | 93.9 | 28.9 |
| | BA | 75.3 | 9.8 | 99.7 | 100.0 | 97.3 | 25.3 |
| | SBM | 68.6 | 13.0 | 100.0 | 99.7 | 96.3 | 27.0 |
| | SFN | 88.3 | 16.3 | 99.8 | 99.9 | 96.8 | 70.1 |

Table 11: GraphGem-R (H) performance across graph generators and encodings.