

DuoKGQA: A Novel Dual-Module Framework for Cypher Query Generation in Knowledge Graph Question Answering

You Li Chong

Multimedia University

Chin Poo Lee

Chin-Poo.Lee@nottingham.edu.cn

University of Nottingham Ningbo China

Kian Ming Lim

University of Nottingham Ningbo China

Research Article

Keywords: knowledge graph, question answering, language model, deep learning, machine learning

Posted Date: October 27th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-6900829/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

DuoKGQA: A Novel Dual-Module Framework for Cypher Query Generation in Knowledge Graph Question Answering

You Li Chong^{1*}, Chin Poo Lee^{2*} and Kian Ming Lim^{2*}

^{1*}Faculty of Information Science and Technology, Multimedia University,
Jalan Ayer Keroh Lama, Ayer Keroh, 75450, Melaka, Malaysia.

^{2*}School of Computer Science, University of Nottingham Ningbo China,
199 Taikang East Road, Yinzhou District, Ningbo, 315100, Zhejiang
Province, China.

*Corresponding author(s). E-mail(s): 1181101060@student.mmu.edu.my;
Chin-Poo.Lee@nottingham.edu.cn; Kian-Ming.Lim@nottingham.edu.cn;

Abstract

The exponential growth of digital information and the complexity of academic inquiries demand advanced question answering systems capable of interpreting diverse natural language queries. This paper introduces DuoKGQA, a dual-module Knowledge Graph Question Answering (KGQA) system for generating precise Cypher queries in the academic domain. DuoKGQA leverages state-of-the-art natural language processing techniques, including transformer-based models, sentence embeddings, and deep learning architectures, to enhance query understanding and accuracy. The system comprises two key modules: the Question Type Classification Module, which uses RoBERTa embeddings and an ensemble of BiGRU and CNN-BiGRU models with weighted majority voting for precise query classification, and the Program Name Identification Module, which employs MiniLM embeddings and cosine similarity to resolve program names in Cypher templates. These modules integrate seamlessly to form executable Cypher queries processed on a Neo4j knowledge graph. Experimental evaluations demonstrate DuoKGQA's effectiveness, achieving 0.946 accuracy in question type classification and 0.890 exact match score for overall query generation. DuoKGQA provides a robust solution for academic question answering and establishes a foundation for future advancements in KGQA systems.

Keywords: knowledge graph, question answering, language model, deep learning, machine learning

1 Introduction

The rapid growth of digital information and increasingly complex user queries have driven significant advancements in question answering (QA) systems. These systems aim to deliver accurate and contextually relevant responses from vast and diverse datasets, encompassing structured and unstructured information. However, traditional retrieval-based and generative QA models often struggle with interpreting complex queries requiring reasoning over structured data. To address these limitations, Knowledge Graph Question Answering (KGQA) has emerged as a promising paradigm, leveraging structured knowledge representations to improve the accuracy, interpretability, and scalability of QA systems. The core techniques of these systems often involve a pipeline of identifying entities, disambiguating them, and finding the correct relations, which has been a primary focus of research in the field [1].

Knowledge graphs represent data as interconnected entities and relationships, enabling reasoning over entity interactions. Despite progress, significant technical challenges persist in their development, particularly in resolving contextual ambiguities in user queries [2]. This issue is especially pronounced in academic domains, where queries span diverse topics such as admissions, scholarships, and university services, often involving intricate relationships and dependencies. Query ambiguity remains a significant challenge in knowledge graph question answering systems, where limited contextual information can deteriorate user experience [3].

Current KGQA systems frequently face difficulties in interpreting natural language queries and capturing the user’s true intent due to the inherent ambiguity of unstructured questions [4]. Research has identified two major sources of this ambiguity: intent ambiguity and entity ambiguity [5]. Intent ambiguity occurs when questions with similar linguistic structures convey different meanings. For instance, “What is the structure of the data science program?” and “What is the fee structure for the data science program?” may appear similar but require entirely different query formulations. Entity ambiguity, meanwhile, complicates the accurate identification of specific academic entities, particularly when users refer to programs or services using varied, informal, or incomplete phrasing. These ambiguities present a significant challenge for KGQA systems, especially in academic domains where precision is critical.

To tackle these challenges, this study introduces DuoKGQA, a dual-module system tailored for academic domain queries. DuoKGQA interprets complex natural language queries and transforms them into precise Cypher queries for efficient knowledge graph retrieval. The system integrates two complementary modules. The Question Type Classification Module identifies the query category using transformer-based models like RoBERTa to encode input sequences. These representations are processed by deep learning architectures, including BiGRU and CNN-BiGRU, ensuring accurate classification and appropriate Cypher query template selection. The Program Name Identification Module extracts program-related information critical for academic queries. It employs a fine-tuned Sentence Transformer to generate embeddings that are compared with predefined program names via cosine similarity, ensuring precise entity recognition.

Combining these modules, DuoKGQA generates executable Cypher queries executed on a Neo4j knowledge graph enriched with academic data such as admission

criteria, tuition fees, and program structures. This dual-module architecture enhances query understanding and entity recognition, enabling accurate responses to a wide range of academic queries. The primary contributions of this work are summarized as follows:

- **Domain-Specific Knowledge Graph Development:** A comprehensive academic knowledge graph is constructed and enriched through systematic data collection, preprocessing, and relationship extraction, forming a robust foundation for KGQA.
- **Advanced Question Type Classification:** The RoBERTa-based Question Type Classification Module captures semantic intricacies of user queries using its transformer-based architecture, and an ensemble approach combining BiGRU and CNN-BiGRU enhances classification accuracy.
- **Precision in Program Name Identification:** The Program Name Identification Module leverages MiniLM embeddings and cosine similarity to precisely match queries with relevant academic programs, ensuring accurate entity resolution.
- **Unified DuoKGQA Framework:** The integration of the classification and identification modules results in a cohesive system capable of transforming natural language queries into Cypher queries, optimized for execution against the Neo4j knowledge graph.

The remainder of this paper is organized as follows. Section 2 reviews existing literature on KGQA, focusing on advancements in query understanding, entity linking, and machine learning techniques. Section 3 details the data preparation, including corpus construction, knowledge graph development, and Cypher query formulation. The architecture of the proposed DuoKGQA system is described in Section 4. Experimental results and analyses are presented in Section 5, followed by a summary of contributions and potential future directions in Section 6.

2 Related Work

This section summarizes advancements in KGQA, focusing on methodologies leveraging knowledge graphs for query interpretation and response generation, with emphasis on graph reasoning, neural models, and query generation techniques.

Cui et al. [6] introduced the Dynamic Reasoning Network (DRN) for multi-hop KGQA, framing it as a stepwise relation extraction problem. DRN employs attention mechanisms and dynamic question representation to reduce search complexity, achieving Hits@1 scores exceeding 90% across multiple datasets, including WC2014 and MetaQA.

Du et al. [7] proposed a contrastive learning framework to mitigate exposure bias in KGQA. By augmenting logical forms and employing contrastive objectives, their method improved accuracy on KQA Pro, OVERNIGHT, and WebQSP, achieving up to 93.65%.

Jiang et al. [8] presented ReasoningLM, a language model that integrates subgraph-aware self-attention for structural KG reasoning. Adaptation and fine-tuning strategies

enabled strong performance on WebQSP, ComplexWebQuestions, and MetaQA, with Hits@1 scores above 90% on MetaQA.

Luo et al. [9] implemented ChatKBQA, a modular generate-then-retrieve framework using fine-tuned large language models. By integrating unsupervised retrieval for efficiency, the system achieved F1-scores of 83.50% and 81.30% on WebQSP and ComplexWebQuestions, respectively.

Wang et al. [10] introduced Hic-KGQA, leveraging hypergraph-based techniques and inference chain modeling for multi-hop KGQA. The approach demonstrated competitive Hits@1 scores across MetaQA, WebQSP, and ComplexWebQuestions datasets.

Wang et al. [11] proposed Keqing, a KBQA framework using logical chains as guides for large language models (LLMs). The workflow involves decomposing complex questions into sub-questions, retrieving candidates along logical chains, reasoning over retrieved triplets, and generating responses. By aligning sub-questions with logical chains, Keqing facilitates effective multi-hop reasoning, achieving Hits@1 scores of 98.40%, 99.90%, and 99.60% for 1, 2, and 3-hop MetaQA tasks, and an F1-score of 74.90% on WebQSP.

Wang et al. [12] introduced RKEKGQA, a multi-hop KGQA method enhancing relational reasoning. The framework incorporates knowledge aggregation, encoding, text interaction, and multi-hop reasoning. It employs global and local attention mechanisms for effective entity-text interaction and relational scoring. RKEKGQA achieved Hits@1 scores of 97.30%, 99.10%, and 98.40% on 1, 2, and 3-hop MetaQA tasks, and 71.60% and 49.20% on WebQSP and ComWebQ datasets, respectively.

Xu et al. [13] developed BigText-QA, which integrates information extraction and retrieval within a hybrid knowledge graph framework. BigText represents documents as interlinked graphs with syntactic and semantic dependencies. For QA, it constructs subgraphs, translates them into quasi-graph structures, and applies the Group Steiner Tree algorithm for answer ranking. BigText-QA achieved mean reciprocal ranks of 0.398 and 0.412 on CQ-W and TriviaQA datasets, respectively.

Zuo et al. [14] proposed a Chunked Learning Network (CLN) for KGQA, comprising modules for pruning, head-entity learning, relation learning, and answer selection. The pruning module uses Bi-LSTM for entity recognition to reduce the search space, while the head-entity and relation-learning modules employ Bi-SRU, CNN, and Graph Convolutional Networks with attention mechanisms for feature fusion. The answer selection module ranks triplets based on joint distance metrics. CLN achieved an accuracy of 75.3% on the SimpleQuestions dataset.

Faldu et al. [15] developed RetinaQA, a KBQA model addressing incomplete KBs through a retrieve, generate, and rank approach. Logical forms are generated via KB-traversal and sketch-filling methods, then ranked using a T5 encoder-decoder. RetinaQA effectively handles both answerable and unanswerable questions, achieving F1-scores of 64.68% on GrailQAbility and 83.33% on GrailQA, and 75.30% on WebQSP for answerable datasets.

Formica et al. [16] implemented a supervised template-based QA system with steps for feature extraction, classification, entity recognition, and SPARQL query generation. The system utilizes syntactic and semantic features, achieving an F1-score of

1.00 on the CSQA dataset when combining both feature types, demonstrating robust performance across various question categories.

Kang et al. [17] proposed the Knowledge-Aware Adaptive Graph Network (KA-AGN), a model for commonsense QA integrating semantic understanding, subgraph construction, adaptive graph networks, and joint reasoning. KA-AGN combines dependency parse trees with subgraphs from ConceptNet using Relation Graph Convolutional Networks (RGCN) and dense bidirectional attention for reasoning. The model achieved 78.97% accuracy on CommonsenseQA and 95.98% on OpenBookQA with AristoROBERTa.

Li et al. [18] introduced FlexKBQA, a framework for few-shot KBQA utilizing LLMs and structured query templates. The approach combines program sampling, low-resource program translation, and execution-guided self-training (EGST) to address distribution shifts between synthetic and real queries. FlexKBQA achieved F1-scores of 63.80% on GrailQA, 60.60% on WebQSP, and an accuracy of 46.83% on KQA Pro.

Tian et al. [19] proposed Graph Neural Prompting (GNP), leveraging knowledge graphs to create soft prompts for pre-trained LLMs. GNP integrates GNN encoders, cross-modality pooling, domain projectors, and link prediction to align graph and text representations. Tested on six datasets, GNP achieved top accuracies, including 89.60% on OpenBookQA, 78.71% on ARC, and 94.31% on BioASQ.

Thambi et al. [20] proposed enhancing KGQA systems using GNNs with relevance scoring. The approach constructs question-specific subgraphs, incorporates weighted nodes and edges, and prunes irrelevant nodes. By employing BERT-based GNNs, subgraph embeddings are generated, and entities are ranked using cosine similarity. Among various GNN models, GENConv performed best, achieving 98.50%, 96.97%, and 95.30% for 1, 2, and 3-hop MetaQA tasks, and 74.64% on WebQSP.

Zhang et al. [21] introduced MRGNN, a multi-hop reasoning model for commonsense QA that fuses meta-paths and combines language models with knowledge graphs. Meta-paths align structural knowledge with QA contexts, enhancing dynamic interactions through multi-level attention graphs. Benchmarked on CommonsenseQA and OpenBookQA, MRGNN achieved 77.80% accuracy on CommonsenseQA and 83.60% on OpenBookQA using RoBERTa-based models.

Chong et al. [22] presented TransKGQA, leveraging Sentence Transformers for improved question encoding in KGQA tasks. Questions are encoded, compared via cosine similarity, and matched to Cypher queries executed in Neo4j. The best-performing model, "sentence-transformers/all-MiniLM-L12-v2," achieved an F1-score of 0.78 on a university corpus, demonstrating the efficacy of semantic encoding for KGQA.

Guo et al. [23] introduced LightRAG, which enhances standard RAG by embedding graph structures into both retrieval and generation. It extracts entities and relations using LLMs, constructs a knowledge graph, and applies deduplication for efficient entity-level retrieval. A dual-level retrieval mechanism supports both fine-grained and abstract information. LightRAG achieved 86.4% win rate in diversity and 84.8% in overall quality on a Legal dataset, addressing limitations of flat indexing and limited contextual reasoning.

Sanmartin [24] integrates structured knowledge graphs into RAG to improve multi-hop reasoning and factual accuracy (KG-RAG). It converts text into KGs with LLM-extracted triples, including nested “triple hypernodes.” Retrieval is guided by a Chain of Explorations (CoE), combining Cypher queries, vector search, and LLM reasoning. GPT-4 Turbo then generates grounded answers. On CWQ, KG-RAG reached 49.00% EM, 58.90% F1, and 69.70% Accuracy, with only 6.70% hallucination.

Li et al. [25] presented SubgraphRAG, a scalable RAG framework that retrieves relevant KG subgraphs using a lightweight MLP with Directional Distance Encoding. It avoids fine-tuning, supports multiple LLMs (e.g., GPT-4o, LLaMA3), and produces explainable, structured prompts. On WebQSP and CWQ-sub, GPT-4o achieved Macro-F1 scores of 77.61 and 65.99, outperforming RoG and GNN-RAG while reducing hallucination and maintaining cross-dataset generalization.

Mavromatis et al. [26] combines GNNs and RAG to support multi-hop reasoning in KGQA (GNN-RAG). GNNs process subgraphs to identify answer paths, which are then verbalized and fed into fine-tuned LLMs (e.g., LLaMA2-Chat-7B). The model captures node and relational features and integrates GNN-LLM outputs. On WebQSP and CWQ, GNN-RAG with retrieval augmentation achieved Hits@1 of 82.80 and F1-score of 62.80.

2.1 Research Gaps

Despite advancements in KGQA systems, challenges persist in addressing complex query structures and entity disambiguation, particularly in academic domains. Existing approaches often struggle with queries involving intricate relationships between multiple entities and properties, leading to suboptimal performance. Academic queries spanning admissions, scholarships, and program structures exacerbate these limitations, reducing system applicability in real-world scenarios.

A key limitation lies in entity linking. Many systems rely on heuristic-based methods or predefined templates, which are insufficient for capturing the variability of natural language expressions in academic queries. For example, identifying specific programs or admission conditions becomes difficult when users phrase them informally or inconsistently. While neural-based models offer greater flexibility, they often ignore domain-specific constraints, resulting in poor generalization and reduced precision [4].

Recent models such as the Dynamic Reasoning Network (DRN) [6] and RKEKGQA [12] have advanced multi-hop reasoning by focusing on relation path prediction. However, these systems are still sensitive to errors in initial entity linking, which can propagate through the reasoning chain. Logical form generation approaches [7, 9] attempt to translate natural language into formal queries, but they tend to be brittle and highly dependent on generating complete and correct logical forms, which remains difficult for nuanced academic questions. Template-based methods like that of [16] offer precision within fixed query patterns but lack the adaptability to handle diverse, domain-specific phrasings.

Efforts to ground KGQA responses in retrieved knowledge, such as KG-RAG [24], LightRAG [23], and SubgraphRAG [25] have improved factual consistency. Yet, these methods rely heavily on retrieval quality. If relevant subgraphs are missed, the generated answers may still be incomplete or incorrect. Graph-based approaches like

GNN-RAG [26] and [20] enhance structural reasoning but face computational overhead and difficulties with fine-grained entity distinctions. Even models like TransKGQA [22], which use Sentence Transformers for better encoding, depend on fixed similarity thresholds, which can fail in dense or semantically subtle academic graphs.

The proposed DuoKGQA system addresses these gaps with a dual-module architecture. The Question Type Classification Module combines RoBERTa embeddings with an ensemble of models, including BiGRU, CNN-BiGRU, and CNN-BiLSTM, to capture local and long-range dependencies, improving query interpretation. The Program Name Identification Module leverages fine-tuned Sentence Transformers to generate dense embeddings compared via cosine similarity, enabling precise entity recognition and reducing ambiguity. By integrating advanced query classification and entity recognition techniques, DuoKGQA enhances accuracy and scalability for complex academic queries, advancing KGQA performance in domain-specific applications.

3 Corpus Collection and Preprocessing

The corpus collection and preprocessing process involves multiple stages, beginning with raw data collection from the official university website and culminating in a structured dataset suitable for knowledge graph construction and machine learning tasks. This pipeline ensures data consistency, accuracy, and adaptability to support the requirements of KGQA systems. Fig. 1 illustrates the complete pipeline for corpus collection and preprocessing.

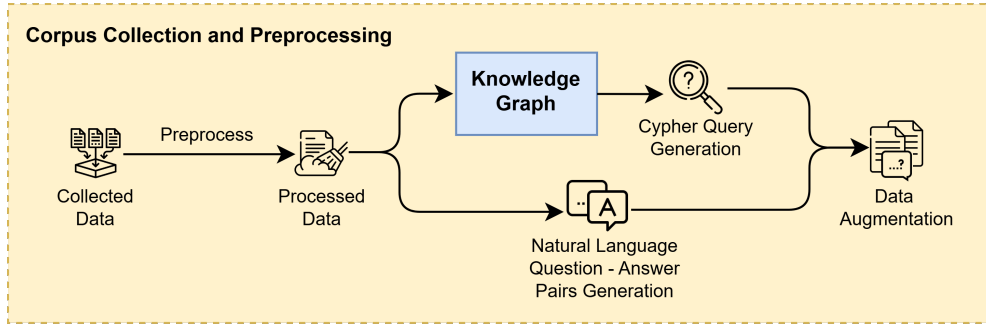


Fig. 1: Corpus collection and preprocessing pipeline. The diagram outlines the workflow for collecting raw data, preprocessing it into a structured format, and utilizing it for creating a knowledge graph, generating question-answer pairs, and augmenting the dataset for deep learning.

The corpus was sourced from publicly accessible pages on the university’s official website. Automated web scraping techniques were employed to streamline the data collection process, ensuring efficiency, consistency, and minimization of human errors. The scraping process adhered to ethical guidelines by limiting operations to publicly available web pages and implementing rate limits to avoid overburdening the university’s servers.

Fig. 2 presents the automated web scraping workflow. A custom script, developed using Selenium’s WebDriver along with Python libraries such as Requests and BeautifulSoup, systematically navigated the website to extract information from various links. Relevant details, including program names, faculties, descriptions, tuition fees, duration, entry requirements, and scholarship opportunities, were compiled into a structured dataframe. This initial dataset, saved as a CSV file, formed the foundation for subsequent preprocessing and knowledge graph construction.

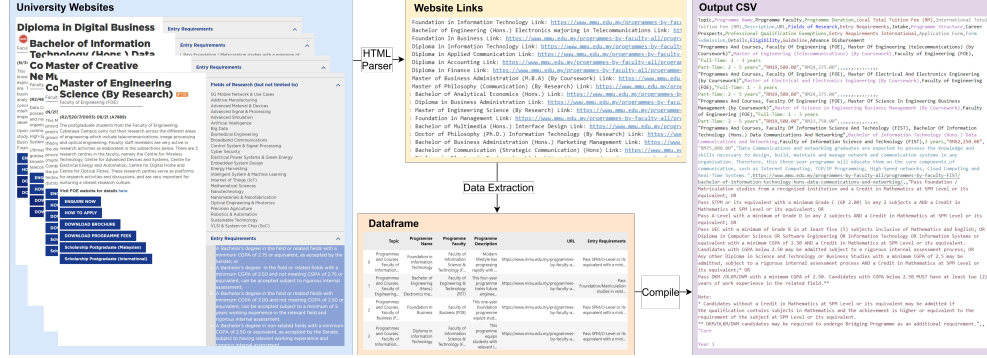


Fig. 2: Automated web scraping process. The workflow demonstrates how program-related information, such as program names, faculties, entry requirements, duration, and fees, is extracted, organized into a structured dataframe, and exported as a CSV file for further processing.

To ensure uniformity and accuracy across the dataset, a comprehensive standardization step was undertaken. Key entities, such as program names, faculty titles, and course codes, were normalized to maintain consistency in terminology. Following standardization, a manual consistency check was conducted, cross-referencing the preprocessed data with the original website content to identify and rectify discrepancies. This rigorous approach minimized errors and ensured that the dataset accurately represented the academic offerings of the university.

The cleaned and standardized dataset served as the foundation for constructing a domain-specific knowledge graph. This graph represents the university’s academic offerings as interconnected entities and relationships, facilitating structured query answering. Additionally, natural language question-answer pairs were generated from the dataset to support KGQA tasks. To meet the extensive data requirements of deep learning models, both question-answer pairs and Cypher queries were augmented using data augmentation techniques. This augmentation expanded the dataset’s diversity and size, enhancing the robustness of the KGQA model during training and evaluation.

Ethical compliance was prioritized throughout the data collection process. The scraping operations were restricted to publicly accessible web pages, and rate-limiting mechanisms were implemented to prevent potential disruptions to the university’s servers. By adhering to these measures, the data collection process ensured respect for institutional resources and maintained transparency.

This comprehensive pipeline for corpus collection and preprocessing addresses the challenges of data quality and consistency, providing a robust foundation for constructing the knowledge graph and training the KGQA system.

3.1 Natural Language Question-Answer Pairs Generation

The generation of natural language question-answer pairs is a critical step in developing an effective question answering system. This process involves crafting questions and their corresponding answers based on the information in the collected corpus. The questions are designed to encompass diverse aspects of academic programs, addressing typical queries from prospective students, parents, and academic advisors. By aligning the questions with common user needs, the system ensures relevance and practicality.

The collected corpus is categorized into two primary groups: program-specific and non-program-specific questions. This classification facilitates comprehensive coverage of inquiries, enabling the system to address both detailed program-related and general academic questions.

Program-specific questions focus on 10 distinct aspects of academic programs, providing users with detailed insights into various program attributes:

- **Program Description:** Questions about the overall structure, objectives, and learning outcomes of the program.
- **Entry Requirements:** Queries regarding academic qualifications, test scores, or other criteria needed for admission.
- **Application Process:** Questions on the steps and deadlines for applying to a program.
- **Career Prospects:** Inquiries about potential career paths and job opportunities available to graduates.
- **Research Field:** Questions concerning focus areas and research opportunities within specific programs.
- **Intake Information:** Queries about program intake dates and schedules.
- **Tuition Fees:** Questions on the total cost of tuition and associated financial obligations.
- **Program Duration:** Queries about the time required to complete the program.
- **Faculty Offering the Programs:** Questions about the department or faculty responsible for delivering the program.
- **Program Structure:** Inquiries regarding the curriculum, including required courses, electives, and specialization options.

Non-program-specific questions address general inquiries relevant to all programs, with a focus on 5 broad categories:

- **PTPTN:** Queries about the Malaysian government education loan scheme.
- **Scholarship:** Questions concerning available scholarships and financial aid options.
- **Loyalty Program:** Inquiries into special benefits or programs for alumni or returning students.
- **B40:** Questions related to financial aid or assistance for students from the bottom 40% income group in Malaysia.

- **Deadlines:** Queries on application and scholarship deadlines.

These question types serve as templates for generating natural language questions that align with users’ specific information needs. Each template is designed to capture frequent inquiries, ensuring that the resulting question-answer pairs are both contextually relevant and practical.

To ensure comprehensive coverage, the generation process systematically iterates through each question type for all relevant entities in the corpus. By leveraging this structured approach, the system creates a diverse and extensive dataset that supports accurate and efficient information retrieval. This dataset is integral to training and evaluating the KGQA model, enabling it to deliver precise and contextually appropriate responses to a wide range of user queries.

3.2 Knowledge Graph Creation

The creation of a knowledge graph is a fundamental step in systematic knowledge representation, complementing the natural language question-answer pairs. The knowledge graph encapsulates the structured relationships between entities, enabling efficient information retrieval for the question answering system. Neo4j serves as the backbone for generating the knowledge graph from the preprocessed corpus, transforming raw data into meaningful graph-based structures.

The preprocessed corpus is converted into a series of triplets, where each triplet represents an entity-relationship-entity tuple. These triplets form the foundational building blocks of the knowledge graph. Fig. 3 provides illustrative examples of triplets related to academic programs and their associated local fees. Each triplet comprises a “programme_name” entity, a “local_fees” entity, and a “costs” relationship that links them. This structure establishes a direct association between academic programs and their corresponding fees, demonstrating how key attributes are systematically interconnected.

These triplets are used to construct a comprehensive knowledge graph in Neo4j, where entities are represented as nodes and relationships are depicted as directed edges. Each edge is labeled to indicate the type of relationship between the nodes. For example, relationships such as “costs”, “links”, and “entry requirements” connect entities like program names, fees, and admission criteria. Fig. 4 showcases a visual representation of a portion of the knowledge graph, generated using Neo4j Bloom. The visualization highlights the interconnected nodes and their labeled relationships, providing an intuitive understanding of the graph structure.

In the knowledge graph, nodes represent specific entities such as academic programs, faculties, tuition fees, durations, and entry requirements. The complete knowledge graph encompasses 852 nodes and 1,095 relationships, effectively representing the rich and intricate relationships inherent in the dataset. This graph structure enables seamless information retrieval, serving as a robust foundation for the question answering system. By organizing data into interconnected nodes and edges, the knowledge graph facilitates efficient query execution and ensures high accuracy in providing contextually relevant answers to user queries.

```

{'local_fees': 'RM6,750.00', 'programme_name': 'Foundation in Management',
'relation_type': 'costs'}
{'local_fees': 'RM73,000.00', 'programme_name': 'Bachelor of Accounting
(Hons.)', 'relation_type': 'costs'}
{'local_fees': 'RM58,170.00', 'programme_name': 'Bachelor of Financial
Engineering (Hons.)', 'relation_type': 'costs'}
{'local_fees': 'RM76,000.00', 'programme_name': 'Bachelor of Engineering
(Hons.) Electronics majoring in Robotics & Automation', 'relation_type':
'costs'}
{'local_fees': 'RM3,000.00/year', 'programme_name': 'Doctor of Philosophy
(Ph.D.) Engineering (By Research)', 'relation_type': 'costs'}
{'local_fees': 'RM75,000.00', 'programme_name': 'Bachelor of Law (Hons.)',
'relation_type': 'costs'}
{'local_fees': 'RM43,300.00', 'programme_name': 'Doctor of Business
Administration (By Coursework and Dissertation)', 'relation_type': 'costs'}
{'local_fees': 'RM15,400.00', 'programme_name': 'Master of Computer Science
in Software Engineering via Coursework (Online & Distance Learning)',
'relation_type': 'costs'}
{'local_fees': 'RM22,000.00', 'programme_name': 'Master of Creative
Multimedia (By Coursework)', 'relation_type': 'costs'}
{'local_fees': 'RM18,750.00', 'programme_name': 'Foundation in Law',
'relation_type': 'costs'}

```

Fig. 3: Examples of triplets connecting “programme_name” nodes to “local_fees” nodes through the “costs” relation type.

3.3 Cypher Query Generation

Cypher is a declarative query language specifically designed for interacting with Neo4j knowledge graphs. In this study, Cypher queries are utilized to extract information from the constructed knowledge graph, enabling precise and contextually relevant responses to user queries. Cypher queries adhere to a structured format comprising clauses such as *MATCH*, *WHERE*, and *RETURN*, which collectively facilitate the retrieval of targeted information.

To bridge the gap between natural language questions and Cypher queries, a set of 715 unique Cypher queries was constructed, corresponding to typical user inquiries. These queries retrieve specific properties from the knowledge graph based on user-defined conditions.

The structured approach ensures that each query is precise, leveraging the inherent relationships and attributes within the knowledge graph. This methodology not only supports accurate information retrieval but also enables the KGQA system to provide contextually appropriate responses to a wide range of user inquiries.

3.4 Data Augmentation

A diverse and extensive dataset of natural language questions is essential for training a robust language model for KGQA. Initially, the dataset comprised 2,240 questions spanning 15 distinct categories, addressing various informational needs within the knowledge graph. However, this dataset was insufficient for generalizing across the

Table 1: Number of Questions in Each Question Type Before and After Data Augmentation

No.	Question Type	Number of Question	
		Before	After
1	Programme Description	158	790
2	Entry Requirements	141	705
3	Application Process	332	1660
4	Career Prospects	120	600
5	Research Field	109	545
6	Intake Information	125	625
7	Tuition Fees	110	550
8	Programme Duration	168	840
9	Faculty Offering the Programmes	62	310
10	Programme Structure	172	860
11	PTPTN	70	350
12	Scholarship	587	2935
13	Loyalty Programme	43	215
14	B40	29	145
15	Deadlines	14	70
TOTAL		2240	11200

the Program Name Identification Module, each addressing specific aspects of query processing.

The Question Type Classification Module identifies the structure and intent of user queries. Using RoBERTa embeddings, it captures semantic information and processes it through an ensemble of BiGRU and CNN-BiGRU models to extract diverse features. The ensemble output determines the question type, which is then used to select a Cypher query template for non-program-specific queries or as a basis for further processing in program-specific queries.

For program-specific queries, the Program Name Identification Module identifies the relevant program name. It generates query embeddings using Sentence Transformers, compares them with precomputed program name embeddings via cosine similarity, and identifies the closest match. This ensures accurate integration of program names into Cypher queries.

The outputs of both modules are combined to form a complete Cypher query, tailored to the user’s query type and program context. The final query is executed on a Neo4j knowledge graph, delivering precise answers. DuoKGQA’s dual-module design enables it to handle diverse query types with high accuracy and adaptability, offering a robust solution for KGQA tasks.

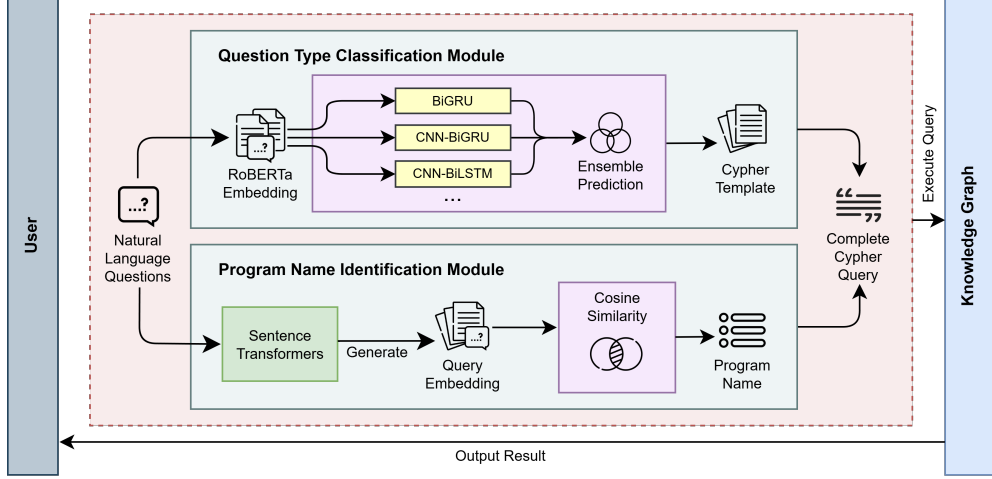


Fig. 5: The DuoKGQA framework integrates the Question Type Classification Module and the Program Name Identification Module. The former processes natural language questions using RoBERTa embeddings and deep learning models (BiGRU and CNN-BiGRU) to select Cypher query templates. The latter identifies program names using Sentence Transformers and cosine similarity. Outputs from both modules are combined to form a complete Cypher query for execution on the knowledge graph.

4.1 Question Type Classification

The Question Type Classification Module is central to the DuoKGQA system, categorizing user queries into predefined types to select the appropriate Cypher query template. This step is crucial for ensuring accurate responses in the question-answering pipeline.

The module utilizes RoBERTa to tokenize and encode input sequences, leveraging its advanced transformer-based architecture to extract rich contextual embeddings. These embeddings are processed by an ensemble of three high-performing deep learning models, which extract semantic features and utilize contextual information to accurately predict question types.

4.1.1 RoBERTa Embedding

The Question Type Classification Module uses RoBERTa to generate contextual embeddings for natural language questions. RoBERTa builds on BERT with enhancements such as dynamic masking, larger batch sizes, and longer sequence lengths, improving its capacity to capture nuanced context.

Given an input question as a sequence of tokens $\{w_1, w_2, \dots, w_n\}$, RoBERTa computes token embeddings by combining token, position, and segment embeddings. These are processed through multiple transformer layers to produce contextualized token representations:

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}, \quad \mathbf{x}_i \in \mathbb{R}^d \quad (1)$$

where $\mathbf{x}_i$ is the embedding for token w_i .

RoBERTa employs multi-head self-attention to model token relationships, with attention weights emphasizing relevant context:

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V \quad (2)$$

where Q , K , and V are query, key, and value matrices derived from embeddings, and d_k is the query and key dimension. Outputs are refined through a position-wise feed-forward network (FFN):

$$\text{FFN}(\mathbf{x}_i) = \text{ReLU}(\mathbf{x}_i W_1 + b_1) W_2 + b_2 \quad (3)$$

where W_1 , W_2 , b_1 , and b_2 are learnable parameters.

The final output embeddings, refined across all layers, include the embedding for the special [CLS] token, representing the entire input:

$$\mathbf{x}_{[\text{CLS}]} = \mathbf{h}_{[\text{CLS}]}^L \quad (4)$$

where $\mathbf{h}_{[\text{CLS}]}^L$ is the hidden state from the last layer L . This [CLS] embedding serves as a fixed-size vector encapsulating the question’s semantic content, passed to subsequent layers for question type classification and query generation.

4.1.2 BiGRU

The Question Type Classification Module utilizes a Bidirectional Gated Recurrent Unit (BiGRU) to capture contextual dependencies in both forward and backward directions, leveraging RoBERTa embeddings to model sequential relationships effectively.

Given input embeddings $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, a GRU regulates information flow with two gates: the reset gate, r_t , and the update gate, z_t . The reset gate determines the influence of past hidden states on candidate hidden states:

$$r_t = \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1} + b_r) \quad (5)$$

The update gate balances past and current inputs:

$$z_t = \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1} + b_z) \quad (6)$$

The candidate hidden state $\tilde{\mathbf{h}}_t$ is computed as:

$$\tilde{\mathbf{h}}_t = \tanh(W_h \mathbf{x}_t + r_t \odot (U_h \mathbf{h}_{t-1}) + b_h) \quad (7)$$

The final hidden state $\mathbf{h}_t$ interpolates between $\mathbf{h}_{t-1}$ and $\tilde{\mathbf{h}}_t$:

$$\mathbf{h}_t = z_t \odot \mathbf{h}_{t-1} + (1 - z_t) \odot \tilde{\mathbf{h}}_t \quad (8)$$

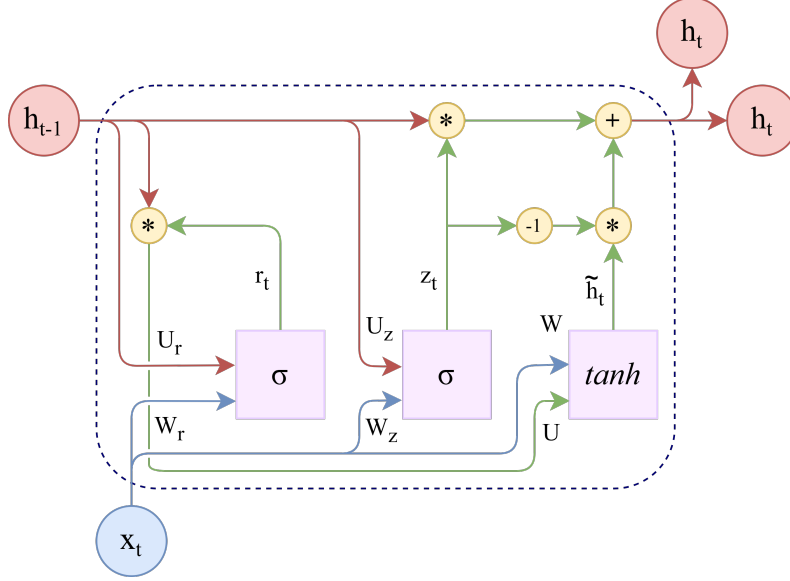


Fig. 6: GRU architecture with reset gate r_t and update gate z_t . These gates regulate information flow, allowing selective retention or forgetting of past states.

In a BiGRU, forward and backward GRUs process sequences in opposite directions:

$$\vec{\mathbf{h}}_t = \text{GRU}_{\text{fwd}}(\mathbf{x}_t, \vec{\mathbf{h}}_{t-1}), \quad \overleftarrow{\mathbf{h}}_t = \text{GRU}_{\text{bwd}}(\mathbf{x}_t, \overleftarrow{\mathbf{h}}_{t+1}) \quad (9)$$

The final hidden state is the concatenation of forward and backward states:

$$\mathbf{h}_t = [\vec{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t] \quad (10)$$

The sequence of hidden states $\{\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n\}$ is passed to a fully connected layer to compute logits:

$$\mathbf{z} = W_o \mathbf{h}_n + b_o \quad (11)$$

The logits are converted into class probabilities via the softmax function:

$$\mathbf{p}_{\text{BiGRU}} = \frac{\exp(\mathbf{z}_c)}{\sum_{k=1}^C \exp(\mathbf{z}_k)} \quad (12)$$

Fig. 6 illustrates the GRU architecture. The BiGRU's bidirectional processing captures both past and future contexts, enabling robust and context-aware representations essential for accurate question classification.

4.1.3 CNN-BiGRU

The CNN-BiGRU architecture in the Question Type Classification Module combines 1D Convolutional Neural Networks (1D-CNN) and Bidirectional Gated Recurrent

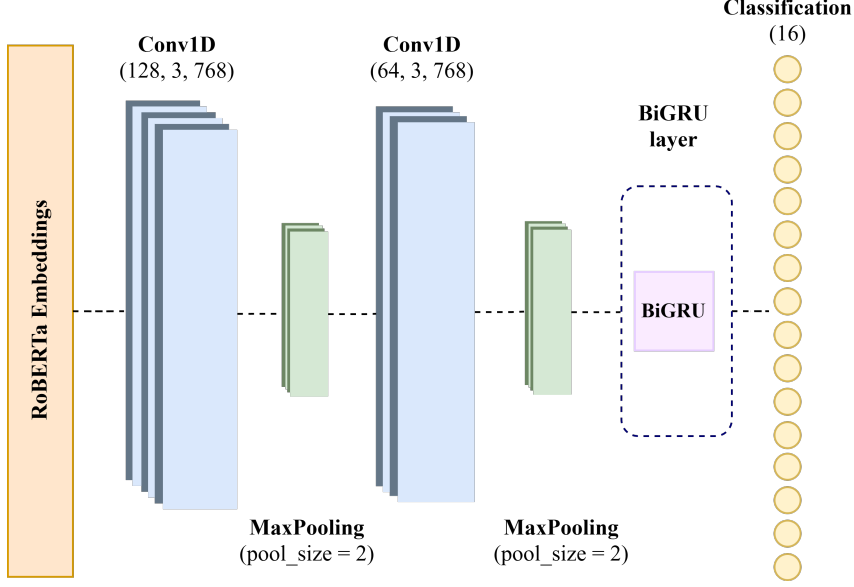


Fig. 7: Overview of the CNN-BiGRU model using RoBERTa embeddings as input. The model processes embeddings of size 768 through three 1D convolutional layers (filter size: 128) to capture local patterns. A max-pooling layer reduces dimensionality, and the pooled features are passed to a BiGRU layer with 50 hidden units per direction (100 total) to capture long-range dependencies. The output is fed into a softmax layer for class probabilities.

Units (BiGRU) to capture both local patterns and global dependencies in RoBERTa embeddings. This hybrid model provides a comprehensive representation of input sequences, modeling both short-term and long-term relationships. Fig. 7 illustrates the architecture.

Given RoBERTa embeddings $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, the CNN layer applies filters to detect local dependencies:

$$\mathbf{c}_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+k-1} + b) \quad (13)$$

where $f(\cdot)$ is a ReLU activation, $\mathbf{w}$ is the filter, and b is a bias term. The resulting feature map $\mathbf{C}$ undergoes max-pooling:

$$\mathbf{p}_i = \max(\mathbf{c}_{i:i+p-1}) \quad (14)$$

yielding a reduced feature map $\mathbf{P}$ with salient features.

The pooled features $\mathbf{P}$ are fed into a BiGRU layer to capture bidirectional dependencies:

$$\vec{\mathbf{h}}_i = \text{GRU}_{\text{fwd}}(\mathbf{p}_i, \vec{\mathbf{h}}_{i-1}), \quad \overleftarrow{\mathbf{h}}_i = \text{GRU}_{\text{bwd}}(\mathbf{p}_i, \overleftarrow{\mathbf{h}}_{i+1}) \quad (15)$$

The forward and backward hidden states are concatenated:

$$\mathbf{h}_i = [\vec{\mathbf{h}}_i; \overleftarrow{\mathbf{h}}_i] \quad (16)$$

producing a sequence of bidirectional hidden states $\mathbf{H}$ encapsulating global context.

The BiGRU output is passed to a classification layer to compute logits:

$$\mathbf{z} = W_o \mathbf{h}_n + b_o \quad (17)$$

where W_o and b_o are learnable parameters. The logits are transformed into probabilities using the softmax function:

$$\mathbf{P}_{\text{CNN-BiGRU}} = \frac{\exp(\mathbf{z}_c)}{\sum_{k=1}^C \exp(\mathbf{z}_k)} \quad (18)$$

This architecture effectively combines the local feature detection of CNN with the long-range dependency modeling of BiGRU, enabling robust question type classification. By leveraging the strengths of both components, CNN-BiGRU enhances the overall performance of the DuoKGQA system.

4.1.4 Ensemble Prediction

The Question Type Classification Module employs an ensemble model that combines the outputs of BiGRU and CNN-BiGRU models to enhance accuracy and robustness by leveraging their complementary strengths.

The predicted probability distributions from each model are denoted as:

$$\mathbf{P}_{\text{BiGRU}} = \{p_{\text{BiGRU},1}, p_{\text{BiGRU},2}, \dots, p_{\text{BiGRU},C}\} \quad (19)$$

$$\mathbf{P}_{\text{CNN-BiGRU}} = \{p_{\text{CNN-BiGRU},1}, p_{\text{CNN-BiGRU},2}, \dots, p_{\text{CNN-BiGRU},C}\} \quad (20)$$

where $\mathbf{P}_{\text{BiGRU}}$ and $\mathbf{P}_{\text{CNN-BiGRU}}$ represent the predicted probability distributions for each class c . The ensemble prediction is calculated using a weighted sum:

$$\mathbf{P}_{\text{ensemble}} = 0.6 \cdot \mathbf{P}_{\text{BiGRU}} + 0.4 \cdot \mathbf{P}_{\text{CNN-BiGRU}} \quad (21)$$

The final class prediction is:

$$\hat{y} = \arg \max_c p_{\text{ensemble},c} \quad (22)$$

This weighted majority voting scheme ensures the final prediction benefits from both models, with BiGRU capturing long-range dependencies and CNN-BiGRU detecting local patterns.

For non-program-specific question types, such as PTPTN loans or scholarships, the module generates a complete Cypher query for direct retrieval from the knowledge graph. For program-specific types, such as program descriptions or tuition fees,

it produces a skeletal Cypher query with placeholders, later populated by the Program Name Identification Module. This two-step approach ensures precise tailoring to user inquiries, enhancing flexibility and accuracy. The ensemble approach mitigates biases of individual models and balances their strengths, underpinning the overall effectiveness of DuoKGQA’s query generation process.

4.2 Program Name Identification

The Program Name Identification Module is the second core component of the DuoKGQA system, tasked with identifying the most relevant program name to populate placeholders in the Cypher query template. This module leverages the MiniLM-L6-v2 model, a compact pre-trained Transformer, to generate dense vector representations (embeddings) for both questions and program names. By comparing these embeddings, the module determines the best match to ensure accurate and context-specific query generation.

The MiniLM-L6-v2 model begins with tokenization, where the input text—be it a question or program name—is split into subword units. This tokenized sequence is denoted as $w = \{w_1, w_2, \dots, w_n\}$, where w_i is the i -th token, and n is the sequence length. The tokenized input is then transformed into embeddings using a combination of token embeddings, positional encodings, and segment embeddings.

Positional encodings, critical for capturing token order, are added to the token embeddings. These encodings are defined as:

$$\mathbf{PE}_{2k}(i) = \sin\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right) \quad (23)$$

$$\mathbf{PE}_{2k+1}(i) = \cos\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right) \quad (24)$$

where i is the token position, k indexes the encoding dimension, and d_{model} is the embedding dimension. These sinusoidal encodings ensure smooth and continuous positional patterns, allowing the model to retain order information in a non-sequential architecture.

MiniLM-L6-v2 employs self-attention to compute relationships between tokens within the sequence. Given input embeddings $\mathbf{X} \in \mathbb{R}^{n \times d_{\text{model}}}$, the model projects them into query, key, and value matrices:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V \quad (25)$$

Here, $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are learned weight matrices. The self-attention mechanism computes attention scores using scaled dot-product attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \mathbf{V} \quad (26)$$

The softmax function normalizes the attention scores, enabling the model to assign weights to tokens based on their relevance to the current context.

To enhance its capacity to focus on different aspects of the sequence, MiniLM-L6-v2 employs multi-head attention. Multiple attention heads independently compute attention, capturing diverse relational patterns:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h) \mathbf{W}_O \quad (27)$$

Each head computes $\text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i)$, and h denotes the number of heads. The combined output undergoes further transformations via a feed-forward network (FFN):

$$\text{FFN}(\mathbf{z}) = \max(0, \mathbf{z} \mathbf{W}_1 + \mathbf{b}_1) \mathbf{W}_2 + \mathbf{b}_2 \quad (28)$$

Here, $\mathbf{W}_1$ and $\mathbf{W}_2$ are learned weight matrices, and $\mathbf{b}_1$ and $\mathbf{b}_2$ are bias terms. The FFN adds non-linearity, enabling the model to learn complex patterns.

The final output of the MiniLM-L6-v2 model is derived from the [CLS] token, which represents the entire input sequence. The hidden state $\mathbf{h}^{[CLS]}$ is projected into a lower-dimensional embedding space:

$$\mathbf{e}_{q_i} = W_e \cdot \mathbf{h}^{[CLS]}(q_i), \quad \mathbf{e}_{p_j} = W_e \cdot \mathbf{h}^{[CLS]}(p_j) \quad (29)$$

where $\mathbf{e}_{q_i}$ and $\mathbf{e}_{p_j}$ are the embeddings for the question q_i and program name p_j , respectively. The similarity between the embeddings is measured using cosine similarity:

$$\text{cosine_similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} \quad (30)$$

The program name with the highest similarity score is selected:

$$j^* = \arg \max_j \text{cosine_similarity}(\mathbf{e}_{q_i}, \mathbf{e}_{p_j}) \quad (31)$$

The identified program name replaces the placeholder in the skeletal Cypher query generated by the Question Type Classification Module. This integration ensures that the final query is fully contextualized and tailored to the user’s input, allowing the DuoKGQA system to retrieve precise information from the knowledge graph. Algorithm 1 outlines the procedures of the proposed DuoKGQA framework. Fig. 8 illustrates an example of the DuoKGQA system utilizing a dual-module approach for answering knowledge graph-based queries.

5 Experimental Results and Analysis

The evaluation of the DuoKGQA system used a corpus of 11,200 questions, each annotated with a natural language query, its question type, and a corresponding Cypher query. The dataset was split into training and testing sets in an 80:20 ratio, stratified by question type to ensure proportional representation and reduce bias, enhancing the evaluation’s reliability and generalization assessment.

Experiments were conducted on the Kaggle platform using an NVIDIA Tesla P100 GPU with 16 GB HBM2 memory and 3584 CUDA cores, which accelerated training

Algorithm 1 Algorithm of the Proposed DuoKGQA Framework

Input: Natural language question q , knowledge graph KG , program names $\mathcal{P}$.

Output: Complete Cypher query Q_T and retrieved answer a .

Procedure:

1: **Question Type Classification:**

- a. Encode the input question q using the RoBERTa model to generate contextual embeddings $\mathbf{X}$.
- b. Obtain class probability distributions $\mathbf{p}_{\text{BiGRU}}$ and $\mathbf{p}_{\text{CNN-BiGRU}}$ from BiGRU and CNN-BiGRU models.
- c. Compute final class probabilities using weighted majority voting:

$$\mathbf{P}_{\text{ensemble}} = w_{\text{BiGRU}} \cdot \mathbf{p}_{\text{BiGRU}} + w_{\text{CNN-BiGRU}} \cdot \mathbf{p}_{\text{CNN-BiGRU}}$$

- d. Identify the question type T as:

$$T = \arg \max_c p_{\text{ensemble},c}$$

- e. Generate a Cypher query template Q_T based on the identified question type T .

2: **Program Name Identification (if applicable):**

- a. Encode q and program names $\mathcal{P}$ using the MiniLM-L6-v2 model to produce embeddings $\mathbf{e}_q$ and $\mathbf{e}_{p_j}$ for all $p_j \in \mathcal{P}$.
- b. Compute cosine similarity scores $S_{i,j}$ between $\mathbf{e}_q$ and $\mathbf{e}_{p_j}$:

$$S_{i,j} = \frac{\mathbf{e}_q \cdot \mathbf{e}_{p_j}}{\|\mathbf{e}_q\| \|\mathbf{e}_{p_j}\|}$$

- c. Identify the program name p^* with the highest similarity:

$$p^* = \arg \max_j S_{i,j}$$

- d. Populate placeholders in Q_T with p^* .

3: **Query Execution and Answer Retrieval:**

- a. Execute the completed Cypher query Q_T on the knowledge graph KG .
 - b. Retrieve the answer a based on the query result.
-

of transformer-based models. A quad-core Intel Xeon CPU with 32 GB RAM handled preprocessing and evaluation tasks, while 75 GB of SSD storage supported data loading and intermediate result storage, ensuring efficient resource utilization.

5.1 Question Type Classification

The selection of the deep learning models for the ensemble was determined through a rigorous comparative analysis to identify the optimal balance of accuracy and efficiency. As detailed in the following subsection, several architectures were evaluated,

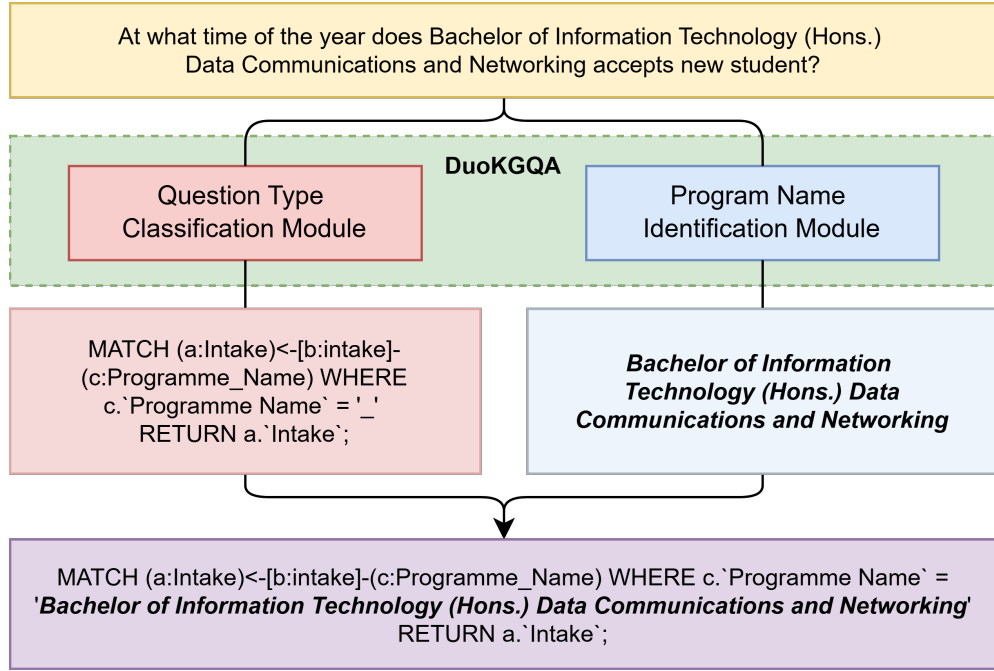


Fig. 8: Example of DuoKGQA in action. The Question Type Classification module first determines the question type to generate the appropriate Cypher query template. The Program Name Identification module then identifies the specific program name in the user’s query, which is inserted into the template to form a complete query ready for execution on the knowledge graph.

with the BiGRU and CNN-BiGRU models ultimately being selected for their complementary strengths and superior performance on our dataset. This subsection outlines the hyperparameter configurations for the Question Type Classification Module, encompassing the RoBERTa, BiGRU, and CNN-BiGRU models. Table 2 summarizes the hyperparameter settings.

5.1.1 Selection of Deep Learning Models

Several deep learning architectures, including GRU, BiGRU, LSTM, BiLSTM, and their combinations with 1D-CNN, were evaluated for question type classification. The dataset consisted of “Natural Language Questions,” containing user queries ranging from simple to complex, and “Question Types,” with categorical labels for 15 distinct question types. Table 3 summarizes the experimental results, including precision, recall, F1-score, accuracy, and processing time.

The evaluation identified BiGRU and CNN-BiGRU as the best candidates, balancing accuracy and processing time. BiGRU achieved 0.994 accuracy, effectively capturing past and future contexts through its bidirectional architecture, with a processing time of 576.36 seconds, suitable for scenarios tolerating moderate latency.

Table 2: Hyperparameter Settings for the Question Type Classification Module

Hyperparameter Settings		
Component	Parameter	Value
RoBERTa	Model Variant	roberta-base
	Max Sequence Length	386
	Learning Rate	3×10^{-5}
	Document Stride	128
	Max Query Length	64
CNN	Conv1D	64 filters, kernel size = 3
	MaxPooling1D	Pool size = 2
	Conv1D	128 filters, kernel size = 3
BiGRU / CNN-BiGRU	Optimizer	Adam
	Loss Function	Categorical Cross-Entropy
	Batch Size	64
	Maximum Epochs	100
	Early Stopping	Monitor: Validation Loss
		Patience: 10 epochs Restore Best Weights: True

Table 3: Experimental Results of Deep Learning Models on Question Type Classification

Model	Precision	Recall	F1-Score	Accuracy	Time (s)
GRU	0.992	0.992	0.992	0.992	708.51
BiGRU	0.994	0.994	0.994	0.994	576.36
LSTM	0.991	0.990	0.990	0.990	925.06
BiLSTM	0.989	0.988	0.988	0.989	607.79
CNN-GRU	0.990	0.990	0.990	0.990	511.83
CNN-BiGRU	0.993	0.992	0.992	0.993	334.98
CNN-LSTM	0.985	0.985	0.985	0.985	405.15
CNN-BiLSTM	0.986	0.985	0.985	0.985	419.12

CNN-BiGRU demonstrated comparable accuracy (0.993) with a significantly lower processing time (334.98 seconds). Its hybrid design combined CNN’s local feature extraction with BiGRU’s ability to model long-range dependencies, making it highly efficient for near-real-time applications without sacrificing precision.

5.1.2 Ensemble Deep Learning Models

While individual models like BiGRU and CNN-BiGRU exhibit strong performance, their architectural constraints limit their ability to address all facets of input data. To overcome these limitations, an ensemble model was introduced, combining the complementary strengths of BiGRU and CNN-BiGRU. Table 4 compares the performance of individual models with the ensemble approach.

The BiGRU model processes input data bidirectionally, effectively capturing sequential dependencies and contextual relationships. It achieves a precision, recall, F1-score, and accuracy of 0.994, demonstrating its reliability in modeling intricate dependencies.

The CNN-BiGRU model combines convolutional layers for local feature extraction with BiGRU for long-range dependency modeling, achieving 0.993 in precision, recall, F1-score, and accuracy. This hybrid architecture enhances its capability to handle complex data interactions.

The ensemble model integrates BiGRU’s sequential modeling with CNN-BiGRU’s local feature extraction, reducing false positives and negatives while improving prediction consistency. It achieves near-perfect performance, with precision, recall, F1-score, and accuracy of 0.999, significantly outperforming individual models.

Table 4: Performance Comparison of Individual Models and the Ensemble Model for Question Type Classification in DuoKGQA.

Models	Precision	Recall	F1-Score	Accuracy
BiGRU	0.994	0.994	0.994	0.994
CNN-BiGRU	0.993	0.992	0.992	0.993
BiGRU & CNN-BiGRU (Ensemble)	0.999	0.999	0.999	0.999

5.2 Program Name Identification

The Program Name Identification Module matches user queries with a predefined list of 91 program names by encoding both natural language questions and program names using a Sentence Transformer model. Cosine similarity between embeddings determines the best match.

Table 5 compares the performance of various Sentence Transformer models. The **paraphrase-MiniLM-L6-v2** model emerged as the top performer, achieving precision, recall, F1-score, and accuracy of 0.998 with an efficient processing time of 30.68 seconds. Designed for semantic similarity tasks, it produces compact embeddings that effectively capture the semantic meaning of inputs, ensuring accurate and efficient program name recognition.

Other models, such as **multi-qa-MiniLM-L6-cos-v1** and **all-MiniLM-L6-v2**, performed well but fell short of the precision and speed of **paraphrase-MiniLM-L6-v2**. Larger models, like **all-MiniLM-L12-v2**, exhibited slower processing times with

slightly lower accuracy, reinforcing the suitability of `paraphrase-MiniLM-L6-v2` for this task.

Table 5: Performance of Sentence Transformer Models on Program Name Identification

Model	Precision	Recall	F1-Score	Accuracy	Time (s)
paraphrase-MiniLM-L6-v2	0.998	0.998	0.998	0.998	30.68
multi-qa-MiniLM-L6-cos-v1	0.974	0.974	0.974	0.974	31.06
thenlper/gte-small	0.971	0.971	0.971	0.971	52.01
all-MiniLM-L6-v2	0.969	0.969	0.969	0.969	31.19
msmarco-MiniLM-L-12-v3	0.964	0.964	0.964	0.964	50.79
all-MiniLM-L12-v2	0.949	0.949	0.949	0.949	52.66
clip-ViT-B-32-multilingual-v1	0.937	0.937	0.937	0.937	32.62
paraphrase-TinyBERT-L6-v2	0.925	0.925	0.925	0.925	30.78

5.3 Unified DuoKGQA System

The DuoKGQA system integrates the Question Type Classification and Program Name Identification modules into a unified framework to generate accurate Cypher queries from natural language questions. The workflow begins with question type classification to select a corresponding Cypher template. For program-specific queries, the Program Name Identification Module determines the relevant program name to complete the query. Non-program-specific queries directly utilize the selected template without program-specific identification.

The system’s performance, summarized in Table 6, demonstrates strong results for program-specific queries. For instance, the “Career Prospect” question type achieved perfect scores (1.000), while “Duration,” “Entry Requirement,” and “Faculty Offering the Programs” attained F1-scores of 0.997, 0.944, and 0.984, respectively. Non-program-specific queries exhibited more variability, with categories like “Scholarship” achieving an F1-score of 0.985, while “Deadline” and “Program Description” scored lower at 0.783 and 0.861, highlighting challenges with similar language constructs.

The overall exact match score, calculated as the ratio of correct Cypher queries to total queries, was 0.890, showcasing the system’s effectiveness in generating accurate queries. Fig. 9 provides a confusion matrix, illustrating misclassification patterns and prediction distributions.

5.4 Comparative Analysis with the Baseline System

To evaluate the effectiveness of the proposed DuoKGQA framework, a comparative analysis is conducted against a relevant baseline. While various KGQA models have been proposed in recent years, most are designed for open-domain datasets such as

Table 6: Performance of the Unified DuoKGQA System on Question Type Classification

Question Type	Precision	Recall	F1-Score	Accuracy
Application Process	0.888	1.000	0.941	1.000
B40	1.000	0.966	0.982	0.966
Career Prospect	1.000	1.000	1.000	1.000
Deadline	1.000	0.643	0.783	0.643
Program Description	0.984	0.766	0.861	0.766
Duration	0.994	1.000	0.997	1.000
Entry Requirement	0.992	0.901	0.944	0.901
Faculty Offering	1.000	0.968	0.984	0.968
Tuition Fees	1.000	0.918	0.957	0.918
Research Field	0.933	0.899	0.916	0.899
Intake Information	0.977	1.000	0.988	1.000
Loyalty Program	0.909	0.930	0.920	0.930
PTPTN	0.978	0.643	0.776	0.643
Scholarship	0.980	0.990	0.985	0.990
Structure	0.778	0.959	0.859	0.959
Macro Average	0.961	0.905	0.926	0.905
Weighted Average	0.952	0.946	0.945	0.946

WebQSP or MetaQA and are not capable of generating Cypher queries for domain-specific knowledge graphs. As such, direct comparisons with these models are not appropriate in this context.

TransKGQA [22] is selected as the baseline for this study, as it is specifically developed to translate natural language questions into Cypher queries within the academic domain. It employs a single Sentence Transformer to jointly perform question intent classification and entity identification based on cosine similarity. Although this design is efficient, it often struggles with questions that require precise interpretation of intent and accurate resolution of semantically similar entities—challenges that are common in academic settings.

DuoKGQA addresses these limitations through a dual-module architecture that separates question type classification from program name identification. This design enables each module to be independently optimized for its specific subtask, resulting in improved accuracy when handling diverse question structures and closely related entity mentions.

As shown in Table 7, DuoKGQA achieves an accuracy of 94.60% and an exact match score of 0.890, significantly outperforming TransKGQA, which attains 76.30% accuracy and an exact match score of 0.763. These results highlight the benefits of modularization and demonstrate the effectiveness of DuoKGQA in translating natural language questions into accurate Cypher queries for academic knowledge graphs.

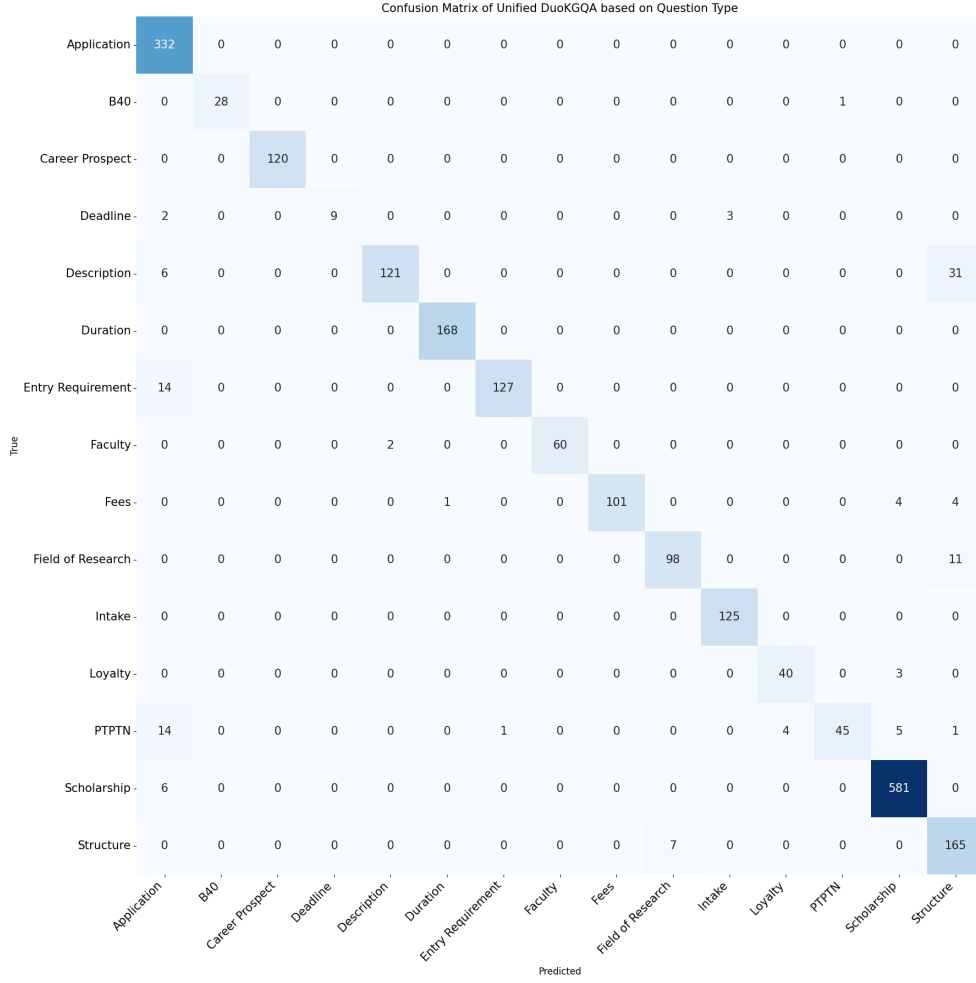


Fig. 9: Confusion matrix showcasing the performance of the DuoKGQA system.

Table 7: Comparative Results of TransKGQA and DuoKGQA

Methods	Accuracy (%)	Exact Match
TransKGQA [22]	76.30	0.763
DuoKGQA	94.60	0.890

6 Conclusion

This paper introduces DuoKGQA, a dual-module Knowledge Graph Question Answering (KGQA) system tailored for the academic domain. The system translates natural

language queries into precise Cypher queries using advanced natural language processing and machine learning techniques. DuoKGQA’s architecture integrates data preprocessing, knowledge graph construction, Cypher query generation, question type classification, and program name identification, providing a comprehensive approach to academic query answering.

The Question Type Classification Module employs RoBERTa for contextual encoding and an ensemble of BiGRU and CNN-BiGRU models to improve classification accuracy. The Program Name Identification Module uses Sentence Transformers and cosine similarity to identify program names in user queries. These modules work together to generate Cypher queries executed on a Neo4j knowledge graph for precise answer retrieval.

Empirical evaluations demonstrate DuoKGQA’s effectiveness, achieving an accuracy of 0.946 and an exact match (EM) score of 0.890, highlighting its ability to handle both program-specific and non-program-specific queries. However, challenges remain in processing queries with similar program names or multifaceted contexts, occasionally leading to misclassifications. Future work could address these issues by incorporating several key advancements. To handle more intricate academic queries involving multi-entity conditions, the system could be enhanced by exploring hypergraph-based knowledge representation techniques. The Question Type Classification Module could be further optimized by implementing a dynamic weighting scheme for the ensemble model to adapt to different query types. These improvements, along with advanced entity disambiguation, multi-hop reasoning, and larger, more diverse knowledge graphs, would form a robust foundation for future research. Finally, domain adaptation techniques could also extend the system to other specialized fields.

References

- [1] Diefenbach, D., Lopez, V., Singh, K., Maret, P.: Core techniques of question answering systems over knowledge bases: a survey. *Knowledge and Information systems* **55**, 529–569 (2018)
- [2] Peng, C., Xia, F., Naseriparsa, M., Osborne, F.: Knowledge graphs: Opportunities and challenges. *Artificial Intelligence Review* **56**(11), 13071–13102 (2023)
- [3] Liu, L., Chen, Y., Das, M., Yang, H., Tong, H.: Knowledge graph question answering with ambiguous query. In: *Proceedings of the ACM Web Conference 2023*, pp. 2477–2486 (2023)
- [4] Hu, S., Zou, L., Yu, J.X., Wang, H., Zhao, D.: Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering* **30**(5), 824–837 (2017)
- [5] Wen, L., Xiong, G., Mo, T., Li, B., Li, W., Zhao, W.: Clear-kgqa: Clarification-enhanced ambiguity resolution for knowledge graph question answering. *arXiv preprint arXiv:2504.09665* (2025)

- [6] Cui, H., Peng, T., Bao, T., Han, R., Han, J., Liu, L.: Stepwise relation prediction with dynamic reasoning network for multi-hop knowledge graph question answering. *Applied Intelligence* **53**(10), 12340–12354 (2023)
- [7] Du, H., Zhang, X., Wang, M., Chen, Y., Ji, D., Ma, J., Wang, H.: A contrastive framework for enhancing knowledge graph question answering: Alleviating exposure bias. *Knowledge-Based Systems* **280**, 110996 (2023)
- [8] Jiang, J., Zhou, K., Zhao, W.X., Li, Y., Wen, J.-R.: Reasoninglm: Enabling structural subgraph reasoning in pre-trained language models for question answering over knowledge graph. *arXiv preprint arXiv:2401.00158* (2023)
- [9] Luo, H., Tang, Z., Peng, S., Guo, Y., Zhang, W., Ma, C., Dong, G., Song, M., Lin, W., et al.: Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *arXiv preprint arXiv:2310.08975* (2023)
- [10] Wang, J., Li, W., Liu, F., Sheng, B., Liu, W., Jin, Q.: Hic-kgqa: Improving multi-hop question answering over knowledge graph via hypergraph and inference chain. *Knowledge-Based Systems* **277**, 110810 (2023)
- [11] Wang, C., Xu, Y., Peng, Z., Zhang, C., Chen, B., Wang, X., Feng, L., An, B.: keqing: knowledge-based question answering is a nature chain-of-thought mentor of llm. *arXiv preprint arXiv:2401.00426* (2023)
- [12] Wang, T., Huang, R., Wang, H., Zhi, H., Liu, H.: Multi-hop knowledge graph question answer method based on relation knowledge enhancement. *Electronics* **12**(8), 1905 (2023)
- [13] Xu, J., Biryukov, M., Theobald, M., Venugopal, V.E.: Bigtext-qa: Question answering over a large-scale hybrid knowledge graph. In: *International Conference on Big Data Technologies and Applications*, pp. 33–48 (2023). Springer
- [14] Zuo, Z., Zhu, Z., Wu, W., Wang, W., Qi, J., Zhong, L.: Improving question answering over knowledge graphs with a chunked learning network. *Electronics* **12**(15), 3363 (2023)
- [15] Faldu, P., Bhattacharya, I., et al.: Retinaqa: A knowledge base question answering model robust to both answerable and unanswerable questions. *arXiv preprint arXiv:2403.10849* (2024)
- [16] Formica, A., Mele, I., Taglino, F.: A template-based approach for question answering over knowledge bases. *Knowledge and Information Systems* **66**(1), 453–479 (2024)
- [17] Kang, L., Li, X., An, X.: Knowledge-aware adaptive graph network for commonsense question answering. *Journal of Intelligent Information Systems*, 1–20

(2024)

- [18] Li, Z., Fan, S., Gu, Y., Li, X., Duan, Z., Dong, B., Liu, N., Wang, J.: Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, pp. 18608–18616 (2024)
- [19] Tian, Y., Song, H., Wang, Z., Wang, H., Hu, Z., Wang, F., Chawla, N.V., Xu, P.: Graph neural prompting with large language models. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, pp. 19080–19088 (2024)
- [20] Thamby, S.V., Reghu Raj, P.: A novel technique using graph neural networks and relevance scoring to improve the performance of knowledge graph-based question answering systems. *Journal of Intelligent Information Systems*, 1–24 (2024)
- [21] Zhang, M., He, T., Dong, M.: Meta-path reasoning of knowledge graph for commonsense question answering. *Frontiers of Computer Science* **18**(1), 181303 (2024)
- [22] Chong, Y.L., Lee, C.P., Muhd-Yassin, S.Z., Lim, K.M., Samingam, A.K.: Transkgqa: Enhanced knowledge graph question answering with sentence transformers. *IEEE Access* (2024)
- [23] Guo, Z., Xia, L., Yu, Y., Ao, T., Huang, C.: Lightrag: Simple and fast retrieval-augmented generation (2024)
- [24] Sanmartin, D.: Kg-rag: Bridging the gap between knowledge and creativity. *arXiv preprint arXiv:2405.12035* (2024)
- [25] Li, M., Miao, S., Li, P.: Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation. *arXiv preprint arXiv:2410.20724* (2024)
- [26] Mavromatis, C., Karypis, G.: Gnn-rag: Graph neural retrieval for large language model reasoning. *arXiv preprint arXiv:2405.20139* (2024)