Fact Verification on Knowledge Graph via Programmatic Graph Reasoning

Yuanzhen Hao
University of Chinese Academy of Sciences
Beijing, China
haoyuanzhen22@mails.ucas.ac.cn

Desheng Wu
University of Chinese Academy of Sciences
Beijing, China
dwu@ucas.ac.cn

Abstract

Fact verification on knowledge graphs (KGs) uses the structured representation of entities and relations as evidence for validating claims. Previous methods for KG-based fact verification predominantly use natural language inference (NLI) models to predict entailment between claims and KG triples, based on implicit reasoning. We propose Programmatic Graph Reasoning (PGR), a novel framework that integrates large language models (LLMs) for fact verification on KGs. PGR explicitly encodes the reasoning process as a graph reasoning program composed of predefined functions to verify claims step by step. These functions are executed sequentially for graph reasoning and final result prediction. By making the graph reasoning process explicit, PGR ensures more precise and transparent reasoning steps compared to implicit methods. Experimental results on the FactKG dataset demonstrate that PGR achieves state-of-the-art performance with 86.82% accuracy, outperforming all the baseline models. Further analysis confirms the interpretability and effectiveness of our method in handling complex graph reasoning.[1]

1 Introduction

Fact verification has become an essential task in natural language processing (NLP) due to the growing prevalence of misinformation across various platforms. Fact verification based on knowledge graphs (KGs) leverages the structured representation of entities and relationships as evidence to support or refute claims (Kim et al., 2023b). Existing approaches typically retrieve relevant triples from the KG and use them as evidence to verify the claim. These methods follow an NLI-based reasoning paradigm, employing natural language inference (NLI) models (Liu et al., 2019) to predict the entailment between the evidence and the claim.

Recent advances in large language models (LLMs) have demonstrated remarkable capabilities in natural language understanding and reasoning (Kojima et al., 2022; Touvron et al., 2023; Achiam et al., 2023; Yang et al., 2024). Existing methods (Kim et al., 2023a; Jiang et al., 2023) also leverage LLMs to improve accuracy in KG-based fact verification. By utilizing the in-context learning capabilities of LLMs, these approaches can achieve high accuracy in few-shot settings. The reasoning process in these methods resembles NLI models, where the LLM verifies the claim using an evidence-claim pair.

The reasoning process for KG-based fact verification involves logical reasoning over subgraphs in the knowledge graph that correspond to the given claim. This graph reasoning process can be performed step by step, similar to the reasoning approach used by human fact-checkers (Nakov et al., 2021). Considering the complexity of graph reasoning involving entities, relations, and logical structures, a programmatic approach can provide an efficient way to manage multi-variable and multi-step operations, ensuring precise graph reasoning.

We propose a novel approach called *Programmatic Graph Reasoning (PGR)*, which leverages the language understanding and programming capabilities of LLMs for fact verification on KGs. In this framework, we define the fundamental functions—`MATCH`, `SEARCH`, and `VERIFY`—with clearly specified input-output formats and functionalities. PGR consists of two main steps: *program generation* and *graph reasoning*. In program generation, a graph reasoning program is generated for claim verification, constructing reasoning steps using the defined functions. In graph reasoning, the program is executed step by step to predict the final verification result. Unlike NLI-based methods with implicit reasoning, PGR explicitly represents the reasoning process as a structured program. This explicit representation not only enhances interpretability but

[1] https://github.com/hydragon/PGR

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 5480–5495
November 4-9, 2025 ©2025 Association for Computational Linguistics

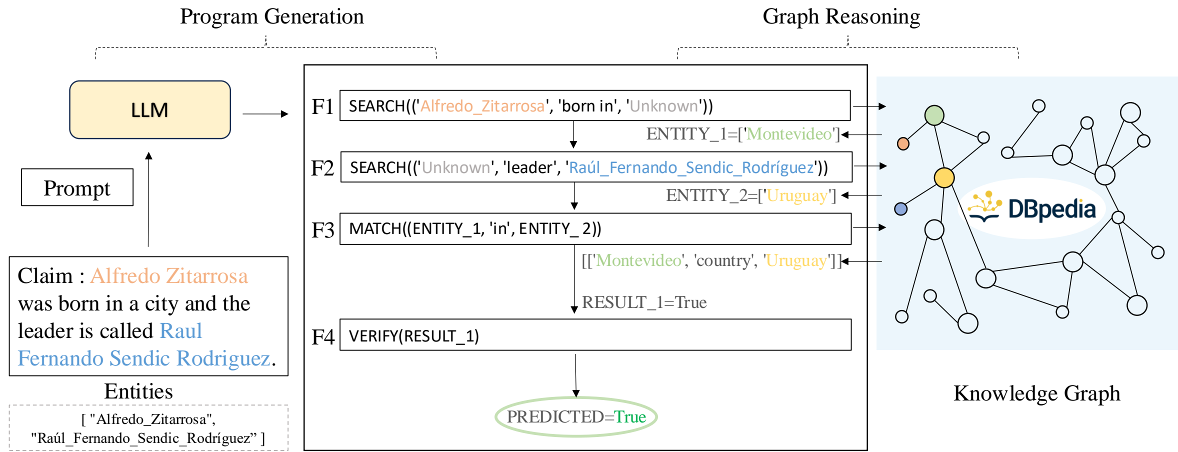

Figure 1: Framework of Fact Verification on KG via Programmatic Graph Reasoning.

also ensures more accurate reasoning by leveraging the clear and well-defined logical structure inherent in the program, compared to implicit reasoning.

We evaluated our proposed method on the FactKG dataset (Kim et al., 2023b). The experimental results show that our approach achieves the state-of-the-art result across all model evaluations, reaching 86.82% accuracy. In the few-shot model comparison, PGR consistently outperforms others, obtaining the highest accuracy across all five claim types in the dataset, further validating the effectiveness of the our method. Additionally, we analyzed the interpretability of PGR and the effectiveness of the generated programs, underscoring the advantages of explicit reasoning in enhancing both predictive performance and explainability.

2 Related Work

Fact-checking with knowledge graphs (KGs) has emerged as a promising approach to verify the veracity of claims by leveraging structured knowledge. FactKG (Kim et al., 2023b) is a fact verification dataset built on DBpedia (Lehmann et al., 2015), a large-scale general-purpose knowledge graph. The claims in FactKG are generated based on the WebNLG corpus (Gardent et al., 2017), with additional data generation steps to create more diverse claims that incorporate more KG triples, requiring complex reasoning for verification. Existing methods for fact-checking on FactKG include BERT-based (Devlin et al., 2019) approaches and GEAR (Zhou et al., 2019), a multi-evidence reasoning framework. GEAR has been adapted and optimized for KG-based fact verification, improving its performance in FactKG dataset (Kim et al., 2023b).

The rapid development of large language models (LLMs) has significantly enhanced both language understanding and logical reasoning capabilities (Kojima et al., 2022; Touvron et al., 2023; Achiam et al., 2023; Yang et al., 2024). The advent of GPT-4 (Achiam et al., 2023) further extends the applications of LLMs to a wide range of tasks, including those related to reasoning and knowledge graphs (Pan et al., 2024; Zhu et al., 2024; Choi and Ferrara, 2024; Wang et al., 2024). Recent research has explored the use of LLMs in KG-based fact verification, particularly through few-shot learning approaches. KG-GPT (Kim et al., 2023a), similar to StructGPT (Jiang et al., 2023), integrates LLMs with few-shot learning for fact verification on KGs. These methods demonstrate that LLMs can be effectively applied to KG-based tasks and show strong reasoning capabilities in few-shot settings, offering new possibilities for improving performance in this domain.

Large language models (LLMs) have demonstrated remarkable versatility in solving various tasks including reasoning(Yao et al., 2023) through in-context learning (Brown et al., 2020), where only a few prompt examples are required to guide the model. ProgramFC (Pan et al., 2023) introduces a novel fact-checking approach that leverages LLMs to generate programs for claim verification. This method decomposes the verification of a complex claim into multiple sub-claims, which are verified independently in a stepwise manner. Such a multi-step reasoning strategy closely resembles the human fact-checking process (Nakov et al.,

2021).

In addition to program-based reasoning, the reasoning performance of LLMs can be significantly enhanced through Chain-of-Thought (CoT) prompting and Knowledge-Grounded Reasoning (Kojima et al., 2022; Wang et al., 2023; Wang and Shu, 2023), which enables the model to perform more accurate multi-step reasoning. Recent researches have shown that decomposing multi-step reasoning tasks using LLMs not only improves accuracy but also offers stronger interpretability, making the verification process more transparent and comprehensible (Pan et al., 2023).

3 Programmatic Graph Reasoning

This section presents our proposed method, Programmatic Graph Reasoning(PGR), for fact verification on KG, which defines essential functions for retrieving and validating evidence from a knowledge graph. The methodology focuses on the definition of functions, the generation of graph reasoning programs, and the execution of programs for accurate fact verification. Figure 1 illustrates the complete process of the proposed PGR framework for KG-based fact verification. The PGR framework consists of two main components: 1. **Program Generation**: Given a claim, PGR generates a structured program composed of graph reasoning functions that represent the reasoning process. 2. **Graph Reasoning**: The generated program is executed to perform step-by-step graph reasoning, with each function operating on KG to verify the claim.

3.1 Problem Definition

We focus on the task of KG-based fact verification. Formally, a knowledge graph is defined as $\mathcal{G} = (V, E)$, where V is the set of vertices representing entities and E is the set of edges representing relationships between entities. Given a claim C and its associated entities $V_C \subset V$ in the knowledge graph $\mathcal{G}$, the goal of fact verification is to check whether the claim aligns with the fact information encoded in $\mathcal{G}$.

This verification process can be conceptualized as identifying a subgraph $\mathcal{G}_C = (V_C, E_C)$ within $\mathcal{G}$ that is relevant to the claim and performing graph reasoning over this subgraph to assess the factuality of the claim.

We aim to design a predictive model $\mathcal{M}$ that takes the claim C, its associated entities V_C, and the knowledge graph $\mathcal{G}$ as inputs and predicts the label $y \in \{\textit{true},\textit{false}\}$ of the claim:

$$\hat{y} = \mathcal{M}(C, V_C, \mathcal{G}), \tag{1}$$

where $\hat{y} \in \{\textit{true},\textit{false}\}$ is the predicted result for C. The primary challenge lies in effectively leveraging the structured information in $\mathcal{G}$ and reasoning over the subgraph $\mathcal{G}_C$ to ensure accurate predictions.

3.2 PGR Formulation

Our proposed method Programmatic Graph Reasoning (PGR) perform multi-step graph reasoning for fact verification. The model $\mathcal{M}$ is designed as a composition of multiple step functions $\{f_1, f_2, \ldots, f_n\}$, where each step function f_i belongs to a predefined function set and is executed sequentially according to the program's flow. The overall process can be formally expressed as:

$$\mathcal{M}(C, V_C, \mathcal{G}) = f_n \circ \cdots \circ f_2 \circ f_1(C, V_C, \mathcal{G}) \tag{2}$$

where each f_i performs an intermediate reasoning step, updating the claim graph and accumulating evidence.

At each step i, the step function f_i takes the current claim graph $\mathcal{G}_C^{i-1}$ and the accumulated evidence set $\mathcal{E}_{i-1}$ as input and produces an updated claim graph $\mathcal{G}_C^{i}$ and an updated evidence set $\mathcal{E}_i$:

$$f_i : (\mathcal{G}_C^{i-1}, \mathcal{E}_{i-1}) \rightarrow (\mathcal{G}_C^{i}, \mathcal{E}_i) \tag{3}$$

where $\mathcal{G}_C^i = (V_C^i, E_C^i)$ represents the graph structure associated with the claim at step i ($\mathcal{G}_C^i \subset \mathcal{G}$), and $\mathcal{E}_i$ denotes the accumulated evidence from all previous steps. For each $e_j \in \mathcal{E}_i$ where $j < i$, e_j represents the output of function f_j, which is utilized for the final prediction.

The final step function f_n is responsible for predicting the factuality label of the claim based on the fully constructed graph $\mathcal{G}_C^{n-1}$ and the accumulated evidence set $\mathcal{E}_{n-1}$. This step can be represented as:

$$f_n : (\mathcal{G}_C^{n-1}, \mathcal{E}_{n-1}) \rightarrow \{\textit{true},\textit{false}\} \tag{4}$$

By modeling $\mathcal{M}$ as a sequence of executable functions, PGR enables flexible and interpretable multi-step reasoning over the claim subgraph $\mathcal{G}_C$. Each step refines the graph and evidence set, ultimately generating a prediction that reflects the underlying reasoning process.

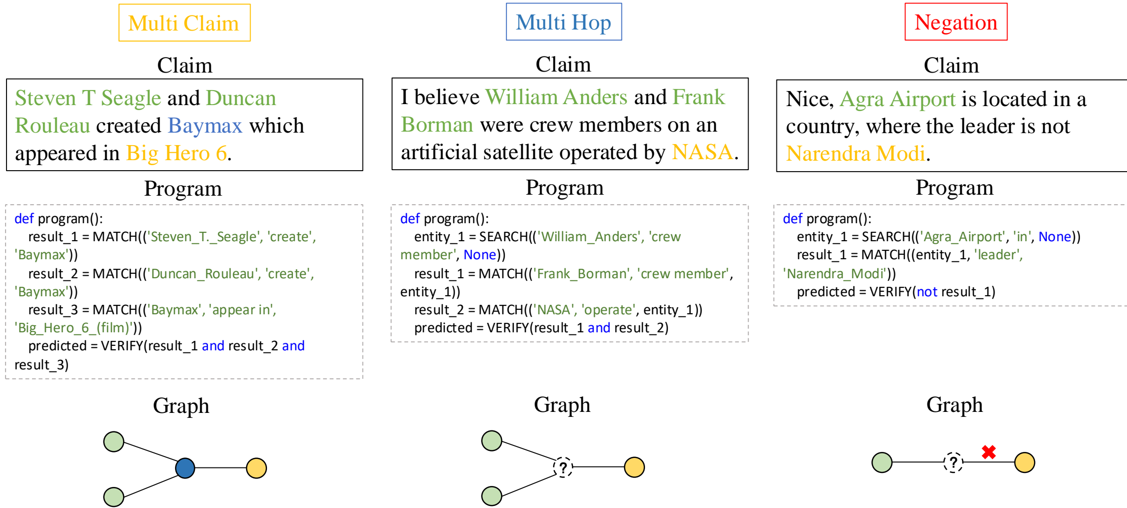

Figure 2: Examples of Programs generated by Programmatic Graph Reasoning.

3.3 Graph Reasoning Functions

The process of verifying claims based on KG as evidence can be viewed as identifying a corresponding subgraph composed of entities and relations from the KG that align with the claim. This process involves converting the claim into a subgraph structure by decomposing it into multiple triple retrieval tasks. The subgraph is then constructed by linking these triples based on the relations described in the claim.

During the subgraph construction, the triple retrieval can be categorized into two fundamental operations: *retrieving missing entities* and *matching complete triples*. To facilitate graph reasoning, we define three fundamental functions—**MATCH**, **SEARCH**, and **VERIFY**:

MATCH

- **Input**: A complete triplet in form of $(entity, relation, entity)$, where the entity may be directly provided in the claim or derived from the output of the **SEARCH** function.
- **Functionality**: Checks whether the specified triplet exists within the KG.
- **Output**: Returns `True` if the triplet is present; otherwise, returns `False`.

SEARCH

- **Input**: A triplet with one `None` value for one of the entity, representing a missing entity (e.g., $(entity, relation, \texttt{None})$ or $(\texttt{None}, relation, entity)$).
- **Functionality**: Identifies the missing component by querying the KG based on the other specified parts of the triplet.
- **Output**: Produces a list of potential entity candidates for the missing component.

VERIFY

- **Input**: The Boolean results from the **MATCH** function.
- **Functionality**: Validates the claim's factual consistency based on the presence or absence of the triplet in the KG.
- **Output**: Returns `True` if the verification confirms the claim; otherwise, returns `False`.

Given a triple (h, r, t), the knowledge graph $\mathcal{G}$, and the claim sentence C, `SEARCH` function returns a set of matching triples $\mathcal{T}_r \subseteq \mathcal{G}$ whose entity and relation match the input triple (h, r, t) with one entity unknown:

$$\texttt{SEARCH} : ((h, r, t)\,,\ \mathcal{G},\ C\ ) \longrightarrow\ \mathcal{T}_r \qquad (5)$$

`MATCH` retrieves the best-matching triple $\tau^* \in \mathcal{G}$ that supports the input triple based on semantic and structural alignment:

$$\texttt{MATCH} : ((h, r, t)\,,\ \mathcal{G},\ C\ ) \longrightarrow\ \tau^* \qquad (6)$$

As the fundamental functions in our method, the `MATCH` and `SEARCH` are designed to handle various types of claims for graph reasoning. Following the reasoning types of claims defined by Kim

et al. (2023b), we analyze how these functions are utilized to address different reasoning requirements. The reasoning types include one-hop, conjunction, existence, multi-hop, and negation, with each type corresponding to specific combinations of the graph reasoning functions.

For one-hop and conjunction claims, the reasoning process primarily relies on the `MATCH` function to match the specific triple or a combination of multiple triples in the knowledge graph. On the other hand, existence and multi-hop reasoning involve both `MATCH` and `SEARCH` functions. For example, existence reasoning validates the presence of an entity or relationship, and multi-hop queries are used to identify paths spanning multiple relationships. The negation category is built upon other claim categories with additional negation logic. Therefore, for negation claims, the `VERIFY` function incorporates appropriate negation operations during the result evaluation to ensure accurate verification. The program structure dynamically adjusts to the reasoning type by incorporating the appropriate functions and operations.

3.4 Program Generation

With the graph reasoning functions defined, we generate reasoning programs to conduct graph reasoning on fact verification task. Each program includes a sequence of operations aimed to extract evidence and finally verify the claim. Specifically, the program serves two primary roles: Evidence Retrieval and Claim Verification. The program utilizes `SEARCH` and `MATCH` functions to identify relevant subgraphs from the KG. By executing the `VERIFY` function, the program synthesizes the retrieved evidence to determine whether the claim is supported or refuted.

We adopt a prompt-based LLM approach to generate programs for knowledge graph reasoning. Given a claim, the LLM is prompted with carefully designed examples in a few-shot setting to generate a program composed of predefined graph reasoning functions. To improve the correctness of program generation, we design the prompt with a structured format consisting of four sequential components: (1) *Task Description*, (2) *Function Definition*, (3) *Examples*, and (4) *Content for Task Execution*. This structured prompt ensures clarity and consistency, guiding the model to generate the desired output more effectively. The detailed prompts can be found in Appendix A

3.5 Program Execution for Graph Reasoning

After generating the program, the graph reasoning process is completed by executing each function in the program. Different functions have distinct execution strategies, with their input, output, and functionality clearly defined according to their specifications.

The `SEARCH` function identifies the missing entity in the input triple by querying the knowledge graph. The `MATCH` function verifies whether a given triple exists in the KG. The prompts used in these functions are given in Appendix A. Finally, the `VERIFY` function aggregates all intermediate results to perform the final logical evaluation of the claim. By executing all functions in the program, PGR provides a step-by-step reasoning process, ultimately predicting the verification result for the claim. The algorithm details of `SEARCH` and `MATCH` are given in Appendix B.

To provide a more intuitive illustration of program execution, Figure 2 presents three examples, each showing a generated program alongside the corresponding claim graph structure. These examples demonstrate how the execution of program functions aligns with the underlying graph reasoning logic:

Multi-claim: In multi-claim example, the corresponding subgraph contains multiple triples, all of which are fully specified in the claim text. Thus, each triple can be directly verified using multiple steps of the `MATCH` function. The program iteratively matches each triplet in the KG, ensuring that all components of the claim are validated.

Multi-hop: Multi-hop claim requires reasoning across multiple connected entities in the KG. These claims include triples with missing entities, which must be identified before completing the verification process. The program uses the `SEARCH` function to locate the missing entity in the KG and subsequently applies the `MATCH` function to validate the remaining triples.

Negation: Negation claim introduces an additional layer of complexity due to the presence of negative logic. The corresponding KG structure often includes missing entities that must be identified through the `SEARCH` function. Once the relevant triples are retrieved, the program applies the `VERIFY` function with the ‘not‘ operation to perform logical negation.

Our framework is designed to be modular by construction, allowing different functions to be selectively composed depending on the task requirements. Furthermore, during program execution, the intermediate outputs of individual functions are explicitly accessible. This facilitates task-specific post-processing and enables transparent interpretation of the reasoning path, which is particularly beneficial in multi-stage or interactive applications. The modular nature of the framework supports reuse and extensibility, making it adaptable across various scenarios. Additional examples illustrating the modular usage can be found in Appendix E.

4 Experiment

4.1 Dataset

We evaluate the proposed Programmatic Graph Reasoning (PGR) method using the FACTKG dataset, a benchmark dataset designed for fact verification on knowledge graphs (KGs). FACTKG contains 108,674 claims, each written in natural language based on factual information from the DBpedia knowledge graph. Each instance in the dataset consists of a claim, the corresponding DBpedia entities mentioned in the claim, a binary label indicating whether the claim is `True` or `False`, and a reasoning type.

Our evaluation uses the FACTKG test set with 9,041 claims, including 4,398 labeled as `True` and 4,643 as `False`. The set covers diverse reasoning types: 1,914 *one-hop* claims, 3,069 *multi-claim (conjunction)* claims, 870 *existence* claims, 1,874 *multi-hop* claims, and 1,314 *negation* claims, making FACTKG a comprehensive benchmark for fact checking. Furthermore, we utilize the DBpedia 2015 knowledge graph as the evidence resource during the verification process. FACTKG allows us to test the effectiveness of the PGR method on the graph reasoning task with different claim complexities, ensuring a robust evaluation.

4.2 Baselines

We followed the baseline setup of Kim et al. (2023b) and categorized the baselines into two groups: *Without Evidence* and *With Evidence*. Based on training strategies, we further classify the methods into *full-training*, *zero-shot*, and *few-shot* settings. The *Without Evidence* baselines rely solely on the claim during inference without incorporating external evidence for decision-making.

For the *Without Evidence* baselines, we utilized three transformer-based text classifiers: **BERT**(Devlin et al., 2019), **BlueBERT**(Peng et al., 2019), and **Flan-T5**(Raffel et al., 2020; Chung et al., 2024), with experimental settings and results consistent with the approach of Kim et al. (2023b). All three models were trained under the *full-training* strategy. Additionally, we included a 12-shot ChatGPT baseline, using the settings and results of Kim et al. (2023a).

In the *With Evidence* category, we compared **GEAR**, **KG-GPT** and **ProgramFC**. GEAR is a model specifically designed and optimized by Kim et al. (2023b) for the FactKG task using the *full-training* strategy. KG-GPT(Kim et al., 2023a), on the other hand, leverages LLMs for reasoning over knowledge graphs in a *few-shot* setting, achieving strong performance in FactKG experiments. The original KG-GPT method employed the `gpt-3.5-turbo-0613` model as the LLM. In our experiments, we replaced it with the latest high-performance models—`gpt-4o-2024-08-06` (KG-GPT†) and `gpt-4o-mini-2024-07-18` (KG-GPT*)—while keeping other parameters consistent with the original paper. KG-GPT-*d* used an open-source LLM `Deepseek-V3`[3]. ProgFC refers to the **ProgramFC** method(Pan et al., 2023), which has been adopted to the FACTKG task. We used the graph extraction method similar to KG-GPT for evidence retrieval in ProgramFC function execution.

4.3 Implementation

We evaluate our proposed method, **PGR** (Programmatic Graph Reasoning), using the `gpt-4o-2024-08-06`[2] model to generate programs from claims. The program generation process adopts a 12-shot prompting strategy with carefully selected examples to guide the LLM. During program execution, the `MATCH` and `SEARCH` functions are implemented using the same `gpt-4o-2024-08-06` model to filter relevant knowledge graph triples. The prompting for these operations adopts a 4-shot strategy. The hyperparameters of LLM are set as follows: temperature = 0.2 and top-p = 0.1.

To evaluate the impact of LLM performance on the PGR method, we also use `gpt-4o-mini-2024-07-18`[2] as the underlying LLM, referred to as **PGR-mini**. **PGR-*d*** refers to the variant of our method using DeepSeek-V3[3], an open-source LLM, which is employed to evaluate

[2]https://platform.openai.com/docs/models
[3]https://www.deepseek.com

Methods	Evi.	Strategy	1-Hop	Existence	Mul. Hop	Mul. Claim	Nega.	Acc.
BERT	w/o	full	69.64	61.84	70.06	63.31	63.62	65.20
BlueBERT		full	60.03	59.89	57.79	60.15	58.90	59.93
Flan-T5		0-shot	62.17	55.29	60.67	69.66	55.02	62.70
ChatGPT		12-shot	-	-	-	-	-	68.48
GEAR	w/	full	83.23	81.61	68.84	77.68	79.41	77.65
ProgFC		12-shot	88.11	64.42	63.13	87.16	57.61	74.62
KG-GPT		12-shot	-	-	-	-	-	72.68
KG-GPT-*d*		12-shot	87.34	79.21	62.61	88.08	61.06	77.43
KG-GPT*		12-shot	80.56	63.19	55.84	69.93	59.19	67.04
KG-GPT†		12-shot	90.26	78.22	59.79	81.01	80.97	78.55
PGR-*d*		12-shot	90.41	89.17	73.14	89.32	78.24	84.18
PGR-mini		12-shot	87.36	78.60	57.82	77.31	57.28	72.62
PGR		12-shot	93.30	93.10	75.96	89.95	81.16	**86.82**

Table 1: The performance of the models on FACTKG. Models are categorized in Input Type(w/o or w/ Evidence) and Training Strategy.

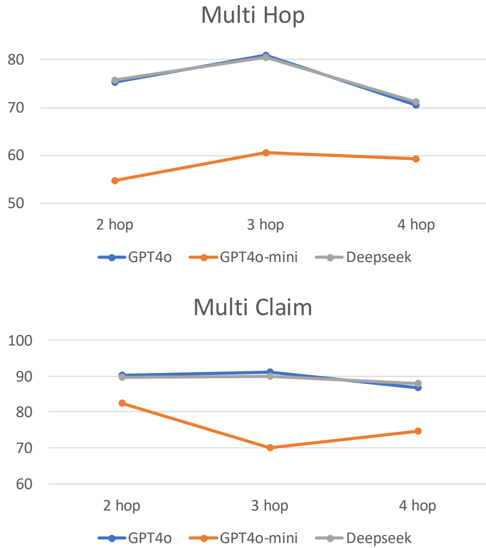

Figure 3: Results of Multi Hop and Multi Claim fact check on 2-hop, 3-hop and 4-hop.

the scalability of our approach. All other parameter settings remain consistent with the original PGR configuration. The cost of LLM used in our method is given in Appendix D.

4.4 Results

Table 1 presents the experimental results comparing the prediction accuracy of all models. Our proposed PGR model outperforms all baseline models, including both few-shot and full-training settings. Notably, PGR achieves the highest accuracy in the few-shot scenario and gets the best performance in all of the claim categories, which demonstrates that PGR is highly effective for KG-based reasoning and fact verification in few-shot settings.

When comparing PGR with KG-GPT† and PGR-mini with KG-GPT*, all under the same 12-shot condition and identical model configurations, PGR consistently achieves significant improvements in accuracy. Using the same LLM configuration, `gpt-4o-2024-08-06`, PGR surpasses KG-GPT† by 8.27% in overall accuracy. The performance gains are most notable in the multi-hop claim category, showing a 16.17% increase in accuracy. These results demonstrate that PGR effectively leverages LLMs for KG-based fact verification, particularly in complex reasoning tasks.

PGR performs effective graph reasoning in complex claims. As shown in Table 1, PGR achieves significant improvements over KG-GPT† in both multi-hop and multi-claim scenarios, which involve claims with complex graph structures. These results indicate that the proposed PGR method is more effective for complex graph reasoning tasks compared to existing baselines. In Figure 3, we further compare the accuracy of PGR, PGR-d and PGR-mini across different levels of graph reasoning complexity. For the multi-hop and multi-claim types, we categorize the claims based on the number of associated knowledge graph triples into 2-hop, 3-hop, and 4-hop categories, with increasing complexity. The accuracy of PGR decreases in the most complex 4-hop scenario, while PGR-mini shows consistently lower accuracy across different complexity levels. PGR achieves a stable improvement in accuracy for complex graph reasoning across varying levels of com-

Error Type	PGR			PGR-mini		
	multi claim	multi hop	negation	multi claim	multi hop	negation
Syntactic error	0%	0%	0%	0%	0%	0%
Variable error	0%	23%	10%	0%	11%	0%
Logical error	52%	26%	27%	93%	34%	39%
—-Graph	*45%*	*17%*	*22%*	*91%*	*34%*	*36%*
—-Verify	*7%*	*9%*	*5%*	*2%*	*0%*	*3%*
Execution error	48%	51%	63%	7%	55%	61%

Table 2: The Error examples of the models PGR and PGR-mini on FactKG.

plexity by enhancing the performance of the underlying LLM. We also conduct an experiment on the MetaQA dataset(Zhang et al., 2018) to verify the effectiveness of PGR and the results are given in Appendix E.

4.5 Interpretability

The interpretability of the PGR method is well demonstrated through the examples of program generation and corresponding graph structures in Figure 2. The generated program clearly decomposes multiple triples in a claim and connects unknown entities with other entities in multi-hop reasoning through variable passing. In the case of negation claims, PGR effectively captures the internal logical relations within the graph structure.

The readability of the generated program and the transparency of its logical reasoning process facilitate better understanding and more efficient error diagnosis.

Furthermore, the error analysis in Table 2 highlights that PGR leverages the improved capabilities of the underlying LLM to generate more accurate programs. The readability of these programs also makes it easier to identify specific errors, providing valuable insights for further optimization and improvement. This interpretability not only enhances the debugging process but also promotes deeper insights into the model's reasoning behavior, offering a foundation for future refinements.

4.6 Error analysis

To better analyze how different reasoning stages affect accuracy, we performed an error analysis focusing on three types of claims where PGR exhibited lower accuracy: multi-claim, multi-hop, and negation claims. Based on the types of errors observed during the reasoning process, we categorized them into four primary classes:

1. *Syntactic error* occurs when the generated program contains syntax issues, rendering it unexecutable. 2. *Variable error* arises from incorrect parameter variables in function arguments. 3. *Logical error* refers to incorrect reasoning logic in the graph structure formed by all functions in the program. Logical error can be further divided into two subcategories: graph error, which results from incorrect graph structures generated after claim decomposition, and verify error, which is caused by incorrect binary value computations in the `VERIFY` function. 4. Finally, *execution error* occurs when a syntactically and logically correct program fails during the execution stage, typically due to errors in KG search and entity matching.

As shown in Table 2, PGR demonstrates a significantly lower proportion of logical errors compared to PGR-mini, suggesting that improvements in LLM performance can enhance the logical correctness of program generation. Lower count of variable errors in PGR-mini is primarily a result of an increased number of logical errors, which reflects a shift in error types. Most errors in PGR arise during the execution stage, suggesting that despite the program's correct logical structure, further improvements are needed in the function execution process. Detailed error examples can be found in Appendix C.

5 Conclusion

In this paper, we propose a novel approach, Programmatic Graph Reasoning (PGR), for KG-based reasoning. Experimental results on the FactKG dataset for KG-based fact verification show that PGR achieves the highest accuracy among competing methods. PGR converts the claim verification process into a step-by-step graph reasoning process by generating executable programs composed of fundamental graph reasoning functions. Our approach effectively handles complex graph reasoning tasks while maintaining strong interpretability.

The program generation paradigm offers high

generalizability. By leveraging LLMs with few-shot prompting, PGR can perform graph reasoning tasks with minimal training resources. Unlike NLI approaches, PGR provides an explicit reasoning chain in the form of graph-based logic, ensuring a more interpretable and accurate reasoning process.

PGR introduces a modular design for KG-based reasoning tasks, offering a promising foundation for future extensions. The modularity of PGR enables flexible adaptation by reconfiguring the combination of core functions and customizing the handling of intermediate output. In future work, we plan to explore the applicability of this framework to more challenging reasoning settings.

Limitations

In our proposed method, the algorithm designed for program execution is relatively simple and does not account for more complex scenarios or potential optimizations. As indicated by the error analysis, this limitation contributes to execution errors and needs more improvement. Additionally, the few-shot learning approach used for program generation relies on a limited set of examples, which may not cover all claim types. This can lead to incorrect program generation when the LLM misinterprets the execution logic for certain unseen claim structures, resulting in errors. Future work could focus on refining and enhancing both the program generation strategy and the execution functions to improve robustness and generalization capability, further boosting the overall accuracy of the method.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 72210107001, and by the CAS PIFI International Outstanding Team Project (2024PG0013).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Eun Cheol Choi and Emilio Ferrara. 2024. Fact-gpt: Fact-checking augmentation via claim matching with llms. In *Companion Proceedings of the ACM Web Conference 2024*, WWW '24, page 883–886, New York, NY, USA. Association for Computing Machinery.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The webnlg challenge: Generating text from rdf data. In *10th International Conference on Natural Language Generation*, pages 124–133. ACL Anthology.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Jiho Kim, Yeonsu Kwon, Yohan Jo, and Edward Choi. 2023a. KG-GPT: A general framework for reasoning on knowledge graphs using large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9410–9421, Singapore. Association for Computational Linguistics.

Jiho Kim, Sungjin Park, Yeonsu Kwon, Yohan Jo, James Thorne, and Edward Choi. 2023b. FactKG: Fact verification via reasoning on knowledge graphs. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16190–16206, Toronto, Canada. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. 2015. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019.

Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Preslav Nakov, David P. A. Corney, Maram Hasanain, Firoj Alam, Tamer Elsayed, Alberto Barrón-Cedeño, Paolo Papotti, Shaden Shaar, and Giovanni Da San Martino. 2021. Automated fact-checking for assisting human fact-checkers. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 4551–4558. ijcai.org.

Liangming Pan, Xiaobao Wu, Xinyuan Lu, Anh Tuan Luu, William Yang Wang, Min-Yen Kan, and Preslav Nakov. 2023. Fact-checking complex claims with program-guided reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6981–7004, Toronto, Canada. Association for Computational Linguistics.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Trans. Knowl. Data Eng.*, 36(7):3580–3599.

Yifan Peng, Shankai Yan, and Zhiyong Lu. 2019. Transfer learning in biomedical natural language processing: An evaluation of BERT and ELMo on ten benchmarking datasets. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 58–65, Florence, Italy. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Haoran Wang and Kai Shu. 2023. Explainable claim verification via knowledge-grounded reasoning with large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 6288–6304, Singapore. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Yu Wang, Nedim Lipka, Ryan A Rossi, Alexa Siu, Ruiyi Zhang, and Tyler Derr. 2024. Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19206–19214.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*, 18(6):1–32.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6069–6076. AAAI Press.

Jie Zhou, Xu Han, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2019. GEAR: Graph-based evidence aggregating and reasoning for fact verification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 892–901, Florence, Italy. Association for Computational Linguistics.

Yuqi Zhu, Xiaohan Wang, Jing Chen, Shuofei Qiao, Yixin Ou, Yunzhi Yao, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2024. Llms for knowledge graph construction and reasoning: recent capabilities and future opportunities. *World Wide Web (WWW)*, 27(5):58.

A Prompts

The prompt of Graph Reasoning Program Generation is shown in Tabel 7. Table 8 and Table 9 show the prompts used in `SEARCH` funtion and `MATCH` function.

Our framework demonstrates strong performance under a few-shot setting, where the prompt for each reasoning type subcase includes only 0–3 in-context exemplars on average, as shown in Table 3. Despite the limited supervision, our model generalizes well across reasoning types. This result suggests that our framework is not only effective but also data-efficient and robust in handling diverse reasoning types.

B Execution of SEARCH and MATCH

Given an input triple $(head, relation, tail)$, where $head$ or $tail$ may be `None`, our system performs

Reasoning Type	# In-Context Examples	Reasoning Type (Negation)	# In-Context Examples
1-hop	2	negation / 1-hop	1
existence	1	negation / existence	0
multi-hop	3	negation / multi-hop	2
multi-claim	3	negation / multi-claim	0

Table 3: Average number of in-context exemplars per reasoning type and its negation counterpart.

entity retrieval or triple matching through two functions: SEARCH and MATCH. The corresponding execution processes are outlined in Algorithm 1 and Algorithm 2. In the algorithms, function REL_MATCH and TRI_MATCH will use prompts in Table 8 and Table 9 to prompt LLMs for searching and matching triples. To handle the intermediate function failure, if any intermediate function (MATCH or SEARCH) fails, the program immediately ends and determines the claim as refuted.

Algorithm 1: SEARCH

Input: Triple $(head, relation, tail)$, Graph G, Sentence s
Output: A list of triples matching the relation

```
1  if head ≠ None then
2  |   node ← head ;
3  else
4  |   node ← tail ;
5  edges ← all relations connected to node in G ;
6  if relation ∈ edges then
7  |   return all triples from G where node is
   |     connected via relation ;
8  relation_candidates ← all relation types
     from node in G ;
9  best_relation ← REL_MATCH(s,
10     (head, relation, tail),
11     relation_candidates) ;
12 return all triples from G where node is
     connected via best_relation ;
```

Algorithm 2: MATCH

Input: Triple $(head, relation, tail)$, Graph G, Claim s
Output: A triple in G matching the input triple

```
1  triple_candidates ← all triples with
     relations connecting head and tail in G ;
2  if (head, relation, tail) ∈
     triple_candidates then
3  |   return True ;
4  matched_triple ← TRI_MATCH(s,
5      (head, relation, tail),
6      triple_candidates) ;
7  if matched_triple is not None then
8  |   return True ;
9  return False;
```

C Error Examples

Figure 4 presents examples of different error types encountered during the fact verification process with PGR. These examples illustrate common issues in program generation and execution, categorized into parameter errors, logical errors, and execution errors.

Example (a) represents a parameter error, where the `MATCH` function incorrectly takes the pronoun "He" as an entity in its input, resulting in an execution failure.

Example (b) demonstrates a logical error involving an incorrect graph structure. The entities in the sentence are directly connected, indicating a multi-claim reasoning type. However, the generated program incorrectly applies the `SEARCH` function twice, creating a multi-hop structure inconsistent with the original claim.

Example (c) shows a logical error within the `VERIFY` function. Although no negation logic is present in the claim, the final `VERIFY` function incorrectly applies a negation using "not," leading to an incorrect binary judgment despite the preceding functions being executed correctly.

Finally, example (d) illustrates an execution error. The generated program logic and parameters are correct, and the `SEARCH` function successfully identifies the missing entity. However, during the `MATCH` step, the correct triple found in the knowledge graph is not recognized as a match, resulting in a false outcome.

These examples highlight key challenges in pro-

gram generation and execution, providing insights for future improvements.

D Cost Analysis

The cost of LLM is a concerning situation in practical usage. One practical advantage of our approach lies in its reduced training cost by leveraging LLMs as few-shot learning methods. However, it is important to note that the primary computational cost in LLM-based systems shifts to the inference stage closely tied to the tokens of input-output sequences. We give a statistical analysis about the average LLM cost of the PGR process on fact verification in Table 4. In our approach, the most cost of LLM is the usage of input tokens. For the prompt length limitation of LLM, we control the max length of input tokens less than 8192 to ensure the correct execution in program.

Metric	Value
Average LLM calls per claim	2.2
Average output tokens per claim	135
Average input tokens per claim	2590

Table 4: LLM usage statistics.

E MetaQA Experiment

We evaluated our PGR framework on the MetaQA dataset (Zhang et al., 2018), a widely used benchmark for multi-hop question answering over knowledge graphs. Specifically, we focus on the MetaQA 3-hop split, which requires reasoning over three connected triples to answer a single question. This setting poses significant challenges in multi-hop reasoning and thus serves as a suitable dataset to assess the effectiveness and modularity design of our method.

Given that MetaQA is formulated as a question answering task, we adapt our framework accordingly. To adapt PGR method on MetaQA task, only the output of the SEARCH function is used to generate the final answer. The preceding modules (e.g., MATCH) are not used in MetaQA task, as the task assumes a question mapped to an incomplete triple, where the answer is derived from completing the triple through entity retrieval.

Table 5 shows the performance of our method on the MetaQA 3-hop dataset. As shown, PGR approach achieves strong performance, demonstrating its capability to reason over multi-hop QA tasks and retrieve correct answer entities effectively. Notably, under the few-shot training setting, our method outperforms other methods, indicating its generalizability in low-resource scenarios.

Methods	Strategy	MetaQA 3-hop
EmbedKGQA	full	94.8
NSM	full	98.9
UniKGQA	full	99.1
KG-GPT	12-shot	94.0
PGR	12-shot	95.8

Table 5: Performance of different models on MetaQA 3-hop dataset. The metric is Hits@1 and methods are trained in different strategies.

F Entity Extraction and Disambiguation

We extend our framework to enable fact verification using only the claim sentence as input by incorporating a lightweight entity extraction and disambiguation module. Specifically, few-shot prompting is employed to extract candidate entity mentions from the claim. KG nodes are pre-encoded using a BERT model, and candidate entities are also transformed into vectors via BERT-based encoding. Cosine similarity is then used to match candidate mentions with the most relevant KG entities. The disambiguated entities are subsequently fed into the reasoning module, completing the verification pipeline.

Although the accuracy of automatically extracted entities is lower than using gold entities, our method achieves clear improvements over baselines in fact verification. Table 6 compares KG-GPT and our proposed PGR across different reasoning types.

These results indicate that even without gold entities, our approach remains competitive and achieves improvements over baselines. With the integration of entity extraction and disambiguation, the framework can support a complete and effective KG-based fact verification pipeline.

Method	1-hop	Existence	Multi-hop	Multi-claim	Negation	All
KG-GPT w/ ex.	78.18	68.61	55.64	63.83	67.40	66.14
KG-GPT	90.26	78.22	59.79	81.01	80.97	78.55
PGR w/ ex.	83.51	78.28	60.79	69.07	69.69	71.38
PGR	93.30	93.10	75.96	89.95	81.16	86.82

Table 6: Comparison of KG-GPT and our proposed PGR across reasoning types with entity extraction module(w/ ex.).

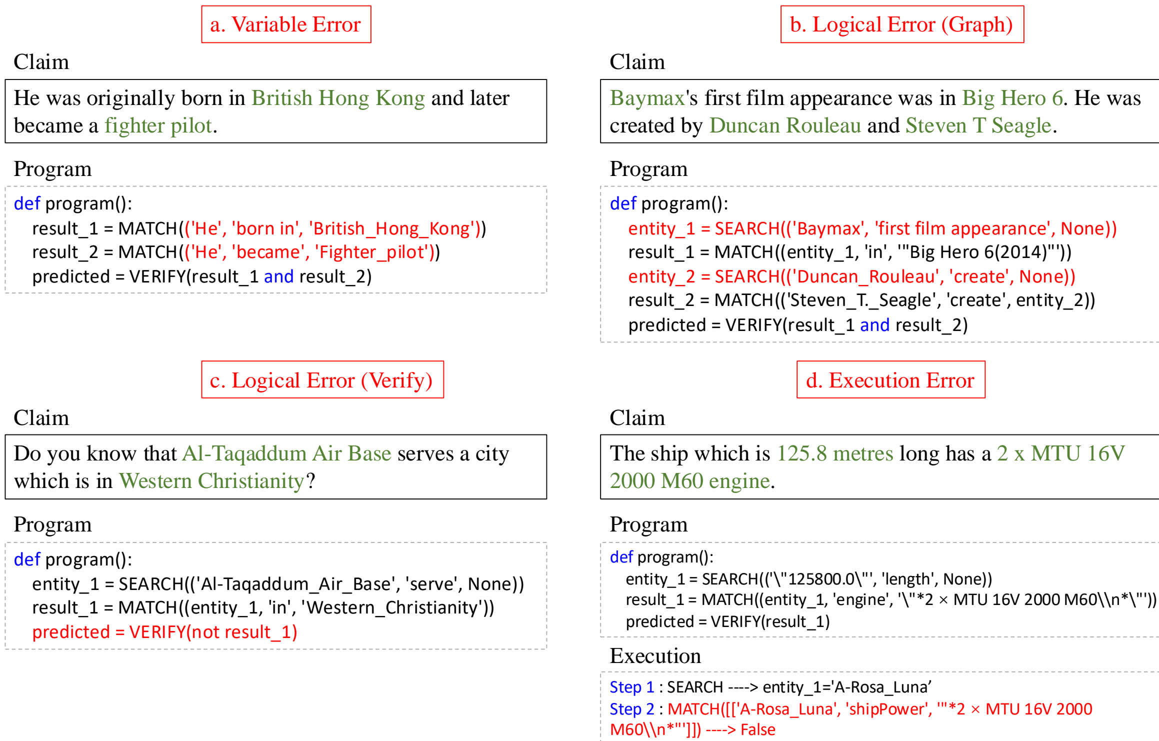

Figure 4: Error examples of PGR.

Please generate a program for the given sentence.
Use the 3 functions ’MATCH’, ’SEARCH’ and ’VERIFY’ in the program.
1. **MATCH**:
- Input: A complete triplet (entity, relation, entity), entity is given in the sentence or output of **SEARCH**.
- Functionality: Checks if the triplet exists in the graph.
- Output: Returns `True` if the triplet exists; otherwise, `False`.

2. **SEARCH**:
- Input: A triplet with one `None` value (missing entity).
- Functionality: Identifies the possible missing entity based on the other components.
- Output: A list of possible entities for the missing part.

3. **VERIFY**:
- Input: The result from the **MATCH** function (`True` or `False`).
- Functionality: Verify the final result.
- Output: Returns `True` or `False`.

Examples:
Abilene, Texas is part of Taylor County, Texas.
Entities = ["Abilene,_Texas", "Taylor_County,_Texas"]
–>Generated:
```
def program():
 result_1 = MATCH((’Abilene,_Texas’, ’part of’, ’Taylor_County,_Texas’))
 predicted = VERIFY(result_1)
```

Lake Placid, N.Y. is served by the Adirondack Regional Airport.
Entities = ["Adirondack_Regional_Airport", "Lake_Placid,_New_York"]
–>Generated:
```
def program():
 result_1 = MATCH((’Adirondack_Regional_Airport’, ’serve’,
 ’Lake_Placid,_New_York’))
 predicted = VERIFY(result_1)
```

...

The Ariane 5 was launched from the ELA-3 launchpad at the Guiana Space Centre.
Entities = ["ELA-3", "Ariane_5", "Guiana_Space_Centre"]
–>Generated:
```
def program():
 result_1 = MATCH((’ELA-3’, ’launch’, ’Ariane_5’))
 result_2 = MATCH((’ELA-3’, ’at’, ’Guiana_Space_Centre’))
 predicted = VERIFY(result_1 and result_2)
```

...

Your Task:
< < < <S> > > >
Entities = < < < <ENTITIES> > > >
–>Generated:

Table 7: Prompt of Graph Reasoning Program Generation

Please find the triplet in "Triplet List" that aligns with the "Target Triplet" of "Target Claim". Output the matched triplets in JSON format.

Examples:

Triplet List : [["William_Anders", "almaMater", ""AFIT, M.S. 1962""]]
Target Claim : An astronaut graduated with an M.S. in 1962 from AFIT, became a member of the Apollo 8 team and retired on September 1st, 1969.
Target Triplet : ["Unknown", "graduate", ""AFIT, M.S. 1962""]
–>Output : [["William_Anders", "almaMater", ""AFIT, M.S. 1962""]]

Triplet List : [["AZAL_PFK", "capacity", ""3500""], ["AZAL_PFK", "ground", "Arena"], ...]
Target Claim : AZAL PFK is in the league but Qarabag FK are the champions.
Target Triplet : ["AZAL_PFK", "in", "Unknown"]
–>Output : [["AZAL_PFK", "league", "Azerbaijan_Premier_League"]]

Triplet List : [["Alliant_Techsystems", "product", "XM25_CDTE"]]
Target Claim : They also produce the XM25 CDTE and make the ALV X-1.
Target Triplet : ["Unknown", "produce", "XM25_CDTE"]
–>Output : [["Alliant_Techsystems", "product", "XM25_CDTE"]]

Triplet List : [["Military_of_Paraguay", "country", "Paraguay"], ...]
Target Claim : Yes, he was born in Paraguay and died in Asuncion.
Target Triplet : ["Unknown", "born in", "Paraguay"]
–>Output : [["Alfredo_Stroessner", "birthPlace", "Paraguay"], ...]

Your Task:

Triplet List : < < < <LIST> > > >
Target Claim : < < < <CLAIM> > > >
Target Triplet : < < < <TARGET> > > >
–>Output :

Table 8: Prompt of SEARCH function.

Please find the triplet in "Triplet List" that aligns with the "Target Triplet" of "Target Claim". Output the matched triplets in JSON format.

Examples:
Triplet List : [["Ahmedabad", "country", "India"]]
Target Claim : There is a museum in Ahmedabad, India.
Target Triplet : [["Ahmedabad", "in", "India"]]
–>Output : [["Ahmedabad", "country", "India"]]

Triplet List : [["Alfredo_Zitarrosa", "birthPlace", "Uruguay"], ...]
Target Claim : Alfredo Zitarrosa died in a country led by Tabaré Vázquez.
Target Triplet : [["Alfredo_Zitarrosa", "died in", "Uruguay"]]
–>Output : [["Alfredo_Zitarrosa", "deathPlace", "Uruguay"]]

Triplet List : [["1101_Clematis", "Planet/apoapsis", "5̈.20906E8"̈]]
Target Claim : well it has a mass of 5.7 kilograms and an apoapsis of 520906000.0 km.
Target Triplet : [["1101_Clematis", "apoapsis", "5̈.20906E8"̈]]
–>Output : [["1101_Clematis", "Planet/apoapsis", "5̈.20906E8"̈]]

Triplet List : [["Alfredo_Zitarrosa", "associatedBand", "Ciro_Pérez"], ...]
Target Claim : Alfredo Zitarrosa is related to the musician Ciro Pérez.
Target Triplet : [["Alfredo_Zitarrosa", "related to", "Ciro_Pérez"]]
–>Output : [["Alfredo_Zitarrosa", "associatedBand", "Ciro_Pérez"], ...]

Your Task:
Triplet List : < < < <LIST> > > >
Target Claim : < < < <CLAIM> > > >
Target Triplet : < < < <TARGET> > > >
–>Output :

Table 9: Prompt of MATCH function.