Graph-ToolFormer: To Empower LLMs with Graph Reasoning Ability via Prompt Dataset Augmented by ChatGPT

Jiawei Zhang
jiawei@ifmlab.org
IFM Lab
Department of Computer Science,
University of California, Davis
Davis, California, USA

ABSTRACT

In this paper, we aim to develop a large language model (LLM) with the reasoning ability on complex graph data. Currently, LLMs have achieved very impressive performance on various natural language learning tasks, extensions of which have also been applied to study the vision tasks with multi-modal data. However, when it comes to the graph learning tasks, existing LLMs present very serious flaws due to their several inherited weaknesses in performing multi-step logic reasoning, precise mathematical calculation and perception about the spatial and temporal factors.

To address such challenges, in this paper, we will investigate the principles, methodologies and algorithms to empower existing LLMs with graph reasoning ability, which will have tremendous impacts on the current research of both LLMs and graph learning. Inspired by the latest ChatGPT and Toolformer models, we propose the Graph-ToolFormer (Graph Reasoning oriented Toolformer) framework to teach LLMs themselves with prompts augmented by ChatGPT to use external graph reasoning API tools. Specifically, we will investigate to teach Graph-ToolFormer to handle various graph data reasoning tasks in this paper, including both (1) very basic graph data loading and graph property reasoning tasks, ranging from simple graph order and size to the graph diameter and periphery, and (2) more advanced reasoning tasks on real-world graph data, such as bibliographic networks, protein molecules, sequential recommender systems, social networks and knowledge graphs.

Technically, to build Graph-ToolFormer, we propose to handcraft both the instruction and a small-sized of prompt templates for each of the graph reasoning tasks, respectively. Via in-context learning, based on such instructions and prompt template examples, we adopt ChatGPT to annotate and augment a larger graph reasoning statement dataset with the most appropriate calls of external API functions. Such augmented prompt datasets will be post-processed with selective filtering and used for fine-tuning existing pre-trained causal LLMs, such as the GPT-J, to teach them how to use graph reasoning tools in the output generation. To demonstrate the effectiveness of Graph-ToolFormer, we conduct some preliminary experimental studies on various graph reasoning datasets and tasks, and will launch a LLM demo online with various graph reasoning abilities.

KEYWORDS

Tool Transformer; In-Context Learning; Language Model; Graph Learning

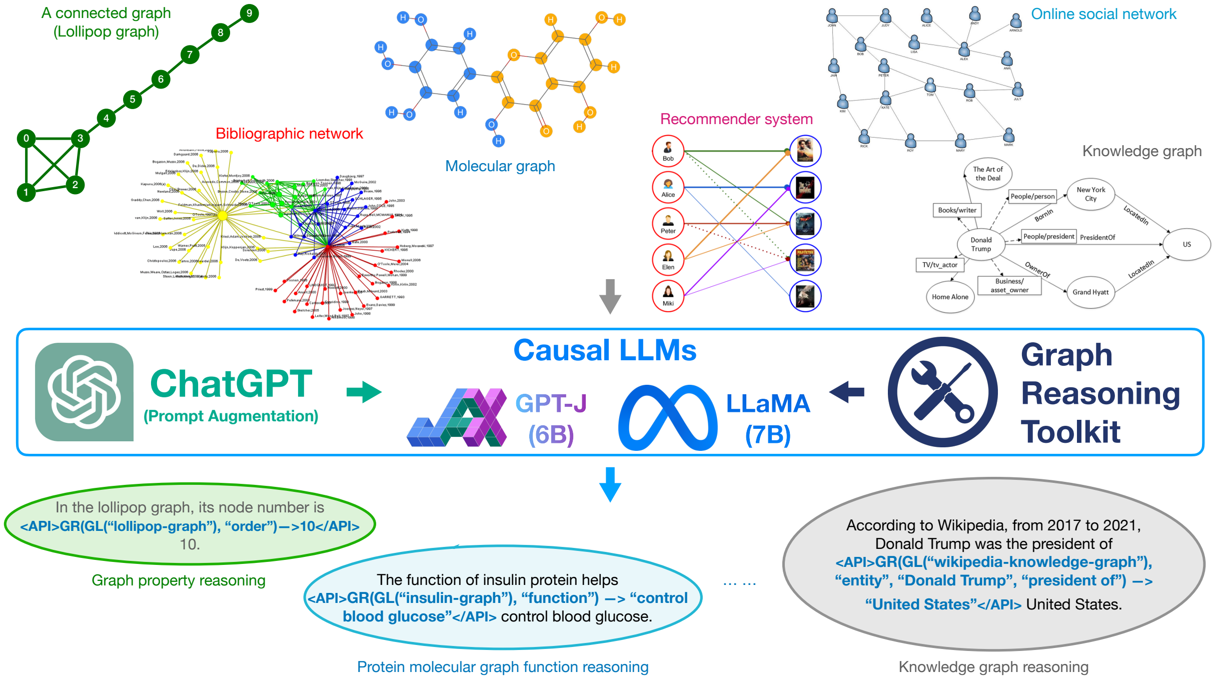
1 INTRODUCTION

In recent years, large language models (LLMs) [5, 26, 41] have achieved very impressive performance on a variety of natural language processing tasks [24, 26, 40], extensions of which have also been extensively applied to solve many other problems with data in different modalities [7, 24, 32, 33] as well. With the launch of ChatGPT and new Microsoft Bing Chat based on both GPT-3.5 and GPT-4, LLMs have also been widely used in people's daily production and life. At the same time, due to their inherent limitations, such LLMs have also received lots of criticisms in their usages due to their inherited weakness, like *inability in performing precise calculations* [28], *difficulty in addressing multi-step logic reasoning problems* [3], incapable to conduct spatial and topological reasoning [1], and *unawareness of progression of time* [6].

With the parallel development of natural language processing and computer vision, transformer based deep learning models on graph structured data has also received lots of attention from the community in recent years [12, 46, 48]. Graph provides a unified representation for many inter-connected data in the real-world, which models both the diverse attributes of the nodes and the extensive links connecting the nodes with each other. Besides the classic graph structures we learn in the *discrete math course*, as shown in Figure 1, lots of real-world data can also be modeled as graphs [36], like bibliographic networks [38], protein molecular graphs [43], recommender systems [21], online social networks [23], and knowledge graphs [14]. In addition to the transformer based deep models, various other graph neural network models have also been proposed for graph data representation learning by now [15, 37, 42], which have achieved outstanding performance on various graph learning tasks already.

Meanwhile, compared with the prosperous research explorations on incorporating vision and language data into LLMs for designing the ambitious AIGC and AGI development plan [25], it seems researchers have either "unintentionally" or "intentionally" ignored the widely existed graph data and don't seem to have any plans to include them into the LLMs building for achieving the AGI.

Here, we say researchers have "unintentionally" ignored graphs, since compared with texts and images that we deal with everyday, graph has long-time been merely used as an intermediate modeling data structure for real-world data and we normally have no very direct interactions with graph actually. It is naturally that people may mistakenly think graph should not be the focus at the current stage for creating AIGC and building the AGI systems. At the same time, we say researchers may have "intentionally" ignored graphs, since graph learning may involve (1) lots of precise mathematical



In the lollipop graph, its node number is <API>GR(GL(“lollipop-graph”), “order”)—>10</API> 10.

The function of insulin protein helps <API>GR(GL(“insulin-graph”), “function”) —> “control blood glucose”</API> control blood glucose.

According to Wikipedia, from 2017 to 2021, Donald Trump was the president of <API>GR(GL(“wikipedia-knowledge-graph”), “entity”, “Donald Trump”, “president of”) —> “United States”</API> United States.

Figure 1: An Illustration of LLMs based Graph Reasoning Tasks. (Based on the input graph data from various domains and a handful number of prompt examples with brief instructions, we propose to use ChatGPT to annotate and augment a large prompt dataset that contains graph reasoning API calls of external graph reasoning tools for each type of the input graph. The generated prompt dataset will be used to fine-tune the existing pre-trained LLMs, like GPT-J or LLaMA, to enable them to automatically include the API calls in the result.)

calculations of graph properties, (2) multi-hop logical reasoning through the links, (3) capturing the extensively connected graph spatial topology structures, and (4) sometimes we also need to handle the dynamics of graphs that are changing with time. Careful readers may have noticed that these requirements mentioned for graph learning actually hit the nail on the head, which exactly correspond to the weaknesses of the current LLMs we mentioned at the very beginning.

Regardless of the potential challenges ahead of us, “an AGI without graph reasoning ability will never be the AGI we may desire”. As a researcher who has been working on graph learning for more than ten years, we write this paper trying to incorporate graph data into LLMs for various graph reasoning tasks. On the one hand, we really hope the currently AI-leading companies like OpenAI, Microsoft and Google can take graph structured data reasoning into consideration when they develop their missions and plans for achieving the AGI, so that the graph learning community will be able to contribute our efforts to building the AGI system together with the language and vision communities. On the other hand, we hope to empower the existing LLMs with the ability to overcome the existing weaknesses in their performance when handling graph structured data for complex graph reasoning tasks. So, the latest developed LLMs can also benefit the graph learning community for solving various graph reasoning tasks as well.

Considering the current language models and their extremely high pre-training costs, we cannot fundamentally re-design a new LLM with pre-training to equip them with graph reasoning components without hurting their outstanding language generation performance. Pre-training such LLMs from scratch is an infeasible task for most research groups in academia and majority of companies in the industry as well. To adapt to the common practices of NLP approaches, we will introduce the Graph Reasoning oriented Toolformer framework (Graph-Toolformer) in this paper. Technically, as illustrated in Figure 1, based on the latest ChatGPT from OpenAI and Toolformer model from Meta [34], we propose to provide the existing pre-trained LLMs (*e.g.,* GPT-J or LLaMA) with the ability to perform various complex graph reasoning tasks by allowing them to use external graph learning tools, such as other pre-trained graph neural network models and existing graph reasoning approaches. Instead of manually hard-coding the graph data loading and external graph learning tool usage function calls in the sentences, to make Graph-Toolformer as a general graph reasoning interface, we will train the LLMs in a self-supervised manner to enable the model to decide not only *where* to retrieve the graph data, but also *what* tools to be used, as well as *when* and *how* to use the tools. More technical details about the Graph-Toolformer model will be introduced in the following methodology section.

As the first exploration attempt to use LLMs for general graph reasoning tasks, we summarize the contributions of this paper as follows:

- **Graph Reasoning with LLMs**: This paper is the first paper that attempts to propose a general LLM, *i.e.,* Graph-ToolFormer, that can handle graph reasoning tasks. It effectively remedies the weaknesses of existing LLMs on graph reasoning tasks. More importantly, it helps bridge the graph learning community with the latest development on LLMs and AIGC led by the language and vision learning communities.
- **Graph Oriented Language Dataset**: In this paper, we create a handful number of human-written language instructions and prompt examples of how graph learning tools can be used. Based on the self-supervised in-context learning, we use ChatGPT to annotate and augment a large graph reasoning dataset with API calls of different external graph learning tools, which will also be post-processed with selective filtering. We will release both the human-written examples and the generated graph reasoning dataset with the community for future explorations.
- **Extensive Experimental Testing**: We test the effectiveness of our proposed Graph-ToolFormer with various graph reasoning based application tasks studied in the real-world, which include the most basic graph data loading and general graph property computation of graphs, as well as some more challenging graph reasoning tasks, like paper topic inference in bibliographic networks, protein molecular graph function prediction, online social network community detection, personalized sequential product recommendation in e-commerce recommender systems and knowledge graph reasoning for conversations.

The remaining parts of this paper are organized as follows. We will introduce the related work in Section 2. A detailed information about the Graph-ToolFormer model will be introduced in Section 4, which will include several detailed steps for dataset annotation and model tuning. The effectiveness of Graph-ToolFormer will be tested with some preliminary experiments on real-world benchmark graph datasets in Section 5. Finally, we will conclude this paper in Section 6.

2 RELATED WORK

In this section, we will discuss about the existing works related to our Graph-ToolFormer framework proposed in this paper, which include *graph learning*, *language models* and *prompts for fine-tuning*.

2.1 Graph Learning

Representative examples of GNNs proposed by present include GCN [15] and Graph-Bert [48], based on which various extended models [16, 37, 42] have been introduced as well. As mentioned above, GCN and its variant models are all based on the approximated graph convolutional operator [10], which may lead to the suspended animation problem [47] and over-smoothing problem [18] for deep model architectures. Theoretic analyses of the reasons are provided in [9, 18, 47]. To handle such problems, [47] generalizes the graph raw residual terms and proposes a method based on graph residual learning; [18] proposes to adopt residual/dense connections and dilated convolutions into the GCN architecture. Several other works [13, 37] seek to involve the recurrent network for deep graph representation learning instead.

2.2 Language Models

Since the propose of Transformer [41], large language models (LLMs) have become the main stream of algorithms for various NLP tasks. Assisted with pre-training, different tech companies also introduce and release their own versions of the LLMs, like BERT from Google [5], BART from Facebook [17], GPT from OpenAI [2, 30, 31], ELMo from AI2 [29] and MT-DNN from Microsoft [20], many of them are also open-sourced and shared to the community for both research and application purposes. One research paper closely related to this work is Toolformer [34] from Meta AI, which proposes to incorporate external APIs into language models. Equipped with such external APIs, the models will be able to decide when and how to use which tool. Prior to Toolformer, several other paper [22, 27] has also explored to augment language models with external tools.

2.3 Prompts for Model Fine-Tuning

Prompts have been shown to be effective in tuning the pre-trained language models with zero-shot or few-shot learning manner [2], which can help language models learn faster than traditional fine tuning tasks. By now, we witnessed three categories of prompt tuning approaches, *i.e.,*, *discrete prompts* [35], *continuous prompts* [19] and *priming* [2]. Discrete prompts [35] reformat data instance with some template text, e.g., "*{ premise } Should we assume that { hypothesis }? [prediction]*", will typically tune all parameters of the model. On the other hand, continuous prompts [19] will prepend examples with embedding vectors of special tokens, which will only update a much smaller set of model parameters. Priming [2] initially adopted in GPT-3 will prepend several priming examples to the target evaluation example instead, e.g., "*{ sentence$_1$ } True or False? { label$_1$ }. $\cdots$ { sentence$_k$ } True or False? { label$_k$ }. { eval-sentence } True or False? [prediction]*". According to the analysis reported in [45], discrete prompts works very well in few-shot tuning, continuous prompts have not yet reported success in few-shot setting yet, while priming is very costly and seems only work for the largest GPT-3 (175B) model.

3 TERMINOLOGY DEFINITION AND PROBLEM FORMULATION

In this section, we will first introduce the notations used in this paper. After that, we will provide the definitions of several important terminologies and the formulation of problem studied in this paper.

3.1 Notations

In the sequel of this paper, we will use the lower case letters (e.g., x) to represent scalars, lower case bold letters (e.g., $\mathbf{x}$) to denote column vectors, bold-face upper case letters (e.g., $\mathbf{X}$) to denote matrices, and upper case calligraphic letters (e.g., $\mathcal{X}$) to denote sets or high-order tensors. Given a matrix $\mathbf{X}$, we denote $\mathbf{X}(i, :)$ and $\mathbf{X}(:, j)$ as its i_{th} row and j_{th} column, respectively. The (i_{th}, j_{th}) entry of matrix $\mathbf{X}$ can be denoted as either $\mathbf{X}(i, j)$. We use $\mathbf{X}^\top$ and $\mathbf{x}^\top$ to represent the transpose of matrix $\mathbf{X}$ and vector $\mathbf{x}$. For vector $\mathbf{x}$, we represent its

L_p-norm as $\|\mathbf{x}\|_p = (\sum_i |\mathbf{x}(i)|^p)^{\frac{1}{p}}$. The Frobenius-norm of matrix $\mathbf{X}$ is represented as $\|\mathbf{X}\|_F = (\sum_{i,j} |\mathbf{X}(i,j)|^2)^{\frac{1}{2}}$. The element-wise product of vectors $\mathbf{x}$ and $\mathbf{y}$ of the same dimension is represented as $\mathbf{x} \otimes \mathbf{y}$, whose concatenation is represented as $\mathbf{x} \sqcup \mathbf{y}$.

3.2 Terminology Definitions

In this paper, we will focus on the graph structured data reasoning tasks, and the graphs to be studied from different sources can have different structures and properties. In this part, we will provide the definition of the general representation of the graphs to be analyzed in this paper as follows.

Definition 1. *(Graph): Generally, the graph studied in this paper can be represented as $G = (\mathcal{V}, \mathcal{E})$. In the notation, $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ is the set of n nodes in the graph and $\mathcal{E} = \left\{e_{i,j} = (v_i, v_j)\right\}_{v_i, v_j \in \mathcal{V}}$ is the set of m links among these nodes, where $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$.*

Depending on the applications and domains, the graph data to be studied may carry different property and structural information. For instance, for some graph, the nodes may carry some feature and label information, which can be represented via mappings $x : \mathcal{V} \to \mathbb{R}^{d_x}$ and $y : \mathcal{V} \to \mathbb{R}^{d_y}$, respectively. For each node $v_i \in \mathcal{V}$, we can represent its features as $x(v_i) = \mathbf{x}_{v_i} \in \mathbb{R}^{d_x}$ and its one-hot label vector as $y(v_i) = \mathbf{y}_{v_i} \in \mathbb{R}^{d_y}$, where d_x and d_y denote the feature and label space dimensions, respectively. If there is also feature and labels attached to the links, we can also represent the corresponding feature and one-hot label vector of link $e_{i,j} \in \mathcal{E}$ in a similar way as $\mathbf{x}_{e_{i,j}} \in \mathbb{R}^{d_x}$ and $\mathbf{y}_{e_{i,j}} \in \mathbb{R}^{d_y}$, respectively.

For the graph data from many domains, like bibliographic network, online social network, recommender system, knowledge graph, there will exist one single large-scale graph in the dataset, but the graph may contain thousands, millions or even billions of nodes and links. Such large-scale graphs can be perfectly represented with the above definition. Meanwhile, for graphs from many other domains, like bio-chemical molecular graphs, there will exist a large number of much smaller graph instances, and each graph instance normally contain tens or a few hundred nodes and links instead. To represent the set of small-sized graph instances, we define the concept of graph set as follows.

Definition 2. *(Graph Set): For the randomly generated graph instances or the bio-chemical molecular graph instances, we can represent the set of graph instances in the dataset as $\mathcal{G} = \{g_1, g_2, \cdots, g_l\}$, where each graph instance $g_i = (\mathcal{V}_{g_i}, \mathcal{E}_{g_i})$ can be represented according to the above graph definition.*

For the graph dataset, each graph instance may also have its unique features and label information, denoting its topological properties and tags of the graph instance. Formally, for a graph instance $g_i \in \mathcal{G}$, we can represent its raw feature vector and one-hot label vector as $\mathbf{x}_{g_i} \in \mathbb{R}^{d_x}$ and $\mathbf{y}_{g_i} \in \mathbb{R}^{d_y}$, respectively.

3.3 Problem Formulation

In this paper, we aim to enable the existing pre-trained LLMs to carry out graph reasoning. As introduced before, the graph reasoning tasks studied in this paper include (1) *graph general property reasoning*, (2) *bibliographic paper topic reasoning*, (3) *protein molecular graph function reasoning*, (4) *recommender system reasoning*, (5) *online social network community reasoning*, and (6) *knowledge graph reasoning*. Specifically, these graph reasoning tasks studied in this paper are carefully selected, which can be categorized into several types of graph learning problems listed as follows:

- *Attribute Calculation*: For the tasks like *graph property reasoning*, we actually aim to calculate either explicit or implicit attributes of the input graph data, ranging from the simple *number of nodes/links* in the graph, to the *graph radius and diameter*, and more complex *graph periphery and node pairwise short path length.*
- *Node Classification*: For the *bibliographic paper topic reasoning*, we aim to predict the topic of the academic papers in the bibliographic network, which can be modeled as the node classification task actually. Via the raw features of the target paper node and nearby neighbor nodes, we can classify the paper into different classes, each of which corresponds to one specific topic.
- *Graph Classification*: For the *protein molecular graph function reasoning*, based on the molecular graph structures, we aim to infer the potential functions of the protein, which can be defined as the graph instance classification task. Via both the protein graph structure and raw attributes, we can classify the graph instance into different classes, each of which correspond to different pre-defined protein functions.
- *Link Prediction*: For the *sequential recommender system reasoning*, we aim to infer the potential preference of users towards certain items in the system, which can be defined as the temporal link prediction task (connecting user and item with a timestamp) in graph learning. Depending on the recommender system settings, we can predict either the link existence label denoting if the user will buy the item or the potential link weight denoting the score the user will rate for the item.
- *Graph Clustering*: For the *online social network community reasoning*, we aim to infer the community structures in online social networks, which can be defined as the graph clustering task. Based on the user social interaction patterns, we want to partition the online social network graphs into different clusters, each of which denote one community detected from the social network.
- *Graph Searching*: For the *knowledge graph reasoning*, we aim to infer the potential entities or relations that can meet the input requirements, which can be modeled as the graph searching problem. Starting from the input entity or relation, we aim to expand and search for the related entities or relations for generating the output.

To address these graph reasoning tasks with LLMs, we propose to include the API calls of external graph learning tools into to the communication text seamlessly. Based on the above notations, we will design a set of graph reasoning related API calls for various graph reasoning tasks in the real-world. Such API calls will include both the external graph learning tool name and the parameters, which will be surrounded with special tokens to differentiate from regular text, specifically. Based on a handful human-written examples, we will generate a large language modeling dataset containing

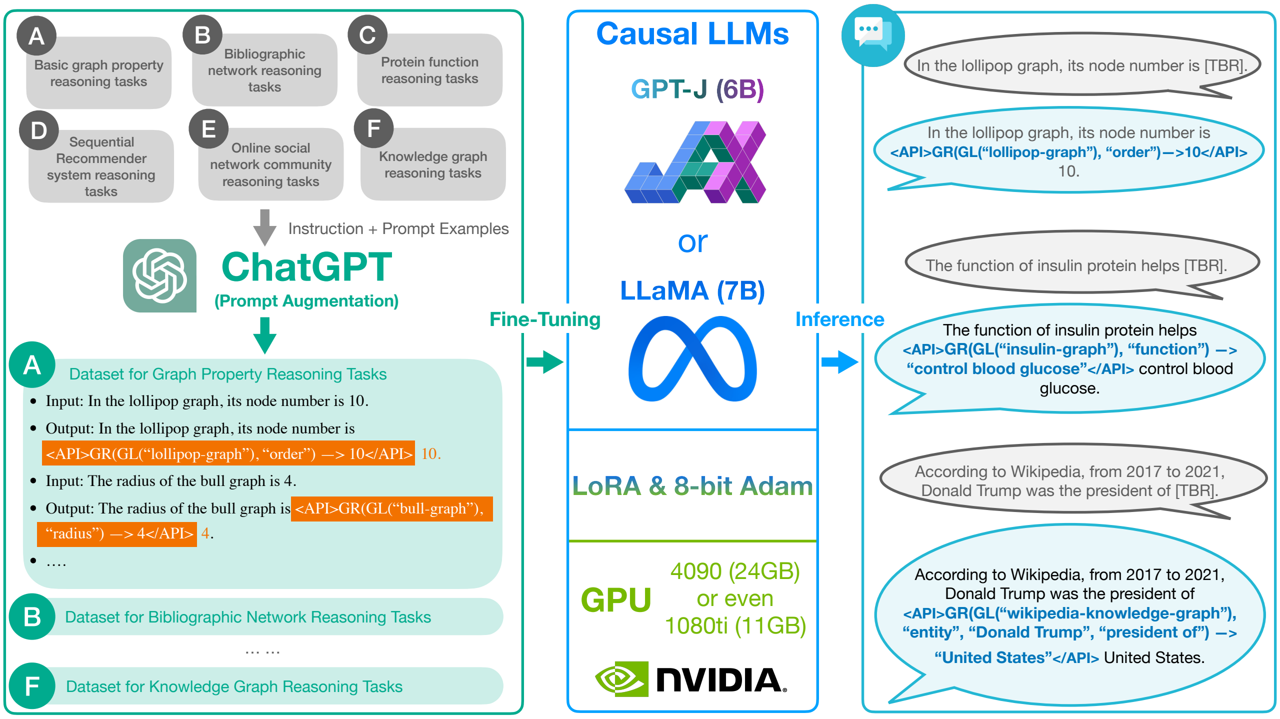

A Basic graph property reasoning tasks
B Bibliographic network reasoning tasks
C Protein function reasoning tasks
D Sequential Recommender system reasoning tasks
E Online social network community reasoning tasks
F Knowledge graph reasoning tasks
Instruction + Prompt Examples
ChatGPT
(Prompt Augmentation)
A Dataset for Graph Property Reasoning Tasks
Input: In the lollipop graph, its node number is 10.
Output: In the lollipop graph, its node number is <API>GR(GL("lollipop-graph"), "order") —> 10</API> 10.
Input: The radius of the bull graph is 4.
Output: The radius of the bull graph is <API>GR(GL("bull-graph"), "radius") —> 4</API> 4.
….
B Dataset for Bibliographic Network Reasoning Tasks
… …
F Dataset for Knowledge Graph Reasoning Tasks
Fine-Tuning
Causal LLMs
GPT-J (6B)
or
LLaMA (7B)
LoRA & 8-bit Adam
GPU
4090 (24GB) or even 1080ti (11GB)
NVIDIA
Inference
In the lollipop graph, its node number is [TBR].
In the lollipop graph, its node number is <API>GR(GL("lollipop-graph"), "order")—>10</API> 10.
The function of insulin protein helps [TBR].
The function of insulin protein helps <API>GR(GL("insulin-graph"), "function") —> "control blood glucose"</API> control blood glucose.
According to Wikipedia, from 2017 to 2021, Donald Trump was the president of [TBR].
According to Wikipedia, from 2017 to 2021, Donald Trump was the president of <API>GR(GL("wikipedia-knowledge-graph"), "entity", "Donald Trump", "president of") —> "United States"</API> United States.

Figure 2: The Outline of the Graph-ToolFormer Framework (the framework has three main parts: (1) prompt data annotation and augmentation with ChatGPT, (2) existing pre-trained causal LLMs fine-tuning with the generated prompt dataset, and (3) inference of the fine-tuned model for adding graph reasoning API calls into statements).

such API calls with ChatGPT, which will be used for fine-tuning the LLMs, like GPT-J (6B) and LLaMA (7B). Such LLMs studied in this paper are all pre-trained and we will only fine-tune them with the generated prompt datasets, which ensures that the model doesn't lose any language generation and modeling ability.

4 PROPOSED METHOD

4.1 Framework Outline

At the beginning of this section, we would like to provide an outline of the Graph-ToolFormer framework in Figure 2 to illustrate how Graph-ToolFormer works. According to the framework outline, based on the hand-crafted instructions and a handful number of prompt examples, we use ChatGPT to annotate and augment a large dataset about graph reasoning API call instances. With the generated prompt dataset, we will fine-tune existing pre-trained causal LLMs, such as GPT-J (6B) [8, 44] and LLaMA (7B) [40] to teach them how to use the external graph reasoning tools. With both LoRA (Low-Rank Adaptation) [11] and 8-bit Adam and quantization techniques [4], Graph-ToolFormer can be fine-tuned on GPUs with small memory space, such as Nvidia GeForce RTX 4090 (24GB RAM) and even Nvidia GeForce RTX 1080Ti (11GB RAM). The fine-tuned Graph-ToolFormer can be used for inference purposes: given the input statements, Graph-ToolFormer will add the corresponding graph reasoning API calls into statements at the most appropriate positions automatically.

In this section, we will introduce the detailed information about the aforementioned components included in the Graph-ToolFormer framework. Specifically, we will first introduce the representation of API calls, and talk about the various graph reasoning tasks studied in this paper, which can all be represented as the API calls of external graph learning tools. Then, we will introduce the approach applied to augment the plain text dataset with the API calls for various graph reasoning via the conversation with ChatGPT for prompt annotation and augmentation. Such augmented prompt datasets will be post-processed with selective filtering. Finally, we will discuss about how to use the augmented dataset to further fine-tune the existing pre-trained LLMs.

4.2 Prompts with API Calls

In this paper, we can represent the API calls of external graph learning tools as $f(args)$, where f is the external tool function name and $args$ denotes the list of parameters of the function. For representation simplicity, we can also represent such an API call as a tuple notation $c = (f, args)$, which will be frequently used in the following part of this paper. Instead of merely generating the external API calls as the output, we propose to inset the API calls into the generated text instead, which enables the LLMs to handle the graph reasoning tasks via regular conversations.

Formally, to insert the API calls into text, we can represent the sequence of tokens for API call $c = (f, args)$ as

$$\mathbf{s}(c) = \text{<API>} f(args) \text{</API>}, \tag{1}$$

Table 1: A summary of API call examples for basic graph Loading and property reasoning studied in this paper. In this table, we use notations $GL(\cdot)$ and $GR(\cdot)$ to represent the graph loading and graph reasoning API calls. Without introducing new special tokens to the pre-trained tokenizer of LLMs, we use "[", "]" and "–>" to represent the "<API>", "</API>" and "→" tokens introduced in this paper. Notation "[TBR]" denotes the "to be reasoned" placeholder token. We refer to the top left Lollipop graph (in green color) illustrated in Figure 1 as the "Lollipop graph" example in the table. In the graph property reasoning API call notations, we use G_l to represent the result ob API call "[GL(file-path:"./graphs/lollipop") → G_l]".

Tasks	API Call Templates	Prompt Examples	
		Inputs	Outputs
Graph Data Loading	$GL(\textit{file-path})$	"The structure of the molecular graph of the benzene ring contains a hexagon."	"The structure of the [GL(file-path="./graphs/benzene-ring")] molecular graph of the benzene ring contains a hexagon."
	$GL(\textit{file-path, node-subset, link-subset})$	"There exist a carbon-oxygen double bond in the Acetaldehyde molecular graph."	"There exist a [GL(file-path="./graphs/acetaldehyde" link-subse={C=O})] carbon-oxygen double bond in the Acetaldehyde molecular graph."
	$GL(\textit{file-path}) \to r$	"Lollipop graph look like a spoon."	"[GL(file-path:"./graphs/lollipop") –> G_l] Lollipop graph look like a spoon."
Graph Property Reasoning	$GR(graph, \text{"}order\text{"}) \to r$	"There exist [TBR] nodes in the lollipop graph."	"There exist [GR(G_l, "order") –> 10] nodes in the lollipop graph."
	$GR(graph, \text{"}size\text{"}) \to r$	"Via [TBR] links, nodes in the lollipop graph are all connected."	"Via [GR(G_l, "size") –> 12] links, nodes in the example lollipop graph are all connected."
	$GR(graph, \text{"}density\text{"}, is\text{-}directed) \to r$	"The undirected lollipop graph has a density of $\frac{4}{15}$."	"The undirected lollipop graph has a density of [GR(G_l, "density", *is-directed*=False) –> $\frac{4}{15}$]."
	$GR(graph, \text{"}eccentricity\text{"}) \to r$	"The long 'tail' will lead to large eccentricity [TBR] for many nodes in the lollipop graph."	"The long 'tail' will lead to large eccentricity [GR(G_l, "eccentricity") –> $\{0:7, 1:7, 2:7, 3:6, 4:5, 5:4, 6:4, 7:5, 8:6, 9:7\}$] for many nodes in the lollipop graph."
	$GR(graph, \text{"}eccentricity\text{"}, \text{node-subset}) \to r$	"The eccentricity of node #4 in the lollipop graph is [TBR]."	"The eccentricity of node #4 in the lollipop graph is [GR(G_l, "eccentricity", node #4) –> 5]."
	$GR(graph, \text{"}radius\text{"}) \to r$	"The radius of the lollipop graph is [TBR]."	"The radius of the lollipop graph is [GR(G_l, "radius") –> 4]."
	$GR(graph, \text{"}center\text{"}) \to r$	"The center of the lollipop graph include node(s) [TBR]."	"The center of the lollipop graph include node(s) [GR(G_l, "center") –> $\{5, 6\}$]."
	$GR(graph, \text{"}shortest\text{-}path\text{"}, node_1, node_2) \to r$	"In the lollipop graph, the length of shortest path between node #1 and node #5 is [TBR]."	"In the lollipop graph, the length of shortest path between node #1 and node #5 is [GR(G_l, "shortest-path", node #1, node #5) –> 3]."
	$GR(graph, \text{"}avg\text{-}shortest\text{-}path\text{"}) \to r$	"The average length of shortest path for all nodes in the lollipop graph is [TBR]."	"The average length of shortest path for all nodes in the lollipop graph is [GR(G_l, "avg-shortest-path") –> 2.86]."
	$GR(graph, \text{"}diameter\text{"}) \to r$	"The diameter of the lollipop graph is [TBR] due to the long 'tail'."	"The diameter of the lollipop graph is [GR(G_l, "diameter") –> 7] due to the long 'tail'."
	$GR(graph, \text{"}periphery\text{"}) \to r$	"The periphery of the lollipop graph includes the nodes [TBR]."	"The periphery of the lollipop graph includes the nodes [GR(G_l, "periphery") –> $\{0, 1, 2, 9\}$]."

or

$$\mathbf{s}(c, r) = \text{<API>} f(args) \to r \text{</API>}, \quad (2)$$

where both "<API>" and "</API>" surrounding the API call function are the special tokens to differentiate it from other tokens in the generated text. For the Graph-ToolFormer model, when it generates the "<API>" and "</API>" tokens, the model will recognize that the tokens inside it denotes the API function call. As to the second API call representation, the term r denotes the return result of $f(args)$. Depends on both the function $f(args)$ and the contexts for the API call in the reasoning task, the LLMs to be fine-tuned later will automatically decide whether the output results are needed or not. We will discuss more about it later as we introduce the reasoning tasks and the prompt templates to be used for these reasoning tasks.

Different from the simple API calls studied in [34], in graph reasoning, some of the API calls may involve a complicated and nested calls of various external functions. Later on, when we discuss about the specific graph reasoning tasks, we will encounter some of such cases, *e.g.*, some of the parameters in one API call can actually be the return result of another API call, or we may need to call multiple sequential APIs concurrently for accomplishing one task. To address such complicated graph reasoning scenarios, in this paper, we will also allow Graph-ToolFormer to generated nested and sequential API calls surrounded by the special tokens "<API>" and "</API>".

For instance, given two API calls $c_1 = (f_1, args_1)$ and $c_2 = (f_2, args_2)$ with their own input parameters, the first API function f_1 needs to use the return result of the second API function f_2 as its input parameter, we can represent such nested API calls as

$$\mathbf{s}(c_1|c_2) = \text{<API>} f_1(args_1 = f_2(args_2)) \text{</API>}, \quad (3)$$

or just simply as

$$\mathbf{s}(c_1|c_2) = \text{<API>} f_1(f_2(args_2)) \text{</API>}, \quad (4)$$

where the notation $c_1|c_2$ denotes these two API calls are nested.

Meanwhile, if a task may need to call multiple sequential APIs simultaneously, *e.g.*, $c_1 = (f_1, args_1)$ and $c_2 = (f_2, args_2)$, we can represent such sequential API calls as

$$\mathbf{s}(c_1, c_2) = \text{<API>} f_1(args_1), f_2(args_2)\text{</API>} \quad (5)$$

$$= \text{<API>} f_1(args_1)\text{</API>}, \text{<API>} f_2(args_2)\text{</API>} \quad (6)$$

$$= \mathbf{s}(c_1), \mathbf{s}(c_2), \quad (7)$$

which is equivalent to two sequential API calls of c_1 and c_2 as well.

For more complicated cases, we can rewrite the above API call representations with either more input parameter denoted by other API calls or deeply nested API calls, *e.g.,*

$$\mathbf{s}\left(c_1 | (c_2, c_3)\right) = \text{<API>} f_1\left(f_2(args_2), f_3(args_3)\right)\text{</API>}, \quad (8)$$

or

$$\mathbf{s}\left(c_1 | (c_2 | c_3)\right) = \text{<API>} f_1\left(f_2\left(f_3(args_3)\right)\right)\text{</API>}, \quad (9)$$

where $c_3 = (f_3, args_3)$ denote a third API call notation.

Such graph reasoning function API calls will be inserted into statements for LLMs fine-tuning later. Without modifying the LLMs' vocabulary set and the pre-trained tokenizer, in implementation, we can replace the special tokens "<API>", "</API>" and → with some less frequently used tokens like "[", "]" and "->" instead. In this paper, we will study several very different graph reasoning tasks involving diverse graph learning API calls, which will be introduced in detail in the following subsection for readers.

4.3 Graph Reasoning Tasks and API Calls

In this paper, we will study several graph reasoning tasks with Graph-ToolFormer, including both the very basic ones, like the general graph property reasoning, and more advanced ones, like the reasoning tasks on graphs from different specific domains. As introduced before in Section 3, these reasoning tasks are all carefully selected, which can be categorized into different types of graph learning tasks, *e.g., graph attribute calculation*, *node classification*, and *graph classification*, etc. All of these reasoning tasks have extensive applications in real-world graph data reasoning tasks. Besides the ones studied in this paper, with minor changes to the API call representations, we can also apply the Graph-ToolFormer to other graph reasoning related application tasks as well.

4.3.1 Graph Data Loading. Different from text and images, the graph data we have in the real-world may have relatively larger sizes, extensively connected structures and complex raw attributes. Except for some small-sized hand-crafted graph examples, it is almost impossible to type in the graph structured data as a sequence of token inputs to LLMs. Therefore, in this paper, we propose to empower the Graph-ToolFormer model with the ability to automatically load the desired graph data from offline files or online repositories based on the provided the dataset name, local file path or online repository URL link.

Technically, the first API call that we will introduce in this paper is for graph data loading, which can load either the whole graph or just a subgraph involving one or a few nodes and links. Specifically, we can represent the graph loading API call as

$$\text{<API>} GL(\textit{file-path}, \textit{node-subset}, \textit{link-subset}) \rightarrow G\text{</API>}, \quad (10)$$

where "$GL()$" denotes abbreviation of the "Graph Loading" function name, and the function parameters "*file-path*", "*node-subset*" and "*link-subset*" specify the local graph data file path (or the online repository URL if the data is stored on the web), subset of specific nodes and links, respectively. The notation "→ G" explicitly represents the loaded graph data with the reference variable $G = (\mathcal{V}, \mathcal{E})$, which is optional actually depending on the application task and settings. What's more, if the local file directory or the online root repository has been pre-provided in the "$GL()$" function already, then we can just assign the "*file-path*" with the specific "graph name" instead when calling this API function.

Furthermore, when the parameters "*node-subset*" and "*link-subset*" are either omitted or assigned with the strings "all nodes" and "all links", respectively, then the API function call will just load the complete graph. For some cases, we can only specify the subset of nodes to be loaded (*e.g.,* $\{v_i, v_j, \cdots, v_k\} \subset \mathcal{V}$ in the graph) but cannot enumerate all the related links, we can just assign the "*node-subset*" and "*link-subset*" parameters with values "$\{v_i, v_j, \cdots, v_k\}$" and "all related links" (or the "*link-subset*" parameter is just omitted). It will provide us with more flexibility in loading sub-graphs based on the provided node set and their internal links. Similarly, we can also only specify the subset of links, by assigning the "*node-subset*" with "all related nodes" or just omitted, it will automatically load the nodes composing those provided links in the graph data as well.

As shown in Table 1, we also provide several prompt examples of the graph data loading API calls, which can retrieve and load the requested data from the files according to the input textual statements.

4.3.2 General Graph Property Reasoning. Graph structured data may have various properties, such as *diameter*, *density* and *center*, which can capture the characteristics of the graph connection structures. For reasoning such graph properties, it usually require the model to not only know the property definition but also have very strong logic reasoning and mathematical calculation ability. In this paper, to empower LLMs with the graph property reasoning ability, we introduce a group of external API calls, which can be used by the models in the conversation. To illustrate the graph properties discussed in this part, we will use the small-sized lollipop graph shown in Figure 1 (the top-left graph in green color) as an example for the property calculation, which can be loaded via the following API calls as introduced before:

$$\text{<API>} GL(\text{“}./graphs/lollipop\text{”}) \rightarrow G_l\text{</API>}, \quad (11)$$

where the loaded the graph can also be referred to by G_l. In the following graph property reasoning API calls, we also need to load graphs from the files as the input, where the graph loading API calls will be nested. In the following discussions, we will use the lollipop graph G_l as an example to introduce the graph property reasoning APIs. To simplify the API call representations, we can just use the loaded graph reference, *i.e.,* G_l, to replace the graph loading API call <API>load("./graphs/lollipop.txt")</API>.

Order and Size: Formally, given a loaded lollipop graph $G_l = (\mathcal{V}, \mathcal{E})$, its *order* denotes the number of nodes in the graph, *i.e.,* $|\mathcal{V}|$, and its *size* is the number of links in the graph, *i.e.,*, $|\mathcal{E}|$. We can represent the API calls for reasoning the *order* and *size* properties of the lollipop graph as

$$\text{<API>} GR(GL(\text{“}./graphs/lollipop\text{”}), \text{“}order\text{”}) \rightarrow r\text{</API>}, \quad (12)$$

$$\text{<API>} GR(GL(\text{“}./graphs/lollipop\text{”}), \text{“}size\text{”}) \rightarrow r\text{</API>}. \quad (13)$$

If the lollipop graph has been pre-loaded via other API calls already and can be referred to as G_l, the above API calls can also be simplified as follows:

$$\text{<API>}GR(G_l, \text{“}order\text{”}) \to r\text{</API>}, \tag{14}$$

$$\text{<API>}GR(G_l, \text{“}size\text{”}) \to r\text{</API>}, \tag{15}$$

where the notation $GR()$ denotes the abbreviated “Graph Reasoning” function name and the parameters “order” and “size” specify the graph properties to be reasoned. The notation “$\to$ r” specifies the output result r by the graph property reasoning API call, and returning the output result of the API calls in the generation result or not is optional, which depends on both the reasoning setting and application tasks as discussed before.

Density: Graph *density* denotes the ratio of existing links in a graph compared with the maximal number of links in a graph. If the input lollipop graph $G_l = (\mathcal{V}, \mathcal{E})$ is *directed*, its *density* can be represented as $\frac{|\mathcal{E}|}{|\mathcal{V}|(|\mathcal{V}|-1)}$; while if G_l is *undirected*, its *density* can be represented as $\frac{2|\mathcal{E}|}{|\mathcal{V}|(|\mathcal{V}|-1)}$. Formally, the API calls that can be used for computing the density of graph can be as

$$\text{<API>}GR(G_l, \text{“}density\text{”}, is\text{-}directed) \to r\text{</API>}, \tag{16}$$

where the boolean “is-directed” parameter differentiates directed graph from undirected ones in the density property reasoning.

Eccentricity: Given the connected lollipop graph $G_l = (\mathcal{V}, \mathcal{E})$, for node $v_i \in \mathcal{V}$, its *eccentricity* denotes the maximum graph distance between v_i and any other node $v_j \in \mathcal{V}$ in graph G_l. For disconnected graph, all nodes are defined to have infinite *eccentricity*. We can compute the *eccentricity* for both the graph and for specific nodes via the following two API calls:

$$\text{<API>}GR(G_l, \text{“}eccentricity\text{”}) \to r\text{</API>}, \tag{17}$$

$$\text{<API>}GR(G_l, \text{“}eccentricity\text{”}, \text{node-subset}) \to r\text{</API>}. \tag{18}$$

Radius: Graph *radius* denotes the is the minimum graph *eccentricity* of any node in a graph. A disconnected graph therefore has infinite radius. The API call for computing a graph *radius* can be represented as

$$\text{<API>}GR(G_l, \text{“}radius\text{”}) \to r\text{</API>}. \tag{19}$$

Center: Formally, the *center* of a graph denotes the set of nodes whose eccentricity is equal to the graph radius. The API call for reasoning a graph *center* can be represented as

$$\text{<API>}GR(G_l, \text{“}center\text{”}) \to r\text{</API>}. \tag{20}$$

Shortest Path: A *shortest path* between two nodes in a graph is a path of shortest possible length connecting them via the nodes and links in the graph. The API call for reasoning the length of the *shortest path* from $node_1$ to $node_2$ in a graph can be represented as

$$\text{<API>}GR(G_l, \text{“}shortest\text{-}path\text{”}, node_1, node_2) \to r\text{</API>}. \tag{21}$$

Meanwhile, the average length of *shortest path* for all nodes in the graph can be obtained via the following API call instead

$$\text{<API>}GR(G_l, \text{“}avg\text{-}shortest\text{-}path\text{”}) \to r\text{</API>}. \tag{22}$$

Besides the average *shortest path length*, we can also reason for the maximum *shortest path length* and minimum *shortest path length* of a graph as follows:

$$\text{<API>}GR(G_l, \text{“}max\text{-}shortest\text{-}path\text{”}) \to r\text{</API>}, \tag{23}$$

$$\text{<API>}GR(G_l, \text{“}min\text{-}shortest\text{-}path\text{”}) \to r\text{</API>}. \tag{24}$$

Diameter: The *diameter* of a graph denotes the “longest shortest path” between any two nodes in the graph, whose API call can be represented as

$$\text{<API>}GR(G_l, \text{“}diameter\text{”}) \to r\text{</API>}, \tag{25}$$

whose result will be equal to the result of the above API call <API>$GR(G_l, \text{“}max\text{-}shortest\text{-}path\text{”})$</API> actually.

Periphery: The *periphery* of a graph is the subgraph of the graph induced by nodes that have graph *eccentricities* equal to the graph diameter, whose API call can be denoted as

$$\text{<API>}GR(G_l, \text{“}periphery\text{”}) \to r\text{</API>}. \tag{26}$$

A Summary of Basic Graph Reasoning API Calls: According to our descriptions above, the readers should have observe that those graph properties may require very complex logic reasoning, and the current LLMs (such as ChatGPT and LLaMA) cannot handle them very well (we will also evaluate their performance in the experimental section later). To incorporate them into language models, we also show some examples of the above API calls in Table 1, which can load the graph data from specified data sources and conduct the reasoning of some general graph properties as discussed above.

4.3.3 Advanced Graph Reasoning Tasks. Besides the general graph property reasoning, we will also study several more advanced reasoning tasks on real-world complex graph data with more complex structures in this paper, which include (1) *academic paper topic reasoning in bibliographic network*, (2) *protein function reasoning based on protein graph structures*, (2) *sequential product recommendation reasoning based on recommender system graphs*, (4) *social community reasoning in online social networks* and (5) *semantic reasoning on knowledge graphs*. These tasks included in this paper are carefully selected and they generally cover different graph learning problem settings, including (1) *node classification*, (2) *graph classification*, (3) *link prediction*, (4) *graph clustering* and (5) *graph searching*, respectively.

For other graph reasoning tasks not studied in this paper, they can be incorporated into Graph-ToolFormer as well, just with very minor changes by adding the corresponding API calls and necessary language model fine-tuning. The Graph-ToolFormer framework can serve as the backbone for hosting various graph reasoning application tasks with LLMs as the interface. We will provide such research and application opportunities for the readers to investigate in their future research projects.

Bibliographic Paper Topic Reasoning: Bibliographic network [38] defines a complex graph structured data representation, which involves diverse entities, such as academic papers, authors, institutions and publication venues, as well as the diverse links among

Table 2: A summary of API call examples for advanced graph reasoning tasks studied in this paper. In this table, we use notations $GL(\cdot)$ and $GR(\cdot)$ to represent the graph loading and graph reasoning API calls. Without introducing new special tokens to the pre-trained tokenizer of LLMs, we use "[", "]" and "–>" to represent the "<API>", "</API>" and "→" tokens introduced in this paper. Notation "[TBR]" denotes the "to be reasoned" placeholder token.

Tasks	API Call Templates	Prompt Examples	
		Inputs	**Outputs**
Bibliographic Paper Topic Reasoning	$GR(graph, \text{“}topic\text{”}, paper\text{-}node) \rightarrow r$	In the core bibliographic network, paper #31366 focuses on the topic of [TBR].	In the core bibliographic network, paper #31366 focuses on the topic of [GR(GL("cora"), "topic", paper#31366)−>Neural Networks].
		Within cora, paper #13195 is dedicated to the study of [TBR].	Within cora, paper #13195 is dedicated to the study of [GR(GL("cora"), "topic", paper#13195)−>Reinforcement Learning].
		The citeseer bibliographic network's paper #2 is concerned with the area of [TBR].	The citeseer bibliographic network's paper #2 is concerned with the area of [GR(GL("citeseer"), "topic", paper#2)−>Agents].
		Paper #3 in the citeseer network investigates the field of [TBR].	Paper #3 in the citeseer network investigates the field of [GR(GL("citeseer"), "topic", paper#3)−>DB].
		Paper #7, situated in the pubmed bibliographic network, is centered around the [TBR] topic.	Paper #7, situated in the pubmed bibliographic network, is centered around the [GR(GL("pubmed"), "topic", paper#7)−>1] topic.
Protein Function Reasoning	$GR(graph, \text{“}protein\text{-}function\text{”}, g_i) \rightarrow r$	The protein molecular graph instance #63 in the PROTEIN dataset has a function of [TBR] for the disease.	The protein molecular graph instance #63 in the PROTEIN dataset has a function of [GR(GL("PROTEIN"), "function", instance#63)−>0] for the disease.
		In PROTEIN, instance #985 of the protein molecular graph demonstrates a function of [TBR] for the disease.	In PROTEIN, instance #985 of the protein molecular graph demonstrates a function of [GR(GL("PROTEIN"), "function", instance#63)−>1] for the disease.
		The chemical molecular graph numbered 63 in PTC is characterized by a function of [TBR].	The chemical molecular graph numbered 63 in PTC is characterized by a function of [GR(GL("PTC"), "function", instance#63)−>1].
		For chemical molecular graph instance #63 in NCI1, its function is [TBR].	For chemical molecular graph instance #63 in NCI1, its function is [GR(GL("NCI1"), "function", instance#63)−>0].
		The molecular graph of chemical compound #121 in MUTAG possesses a function of [TBR].	The molecular graph of chemical compound #121 in MUTAG possesses a function of [GR(GL("MUTAG"), "function", instance#121)−>2].
Sequential Recommender System Reasoning	$GR(graph, \text{“}recommendation\text{”}, u_j, i_l) \rightarrow r$	In the Amazon recommender system, user #A240ORQ2LF9LUI rates item #0077613252 with a score of [TBR].	In the Amazon recommender system, user #A240ORQ2LF9LUI rates item #0077613252 with a score of [GR(GL("amazon"), "recommendation", user#A240ORQ2LF9LUI, item#0077613252)−>4.0].
		Within Last.fm, user #2 awards item #52 with a [TBR] tag.	Within Last.fm, user #2 awards item #52 with a [GR(GL("last.fm"), "recommendation", user#2, item#52)−>41] tag.
		User #196 gives a rating of [TBR] to item #251 at MovieLens.	User #196 gives a rating of [GR(GL("movielens"), "recommendation", user#196, item#251)−>3] to item #251 at MovieLens.
Online Social Network Reasoning	$GR(graph, \text{“}community\text{”}) \rightarrow r$	In the academic collaboration network dblp, scholar #355233 is involved in [TBR] local community formed by his/her collaborators.	In the academic collaboration network dblp, scholar #355233 is involved in [GR(GL("dblp"), "community-count", scholar#355233)−>6] local community formed by his/her collaborators.
		In the email communication social network, there exist a number of [TBR] local communities formed by users.	In the email communication social network, there exist a number of [GR(GL("email"), "community-count")−>42] local communities formed by users.
		The video sharing social network youtube houses the largest user-formed local community, which consists of [TBR] users.	The video sharing social network youtube houses the largest user-formed local community, which consists of [GR(GL("youtube"), "max-community-size")−>3001] users.
Knowledge Graph Reasoning	$GR(graph, reasoning\text{-}type, inputs) \rightarrow r$	According to the Freebase knowledge graph, the relation between entity /m/027rn and entity /m/06cx9 is [TBR].	According to the Freebase knowledge graph, the relation between entity /m/027rn and entity /m/06cx9 is [GR(GL("freebase"), "relation", entity:/m/027rn, entity:/m/06cx9)−>/location/country/form_of_government].
		According to the WordNet knowledge graph, from entity plaything.n.01, via relation _hyponym, we can derive entity [TBR].	According to the WordNet knowledge graph, from entity plaything.n.01, via relation _hyponym, we can derive entity [GR(GL("freebase"), "entity", entity:plaything.n.01, relation:_hyponym)−>swing.n.02].

these entities, such as the citation links, authorship links, affiliation links and publication links. In this part, we will discuss about the academic paper topic reasoning task based on the bibliographic network. The topics of a paper can be inferred with not only its own textual descriptions but also the other papers cited by/citing it, which requires the graph reasoning model to utilize both the raw textual features of the papers and the extensive citation links among the papers.

Formally, based on the terminology definition provided in the previous Section 3.2, we can represent the *bibliographic network* as $G = (\mathcal{V}, \mathcal{E})$, which can be loaded via API call

$$\text{<API>}GL(\text{“}./graphs/bibliographic\text{-}network\text{”}) \to G\text{</API>}. \quad (27)$$

Each paper node $v_i \in \mathcal{V}$ in the bibliographic network has both its raw feature vector $\mathbf{x}_{v_i}$ and label vector $\mathbf{y}_{v_i}$. The raw feature vector includes the textual information about the paper, and its label vector indicates the topics of the paper. Existing graph neural networks (GNN) infer the paper topics by learning their representations with both raw features and connected neighbors' information, which can be further used to infer the topic label vector. Therefore, we can represent the paper topic reasoning via graph neural network model with the following API call:

$$\text{<API>}GR(G, \text{“}topic\text{”}, paper\text{-}node) \to r\text{</API>}. \quad (28)$$

The function notation “$GR(\cdot, \text{“}topic\text{”}, \cdot)$” denotes it is a paper topic reasoning API, which will call the external pre-trained graph neural network models (or other toolkit functions) to reason the topics of the input paper node.

Protein Molecule Function Reasoning: Protein function inference [43] has been a classic problem studied in bio-chemical research for decades, which has fundamental applications in the real-world, such as helping design some new drugs for curing some existing rate diseases. Protein function inference is not an easy task, because homologous proteins often have different functions. Also such a prediction needs to be fine-tuned with respect to some mutations but robust with respect to others. Researchers have been exploring on this problem for many years, and have also developed a relatively large protein function database [39]. However, compared with the number of protein existing in the real world, the specific proteins with known functions included in the database is still very limited. In graph learning, inferring the function of protein molecules based on its structure has also be extensively studied as well. Therefore, in this part, we also include it as a graph reasoning task into LLMs.

Different from the *bibliographic network*, the *protein molecular graphs* have much smaller sizes and there will exist multiple such graph instances in the dataset. What's more, the features and protein function label information of *protein molecular graphs* is also normally about the whole molecular graph, not the individual nodes. As introduced in Section 3.2, we can represent the set of studied *protein molecular graphs* as $\mathcal{G} = \{g_1, g_2, \cdots, g_l\}$, which can be loaded with the following graph loading API call:

$$\text{<API>}GL(\text{“}./graphs/protein\text{-}graph\text{-}set\text{”}) \to \mathcal{G}\text{</API>}. \quad (29)$$

For each $g_i = (\mathcal{V}_{g_i}, \mathcal{E}_{g_i})$ in the dataset $\mathcal{G}$, it denotes an individual protein molecular graph instance. There will also be raw features and labels related to each protein molecular graph instance. For instance, for each protein graph instance $g_i \in \mathcal{G}$, we can represent its raw feature as $\mathbf{x}_{g_i}$ and its label as $\mathbf{y}_{g_i}$, where the label vector also indicate its corresponding protein functions. Based on the protein graph structure and its raw features, we can define the following API call for *protein molecule function reasoning* as follows:

$$\text{<API>}GR(\mathcal{G}, \text{“}protein\text{-}function\text{”}, g_i) \to r\text{</API>}, \quad (30)$$

which will call the pre-trained graph neural network for protein graph representation learning and classification.

Sequential Recommender System Reasoning: In the era of big data, as more and more information and data are generated both online and offline, manual search of desired information from the data sources has become infeasible nowadays and we may need recommender systems [21] to automatically recommend such information for us instead. Based on the historical records, sequential recommender system aims to infer the next item(s) that users may be interested in, which may lead to either the future purchase action or the review rating scores of those items. When studying sequential recommender systems, it is a common way to model recommender systems as the bipartite graphs, where the user-item interaction record also has an attached timestamp. With considerations about the timestamps, sequential recommender systems aim to infer the potential existence (or the weight) of links between user and their interested items for the next future timestamp. In other words, we can define the sequential recommendation problem in recommender systems as a link prediction task with considerations about the temporal factor.

Formally, according to the above description, we can represent the sequential recommender system as a bipartite graph $G = (\mathcal{V}, \mathcal{E})$, where the node set $\mathcal{V} = \mathcal{U} \cup \mathcal{I}$ covers both users and items and the links in set $\mathcal{E} \subset \mathcal{M} \times \mathcal{I}$ only exist between users and item instead. For each user-item pair $(u_j, i_l) \in \mathcal{E}$ in the link set, we can also obtain its timestamp as $t_{j,l}$. The sequential recommender system data can be loaded with the following API call:

$$\text{<API>}GL(\text{“}./graphs/recommender\text{-}system\text{”}) \to G\text{</API>}. \quad (31)$$

For each user u_j and item i_l in the recommender system G, based on the historical interaction records (before the current timestamp), we can also obtain their raw features as vectors $\mathbf{x}_{u_j}$ and $\mathbf{x}_{i_l}$, which will be used to infer the label between them, *i.e.,* $\mathbf{y}_{u_j, i_l}$, in the future. Depending on the modeling approach, the label vector can indicate either whether the user will purchase the item or not (*i.e.,*, binary classification task) or the rating score of the user for the item (*i.e.,* the regression task). Regardless of the specific modeling settings, we can represent the recommender system reasoning API call in LLMs as follows:

$$\text{<API>}GR(G, \text{“}recommendation\text{”}, u_j, i_l) \to r\text{</API>}, \quad (32)$$

which will return either the probability scores that the user u_j will be interested in the item i_l or the specific rating scores that u_j will rate for i_l. In addition, for one specific user u_j, we can also return the list of top-k recommended items with the API call:

$$\text{<API>}GR(G, \text{“}recommendation\text{”}, u_j, k) \to r\text{</API>}. \quad (33)$$

In the above API calls, we can use different pre-trained recommendation models to define the function used in the above API call, *e.g.,* *collaborative filtering* and *graph neural network* based recommendation models.

Online Social Network Community Reasoning: Online social networks [23], like Facebook, Twitter and Tiktok, provides various categories of online services for their users to facilitate their online socialization with family members, colleagues and friends. Users in online social networks tend to interact more frequently with friends with closer social relationships, who will naturally form their online social communities based on their online social behaviors. Reasoning for the social communities of users in online social networks is a very complicated problem for LLMs to handle. We will also introduce the API calls to empower LLMs to reason social communities in this paper as well.

Formally, we can represent the online social network studied in this paper as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of user nodes and $\mathcal{E}$ denotes the social interactions among the users in the network. The online social network data can be loaded with the following API call:

$$\text{<API>}GL(\text{“}./graphs/social\text{-}network\text{”}) \rightarrow G\text{</API>}. \tag{34}$$

Based on G, we can represent its detected social community structure as $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$, where $\bigcup_{i=1}^{k} C_i = \mathcal{V}$. Depending on the problem setting, the social communities to be detected can be either based on hard partition or soft partition of the user nodes. For hard partition, we will have $C_i \cap C_j = \emptyset, \forall i, j \in \{1, 2, \cdots, k\}$ (*i.e.,* there exist no overlap between any two communities); where as for the soft partition, the communities may have overlaps and one user node may be partitioned into multiple communities simultaneously.

Based on the above description, given on the loaded online social network G, we can infer both the communities of the whole social network or the local community for a specific user (*e.g.,* $u_i \in \mathcal{V}$) with the following API calls:

$$\text{<API>}GR(G, \text{“}community\text{”}) \rightarrow r\text{</API>}, \tag{35}$$

$$\text{<API>}GR(G, \text{“}community\text{”}, u_i) \rightarrow r\text{</API>}, \tag{36}$$

which will call various pre-trained social network community detection algorithms to identify the community structures.

Knowledge Graph Semantics Reasoning: Compared with the unstructured documents, knowledge graph [14] aggregates information about entities and their relations from multiple data sources in a well-organized format. Knowledge graph is a powerful tool for supporting a large spectrum of applications in the real-world, like searching, ranking, Q&A and chatbot dialogue systems. Reasoning of knowledge graphs helps provide the evidences for providing the results with factual basis. At the same time, such a reasoning process will also provide the justification and explanation for the obtained results by the current natural language processing systems. In this paper, we will not study how to build the knowledge graph from textual document sources. Instead, we assume the knowledge graph has been built and is ready to be used for reasoning in the various downstream applications.

Formally, we can represent the built knowledge graph (*e.g.,* Wikipedia) as $G = (\mathcal{V}, \mathcal{E})$, where the node set $\mathcal{V}$ covers the set of named entities and the link set $\mathcal{E}$ includes the set of relations among these entities instead. The knowledge graph can be loaded with the following API call:

$$\text{<API>}GL(\text{“}./graphs/knowledge\text{-}graph\text{”}) \rightarrow G\text{</API>}. \tag{37}$$

In the loaded knowledge graph G, there will also be a bunch of attributes attached to both the entities and relations. Both the attributes and the knowledge graph structure can be effectively used in the reasoning process to infer the potential attributes, related entities and the relations between pair of entities. Formally, we can represent the knowledge graph semantics reasoning API call used in this paper as:

$$\text{<API>}GR(G, reasoning\text{-}type, inputs) \rightarrow r\text{</API>}, \tag{38}$$

where the “inputs” denotes the input entity, relation and attribute information and “reasoning-type” can take values from {“entity”, “relation”, “attribute” } that specifies the desired reasoning output type. Various pre-trained knowledge graph representation learning models can be used to define the function called in the above API.

A Summary of Advanced Graph Reasoning API Calls: we also provide a list of API call examples of the advanced graph reasoning tasks mentioned above in Table 2. For each of the tasks, we provide several different input reasoning statements, and try to insert the corresponding reasoning API calls at the most appropriate positions in the statements. As introduced above, some of the API calls introduced above can be used in different ways to reason for different types of desired information, some examples of which have also been provided in Table 2 as well.

4.4 Prompt Augmentation with ChatGPT

For the prompt examples provided in Table 1 and Table 2, they can only cover a handful number of examples about how to use the API calls for different graph reasoning tasks. Such a small number of instances are not sufficient for the fine-tuning of the existing LLMs. In this paper, we propose to augment the prompt instances with ChatGPT (gpt-3.5-turbo), which has demonstrated excellent few-shot and even zero-shot learning ability [2] in many different language learning tasks already.

4.4.1 Graph Loading Prompt Dataset Generation. Similar to [26], to help the generation of prompt examples, we also provide a detailed instruction for ChatGPT to specify its system role. Here, we can take the graph data loading API call as an example, and the instruction together with the prompt examples fed to ChatGPT can be provided as follows. Based on it, we will ask the ChatGPT to generate the graph data loading prompt dataset.

```
Instruction: Your task is to add API calls of Graph Loading functions to a piece of input text for concrete graph data loading.
The function should help load required graph structured data based on the mentioned graph name and its nodes and links.
You can call the Graph Loading API by writing "[GL(graph-name, nodes, links)]", where the "graph" is the target graph data, and "nodes" and "links" are the mentioned nodes and links.
```

If no specific nodes or links are mentioned, then the API will write "all nodes" and "all links" for the "nodes" and "links" parameter entries.
If only nodes are specified, the API will write the mentioned nodes for the "nodes" parameter entry, and write "all related links" for the "links" parameter entry.
If only links are specified, the API will and write "all related nodes" for the "nodes" parameter entry, and write the mentioned links for the "links" parameter entry.

Here are some examples of the API call for loading graph structured data. In the examples, the output will repeat the input, and also insert the API call at the most appropriate position.

- Input: The structure of the benzene ring molecular graph of benzene ring contains a hexagon.
- Output: The structure of the [GL("benzene-ring")] molecular graph of benzene ring contains a hexagon.

- Input: There exist a carbon-oxygen double bond in the Acetaldehyde molecular graph.
- Output: There exist a [GL("acetaldehyde-molecular-graph", {Carbon, Oxygen}, {(Carbon, Oxygen)})] carbon-oxygen double bond in the Acetaldehyde molecular graph.

- Input: The lollipop graph looks like a spoon.
- Output: The [GL("lollipop-graph", "all nodes", "all links")] lollipop graph looks like a spoon.

- Input: The paper#10 in the Cora bibliographic network introduces the Transformer model.
- Output: The [GL("cora", {Paper#10}, "all related citation links")] paper#10 in the bibliographic network introduces the Transformer model.

- Input: Insulin is a small globular protein containing two long amino acid chains.
- Output: [GL("insulin-protein-graph", "all atom nodes", "all atom bond links")] Insulin is a small globular protein containing two long amino acid chains.

- Input: At IMDB recommender system, David rates the "The Avengers" movie with a 10-star review score.
- Output: At IMDB recommender system, [GL("imdb-recommender-system", {"David", "The Avengers"}, {("David", "The Avengers")})] David rates the "The Avengers" movie with a 10-star review score.

- Input: Among the existing online social apps, Tiktok makes it easy for users to socialize with each other online via livestream videos.
- Output: Among the existing online social apps, [GL("tiktok-social-network", "all user and video nodes", "all user-video links and user-user links")] Tiktok makes it easy for users to socialize with each other online via livestream videos.

- Input: According to the Freebase knowledge graph, Donald Trump was born in 1946 at the Jamaica Hospital Medical Center in New York.
- Output: According to the [GL("freebase", {"Donald Trump", "Jamaica Hospital Medical Center", "New York"}, {("Donald Trump", "Jamaica Hospital Medical Center"), ("Jamaica Hospital Medical Center", "New York")})] Freebase knowledge graph, Donald Trump was born in 1946 at the Jamaica Hospital Medical Center in New York.

Query: Based on the instruction and examples, please generate 5000 such input-output pairs for real-world graph data loading. Please make sure the data loaded are in graph structures and the API call is insert ahead of the mentioned graphs or the mentioned nodes or links.

Based on the instruction, examples and query, by calling ChatGPT API, we obtained a prompt dataset with 5, 000 input-output pair instances. With manual removal the incomplete instances and brief proofreading, about 2, 803 graph data loading API call input-output pairs are preserved in the dataset, which will be used for the fine-tuning to be introduced later.

4.4.2 Graph Reasoning Prompt Dataset Generation. As to the other graph reasoning prompts, with the instruction and prompt examples, we can use ChatGPT to generate a large number of similar input-output pairs. Meanwhile, slightly different from graph loading API calls, to ensure the graph reasoning prompts are valid, we propose to compose all the inputs statements manually by calling the graph reasoning toolkits in advance. For instance, for the first paper in the Cora bibliographic network, its topic is about "Neural Networks" and we will compose its input statement as follows:

- Input: The first paper in Cora has a topic of Neural Networks.

We will feed such input to ChatGPT and ask it helps insert the appropriate graph reasoning API calls to the statement with the query.

Query: Based on the instruction and examples, generate the output with graph reasoning API calls for the input. Please make sure the API call is insert at the most appropriate position.

Based on the query and input statement, ChatGPT will return the following output:

- Output: The first paper in Cora has a topic of [GR(GL("cora", "all paper nodes", "all citation links"), "topic", Paper#1) -> r] Neural Networks.

Besides using ChatGPT to annotate the API calls and generate the above output, we also use ChatGPT to rewrite the input statement in other words without changing its semantic meaning. For instance, for the input statement shown above, we also obtain several of its rephrase versions as follows:

- Input: The initial article in Cora focuses on the subject of Neural Networks.

- Input: In Cora, the premier paper addresses Neural Networks as its main theme.

- Input: The foremost paper in the Cora collection pertains to the field of Neural Networks.

- Input: Cora's inaugural publication delves into the subject matter of Neural Networks.

These rephrased input will also be fed to ChatGPT again for the API call annotation as well. Such a process will be done for all the node/graph instances studied in both the basic graph property reasoning tasks and the advanced graph reasoning tasks for generating the input-output pair datasets. Based on the generated dataset, we will run the API calls generated by ChatGPT and compare the return result of the graph reasoning API functions with the true values in the statements. For the outputs whose API calls (1) are not runnable or (2) cannot return the correct result, they will be filtered from the dataset. Finally, after the filtering, the ChatGPT augmented generated datasets will be used for the LLMs fine-tuning, whose statistical information will be provided later in the following experiment section. Meanwhile, detailed information about the instruction and prompt examples fed to ChatGPT for graph reasoning API call dataset generation will be provided for the readers in the Appendix attached at the end of this paper as well. For the other graph reasoning tasks not studied in this paper, their reasoning API call datasets can be generated in a similar way as described above.

4.5 LLMs Fine-Tuning for Graph Reasoning

In this part, we will talk about how to use the ChatGPT augmented dataset to fine-tune other pre-trained LLMs, so they can automatically generate such API calls just in the same manner as regular content generation in the later inference process.

4.5.1 API Call Insertion Position Prediction. Formally, as shown by the prompt examples in Table 1 and Table 2, given the input textual statement with a sequence of words, *i.e.,* $\mathbf{w} = [w_1, w_2, \cdots, w_n]$, we aim to identify the position where we can insert the appropriate API calls with a sequence of special tokens, *i.e.,* $\mathbf{s}(c)$ = <API>$f(args)$</API> as introduced in the previous Section 4.2. The major challenges lie in (1) precisely identify the most appropriate positions to insert the API call, (2) correctly the choose the API functions to be used for the call, and (3) also accurately extract the parameters from the context and feed them to the functions. In this section, we will address all these three challenges.

For the provided input statement, there may exist multiple potential positions for inserting the API calls. Therefore, instead of finding the most likely position to insert the API calls, we propose to identify the top-k positions that we may instead the API calls, where k is a hyper-parameter to be tuned. Formally, based on a pre-trained language model M, given the input statement $\mathbf{w} = [w_1, w_2, \cdots, w_n]$, we can insert an API call at the i_{th} (where $i \in \{1, 2, \cdots, n\}$) position (*i.e.,* right between the words w_{i-1} and w_i) with the probability

$$P(i|\mathbf{w}(1:i-1)) = P_M(\text{<API>}|\mathbf{w}(1:i-1)). \tag{39}$$

Once the LLM M generates the special beginning token <API> at the i_{th} position, the model will know it should insert an API call here. All the tokens generated after the <API> token and before the special ending token </API> will be the API call function name or the input parameters. For the position index with the latest probabilities, *i.e.,* top-k of $\{P(i|\mathbf{w}(1:i-1))\}_{i \in \{1,2,\cdots,n\}}$, they will be selected as the potential positions to insert the API calls.

Meanwhile, different from the simple API functions studied in [34], the graph data reasoning APIs studied in this paper have much higher costs in terms of both computational time and required storage space. Therefore, whether to insert the API calls at those selected positions or not, we also need to further consider the specific API call functions, where duplicated API calls will be filtered to avoid the redundant and unnecessary time costs in calling such APIs. We will discuss them in the following part in detail.

4.5.2 API Call Domain and Function Selection. Different from the very few API call functions studied in [34], the graph reasoning APIs studied in this paper are much more diverse, which may create challenges in the model implementation and tuning. On the one hand, as more graph reasoning tasks and API functions are incorporated into tuning the LLMs, the graph reasoning API function search space will grow exponentially, which makes it harder to select the correct and best API functions in the call. Also some API functions for different graph reasoning tasks may even share similar function names, which may mislead the LLM in selecting the correct ones in the API calls. On the other hand, different graph reasoning tasks may call different API functions from different toolkits, and some may require different graph functions and trained models to be pre-loaded at the backend.

Therefore, we propose to slightly change the graph reasoning API call template introduced in Section 4.2 as follows:

$$\mathbf{s}(c) = \text{<API>}GR(G, domain{:}func, args)\text{</API>}, \tag{40}$$

or

$$\mathbf{s}(c, r) = \text{<API>}GR(G, domain{:}func, args) \to r\text{</API>}, \tag{41}$$

where the corresponding "domain" of the API function is prepend to the specific graph reasoning tasks as a parameter in the API function call. The domain can be either the used toolkit names or the specific pre-trained model names. For instance, for the graph property reasoning of graph diameter with the networkx toolkit[1], we can represent the corresponding parameter as "networkx:diamster"; while for the recommender system reasoning with spectral collaborative filtering [49], we can represent the corresponding parameter as "scf:recommendation" (where "scf" is an abbreviation of the "spectral collaborative filtering" model name). Meanwhile, for some cases, if the domain is not specified, we will just use the function in the default domain for the graph reasoning task.

In other words, when inserting the API function calls into the statement, we need to infer both the domain and function name of the API call. At the inference stage, as the LLMs generate the domain, the system can pre-load the domain code at the backend even before the whole API call output statement generation is completed. At the same time, it will also allow the LLMs to choose the optimal domain to be used in the API call, since to accomplish the same graph reasoning task, there will exist several different approaches with different performance in terms of effectiveness and efficiency.

Technically, within the function parameters, we may need to select the best domain from the available candidate domain set, *i.e.,* $\mathcal{D} = \{d_1, d_2, \cdots, d_n\}$, according to the prefix context, *i.e.,* the selected domain after the sequence "$\mathbf{w}(1 : i-1)\text{<API>}GR(G,$" can be represented as

$$d^i = \arg\max_{d_j \in \mathcal{D}} P_M(d_j | \mathbf{w}(1 : i-1)\text{<API>}GR(G,)). \tag{42}$$

Furthermore, based on the selected domain d^i, we may need to select the best function from it that may meet our needs. Formally, we can represent the available functions from the selected domain d^i as set $\mathcal{F}_{d^i} = \{f_1, f_2, \cdots, f_m\}$, and the function which can maximize the generation probability will be selected, *i.e.,*

$$f^i = \arg\max_{f_l \in \mathcal{F}^i_j} P_M(f_l | \mathbf{w}(1 : i-1)\text{<API>}GR(G, d^i{:}), \tag{43}$$

where the ":" mark is appended after domain d^i automatically. Once the domain and function are selected, the model may also need to fill in the remaining parameters for the functions accordingly, which will be introduced in the following subsection for readers.

4.5.3 API Call Function Parameter Completion. Once the domain and function token $d^i : f^i$ are determined, Graph-ToolFormer will also need to provide the parameters for the selected function based on the statement context. There exist two different ways for completing the parameter entries, *i.e.,* masked parameter completion and causal parameter completion.

For the masked parameter completion, once the domain and function are selected, Graph-ToolFormer will automatically generate and insert the remaining tokens, include the function parameter names, the parentheses, the comma and colon marks, and API call ending special token </API>. For instance, based on the current token sequence "$\mathbf{w}(1 : i-1)\text{<API>}GR(G, d^i{:}f^i,$", Graph-ToolFormer will automatically complete the API call as follows:

$$\begin{aligned} &\mathbf{w}(1 : i-1)\text{<API>}GR(G, d^i{:}f^i, arg_1{:}[MASK_1], && (44)\\ &\qquad arg_2{:}[MASK_2], \cdots, arg_n{:}[MASK_n])\text{</API>}, && (45) \end{aligned}$$

where terms $arg_1, arg_2, \cdots, arg_n$ are the list of parameter names of function $d^i{:}f^i$. In the API call, we mask the parameter values, which will be inferred based on both the prefix context and the function parameter names.

The disadvantages of the above masked parameter completion is that the model need to complete the full list of parameters of the function. However, in the real world function calls, only a few number of parameters will be provided actually, whereas the remaining parameters will use their default values instead. Also the masked parameter completion is inconsistent with the previous special token and domain/function prediction process. Therefore, in this paper, we propose to use the consistent causal parameter completion instead.

For the causal completion of the function parameters, its completion process is similar to the above special token and domain/function selection process. Based on the provided statement and generated tokens, Graph-ToolFormer can generate the list of provided parameter values for the API call, e.g.,

$$arg^i_j = \arg\max P_M(arg | \mathbf{w}(1{:}i-1)\text{<API>}GR(G, d^i{:}f^i,), \tag{46}$$

$$val^i_j = \arg\max P_M(val | \mathbf{w}(1{:}i-1)\text{<API>}GR(G, d^i{:}f^i, arg^i_j{:}). \tag{47}$$

Such a process continues until the end API call special token "</API>" is generated. By adding the generated parameter name and value into the token list, we can get the model generation result to be "$\mathbf{w}(1{:}i-1)\text{<API>}GR(G, d^i{:}f^i, arg^i_j{:}val^i_j, \cdots)\text{</API>}\mathbf{w}(i{:}n)$" or the one with output "$\mathbf{w}(1{:}i-1)\text{<API>}GR(G, d^i{:}f^i, arg^i_j{:}val^i_j, \cdots) \to r\text{</API>}\mathbf{w}(i{:}n)$". For the parameters that are not generated by the Graph-ToolFormer model, we will use the default parameter values in the API function calls in the inference process.

4.5.4 LLMs Fine-Tuning with Augmented API Call Dataset. Formally, given the ChatGPT augmented graph reasoning prompt dataset $\mathcal{D} = \{(\mathbf{w}_1, \bar{\mathbf{w}}_1), (\mathbf{w}_2, \bar{\mathbf{w}}_2), \cdots, (\mathbf{w}_{|\mathcal{D}|}, \bar{\mathbf{w}}_{|\mathcal{D}|})\}$ involving a set of input-output statements pairs, where the notation $\bar{\mathbf{w}}_i$ ($\forall i \in \{1, 2, \cdots, |\mathcal{D}|\}$) is the output statement of $\mathbf{w}_i$ with inserted API call annotations. According to the above generation process, by feeding the input statement $\mathbf{w}_i$ to LLMs, we can represent the generated output by the model as

$$\hat{\mathbf{w}}_i = LLM(\mathbf{w}_i), \forall i \in \{1, 2, \cdots, |\mathcal{D}|\}. \tag{48}$$

[1] https://networkx.org/

Furthermore, by comparing the generation output $\hat{\mathbf{w}}_i$ with the ChatGPT annotated output statement $\bar{\mathbf{w}}_i$, we can define the loss function for fine-tuning the LLM as

$$\ell(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{w}_i, \bar{\mathbf{w}}_i) \in \mathcal{D}} \ell(\hat{\mathbf{w}}_i, \bar{\mathbf{w}}_i) \tag{49}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{w}_i, \bar{\mathbf{w}}_i) \in \mathcal{D}} \sum_{j} \text{cross-entropy}(\hat{\mathbf{w}}_i(j), \bar{\mathbf{w}}_i(j)). \tag{50}$$

4.6 LLMs Inference and Postprocessing

Prior to the inference stage, all the graph loading and reasoning API toolkits and functions will be installed and ready to use already. Also all the graph data to be loaded will be organized into a unified format (or other format) that the graph loading API functions can handle. In the inference, given any graph reasoning input statement $\mathbf{w}$, the fine-tuned LLMs will project the input statement to the corresponding output statement $\hat{\mathbf{w}}$ annotated with the API calls. Furthermore, the generation output $\hat{\mathbf{w}}$ will be further post-processed by detecting and initiating the API calls in it. Depending on whether the API call return result needs to be outputted or not, Graph-ToolFormer will also further replace the API calls with its return result in the statement. We also provide an example about the inference process as follows:

- `Input: The first paper in Cora has a topic of [TBR].`

- `Output: The first paper in Cora has a topic of [GR(GL("cora"), "graph-bert:topic", Paper#1) -> Neural Networks].`

- `Post-processed Output: The first paper in Cora has a topic of Neural Networks.`

What's more, within the Graph-ToolFormer model, we also maintain a small-sized running time memory bank, which keep records of both the recent external API function calls and their output results for the model in the reference stage. For instance, if the API call on graph loading $GL(file\text{-}path, node\text{-}subset, link\text{-}subset)$ or graph reasoning $GR(G, d^i{:}f^i, arg_1{:}val_1, arg_2{:}val_2, \cdots, arg_n{:}val_n)$ has been executed and its read-only result has also been stored in the memory bank, then Graph-ToolFormer will also has a post-processing of the generation result to replace the function call with its read-only results obtained before. Such a memory bank is very helpful, especially for the API function calls like data loading or other graph reasoning API calls with large time or space costs. For instance, as shown in Table 1, after the example lollipop graph (the top left green graph) shown in Figure 1 has been loaded as G_l, we will just replace the data loading file name path with the graph G_l directly. The memory bank has a fixed size in Graph-ToolFormer, and will maintain its stored information similar to a queue (*i.e.*,, FIFO). Once the stored information exceeds the memory bank capacity, the result of very old API calls or the results which has been rewritten already, Graph-ToolFormer will delete them from the memory bank.

5 EXPERIMENTS

In this section, we will conduct extensive experiments to evaluate the performance of Graph-ToolFormer on various graph data reasoning tasks that we discuss before. According to the method section, we will ChatGPT to generate a large-size of graph reasoning API call dataset based on both the textual instructions and a small number of hand-crafted prompt reasoning examples. Via API calling, we will verify that the data instances in the generated dataset are all valid, which can generate the desired reasoning results. Furthermore, based on the filtered prompt datasets, we will further fine-tune existing pre-trained language models (*e.g.*, GPT-J and LLaMA) for the graph reasoning tasks specifically. More details about the experimental settings and some preliminary experimental results will be provided in the following parts of this section.

5.1 Experimental Settings

5.1.1 *Graph Reasoning API Call Design and Data Generation.*

Based on both the instructions and prompt examples provided at the Appendix section attached to this paper, we propose to use the ChatGPT (via OpenAI official API with backend model "gpt-3.5-turbo") for generating a set of similar input-output pairs for *graph data loading*, *graph property reasoning* and all the other *advanced graph reasoning tasks* introduced before in this paper, respectively. To validate such generated datasets are all correct, we screen and filter all the input-output pairs by calling the APIs. For the instances that can work correctly and also generate the desired return values, we will include them into the dataset for language model fine-tuning, while the others will be filtered.

We also provide the dataset statistical information in Table 3, which include both the size of the graph dataset (including numbers of nodes, links, graph instances, features and classes) and the obtained API call instance number. Together with the raw graph datasets, both the hand-crafted prompts (including instructions and examples) and the generated input-output pairs will be shared with the community, which will be posted to the link[2] shortly.

5.1.2 *Adopted Models.*

As to the base model to be used for building Graph-ToolFormer, we have tried several pre-trained language models with open-source model architectures, configurations, tokenizers and parameter checkpoints. Specifically, the base language models used in this experiment include

- **Graph-ToolFormer (GPT-J 6B, 8bit)**: The EleutherAI's GPT-J (6B) is a transformer model trained using Ben Wang's "Mesh Transformer JAX" [44] that has 6 billion parameters. To load GPT-J in float 32, it will require 22+GB RAM to load the model and the fine-tuning will require at least 4x RAM size. To further lower-down the RAM consumption for GPT-J (6B), researchers also propose to quantize it with 8-bit weights [8], which allows scalable fine-tuning with LoRA (Low-Rank Adaptation) and 8-bit Adam and GPU quantization from bitsandbytes. The Graph-ToolFormer (GPT-J 6B, 8bit) model will use GPT-J 6B 8bit as the base model for fine-tuning.

[2] dataset github link: https://github.com/jwzhanggy/Graph_Toolformer/tree/main/data

Table 3: A statistical summary of graph datasets used in the experiments of this paper. For the GPR and molecular graph datasets (including PROTEIN, PTC, NCI1 and MUTAG), the "Node#" and "Edge#" denote the average numbers of nodes and edges for the graph instances in the datasets, respectively. For the graphs without features or labels, we will fill the entries with "NA" in the table.

Tasks	Datasets	Graph Types	Node#	Edge#	Graph#	Feature#	Class#	API Call#
Graph Loading	GL-Prompt	NA	NA	NA	NA	NA	NA	2,802
Property Reasoning	GPR-Prompt	Generated classic graphs	14.70 (avg)	28.27 (avg)	37	NA	NA	2,587
Paper Topic Reasoning	Cora	Bibliographic network	2,708	5,429	1	1,433	7	18,956
	Citeseer	Bibliographic network	3,327	4,732	1	3,703	6	23,184
	Pubmed	Bibliographic network	19,717	44,338	1	500	3	138,019
Molecule Function Reasoning	PROTEIN	Protein molecular graphs	39.05 (avg)	72.82 (avg)	1,113	NA	2	6,678
	PTC	Chemical molecular graphs	25.56 (avg)	25.96 (avg)	344	NA	2	2,064
	NCI1	Chemical molecular graphs	29.86 (avg)	32.30 (avg)	4,110	NA	2	24,660
	MUTAG	Chemical molecular graphs	17.93 (avg)	19.79 (avg)	188	NA	2	1,128
Sequential Recommendation Reasoning	MovieLens	Recommender system	2,625	100,000	1	NA	5 (rating)	500,000
	Last.FM	Recommender system	14,415	71,064	1	NA	7,107 (tag)	355,320
	Amazon	Recommender system	396,810	450,578	1	NA	5 (rating)	2,252,890
Social Community Reasoning	Youtube	Social network	1,134,890	2,987624	1	NA	8,385 (community)	158,040
	Email	Social network	1,005	25,571	1	NA	42 (community)	5,034
	DBLP	Social network	317,080	1,049,866	1	NA	13,477 (community)	783,009
Knowledge Graph Reasoning	Freebase	Knowledge graph	14,951	592,213	1	NA	NA	1,695,651
	WordNet	Knowledge graph	41,105	151,442	1	NA	NA	454,326

More baseline LLMs will be added and compared in the experiments. The source code of the models studied in the experiment will be posted to the link[3] very soon.

5.1.3 Detailed Hardware, Software, Experimental Setups. Based on the datasets introduced Section 5.1.1, we will further fine-tune the above language models. Depending on the machine memory size, we will use different mini-batch sizes in the fine-tuning process. Especially, when the batch size and input/output max token length are assigned with small values (*e.g.,* batch-size: 2 and max-length: 128), we can also fine-tune Graph-ToolFormer (GPT-J 6B, 8bit) model on GPUs with smaller RAM (like Nvidia 1080ti with 11GB memory).

Based on the augmented and filtered graph reasoning dataset, considering the extremely high-cost in generation for evaluation, we split the dataset into training/testing with the size ratio "$\min(N-160, 1,600) : 160$", where N is the complete dataset size and the training set size depends on the size. For the dataset with more than $1,760$ instances, $1,600$ instances will be randomly sampled from it as the training set; whereas for the dataset with less than $1,760$ instances, by excluding the randomly sampled testing set (with 160 instances), all the remaining $N-160$ instances will be used as the training set. For all the reasoning tasks on all the datasets, a fix-sized testing set with 160 instances are used for model performance evaluation purposes.

Specifically, the hardware and software setups for the fine-tuning of the language models in this experiment are provided as follows:

[3] model github link: https://github.com/jwzhanggy/Graph_Toolformer

- **Hardware**: We run the fine-tuning experiments on a standalong workstation with GPUs borrowed from a "rich friend" (really appreciate it). Detailed hardware information about the workstation is as follows: ASUS WS X299 SAGE/10G LGA motherboard, Intel Core i7 CPU 6850K@3.6GHz (6 cores), 1 Nvidia Ampere A100 GPU (80 GB HBM2e DRAM), 1 Nvidia GeForce RTX 4090 Founders Edition GPU (24GB GDDR6X RAM), and 96 GB DDR4 memory and 128 GB SSD swap.
- **System and Software**: We run the experiment on Ubuntu 22.04, with CUDA toolkit version 11.8, Nvidia Driver version 520, PyTorch version 1.13.1 and Python 3.9. For the optimizer of Graph-ToolFormer (GPT-J 6B, 8bit), we use the 8-bit AdamW from bitsandbytes with version 0.37.1. We load the pre-trained GPT-J 6B 8bit from Huggingface with weight parameter checkpoint "hivemind/gpt-j-6B-8bit" and config/tokenizer checkpoint "EleutherAI/gpt-j-6b" as the base model of Graph-ToolFormer, and the installed transformer toolkit version is 4.28.0.dev0.
- **Hyper-parameters**: For fine-tuning Graph-ToolFormer (GPT-J 6B, 8bit), we use AdamW with a very small learning rate 1e-6 with weight decay 1e-2. Both the training and testing instances are divided into batches with shuffle with batch size 8 and we set the max input/output token length as 128. For the generation function of the language model, the following hyper-parameters are used, *i.e.,* num-beams: 5, tok-k: 5, top-p: 0.95, temperature: 1.9, do-sample: True, num-return-sequence: 5, max-length: 128.

5.1.4 *Performance Evaluation.*

For the preliminary performance evaluation of Graph-ToolFormer, we have used several evaluation metrics as follows in the experiments:

- **ROUGE scores**: By comparing the outputs of the Graph-ToolFormer framework with the ground-truth, we calculate the Rouge-1, Rouge-2, Rouge-L and Rouge-LSum scores obtained by the model.
- **BLEU score**: Besides the ROUGE scores, we also evaluate the performance of Graph-ToolFormer with the BLEU and BP metrics by comparing the generation output with the ground-truth.

More evaluations among the API calls on the graph reasoning will be provided in the follow-up version of this paper very soon.

5.2 Experimental Tasks and Preliminary Results

We have obtained some preliminary results of graph reasoning tasks with LLMs in Graph-ToolFormer, which are reported in Table 4. These are the scores we have obtained so far, and we will continue to polish the model and update the scores in the follow-up versions. At the same time, we are also working on the demo. With the external graph learning model integrated, we plan to release the demo with GUI for general graph reasoning tasks shortly.

5.2.1 *Graph Data Loading.*

As shown in Table 3, with the ChatGPT as introduced before, we generate a set of graph data loading API call input-output instances. Based on both the instruction and input-output prompt example pairs, we apply ChatGPT to generate about 5, 000 input-output pairs. Via necessary filtering and clean, about 2, 802 are used for the model fine-tuning in the experiment. According to the experimental settings introduced before, we partition the pairs into training and testing sets, and evaluate the performance of the fine-tuned model to evaluate the performance of the Graph-ToolFormer framework for graph data loading.

The experimental results of Graph-ToolFormer on graph data loading evaluated by Rouge and BLEU metrics are provided in Table 4. According to the Table scores, compared with the ground-truth, the outputs generated by Graph-ToolFormer are of very high quality, which obtained the R1 score of 79.58 and R2 score of 69.97. The BLEU scores obtained by Graph-ToolFormer are also very high, which is 61.37 with BP at 88.14.

5.2.2 *General Graph Property Loading.*

For the *graph property reasoning* task, we manually create a graph dataset, involving 27 small-sized classic graph instances, such as *barbell graph*, *wheel graph* and *lollipop graph*, etc. For each graph instance in the dataset, we manually design a few number of reasoning prompts with API calls for its properties (as discussed in Section 4.3.2). Based on both the property reasoning instructions and the hand-crafted prompt examples, with ChatGPT, we further augment the prompt examples and generate a large set of annotated graph property reasoning API call dataset, which will be used for fine-tuning the LLMs in the Graph-ToolFormer framework.

Some basic statistics about the raw graph dataset and the prompt dataset are provided in the Table 3. On average, the generated graph instances have about 14.70 nodes and 28.27 links. We have created a set of 2, 587 API call input-output pairs for reasoning their different properties. Based on the API call instances, similar to the graph data loading task, the performance of Graph-ToolFormer on the graph property reasoning tasks is also very good, with an over-80 R1 score and about 66 BLEU score as shown in Table 4.

5.2.3 *Bibliographic Paper Topic Reasoning.*

We have studied three bibliographic network datasets in this experiment, which include Cora, Citeseer and Pubmed, which are all the frequently used benchmark datasets studied in graph neural network research work. For each of the bibliographic network dataset, we hand-craft a few prompt examples and also augment them with ChatGPT to rephrase and rewrite more diverse input-output pairs. After data filtering, the valid data instances (which works and can obtain the correct graph reasoning results with the API calls) will be used for the model fine-tuning, whose statistical information are provided in Table 3. The performance of the LLMs studied in the experiments are provided in Table 4. Among the three datasets, the performance of Graph-ToolFormer on pubmed dataset is slightly higher than the other two.

5.2.4 *Protein Molecule Function Reasoning.*

For the bio-chemical molecular graph function reasoning task studied in this paper, we will use four bio-chemical graph classification benchmark dataset in the experiments, which include PROTEIN, PTC, NCI1 and MUTAG. In these dataset, each graph instance has both its molecular graph structure and a label indicating its function. For each graph instance, we hand-craft a few prompt examples about its functions, which will be augmented by ChatGPT to rephrase and generate similar

Table 4: A summary of the experimental results of Graph-ToolFormer on various graph reasoning tasks on the corresponding benchmark datasets. The results are evaluated by the Rouge scores, BLEU and BP scores. Except for the graph loading task, we also evaluate the results on other tasks/datasets by comparing the graph reasoning API calls (other textual contents are excluded) with the ground-truth API calls, and report the Accuracy on reasoning API calls in the table as well.

Tasks	Datasets	Methods	Evaluation Metrics					
			Rouge-1	Rouge-2	Rouge-L	Rouge-LSum	BLEU	BP
Graph Loading	GL-Prompt	Graph-ToolFormer	79.58	69.97	75.31	75.38	61.37	88.14
Property Reasoning	GPR-Prompt	Graph-ToolFormer	80.34	68.71	70.35	70.36	66.84	67.66
Paper Topic Reasoning	Cora	Graph-ToolFormer	73.22	63.04	71.26	71.31	53.96	100.00
	Citeseer	Graph-ToolFormer	71.34	64.41	70.32	70.27	51.10	100.00
	Pubmed	Graph-ToolFormer	80.41	76.94	80.20	80.20	63.98	72.61
Molecule Function Reasoning	PROTEIN	Graph-ToolFormer	84.95	79.11	84.48	84.59	70.61	100.00
	PTC	Graph-ToolFormer	74.24	62.74	70.18	70.17	47.39	100.00
	NCI1	Graph-ToolFormer	85.28	79.06	84.92	84.89	61.94	100.00
	MUTAG	Graph-ToolFormer	61.64	56.52	60.76	60.75	31.43	47.49
Sequential Recommendation Reasoning	MovieLens	Graph-ToolFormer	76.16	69.02	75.65	75.51	63.99	100.00
	Last.FM	Graph-ToolFormer	89.53	85.73	86.22	86.22	77.96	100.00
	Amazon	Graph-ToolFormer	71.14	59.63	68.32	67.51	48.77	81.45
Social Community Reasoning	Youtube	Graph-ToolFormer	61.48	60.39	61.48	61.48	19.32	20.02
	Email	Graph-ToolFormer	85.05	84.69	85.05	85.05	52.60	55.09
	DBLP	Graph-ToolFormer	64.40	52.76	61.68	61.74	41.67	100.00
Knowledge Graph Reasoning	Freebase	Graph-ToolFormer	92.79	90.91	92.62	92.67	85.01	91.53
	WordNet	Graph-ToolFormer	87.90	82.18	86.04	85.99	74.86	100.00

data instances. The statistical information about these four datasets are provided in Table 3. We provide the experimental results of the comparison language models in Table 4, and Graph-ToolFormer works very well on PROTEIN, PTC, NCI1. The scores on the MUTAG is slightly lower, partially due to the small number of graph instances and generated API calls in the dataset.

5.2.5 *Sequential Recommender System Reasoning.* For the sequential recommendation reasoning task, we have used three benchmark datasets studied in recommender systems, which include MovieLens, Last.FM and Amazon Review (Software). In these recommender system datasets, both users and items are represented as individual nodes and the interaction between users and items are represented as the links connecting them annotated with the timestamps. Similar to the previous reasoning tasks, we also design a few reasoning prompt examples and further augment them with ChatGPT to generate a large sequential recommender system reasoning dataset. The statistical information about both the raw datasets and the generated reasoning input-output pairs are provided in Table 3. By comparing the studied language models with each, we provide the experimental results of these comparison methods in Table 4. Graph-ToolFormer works very on inserting the correct API calls to the recommender system reasoning tasks, especially on Last.FM.

5.2.6 *Online Social Network Reasoning.* In the social network community reasoning task, we use three benchmark datasets with community ground-truth in the experiment, which include Youtube (an online video sharing social network), Email (an email exchange social network) and DBLP (an academic collaborative social network). As introduced at page[4], the community ground-truth in these social networks denotes either the connected components or the organizational labels of individuals in the networks. Based on the social network datasets, we also generate the input-output reasoning dataset as illustrated in Table 3. The experimental performance results of comparison methods are provided in Table 4,

[4] http://snap.stanford.edu/data/index.html#communities

considering the size and reasoning API call instance number of DBLP and Youtube network are very large, the performance of Graph-ToolFormer on Email is better than the other two datasets.

5.2.7 Knowledge Graph Reasoning. For the knowledge graph reasoning, we use both two benchmark datasets, Freebase and WordNet, in the experiments, which provides both entities and their internal relations. In Table 3, we provide the statistical information about the knowledge graphs and the generated reasoning API calls. For the Freebase knowledge graph, there exist 1, 345 types of edges in the graph, and the total number of "entity-relation-entity" tuples in the graph is 592, 213; whereas the numbers for the WordNet are 18 and 151, 442 instead. Among all these tasks studied in this paper, we observe that the reasoning performance of Graph-ToolFormer achieve the highest scores on the knowledge graph reasoning API generation. Partial reasons are due to the base model used in Graph-ToolFormer has been pre-trained on large knowledge graph datasets before already, and Graph-ToolFormer can capture the patters of the context and insert the API calls correctly for the reasoning tasks.

6 CONCLUSION

In this paper, we investigate the principles, methodologies and algorithms to empower existing LLMs with graph reasoning ability. We introduce the Graph-ToolFormer framework, and propose to teach LLMs to use external graph data loading and graph reasoning tools for addressing the tasks. Specifically, several representative graph reasoning tasks are studied in this paper, including the basic graph property reasoning task, and the advanced tasks, like bibliographic paper topic reasoning, molecular graph function reasoning, sequential recommender system reasoning, social network community reasoning and the knowledge graph reasoning. For each of these graph reasoning tasks, we select several benchmark graph datasets in the experiment, and design the reasoning API call examples manually. In addition, with the help of ChatGPT, we can further augment the reasoning API call templates to generate large-sized graph reasoning datasets. Via fine-tuning of existing pre-trained LLMs with the generated dataset, the Graph-ToolFormer framework has been demonstrated to be effective with the preliminary results on many of these studied graph reasoning tasks. We will continue to polish the Graph-ToolFormer framework in the coming versions and also build the demo of Graph-ToolFormer with GUIs for diverse graph reasoning tasks.

REFERENCES

[1] Yejin Bang, Samuel Cahyawijaya, Nayeon Lee, Wenliang Dai, Dan Su, Bryan Wilie, Holy Lovenia, Ziwei Ji, Tiezheng Yu, Willy Chung, Quyet V. Do, Yan Xu, and Pascale Fung. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *ArXiv*, abs/2302.04023, 2023.

[2] 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, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff 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. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

[3] Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-inference: Exploiting large language models for interpretable logical reasoning. *ArXiv*, abs/2205.09712, 2022.

[4] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. *ArXiv*, abs/2110.02861, 2021.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

[6] Bhuwan Dhingra, Jeremy R. Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W. Cohen. Time-Aware Language Models as Temporal Knowledge Bases. *Transactions of the Association for Computational Linguistics*, 10:257–273, 03 2022.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.

[8] Hugging Face. hivemind/gpt-j-6b-8bit. https://huggingface.co/hivemind/gpt-j-6B-8bit, 2022. [Online; accessed 19-March-2023].

[9] Nezihe Merve Gürel, Hansheng Ren, Yujing Wang, Hui Xue, Yaming Yang, and Ce Zhang. An anatomy of graph neural networks going deep via the lens of mutual information: Exponential decay vs. full preservation. *ArXiv*, abs/1910.04499, 2019.

[10] David K. Hammond, Pierre Vandergheynst, and Remi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129?150, Mar 2011.

[11] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ArXiv*, abs/2106.09685, 2021.

[12] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[13] Binxuan Huang and Kathleen M. Carley. Inductive graph representation learning with recurrent graph neural networks. *CoRR*, abs/1904.08035, 2019.

[14] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2022.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[16] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Personalized embedding propagation: Combining neural networks on graphs with personalized pagerank. *CoRR*, abs/1810.05997, 2018.

[17] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Annual Meeting of the Association for Computational Linguistics*, 2019.

[18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *CoRR*, abs/1801.07606, 2018.

[19] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, abs/2101.00190, 2021.

[20] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. In *Annual Meeting of the Association for Computational Linguistics*, 2019.

[21] Prem Melville and Vikas Sindhwani. Recommender systems. *IBM T.J. Watson Research Center*, 2010.

[22] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ramakanth Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey. *ArXiv*, abs/2302.07842, 2023.

[23] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, page 29–42, New York, NY, USA, 2007. Association for Computing Machinery.

[24] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[25] OpenAI. Planning for agi and beyond. https://openai.com/blog/planning-for-agi-and-beyond, 2023. [Online; accessed 27-March-2023].

[26] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155, 2022.

[27] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *ArXiv*, abs/2205.12255, 2022.

[28] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics:*

Human Language Technologies, pages 2080–2094, Online, June 2021. Association for Computational Linguistics.

[29] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *North American Chapter of the Association for Computational Linguistics*, 2018.

[30] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[31] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[32] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *ArXiv*, abs/2204.06125, 2022.

[33] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *ArXiv*, abs/2102.12092, 2021.

[34] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761, 2023.

[35] Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Conference of the European Chapter of the Association for Computational Linguistics*, 2020.

[36] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and Philip S. Yu. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29:17–37, 2015.

[37] Ke Sun, Zhouchen Lin, and Zhanxing Zhu. Adagcn: Adaboosting graph convolutional networks into deep models, 2019.

[38] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *Proc. VLDB Endow.*, 4(11):992–1003, aug 2011.

[39] Damian Szklarczyk, Annika L Gable, Katerina C Nastou, David Lyon, Rebecca Kirsch, Sampo Pyysalo, Nadezhda T Doncheva, Marc Legeay, Tao Fang, Peer Bork, Lars J Jensen, and Christian von Mering. The STRING database in 2021: customizable protein–protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic Acids Research*, 49(D1):D605–D612, 11 2020.

[40] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023.

[41] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[42] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[43] Saraswathi Vishveshwara, K. V. Brinda, and N. Kannan. Protein structure: Insights from graph theory. *Journal of Theoretical and Computational Chemistry*, 01(01):187–211, 2002.

[44] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

[45] Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? *ArXiv*, abs/2109.01247, 2021.

[46] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. Graph transformer networks. In *Neural Information Processing Systems*, 2019.

[47] Jiawei Zhang and Lin Meng. Gresnet: Graph residual network for reviving deep gnns from suspended animation. *ArXiv*, abs/1909.05729, 2019.

[48] Jiawei Zhang, Haopeng Zhang, Li Sun, and Congying Xia. Graph-bert: Only attention is needed for learning graph representations. *ArXiv*, abs/2001.05140, 2020.

[49] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S. Yu. Spectral collaborative filtering. *Proceedings of the 12th ACM Conference on Recommender Systems*, 2018.