FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Mo-Lab: Interactieve modelrapportage en -analyse in openCAESAR via verwerking van natuurlijke taal en SPARQL-integratie

Thomas Decloedt
Studentennummer: 01808629

Promotoren: prof. dr. ir. Chris Develder, prof. dr. Mohammad Hamdaqa (Polytechnique Montréal)

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in Computer Science Engineering

Academiejaar 2024-2025

UNIVERSITEIT
GENT

Acknowledgment

I would like to express my deepest gratitude to Professor Dr. Mohammad Hamdaqa from Polytechnique Montréal for his guidance and support throughout this research journey. I am immensely thankful for the opportunity he provided, the intellectually stimulating conversations we shared, and the academic freedom he granted me to explore and develop my research ideas. His encouragement to engage with other experts significantly enriched this work.

I am profoundly grateful to Dr. Maged Elaasar from NASA Jet Propulsion Laboratory (JPL) and Professor Dr. Bentley James Oakes from Polytechnique Montréal for their invaluable insights and constructive feedback. Their expertise and generosity with their time allowed me to refine my ideas and enhance the overall quality of this thesis.

My sincere thanks go to Professor Dr. ir. Chris Develder at Ghent University for his administrative support, detailed feedback, and thorough review of the manuscript. His commitment to excellence and attention to detail were crucial in ensuring this thesis achieved the desired quality.

I also wish to thank François Goybet for granting me the opportunity to utilize his bachelor project, which proved invaluable in testing my proof of concept. I had the pleasure of meeting him through Prof. Dr. Hamdaqa during my exchange in Montréal, where he was also participating in an exchange program at the time. His collaboration and generosity in sharing his work were sincerely appreciated.

I am deeply grateful to the Faculty of Engineering and Architecture (FEA) and Ghent University for enabling me to undertake this research in collaboration with Professor Hamdaqa. This experience has been a highlight of my academic journey. I extend my thanks to the International Relations Office of FEA for facilitating my study exchange and ensuring a rewarding experience, as well as to Polytechnique Montréal for hosting me as an exchange student.

Finally, I want to acknowledge the broader society for the opportunities that allowed me to pursue higher education and embark on this academic journey. The privilege to engage in research and study abroad is something I deeply value.

Above all, my heartfelt gratitude goes to my mother. Her unwavering support, encouragement, and belief in me have been the cornerstone of my achievements. Her strength continues to inspire me every day.

Explanation regarding the master's thesis and the oral presentation

Abstract

Title: Mo-Lab: Interactive model reporting and analysis in openCAESAR through natural language processing and SPARQL integration

Author: Thomas Decloedt **Student Number:** 01808629 **Supervisors:** Prof. Dr. Mohammad Hamdaqa, Prof. Dr. ir. Chris Develder **Degree:** Master of Science in Computer Science Engineering **Academic Year:** 2024-2025

The task of generating reports in Model-Based Systems Engineering often demands significant effort and technical expertise, creating barriers resulting in inflexibility. Recent advancements in AI, particularly in Natural Language Processing, offer promising opportunities to tackle these challenges by facilitating more interactive and less technical approaches.

A significant challenge in applying neural approaches to Model-Based Systems Engineering (MBSE) lies in the scarcity of datasets available in these specialized domains. This thesis addresses the issue of data scarcity primarily through domain adaptation, which leverages data from other domains to benefit specialized MBSE contexts. Additional techniques employed to mitigate data limitations include the use of Controlled Natural Language, which reduces the training data requirements, and Retrieval-Augmented Generation, which integrates unseen domain information to enhance performance and adaptability.

The experimental results demonstrate that data scarcity can be effectively addressed through Domain Adaptation and Controlled Natural Language. Using high-quality synthetic data resulted in excellent performance, even on a challenging benchmark. However, the effectiveness of Retrieval-Augmented Generation remains inconclusive, largely due to limitations inherent to the benchmark used for evaluation. Additionally, as with related approaches, challenges persist regarding the generalizability of the proposed method. Nonetheless, the results reveal promising potential, warranting further exploration.

This thesis contributes to the field of MBSE by introducing an interactive model reporting paradigm, demonstrated through the proof-of-concept implementation, Mo-Lab. The approach addresses the challenges of data scarcity often encountered in specialized domains by designing an architecture tailored to these constraints. Its evaluation on a representative domain underscores its feasibility and effectiveness. Finally, this work demonstrates promising practical applications, paving the way for more accessible and flexible model reporting in MBSE.

Keywords: Model-Based Systems Engineering, openCAESAR, text-to-SPARQL, domain adaptation, data scarcity

Mo-Lab: Interactive model reporting and analysis in openCAESAR through natural language processing and SPARQL integration

Thomas Decloedt

Supervisors: Professor Dr. Mohammad Hamdaqa and Professor Dr. ir. Chris Develder

Abstract—Creating reports in Model-Based Systems Engineering (MBSE) often demands significant effort and technical expertise, creating barriers to flexibility. This work proposes a novel paradigm of interactive model reporting, enabling stakeholders to compose reports without technical barriers. The proof-of-concept implementation, Mo-Lab, targeted at the representative openCAESAR platform, is built around a notebook environment that integrates a Question Answering System capable of answering questions about MBSE models with natural language explanations, tabular data, and visualizations. The key challenge in implementing the proposed paradigm is the scarcity of domain-specific data required to leverage contemporary state-of-the-art neural approaches. To address this challenge, this research employs domain adaptation, leverages a controlled natural language, adopts Retrieval-Augmented Generation, and explores synthetic data generation. The proposed approach tackling the central obstacle of data scarcity demonstrates competitive performance on a challenging benchmark and offers greater applicability to MBSE domains compared to existing methods.

Index Terms—Model-Based Systems Engineering, interactive model reporting, openCAESAR, data scarcity, domain adaptation

I. Introduction

The importance of documenting complex systems effectively is well-recognized across domains. In Machine Learning, model cards and datasheets exemplify this by providing stakeholders with essential information for informed decisions [1], [2]. Similarly, Systems Engineering benefits from clear documentation tailored to stakeholder needs. Model-Based Systems engineering (MBSE) has been gaining more traction, partially because of its benefits in terms of communication with stakeholders. MBSE employs domain models as single sources of truth, often communicated through reports generated from predefined templates called viewpoints. However, they lack flexibility. Creating or updating reports requires both domain expertise and technical knowledge, leading to time and cost inefficiencies.

To address this, a no-code paradigm called model reporting is proposed for MBSE. This approach enables stakeholders to interact with a reporting agent using natural language, bypassing the need for technical knowledge. The agent utilizes advancements in Natural Language Processing (NLP) and Large Language Models (LLMs) to answer questions, produce visualizations, etc. Mo-Lab, a proof-of-concept implementation, demonstrates the proposed paradigm in a notebook environment, prioritizing interactivity and ease of use.

The primary challenge to implementing model reporting lies in data scarcity, brought about by the specialized nature of MBSE domains. This work addresses this challenge through a multipronged approach: leveraging domain adaptation, adopting a controlled natural language to reduce LLM training data requirements [3], integrating the LLM within a Retrieval-Augmented Generation (RAG) framework, and generating synthetic data to emulate realistic use cases.

Hereafter, related work is first presented, then the proposed paradigm is introduced, followed by its proof-of-concept, pen ultimately the experiments evaluating the implementation, and finally a conclusion.

II. Related Work

Automating documentation and reporting has gained significant interest across various domains. In MBSE, traditional methods for generating reports from templates, or model viewpoints, have proven useful but face challenges in terms of flexibility.

Recent advancements in AI have introduced frameworks like text-to-SQL systems for enterprise reporting [4], [5], AI-assisted presentation generation from computational notebooks [6], and question answering systems using retrieval-augmented generation (RAG) [7], [8].

While these systems demonstrate promising results, they primarily target relational databases or general-purpose question answering, which differ from MBSE's specific needs, nor do these approaches typically work when domain-specific training data is not readily available.

III. Model Reporting

The proposed paradigm for model reporting seeks to address the limitations and challenges inherent in traditional reporting approaches within MBSE.

Currently, the template-driven methodologies are of a highly technical nature — for instance, requiring extensive coding skills in a formal query language to interact with the model — which imposes significant barriers and hinders flexible reporting.

This section introduces a novel paradigm that provides an alternative to these conventional methodologies.

Following this, Mo-Lab, the proof-of-concept implementation developed as part of this work, is presented to demonstrate the feasibility and effectiveness of the proposed approach.

Finally, key challenges that underscore the importance and necessity of this research are discussed.

A. Paradigm Overview

Rather than relying on the engineering of templates, model reporting is reimagined as a process where stakeholders interact with an intelligent agent within a notebook environment.

The agent's role is to respond to stakeholder inquiries about the model by providing natural language explanations, visualizations, and other outputs, offering a no-code approach to composing reports.

The environment enables stakeholders to reorganize and customize the content, ensuring that the reports align with their specific requirements.

B. Implementation

Mo-Lab, the proof-of-concept implementation, employs an effective Question Answering pipeline to realize the model reporting paradigm.

The implementation focuses on the openCAESAR platform, a representative MBSE approach that defines its models using the OML language. After model authoring, the OML models are converted into RDF graphs, which are made queryable using the SPARQL language.

The pipeline follows these sequential steps:

1) A stakeholder submits a natural language question related to the system.
2) The inquiry is converted into a query.
3) The query is executed against the RDF graph to retrieve the data answering the stakeholder's question.
4) Post-processing is applied, including natural language explanation, tabulation, and visualization.
5) The final results are returned to the stakeholder's notebook environment.

This process is designed to streamline report generation by automating the extraction and presentation of model information in an intuitive, easy-to-use format.

C. Challenges

The most significant challenge in this proof-of-concept implementation lies in the translation of questions into queries. This task is carried out by a text-to-SPARQL component, which is critical for enabling downstream processes, such as the visualization of query results.

However, in practice, MBSE domains typically lack the vast quantities of data needed to train neural approaches commonly employed for the text-to-SPARQL task. This scarcity of data presents a key obstacle that this work seeks to address in order to realize effective model reporting.

IV. Mo-Lab

To address data scarcity, this research adopts a multifaceted approach, with domain adaptation playing a central role.

A Large Language Model (LLM) is trained on a variation of the text-to-SPARQL task, where queries contain placeholder strings instead of explicit references to any particular domain.

While this approach requires post-generation linking of the LLM output to the target domain, its major advantage lies in its adaptability. The model can be repurposed for other domains through domain adaptation, making it a flexible solution for diverse MBSE contexts.

A linker from the literature is adopted [9].

A. Domain Adaptation

Recognizing the limited availability of domain-specific data, this solution employs domain adaptation techniques to leverage data from other domains for the benefit of specialized domains commonly encountered in MBSE.

The previously mentioned LLM is fine-tuned on a source domain with an expansive dataset, i.e., on Wikidata with LC-QuAD [10], on the variation of the text-to-SPARQL task with placeholders.

This fine-tuned model is then integrated into a Retrieval-Augmented Generation (RAG) architecture as the generator component. A second training phase is performed using target domain data, with the fine-tuned model kept frozen to preserve its acquired knowledge. The target domain in this case is ORKG, and training is conducted using the SciQA [11] dataset.

The objective of this second training phase is to effectively transfer the knowledge gained by the fine-tuned LLM from the source domain to the target domain.

Retrieval is carried out by gathering relevant information in response to a question and post-processing it into a subgraph of the RDF graph representing the model.

This final trained RAG model, specialized for the target domain, is referred to as the domain expert.

B. Controlled Natural Language

In addition to facilitating model repurposing, the general training requirements for the LLM generating queries are further reduced by adapting the text-to-SPARQL task. Instead of SPARQL, a specific controlled natural language called SQUALL is used as the output of the language model.

This approach lowers the LLM's training data requirements due to SQUALL's closer resemblance to natural language [12], while maintaining equivalence to SPARQL queries. SQUALL expressions can be translated into SPARQL using a dedicated translator [13].

C. Retrieval-Augmented Generation

In addition to leveraging data-rich domains and reducing general data requirements, retrieval from non-parametric memory is incorporated into the architecture.

Retrieval-Augmented Generation (RAG) is employed as the core architecture for the domain expert, enabling the integration of information, unseen during training, from target domains.

The LLM's generation process is augmented with a constructed question-relevant graph. This graph is constructed using the Prize-Collecting Steiner Tree (PCST) algorithm, which connects the retrieved vertices and edges into one connected graph while effectively controlling for size.

D. Synthetic Data Generation

Training on synthetic data is explored to evaluate the approach in scenarios that more closely resemble those encountered in practice.

While the benchmark dataset SciQA is also synthetically generated, it is of high quality. In contrast, the synthetic data generated in this work is of much lower quality, providing additional insights into the practical applicability and robustness of the proposed approach.

E. Result Handling

Mo-Lab's includes three illustrative artifacts that are essential for model reporting.

First, natural language explanations are generated by prompting ChatGPT to answer the stakeholder's question based on the results of the executed query produced by the text-to-SPARQL pipeline.

Second, tabulation provides a straightforward, formatted representation of the query results, offering a clear and organized way to present data.

Third, automatic visualization of the results is achieved by passing the generated query to VizKG, a visualization framework adopted from the literature [14], to produce graphical representations of the query output.

V. Experiments

The proposed approach is evaluated across four experimental setups focusing on different performance aspects of the domain expert.

Another experiment examines the synthetic data generation strategy, serving a supportive role by providing insights that inform the other experiments.

The final part of this section concludes with a comparative analysis against benchmarks, providing context and highlighting the significance of the results.

A. Retrieval and Post-Retrieval Analysis

This experiment investigates how the retrieval and post-retrieval components construct relevant subgraphs based on a given question. Key parameters analyzed include the top-k value, which determines the number of vertices and edges retrieved, and the edge cost, which affects the compactness of the resulting subgraph.

The results reveal that increasing the top-k value improves recall but also enlarges the subgraphs, potentially introducing noise and degrading the performance of the domain expert. In contrast, higher edge costs reduce subgraph size while slightly impacting recall. Together achieving an effective balance between informativeness and compactness.

These findings highlight the critical role of parameter tuning in optimizing subgraph construction.

B. Synthetic Dataset Quality

This experiment plays a supportive role, focusing on evaluating the quality of synthetic datasets generated using ChatGPT to train the domain expert in scenarios where high-quality data is scarce. Faithfulness is assessed using BLEU scores by comparing the generated queries with ground truth references. Two key variables are analyzed: variations in the type of demonstration and the number of demonstrations provided to ChatGPT.

The results indicate that increasing the number of examples per prompt (from one to three) enhances faithfulness and, consequently, dataset quality. Additionally, reducing the variety in demonstrations significantly impacts quality, emphasizing the need for diversity. However, excluding one type of question does not affect the quality of the remaining types.

These findings underscore the importance of incorporating both a diverse set of examples and a sufficient quantity of demonstrations in prompts to improve synthetic dataset generation.

C. Text-to-SQUALL

This experiment evaluates the domain expert's capability to generate SQUALL expressions from textual questions. Different configurations for retrieval are tested with BLEU scores used as the primary evaluation function.

The baseline configuration, which does not incorporate retrieval, performs unexpectedly well. This outcome is attributed to the homogeneity of the benchmark dataset, where recurring entities and relationships dominate. While larger and more informative subgraphs improve performance, the baseline remains a robust reference point.

These findings highlight the influence of dataset vocabulary on the impact and usefulness of retrieval. Overall, the performance is excellent, with the baseline achieving a BLEU score of 98, demonstrating the domain expert's strong capabilities in generating accurate SQUALL expressions.

D. Generalization Capabilities

This experiment evaluates the domain expert's ability to generalize to unseen question types and synthetic datasets, simulating data-scarce scenarios. Four setups are tested, combining different training data types (ground truth vs.

synthetic) and question type familiarity, with BLEU scores used as the performance metric.

The results show that performance declines as data quality decreases, particularly for rare question types. Synthetic data performs better than anticipated but remains inferior to ground truth training.

These findings highlight the challenges of achieving robust generalization and emphasize the importance of using representative datasets to effectively train the domain expert.

E. Text-to-SPARQL

This experiment evaluates the full text-to-SPARQL component, which integrates the domain expert, translator, and linker. Performance is measured using BLEU scores, execution accuracy, and F1 scores.

The baseline configuration delivers the best results, achieving a BLEU score of 94.34 and an F1 score of 86.52, with syntactically correct queries exceeding 93% accuracy. Failures, which account for less than 7%, stem from, for example, incomplete outputs by the domain expert and linking inaccuracies.

These results validate the robustness of the pipeline, demonstrating its reliability in generating executable SPARQL queries.

F. Comparative Analysis

This experiment contextualizes the performance of the proposed approach by comparing it to KGQAn [9] — a question answering platform aiming for universality, i.e., applicability to any domain — as well as fine-tuned and few-shot learning LLMs evaluated on the benchmark [12].

1) Realistic Scenario: In a realistic scenario, the proposed approach significantly outperforms KGQAn, which does not use any domain-specific data. KGQAn struggles with a BLEU score of 4.42 and a 54% failure rate due to its dependence on semantic triple generation as an intermediate step, followed by heuristic-based query construction.

2) Ideal Scenario: When benchmarked against a fine-tuned T5 model or a prompt-engineered few-shot learning GPT-3.5-turbo, the proposed approach remains competitive. Although these models achieve near-optimal performance, the proposed architecture delivers results only marginally lower than the benchmarks, validating its strong performance and practical applicability, particularly in data-scarce conditions.

3) Insights: The fine-tuned model exhibited syntactical errors, while the few-shot learning models struggled with entity hallucination and semantic inaccuracies.

Syntactical errors are mitigated in the proposed approach through its initial training phase and by freezing the generator during subsequent training, although further error analysis is required to gain deeper insight. Meanwhile, entity hallucination and semantic inaccuracies are entirely avoided by utilizing placeholders during query generation and resolving them through a dedicated linking process.

It excels in data-scarce scenarios while remaining competitive with state-of-the-art benchmarks.

VI. Conclusion

This work introduces a novel paradigm for model reporting in MBSE, along with its proof-of-concept implementation, Mo-Lab. A key challenge to integrating advanced NLP techniques into MBSE reporting — data scarcity — was identified and addressed through a combination of domain adaptation, the use of controlled natural language with SQUALL, and an architecture.

Comprehensive experiments demonstrate the approach's strong performance in both realistic and ideal scenarios, achieving competitive results against benchmarks and showcasing its robustness in data-scarce conditions.

These findings validate the proposed paradigm's feasibility, highlighting its potential for practical applicability and its ability to provide a flexible and interactive reporting framework for MBSE.

References

[1] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru, "Model Cards for Model Reporting," in Proceedings of the Conference on Fairness, Accountability, and Transparency. Atlanta GA USA: ACM, Jan. 2019, pp. 220–229.

[2] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. D. Iii, and K. Crawford, "Datasheets for datasets," Communications of the ACM, vol. 64, no. 12, pp. 86–92, Dec. 2021.

[3] J. Lehmann, P. Gattogi, D. Bhandiwad, S. Ferré, and S. Vahdati, "Language Models as Controlled Natural Language Semantic Parsers for Knowledge Graph Question Answering," in Frontiers in Artificial Intelligence and Applications, K. Gal, A. Nowé, G. J. Nalepa, R. Fairstein, and R. Rădulescu, Eds. Krakow (Cracovie), Poland: IOS Press, Sep. 2023, pp. 1348–1356.

[4] S. R. Joshi, B. Venkatesh, D. Thomas, Y. Jiao, and S. Roy, "A Natural Language and Interactive End-to-End Querying and Reporting System," in Proceedings of the 7th ACM IKDD CoDS and 25th COMAD. Hyderabad India: ACM, Jan. 2020, pp. 261–267.

[5] C. Zhang, Y. Mao, Y. Fan, Y. Mi, Y. Gao, L. Chen, D. Lou, and J. Lin, "FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis," Jan. 2024.

[6] F. Wang, X. Liu, O. Liu, A. Neshati, T. Ma, M. Zhu, and J. Zhao, "Slide4N: Creating Presentation Slides from Computational Notebooks with Human-AI Collaboration," in Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. Hamburg Germany: ACM, Apr. 2023, pp. 1–18.

[7] Y. Peng, S. Lin, Q. Chen, L. Xu, X. Ren, Y. Li, and J. Xu, "ChatGraph: Chat with Your Graphs," Jan. 2024.

[8] X. He, Yijun Tian, Yifei Sun, N. V. Chawla, T. Laurent, Y. LeCun, X. Bresson, and B. Hooi, "G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering," Feb. 2024.

[9] R. Omar, I. Dhall, P. Kalnis, and E. Mansour, "A Universal Question-Answering Platform for Knowledge Graphs," Proceedings of the ACM on Management of Data, vol. 1, no. 1, pp. 1–25, May 2023.

[10] M. Dubey, D. Banerjee, A. Abdelkawi, and J. Lehmann, "LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia," in The Semantic Web – ISWC 2019, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, Eds. Cham: Springer International Publishing, 2019, vol. 11779, pp. 69–78.

[11] S. Auer, D. A. C. Barone, C. Bartz, E. G. Cortes, M. Y. Jaradeh, O. Karras, M. Koubarakis, D. Mouromtsev, D. Pliukhin, D. Radyush, I. Shilin, M. Stocker, and E. Tsalapati, "The SciQA Scientific Question Answering Benchmark for Scholarly Knowledge," Scientific Reports, vol. 13, no. 1, p. 7240, May 2023.

[12] J. Lehmann, A. Meloni, E. Motta, F. Osborne, D. R. Recupero, A. A. Salatino, and S. Vahdati, "Large Language Models for Scientific Question Answering: An Extensive Analysis of the SciQA Benchmark," in The Semantic Web, A. Meroño Peñuela, A. Dimou, R. Troncy, O. Hartig, M. Acosta, M. Alam, H. Paulheim, and P. Lisena, Eds. Cham: Springer Nature Switzerland, 2024, vol. 14664, pp. 199–217.

[13] S. Ferré, "Squall2sparql: A Translator from Controlled English to Full SPARQL 1.1," in CEUR Workshop Proceedings, vol. 1179, Valencia, Spain, Sep. 2013.

[14] H. Raissya, F. Darari, and F. J. Ekaputra, "VizKG: A Framework for Visualizing SPARQL Query Results over Knowledge Graphs," CEUR Workshop Proceedings, vol. 3023, pp. 95–102, 2021.

Contents

List of Figures

List of Tables

List of Acronyms

ABox Assertional Box. , 34, 35

AI Artificial Intelligence. , 1, 6, 7, 13–15, 23, 26, 27, 30, 100, 101, 116

ANN Artificial Neural Network. , 44

API Application Programming Interface. , 15, 33, 72

AWS Amazon Web Services. , 73

BGP Basic Graph Pattern. , 38, 39, 97

BLEU Bilingual Evaluation Understudy. , 64, 65, 76, 77, 79–81, 85, 87, 88, 90, 93

BNode Blank Node. , 35

CAD Computer-Aided Design. , 6

CF Coherency Factor. , 72

CGP Complex Graph Pattern. , 38, 93, 97

CNL Controlled Natural Language. , 4, 7, 10–12, 16, 30, 51, 54

DA Domain Adaptation. , 7, 8, 10, 16, 101, 102

DPT Demonstration Per Template. , 76–78

FSL Few-Shot Learning. , 92–95

GAT Graph Attention Network. , 44

GED Graph Edit Distance. , 66, 67

GNN Graph Neural Network. , 44

GP Graph Pattern. , 39, 45, 48, 49, 56, 60, 66, 67, 69

GPM Graph Pattern Matching.

ICL In-Context Learning. , 61

IDE Integrated Development Environment.

IR Information Retrieval. , 8, 52, 54, 64, 65

JIT Just-In-Time. , 46, 47

KB Knowledge Base. , 9, 14, 15, 34, 35, 52, 61, 81

KBQA Knowledge Base Question Answering. , 9, 13

KG Knowledge Graph. , 9–11, 13–15, 23, 30, 31, 34, 35, 38, 40, 41, 43, 47, 50–54, 58, 62, 63, 66–71, 73, 74, 79, 80, 91, 92, 96, 102

KGQA Knowledge Graph Question Answering. , 9, 11–13, 15, 23, 24, 26, 30, 54–56, 61, 73, 87, 90

KGQAn Knowledge Graph Question Answering platform. , 90–92, 98

LLM Large Language Model. , 2–4, 10–16, 27, 30, 31, 33, 43, 44, 50–52, 54, 56, 58, 60, 62, 63, 66, 71, 84, 89–95

LM Language Model. , 10–12, 27, 38, 40, 51

LoRA Low-Rank Adaptation. , 14

MBSE Model-Based Systems Engineering. iv, 1, 2, 5–7, 9, 10, 14–20, 26, 29, 30, 33, 41, 50, 52, 55, 67, 68, 83, 84, 95, 98–100, 102

MCS Maximum Common Subgraph. , 66, 67

ML Machine Learning. , 1, 7

MLP Multilayer Perceptron. , 44

NLP Natural Language Processing. , 2, 7, 8, 10, 11, 14, 16, 53, 55, 98–100

NLU Natural Language Understanding.

NMT Neural Machine Translation. , 64, 95

NNQT Normalized Natural Question Template. , 57, 58

OCL Object Constraint Language. , 14

OML Ontological Modeling Language. , 1, 17, 18, 20, 21, 33–36

OOV Out-of-Vocabulary. , 82

ORKG Open Research Knowledge Graph. , 3, 29, 41, 52, 53, 58, 68, 69, 73, 86, 96, 101

OWL Web Ontology Language. , 17, 34–36

OWL2 Web Ontology Language 2. , 33, 34

PCST Prize-Collecting Steiner Tree. , 31, 41–43, 61, 62, 66, 74, 75, 79–81, 85

PEFT Parameter-Efficient Fine-Tuning. , 71

PGP Placeholder Graph Pattern. , 39, 45, 46, 48, 54, 56, 57, 59, 60, 62, 66, 69, 87, 88, 114

PLLM Pre-trained Large Language Model. , 3, 6, 11, 12, 24, 27, 31, 50, 51, 60, 61, 76, 78, 85, 99, 101

PLM Pre-trained Language Model. , 15

QA Question Answering. , 2, 7–12, 15, 16, 27, 33, 62, 64, 92, 101

QLoRA Quantized Low Rank Adaption. , 71

QU Question Understanding. , 93

RAG Retrieval-Augmented Generation. , 4, 7, 11–16, 31, 43, 51, 52, 55, 62, 74, 79, 83, 95–97, 99

RDF Resource Description Framework. , 3, 11, 34–38, 45, 48, 49, 66, 69, 81

S2S SQUALL-to-SPARQL. , 59

SE Systems Engineering. , 1, 17–19, 33

SP Semantic Parsing. , 7–12, 16

SPARQL SPARQL Protocol and RDF Query Language. , 2–5, 7, 9–11, 20, 21, 23, 24, 26, 27, 30, 34, 38, 39, 45, 46, 51, 53–59, 62, 64, 65, 69, 73, 79–81, 86–98, 101, 114

SQL Structured Query Language. , 11, 13–15

SQUALL Semantic Query and Update High-Level Language. , 4, 11, 12, 30, 31, 33, 39, 44, 45, 51–62, 64, 69, 71, 74–76, 78–81, 84–90, 92–98, 101, 114

SWRL Semantic Web Rule Language. , 33, 34

SysML Systems Modeling Language. , 1, 14

TBox Terminological Box. , 34, 35

UML Unified Modeling Language. , 14

URI Uniform Resource Identifier. , 26, 35, 36, 38, 39, 45, 48, 52, 53, 56–58, 69, 79, 81–83, 87, 89, 90, 92, 98

VTL Velocity Template Language. , 2

ZSL Zero-Shot Learning. , 3, 92

List of Listings

1 Introduction

The need to clearly document complex systems is evident across domains, one example hereof is the relatively recent innovation of model cards for Machine Learning (ML) models [1]. The idea behind them is to document various aspects of the ML model, e.g., how it works or for what it is intended, such that the various stakeholders can easily access relevant information. They are meant to be complementary to datasheets for datasets [2] which address the concerns of two stakeholder groups: dataset creators and consumers. For the latter, these datasheets provide the essential information needed to make informed decisions about how to use the data effectively. This need for information communication is not limited to the Artificial Intelligence (AI) community.

Systems Engineering (SE) is one such domain that deals with complex systems and could reap the benefits of improved information communication. The aerospace sector is a pertinent example where SE is crucial, but to deal with the increasing complexity of systems, a particular type of SE is gaining traction: MBSE. It deviates from the document-centric approach of traditional SE, opting instead for domain models that serve as single source of truth.

In MBSE, a system can be documented and presented in a way that is targeted to specific stakeholder groups, — also called reporting — adapting to their expertise and interests [3]. Reports are a type of document-based presentation of the system providing specific information. One example are information reports [4], which are quite similar to the previously mentioned model cards and datasheets, from an information communication point of view. They give insight, provides analyses, and assist in decision-making throughout a system's life cycle.

The current approach to the reporting of models in MBSE is based around pre-defined templates called viewpoints. It involves generating a document-based view, or representation, of the model from the perspective of a certain viewpoint. These views have been recognized for their contribution in enhancing stakeholder engagement [5]. This general approach is adapted in various formal languages or tools, such as the Systems Modeling Language (SysML) and the Cameo Systems Modeler tool, the Arcadia Language and the Capella tool or the Ontological Modeling Language (OML) and the openCAESAR platform. To help with report creation, tools can have report generators and include pre-defined viewpoints which can be quite numerous in quantity [4].

1 Introduction

Although the current industry-standard approach is quite agile — reports can be tailored to specific stakeholder groups — it is not flexible. Creating a new type of report or updating an existing viewpoint to address a stakeholder's interest in a previously unaddressed aspect of the model requires a significant investment of both time and resources. The reason for this is two-fold; in order to create or update a viewpoint both knowledge of the domain and of the specific technology used for engineering the viewpoints is needed. For example, the technologies used might include a formal query language such as SPARQL [3], [4], [6] and a templating language such as Velocity Template Language (VTL) [6]–[8]. Any change thus requires two highly specialized profiles: a domain expert and a technologist, the former having the necessary knowledge of the domain in question — while typically not having a formal education in software engineering [9] — and the latter having the appropriate skills to define the queries, templates, etc.

To address this flexibility gap, model reporting is proposed: a no-code interactive reporting paradigm for MBSE. It is a type of information reporting restricted to and aimed at documenting the model of the system under consideration. Essential information about the system is conveyed to stakeholders through text, diagrams, tables, etc. Different from the current approach to reporting, a technologist is not required. Instead, its role is to be supplanted by an envisioned agent who takes care of the technicalities, performing the tasks given to it by a user through interaction.

A proof-of-concept is put forward based around the notebook environment from within which a user can interact with an agent — through natural language interaction — who answers question about the model with explanations, tabular results and visualizations. Reports can be created in this environment in an easy and quick way without help from a technologist. For example, the need to know a formal query language is obviated and instead delegated to an agent.

This agent takes the role of the technologist and is responsible for generating appropriate responses to user inquiries. In the current approach the technologist usually writes the appropriate formal language queries, for example, in SPARQL. The well-established domain of Question Answering (QA) might be used to replace the technologist, since it traditionally focuses on generating answers to questions, both in natural language. However, model reporting extends beyond text-based answers: visualizations and access to tabular data are essential components too. Therefore, the agent must not only generate natural language responses but also explicitly produce queries that facilitate these additional tasks. To achieve this, the agent leverages advancements in Natural Language Processing (NLP), particularly by harnessing the capabilities of state-of-the-art Large Language Models (LLMs), which are increasingly being used to address complex, multi-faceted challenges in data-driven tasks.

The key problem with explicit query construction for model reporting is the lack of data. The domains of MBSE are often private and highly specialized. Lack of data is of course an extremely common issue, but the above two realities make it especially troublesome. No high-quality, sizeable and domain relevant labeled datasets can be assumed to exist. This challenge is referred to as the "data scarcity scenario" throughout this work. The primary focus of the remainder of this study is to address and overcome this issue of data scarcity, in order to realize model reporting.

To illustrate the issue, consider the following example involving a domain called Open Research Knowledge Graph (ORKG), which includes papers, their contributors, topics, and more. Suppose a non-technical person would like a question answered which has not yet been addressed in any of the previous viewpoints. Given the time-sensitivity of their inquiry, they give it a shot by directly querying the model. They have no extensive knowledge of the query language, and thus give ChatGPT a try, for instance, they might ask: `Provide a list of papers that have utilized the Depth DDPPO model and include the links to their code?` [10]. Despite providing ChatGPT with the relevant information necessary to answer the question — for example, Resource Description Framework (RDF) triples — they would likely find that ChatGPT is unable to generate a sensible query. This issue arises even though ChatGPT is knowledgeable about SPARQL and has access to domain-specific information. The challenge lies in the relative obscurity of ORKG compared to more widely-known domains and the high complexity of the query itself. See Listing 1.1 for the expected answer and an example of a generated query. Evidently, this is a rather naive approach and much more intricate solutions can be built using Pre-trained Large Language Models (PLLMs). Nevertheless, it does get to the heart of the problem: for certain domains, generating complex queries is not trivial. A recent experimental study [11] supports these intuitions by analyzing the performance of LLMs in generating challenging queries for a specialized domain. The findings clearly demonstrate that Zero-Shot Learning (ZSL) (i.e., prompts without examples) is entirely impractical for complex queries. While some generated queries may resemble the expected results, they are never fully correct, with no exact matches observed [12]. Comparable results to fine-tuning can be achieved by selecting an appropriate model and employing extensive prompt engineering, which involves incorporating relevant examples into the prompt.

```
SELECT DISTINCT ?code WHERE {
        ?model a orkgc:Model;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark orkgp:HAS_DATASET ?dataset .
        ?cont orkgp:HAS_BENCHMARK ?benchmark .
        ?cont orkgp:HAS_MODEL ?model;
                orkgp:HAS_SOURCE_CODE ?code .
}

SELECT ?paperTitle ?codeLink WHERE {
         ?paper orkgp:HAS_MODEL orkg:R123481 .
         ?paper rdfs:label ?paperTitle .
         ?paper orkgp:HAS_SOURCE_CODE ?codeLink .
}
```

Listing 1.1: Ground truth SciQA (above) and ChatGTP-generated (below) query for the same question about ORKG [10].

Data scarcity is addressed through a three-pronged approach. First, the generation target language is shifted from a formal query language (i.e., SPARQL) to a Controlled Natural Language (CNL) (i.e., Semantic Query and Update High-Level Language (SQUALL) [13]), see Listing 1.2 for an example. Using CNLs has been shown to reduce data requirements for LLMs because they are closer to natural language [12]. Second, transfer learning is leveraged through a Retrieval-Augmented Generation (RAG) paradigm, involving:

- **Initial Fine-Tuning:** Fine-tuning of an LLM in a data-rich environment, followed by freezing it, as the first phase of a sequential training strategy.
- **Integration in RAG Model:** Using the fine-tuned LLM as a generator within a RAG framework.
- **Retrieval and Augmentation:** Using a static retriever and post-retrieval processing for non-parametric information, and integrating it through a trainable augmentation method incorporated in the intermediate layers of the frozen LLM (soft prompting).
- **Domain Adaptation:** Conduct a second phase of training on the target domain, leveraging the frozen LLM for its retained capabilities while adapting through augmentation.

The resulting model is dubbed the "domain expert". Finally, to simulate a realistic scenario with some available data, synthetic data is generated from the limited dataset to test the effectiveness of the proposed approach. In conclusion, data scarcity is addressed through a combination of strategies, including Domain Adaptation (DA), the use of a Controlled Natural Language (CNL), the application of the RAG paradigm, and the generation of synthetic data.

```
SQUALL:
What is the name of the author-s of PaperX?

SPARQL:
SELECT DISTINCT ?x1 WHERE {
        :PaperX :author ?x2 .
        ?x2 :name ?x1 .
}
```

Listing 1.2: SQUALL-to-SPARQL example.

From the core problem, the following research questions are formulated: **RQ1:** How does the performance of the proposed domain expert compare to other approaches across various data availability scenarios? **RQ2:** Does the domain expert effectively utilize the retrieved information with which it is soft-prompted, and what factors influence the effectiveness of these prompts? **RQ3:** What is the impact of using synthetic data, as opposed to ground truth data, on domain adaptation performance during the second learning phase?

The contributions of this thesis include the introduction of a novel interactive model reporting paradigm for the MBSE community with an accompanying proof-of-concept implementation, Mo-Lab, that allows a previously unattainable level of flexibility. Furthermore, an approach to the text-to-SPARQL task is presented that is specifically designed for realistic data scarce scenarios using a pipeline that incorporates the proposed domain expert. Finally, an evaluation and comparison of the presented approach with similar methods on a representative domain is included.

This work advances the state of the art in reporting for MBSE by its proposed paradigm shift, enabled through an approach that overcomes data scarcity through domain adaptation.

2

Preliminaries

A significant body of research has been dedicated to streamlining and simplifying tasks such as writing code, documentation, reports, etc. characterized by a repetitive, tedious or otherwise prohibitively technical nature. In the same vein current MBSE techniques exist for generating documents such as reports from model viewpoints [14]. Viewpoints are used in Systems Engineering, but in other domains as well, e.g., software engineering. A concrete example is the physical viewpoint in the context of space systems. This viewpoint focuses on aspects such as the physics of motion, system components, their interconnections, and external forces.[1] The output resulting from a specific viewpoint is referred to as a view; Computer-Aided Design (CAD) models representing physical systems are one example, the reports used in MBSE another. While the techniques used in MBSE to generate reports are useful, they can be supplemented with new AI-based approaches.

Some fundamental issues arise in the proposed approaches, often at least one of two common presuppositions ought to be met for the proposition to be viable. There is either an abundance of high-quality information related to the domain, typically in the form of an extensive dataset comprising domain-specific questions and their corresponding queries. Or, the domain is implicitly assumed to be sufficiently well-known, for example, when prompting a PLLM has sufficiently seen information about the domain during its pre-training such that it can be effective given a proper prompting strategy. A contrived example can highlight the issue, take for example a domain that is private for which no data engineering has of yet been done, hence, high-quality datasets are missing. Clearly, when domain-specific information is not publicly available, PLLMs become ineffective. Furthermore, the investment required to create the necessary datasets can be prohibitive, especially if manual annotation is chosen, as it demands a high level of domain-specific expertise from data annotators [15]. When neither of these conditions is met, this situation — hereafter referred to as data scarcity — poses a significant barrier to progress. Addressing and overcoming this challenge is the central motivation for the research presented here, as it has been identified as the primary obstacle in advancing model reporting.

If a method existed that could flawlessly answer questions, generate queries, and adapt effectively to any domain

[1]Example from https://web.archive.org/web/20100527225452/http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/39798/1/06-1543.pdf.

— no matter how obscure — then implementing model reporting would be primarily a matter of engineering rather than a conceptual challenge. However, the reality is more complex. The subsequent sections will explore existing methods, highlighting their relevance while demonstrating that they are either incompatible with data-scarce scenarios or neither straightforward to implement nor trivial to adapt. This analysis underscores the necessity of the research presented in this work. First, the foundational background of the core technologies underpinning this work is introduced. Next, a cursory survey of relevant literature and industry practices for document generation in other domains is presented. Finally, the limitations of previous approached are discussed.

2.1 Relevant NLP Technologies

This section explores key NLP technologies that form the foundation of the research presented in this work. The primary focus is on the following core techniques:

- **Question Answering**: A discipline focused on creating systems capable of automatically answering natural language questions posed by users.
- **Semantic Parsing**: A task that involves mapping natural language to machine-understandable representations.
- **Domain Adaptation**: A field concerned with applying an ML domain, trained on a source domain, to a related but different target domain.
- **Controlled Natural Language**: A subset of natural languages that have restricted grammar and vocabulary, and are thus less complex.
- **Retrieval-Augmented Generation**: A technique that integrates Information Retrieval capabilities with generative AI to enhance the performance of models.

These technologies are critical in addressing the research challenges related to data scarcity and enabling model reporting in MBSE domains. Recent QA systems informed the general architecture for the proof-of-concept implementation of the agent. Meanwhile, Semantic Parsing (SP) played a more specialized role, informing this work on tackling text-to-SPARQL task. It proved essential for realizing the domain expert used for generating the queries necessary to create relevant artifacts like visualizations and explanations required for model reporting. The final three technologies all played a key role in overcoming data scarcity: Domain Adaptation (DA) enabled the transfer of knowledge from well-resourced domains to the data-scarce ones of MBSE, the use of a CNL contributed by lowering training data requirements, and RAG aimed to enhance the performance of the domain expert by incorporating information unseen during training.

2.1.1 Question Answering, Semantic Parsing and Domain Adaptation

This section introduces three foundational technologies: QA, SP, and DA. These are prioritized as they are integral to the approach developed in this work, and their roles are deeply interconnected within the context of model reporting. Following an overview of each technology, their relevance to the research is analyzed, emphasizing the challenges that make their implementation in this domain non-trivial.

QA Firstly, QA is a discipline at the intersection of NLP and Information Retrieval (IR) that studies systems concerned with answering questions posed by humans. Typically, the QA approach involves three key steps: understanding the question (i.e., Natural Language Understanding), retrieving relevant information (i.e., Information Retrieval), and generating the answer. The information source a QA system can retrieve from varies widely depending on the application. One examples would be technical support where the source of information could be a system's documentation. Interest in QA has surged with the advancements in deep learning technologies [16].

SP Semantic parsing is a task in NLP that involves translating natural language statements into corresponding logical forms. If, for example, the following question is posed: `Tell me the name of the author of PaperX?`, it could be parsed into a dependency graph, see Figure 2.1.

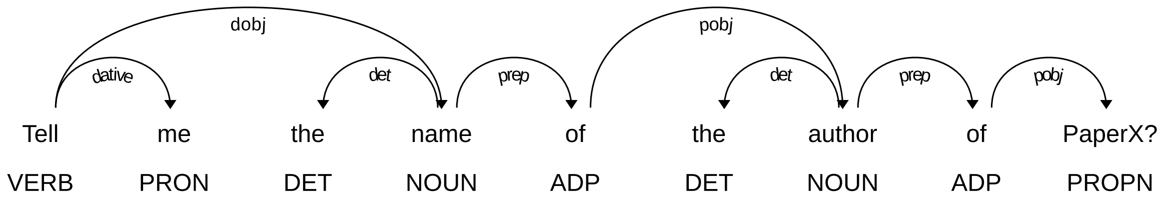


Figure 2.1: The dependency graph of an example question.

The task is alternatively defined as converting natural language to formal representations of meaning or meaning representations. In any case, the resulting output of SP is diverse and includes constituency graphs, dependency graphs, database queries, query graphs [17], semantic graphs [18], lambda-calculus [19], etc. Furthermore, various natural languages are possible, and the logical forms can be task-specific, i.e., dependent on the area of application. Examples of application areas are machine translation and, more pertinently, QA. SP has benefited from deep learning's progress [20], [21].

DA Domain adaptation is categorized as transductive transfer learning in the NLP domain. It aims to leverage data and knowledge from a source domain to the benefit of a target domain under the assumption that source and target domain are distinct, that the task is identical and that some (unlabeled) target domain data is available at training time [22]. As a concrete example, a neural dependency parser could be trained to parse English sentences and be adapted to a less common language through Domain Adaptation.

Relatedness of QA and SP: Domains and Knowledge Bases The domains of QA and SP naturally converge when investigated in the context of Knowledge Bases (KBs). In general, a Knowledge Base (KB) is a structured repository designed to store, manage, and retrieve information efficiently. They are widely used to represent domains and can, for example, be utilized to represent systems in MBSE. Typically, KBs consist of sentences stated in a particular formal language, and information can be retrieved from the repository using another formal language called the query language. When the data in a KB is organized in a graph structure, it is referred to as a Knowledge Graph (KG). A notable example is Wikidata, a KG that serves as a central knowledge repository, supplying structured data to Wikipedia and other Wikimedia projects. The convergence of QA and SP can be illustrated using the previous question if it is stated in the context of, for example, a particular domain consisting of scientific papers and their authors, implemented as a KG. Instead of parsing the question into a constituency graph, it could then be translated into a SPARQL query: `SELECT DISTINCT ?x1 WHERE { :PaperX :author ?x2 . ?x2 :name ?x1 . }`. Execution of the query would retrieve the relevant information needed by a QA system to answer the question. In this fashion a QA system could leverage SP techniques in order to answer a user's question about a particular domain. In QA, when the answers are derived from a KB (or a KG), the task is referred to as Knowledge Base Question Answering (KBQA) (or Knowledge Graph Question Answering (KGQA)). However, SP is not a necessary component of a KBQA system.

Relevance of DA Meanwhile, many SP scenarios are domain-specific [15], meaning that for the same task variations of a semantic parser are needed if the target domains differ. The previous question could be parsed into a different query if the domain did not consist of scientific papers, but instead was concerned with newspapers, yet intuitively there are similarities. Hence, adapting a SP model from a source domain to a target domain has previously been the topic of research [15] using transfer learning with some work aimed focusing on using domain adaptation [23]. Interest in domain adaptation is driven by a desire to overcome data scarcity in domains of interest since contemporary neural semantic parsing techniques usually require significant annotated datasets [15], [23], [24]. However, transferring a trained neural model to a new domain is inherently challenging, primarily due to differences in meaning representations across domains [23].

These three technologies are central to model reporting paradigm. The proof-of-concept implementation of the proposed paradigm centers on openCAESAR, a representative MBSE framework detailed in Chapter 3. This framework employs a KG to model systems. As indicated earlier, in this context answering stakeholder requests entails understanding the question, retrieving information from the KG through querying, and generating the answer in the expected. Hence, from the QA point of view, the proposed implementation of model reporting can be seen as a straightforward application. As will be discussed later on, the proof-of-concept follows a KGQA pipeline combined with further downstream tasks to achieve the expected results such as visualizations and natural language answers. Now it is clear how SP is vital: it enables those later tasks crucial to the paradigm. Namely, SP provides the logical forms which are needed to query the system. Note that although QA cane be done without SP, as mentioned previously, this is not the case for model reporting where an explicit logical form is required. Section 3.3

elaborates further upon this requirement. Finally, overcoming data scarcity, which is ubiquitous in MBSE, will be achieved using effective DA strategies applied to the model used for SP (i.e., the domain expert).

Although these three technologies are applicable to model reporting, it remains a significant challenge to apply previous work because of data scarcity. While some QA and SP approaches do target data-scarce conditions, their applicability is constrained particularly when addressing larger domains, making it hard to apply to MBSE. This section concludes with some examples from the literature. A relevant QA system is Knowledge Graph Question Answering platform (KGQAn) [25], which aims to be universal and capable of answering user questions for any arbitrary KG. It employs a two-stage process: a coarse stage for understanding the question and a fine stage resulting in the final logical forms (e.g., SPARQL queries). The question understanding step uses a domain-independent Language Model (LM) to generate sketches of the logical forms (i.e. lists of triples), while the second step finalizes them without relying on a trained model. The platform's approach diverges from traditional SP in an attempt to be applicable to more than one domain without domain specific training data. The approach results suffers fundamental limitations though as detailed in Section 5.7.3.1. Many natural language utterances cannot be correctly converted into equivalent logical forms due to the limitations of the question understanding LLM during the coarse stage, and the reliance on heuristic methods for the fine stage. These constraints highlight the need for alternative strategies to address the demands of model reporting. ProtoParser [15] exemplifies a neural semantic parser designed for data-scarce conditions. It operates in two stages: template generation and slot filling, leveraging synthetic data generation and transfer learning to manage data scarcity. ProtoParser's evaluation spans three domains with predicate counts of 15, 24, and 88, respectively, where the number of unique predicates is a measure of the size of a domain represented by a KG. Although ProtoParser is evaluated for new, i.e., unseen predicates, it remains unclear how its performance scales to more complex, encompassing domains. For instance, the domain evaluated later in this thesis involves nearly 7 000 unique predicates (see Section 5.1.1). This raises doubts about the feasibility of applying ProtoParser to model reporting tasks, where the diversity and scale far exceed those in ProtoParser's tested scenarios. ProtoParser assumes that domain knowledge — acquired during training on structured data, such as database schematics — can generalize to unseen predicates within the same domain. While ProtoParser does not employ domain adaptation, other approaches do, [23], [26], [27]. However, these methods share similar limitations: they focus on the same domains with relatively few predicates, between 15 and 45 predicates [28], which is not representative of the scale of KGs used in MBSE.

2.1.2 Controlled Natural Languages

A Controlled Natural Language (CNL) is a language characterized by a restricted grammar and vocabulary, making it less complex than natural languages. These CNLs are designed for various purposes, such as enhancing readability for non-native speakers. In the context of this thesis, their primary significance lies in their applicability to NLP. Specifically, they are employed for their capacity to intuitively and naturally represent formal notations, such as SPARQL queries [29]. The semantic parser introduced in this work is tailored to address the challenge of data scarcity, and CNLs play a supporting role in this effort. The following sections will elucidate how leveraging a CNL contributes to mitigating this central obstacle to model reporting. Details of the semantic parser can be found in Chapter 4.

Neural semantic parsers have garnered significant attention, particularly with advancements in LLMs and PLLMs. These parsers typically generate logical forms such as Structured Query Language (SQL) or SPARQL queries. However, alternative output formats have also been investigated. A notable hypothesis proposed by [12] suggests that generating outputs in CNL instead of traditional logical forms can enhance performance while reducing the data requirements for the underlying LLMs. This highlights the relevance of CNLs to this work: their use has the potential to decrease the data demands of the semantic parser, a critical factor in addressing the challenges of data scarcity.

A whole host of CNLs exists, but this thesis only focuses on one: SQUALL. It is a CNL and a formal language that can be unambiguously translated or mapped into SPARQL. SQUALL covers all SPARQL constructs, including many of SPARQL 1.1 [13], e.g., it can be used to query and update RDF graphs. Returning to the earlier example question: `Tell me the name of the author of PaperX?`, which was translated to a SPARQL query: `SELECT DISTINCT ?x1 WHERE { :PaperX :author ?x2 . ?x2 :name ?x1 . }`. This query can be mapped to the following equivalent SQUALL expression: `What is the name of the author-s of PaperX?`, which is not significantly different from the original natural language question upon first sight. However, a very particular sentence structure is used, and the vocabulary is indeed restricted.

As introduced in Section 2.1.1, the proof-of-concept implementation targets a domain represented by a KG, more specifically, an RDF graph. RDF graphs are queried in SPARQL, hence a CNL mappable to SPARQL was needed. It was shown [12] that SQUALL is a good choice likely due to its remarkable similarity to natural language, leading to relatively good performance.

Although the use of CNLs has been identified as a promising new avenue [30] for KGQA and SP, QA implementations applicable to model reporting were not found. Furthermore, a difficulty related to neural SP is that LLMs generally lack extensive knowledge of the target CNLs due to the infrequent occurrence of CNLs in the training data of LLMs. Prompting these models to generate CNL expressions seems highly non-trivial. Indeed, the work [12] that proposed this hypothesis relied on fine-tuning LLMs using custom datasets consisting of natural language utterance/CNL-expression pairs. Thus, data scarcity once again presents a significant and unresolved hurdle.

2.1.3 Retrieval Augmented Generation

RAG models have been suggested [31] as a way to tackle knowledge-intensive NLP tasks by incorporating non-parametric external knowledge in LMs. Since then numerous novel variations have proliferated, occasionally dubbed an architecture, a framework or a paradigm. Irrespective of the particular RAG description utilized three steps and corresponding components are essential: retrieval, augmentation and generation [32]. The first component is responsible for retrieving knowledge relevant to the task (e.g., KGQA). Knowledge can be contained in many formats, including textual documents or graphs. Augmentation refers to the process of enhancing retrieval to optimize LLM generation. This process can be employed at different stages, including LLM pre-training, fine-tuning, or inference. A notable example of augmentation is iterative retrieval, which involves progressively refining search queries based on feedback from previous results. This approach aims to improve the search process by iteratively narrowing the focus to the most relevant information through a dynamic feedback loop [32]. The final component is the generator, typically involving only a generation step by the LLM. However, post-processing the generated output can also be included to enhance results.

The reason for introducing RAG is to solve the problems associated with the lack of an LLM's knowledge. One of these issues are the well-researched hallucinations, shown to have many causes, including the inevitably limited or outdated knowledge contained in the LLM's parameters [32], [33]. QA is a prominent example of a task where RAG is beneficial, but it is applied to a range of generative tasks, for example, dialogue response generation and machine translation [34]. Its usefulness to QA stems from the inclusion of external knowledge not contained in the LM's weights which can increase accuracy and credibility [32].

As before, two complications present themselves: data scarcity and domain size. Both factors increase the complexity of information retrieval. The large domain size, in particular, makes the prompting of PLLMs non-trivial. Thus, identifying the relevant facts from the domain and incorporating them as manageable input to a model is challenging.

2.1.4 Synthesis of Relevant Technologies

To enable effective model reporting, a QA system framework is adopted, with the semantic parser serving as its central component. This parser is designed to generate queries that address questions posed by stakeholders. Leveraging domain adaptation, the model utilizes data from data-rich domains to mitigate the challenge of data scarcity.

Additionally, the semantic parser outputs a CNL known as SQUALL, which has been shown to reduce data requirements, thereby further addressing the issue of data scarcity. This specialized parser is referred to as the SQUALL expert.

Finally, the integration of the SQUALL expert into a RAG architecture incorporates non-parametric domain knowledge. This approach aims to address the inherent limitations of LLMs in handling knowledge-intensive tasks such as QA and SP. The resulting RAG model, previously referred to as the domain expert, represents the culmination of this synthesis of technologies.

2.2 Related Frameworks for Automating Reporting

Numerous paradigms and frameworks have attempted to leverage AI to assist, automate, accelerate or simplify the often tedious and time-consuming tasks of creating documentation, presentations, reports, etc. These approaches span a spectrum of automation, from entirely manual to fully automated, with assisted generation occupying an intermediate position. Interest in this area has grown significantly, and with recent advancements in deep learning, practical implementations have begun to proliferate. Examples include an interactive agent for generating financial reports, such as slide decks, from natural language [35], report generation from medical images [36], [37], bash command generation [38], and agent-assisted presentation generation from data science notebooks [39]. To provide context for this work and clarify the rationale behind its design choices, some examples that have tackled automating reporting are discussed below.

Concretely, this section explores the following:

- **Reporting Systems for SQL Databases**: Systems that simplify querying of SQL-based databases using natural language, facilitating enterprise reporting and analytics, often incorporating RAGs and LLM.
- **AI Agents**: AI assistants that help automate or augment tasks like creating reports or presentations from computational notebooks, emphasizing human-in-the-loop approaches for maintaining coherence and flexibility.
- **KBQA and KGQA Systems**: Frameworks focusing on answering user questions through querying KGs using LLMs and retrieval techniques, aiming to enhance knowledge-intensive tasks with RAG paradigms and non-parametric memory.

2.2.1 Reporting Systems for SQL Databases

Systems simplifying reporting by enabling natural language querying of SQL-based databases have been previously suggested. An end-to-end Query Enterprise Data (QED) system was proposed [40] to facilitate natural language and conversational search on large databases, it is aimed at enterprise reporting and analytics and uses graphical, tabular, and textual reporting interfaces. As possible future research, extending the approach to knowledge graphs was also mentioned. Vanna is another KBQA system whose approach also hinges on the formation of formal language (i.e., SQL) queries explicitly allowing post-processing such as tables and charts.[2] It is a newer framework making use of the RAG paradigm. Finally, FinSQL [41] is a financial analysis framework

[2]Vanna available at https://vanna.ai.

designed for financial professionals also based on text-to-SQL. It addresses the technical hurdles faced by stakeholders lacking programming skills to interact with the KBs used in the finance sector. To overcome these barriers, the authors present a text-to-SQL framework for financial analysis with interactive features — although these are only mentioned in passing such as context retention, natural language summaries, and various forms of visualization.

These reporting systems have quite similar aims as model reporting. Key observations include:

- The need for interactivity and trust in a reporting system, and the use of synthetic data to combat data scarcity [40]. To combat data scarcity in industrial settings, synthetic data is generated to train the models.
- The integration of LLMs and RAG forms the core foundation of Vanna's functionality.
- Achieving data efficient domain transfer (i.e., reusing base model for different databases) through Low-Rank Adaptation (LoRA) modules [41].

However, it is important to note that FinSQL, QED and Vanna are primarily designed for relational databases and the text-to-SQL task. In contrast, the models used in MBSE are defined and queried differently. For instance, SysML/Unified Modeling Language (UML) models are queried using Object Constraint Language (OCL).

Similarly, openCAESAR and its associated models cannot be queried using these techniques. For further details on querying in the context of the proof-of-concept implementation, see Section 4.1.1.2.

2.2.2 AI Agents

Slide4N [39] is an AI assistant that helps data scientists to create slides from computational notebooks. A human-in-the-loop solution was chosen, resulting in an interactive framework, instead of going for full automation. Downsides to full automation that where mentioned include limited choice in generation, and decreased efficiency if generated content is not satisfactory. The architecture consists of a front-end and a NLP powered back-end.

Slide4N's relevancy to model reporting is clear through analogy: ideally an AI assistant would help MBSE stakeholders to create reports from models. Like Slide4N the proposed proof-of-concept also introduces a human-in-the-loop approach instead of going for complete automation. That way some creative control over the final model report is kept.

2.2.3 Knowledge Base/Graph Question Answering

Various frameworks have been suggested in the literature focused on chatting with KGs. Common techniques that stand out are the use of LLMs to power conversations and nearest neighbor-based embedding retrieval techniques. Utilizing LLMs in graph analysis has been explored [42], they submit a framework that allows chatting with for example chemical molecules or social networks. A user's questions are answered through a dialog, and their analysis performed, including graph comparison and cleaning. The graph operations needed to do this are implemented through consecutive Application Programming Interface (API) calls, dubbed chains. Similar to document embeddings, the calls are embedded in a space. The user prompt is made up of an inputted text and an uploaded graph. The first is embedded into the API space after which the most similar calls are retrieved using an approximate nearest neighbor search. While the latter is first sequentialized, then using the retrieved API calls and sequentialized graph, an LLM constructs a chain of calls to operate on the graph in order to achieve the user's question. The G-Retriever chat graphing framework [43] puts emphasis on KGQA for real-world graphs. It tackles the problem using an LLM and the RAG paradigm. Initially the vertices and edges of the target graph are embedded using a Pre-trained Language Model (PLM) and stored in a nearest neighbor data structure. When the user poses a question it is embedded by the same PLM after which the k most similar — based on a cosine similarity function — vertices and edges are retrieved and from which a connected subgraph is constructed using a modified Prize-Collecting Steiner Tree algorithm. The prize, i.e., value of both vertices and graphs, is determined by the cosine similarity. The algorithm finds the connected subgraph with maximal total prize subtracted by a cost function, dependent on the size of the subgraph. The cost functions act as a regularization parameter, without it the biggest connected subgraph would always be chosen. LLMs have limited context windows, hence this regularization helps when scaling to bigger graphs. A final example is RAG-end2end [44], a RAG paradigm proposed to specifically handle QA on KBs of specialized domains.

These first two approaches are relevant to the proof-of-concept implementation as they specifically address inquiring into KGs using natural language while G-Retriever and RAG-end2end exploit the RAG paradigm incorporating a KG and KB as non-parametric memory, respectively. However, these approaches do not perform explicit query construction and the latter two have significant data requirements.

2.2.4 Recap of Related Frameworks

In this section, multiple frameworks were explored that were aimed at automating reporting using AI.

Systems like Vanna and FinSQL focus on natural language querying of SQLs databases, emphasizing LLM and RAG paradigms, though they target relational databases, not MBSE models.

An assistant like Slide4N assists in generating reports from computational notebooks using a human-in-the-loop approach, importantly maintaining some level of creative control.

Frameworks like G-Retriever and RAG-end2end utilize LLMs and RAG paradigms for querying KBs, focusing on incorporating non-parametric memory.

These frameworks highlight valuable techniques, though their focus on relational databases and heavy data dependencies pose difficulties for MBSE and its data-scarce domains.

2.3 Open Challenges

The current state of research presents several limitations to model reporting for MBSE.

Existing approaches in QA and SP (see Section 2.1.1) do not fully address the unique demands of model reporting. The scale and nature of MBSE domains complicate the application of these solutions.

Many QA and SP approaches depend on labeled training data for the specific domains, and generally certain public datasets are used to evaluate them, e.g., LC-QuAD 2.0 and QALD-9-PLUS. These systems are inapplicable when no data is available, although DA and synthetic data generation approaches show some limited success, this is mainly for much smaller domains than encountered in practice.

Recent work has indicated the promising potential of CNLs to further aid in this regard by reducing training data requirements of the neural models used for the SP task. However, current LLMs have limited or no knowledge of CNLs, because of the scarcity of such data in their training corpora. Fine-tuning LLMs for CNL generation tasks still requires substantial datasets, which are typically unavailable in specialized domains.

RAG-based architectures show promise for QA in specialized domains, but in practice SP, i.e., explicit query construction is often eschewed for answers which can complicate or make impossible downstream post-processing tasks that are essential to model reporting.

After exploring these relevant NLP techniques, a closer look was taken into particular frameworks with similar goals as model reporting, including some QA systems and semantic parsers. Although these frameworks make use of the aforementioned techniques — for example, using CNLs to lower data requirements and RAG to incorporate non-parametric memory — it is unclear how they can be adapted or emulated for MBSE.

The fundamental challenge for model reporting lies in the scarcity of domain-specific data, which is inherent to MBSE. Despite the exploration of various frameworks, none of the examined ones have successfully addressed this critical issue. Using the presented NLP techniques this thesis solves the data scarcity issue in MBSE, paving the path for model reporting.

3

Proposed Solution

This chapter presents the proposed paradigm shift along with its proof-of-concept implementation, Mo-Lab. The requirements for the implementation are subsequently defined. Finally, the chapter demonstrates how the proof-of-concept satisfies these requirements through validation. To provide a clearer context, the discussion first begins with a detailed introduction of MBSE.

3.1 Introduction

As previously stated, MBSE is a relatively recent development in SE turning away from the document-centric earlier approaches towards a formalized model-centric one where a model serves as single source of truth instead of a variety of sources such as text files and schematics. Multiple tools exist that cater to this methodology, for example, Cameo Systems Modeler, Innoslate and openCAESAR. Hereafter, the latter will be used to showcase some general capabilities of an MBSE framework — given that the proof-of-concept implementation, presented later on, is targeted at the openCAESAR platform — however, the proposed paradigm of model reporting is meant for all of MBSE.

At the heart of the openCAESAR platform is the OML language used to author models. Model authoring is supported by openCAESAR through several OML workbenches, with the Rosetta workbench being the most prominent [3].[1][2]

A survey of 148 ontology engineering projects from academia and industry provides valuable insights into the preferred languages for ontology development. Notably, Web Ontology Language (OWL) emerged as the most commonly used language, accounting for 30% of the projects [45]. Given that OML is mapped to OWL (for details, see Section 4.1.1.2), openCAESAR can be considered representative. Consequently, the proof-of-concept is applicable to a significant portion of MBSE, further reinforcing its relevance.

[1] Rosetta is an Eclipse RCP, making it comparable in use to the IDEs commonly used in software engineering.

[2] Rosetta is available at https://github.com/opencaesar/oml-rosetta.

Kepler16b Running Example The Kepler16b demo project is used as a running example throughout this chapter with examples taken from the OML tutorials and from the work presenting the openCAESAR platform [3] as well.[3][4][5] Kepler16b is an exoplanet orbiting the binary star system Kepler16. It serves as an illustrative mission and a case study for the openCAESAR platform [3]. The mission comprises two spacecraft: an orbiter and a lander. Each spacecraft is tasked with scientific missions, such as characterizing the planet's atmosphere, environment, and gravitational field. To achieve these objectives, the spacecraft are composed of multiple subsystems, including electrical, thermal, telecom, mechanical, and propulsion, along with their respective components. This example thus represents a complex system, necessitating the use of MBSE methodologies and tool support.

Five key aspects of MBSE are discussed: system design, requirements, analysis, verification and validation.[6]

System Design First, a model representing the system is designed. In openCAESAR's case, the platform enables authors to define their system of interest using a language called OML as the central formalism for representing domain knowledge in models [3]. OML is elaborated upon in Section 4.1.1.2, here an example is more illustrative. See Listing 3.1, it shows the Lander Mission along with its objectives and its components. System design is complex, and to aid in the process MBSE toolchains typically provide ways to visualize the models. An example of this is given by Figure 3.1, which shows the objectives pursued by the familiar Lander Mission and a new Orbiter Mission.

```
ci_lander : mission:Mission [
         base:hasIdentifier "M.02"
         base:hasCanonicalName "Lander Mission"
         mission:pursues_objectives :characterizeatmosphere
         mission:pursues_objectives :characterizeenvironment
         mission:deploys_components :lander-launch-system
         mission:deploys_components :lander-spacecraft
         mission:deploys_components :lander-ground-datasystem
         mission:deploys_components :mission-operationssystem
]
```

Listing 3.1: Kepler16b description model excerpt showcasing the Lander Mission, its objectives and its components [3].

System Requirements Defining a system's requirements is essential to SE. For example, the Ground Datasystem component — deployed by the Lander Mission, see Listing 3.1 — could be required to present a particular interface. See Listing 3.2 for how it can be specified in OML.

[3] Kepler16b demo project available at https://github.com/opencaesar/kepler16b-example.

[4] Kepler16b demo documentation used for Figure 3.2 is available at http://www.opencaesar.io/kepler16b-example/doc.

[5] OML tutorials used for Listing 3.2 is available at https://www.opencaesar.io/oml-tutorials.

[6] Key aspects identified by The International Council on Systems Engineering (INCOSE) as reported at https://sdincose.org/wp-content/uploads/2011/12/SEVision2020_20071003_v2_03.pdf.

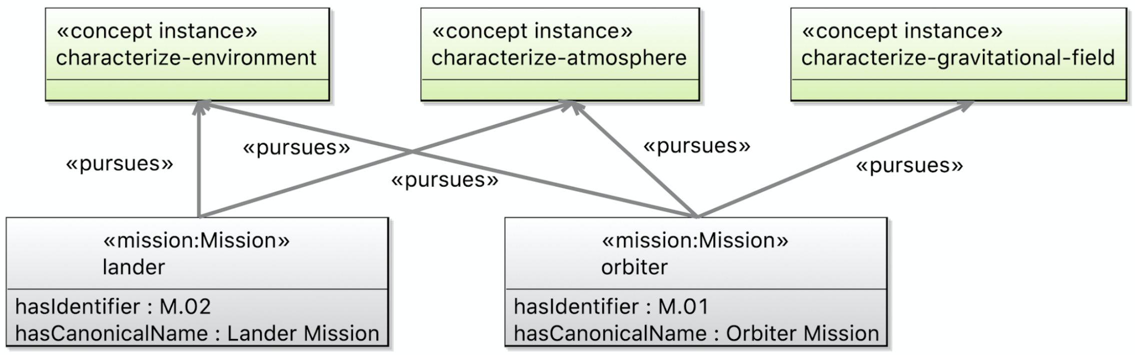


Figure 3.1: Schematic of the objectives pursued by the Lander and Orbiter Mission.

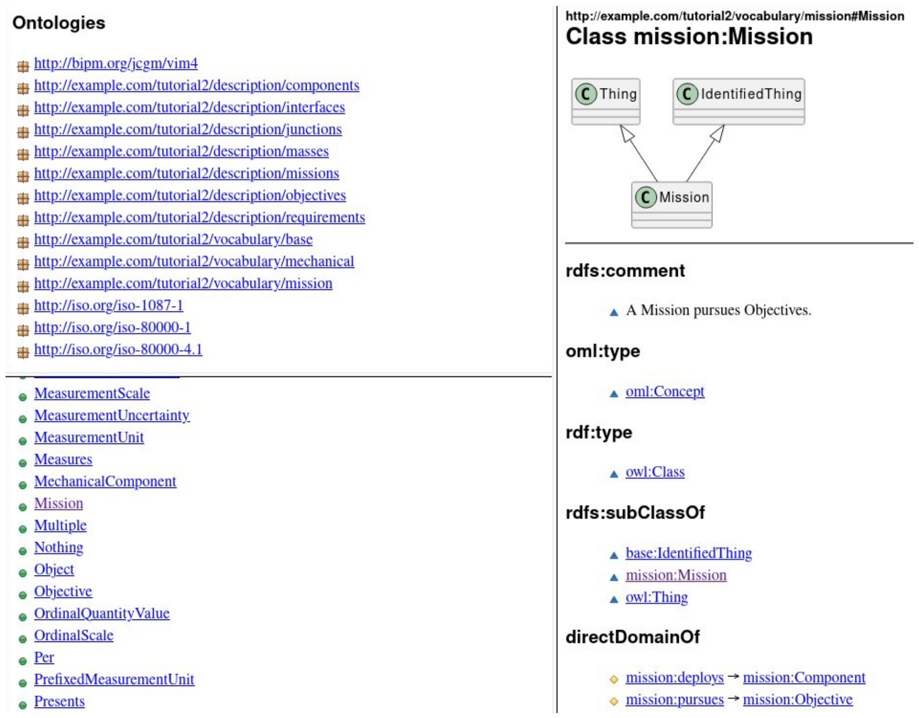
```
instance lander-ground-datasystem-command-to-spacecraft : mission:Requirement [
        base:hasIdentifier "R.04"
        mission:specifies interfaces:lander-ground-datasystem.presents.commandOut
]

// Inclusion of the requirement in the requirement documentation.
> Requirement 'R.04' specifies that component 'Lander Ground Datasystem' shall present
 ↪  interface 'Command Out'.
```

Listing 3.2: Kepler16b example requirement and documentation.

System Analysis Broadly, MBSE tools allow for analysis of the defined system. What the analysis entails depends on the domain of the system, the tools used, etc. For example, if the defined system is a digital twin, i.e., virtual representation of a physical object then an MBSE tool could be used to run simulations with the intent of analyzing how the system might behave under certain conditions. Specifically for openCAESAR, analyses can be run to check for any logical inconsistencies in the model [4]. If, for example, missions were defined to pursue objectives, an error would be indicated upon specifying that the Lander Mission pursues the Lander Ground Datasystem component, given that the latter is not an objective.

System Verification and Validation System validation by stakeholders can be effectively conducted using documentation and reports. Tools can provide automatic documentation of the system as is the case in openCAESAR. Furthermore, MBSE tools often provide graphical wizards that enable users to create reporting templates. These templates are then utilized by the tool's report generator to produce up-to-date reports directly from the model. Unlike traditional document-based approaches to SE, this method ensures that reports remain consistent with the model. Figure 3.2 presents an example of a generated system documentation in openCAESAR, while Figure 2–Figure 4 shows a report. Another example is the generation of requirement documents from the model using a requirement document generator. For the Lander Mission and its Ground Datasystem component, refer to Listing 3.2, which illustrates how the requirement is integrated into the requirement documentation.


Ontologies

- http://bipm.org/jcgm/vim4
- http://example.com/tutorial2/description/components
- http://example.com/tutorial2/description/interfaces
- http://example.com/tutorial2/description/junctions
- http://example.com/tutorial2/description/masses
- http://example.com/tutorial2/description/missions
- http://example.com/tutorial2/description/objectives
- http://example.com/tutorial2/description/requirements
- http://example.com/tutorial2/vocabulary/base
- http://example.com/tutorial2/vocabulary/mechanical
- http://example.com/tutorial2/vocabulary/mission
- http://iso.org/iso-1087-1
- http://iso.org/iso-80000-1
- http://iso.org/iso-80000-4.1

- MeasurementScale
- MeasurementUncertainty
- MeasurementUnit
- Measures
- MechanicalComponent
- Mission
- Multiple
- Nothing
- Object
- Objective
- OrdinalQuantityValue
- OrdinalScale
- Per
- PrefixedMeasurementUnit
- Presents

http://example.com/tutorial2/vocabulary/mission#Mission

Class mission:Mission

rdfs:comment

- A Mission pursues Objectives.

oml:type

- oml:Concept

rdf:type

- owl:Class

rdfs:subClassOf

- base:IdentifiedThing
- mission:Mission
- owl:Thing

directDomainOf

- mission:deploys → mission:Component
- mission:pursues → mission:Objective

Figure 3.2: Kepler16b documentation excerpt.

3.2 System Description

Now that a general impression of MBSE practices is painted, an example follows of a current reporting approach, then the model reporting paradigm is introduced, followed by the proof-of-concept implementation. Attention is paid as to how the implementation practically differs from current MBSE approaches.

3.2.1 The openCAESAR Analysis Pipeline

The openCAESAR framework supports the creation of viewpoints from OML repositories [3]. These viewpoints are templates serving as tailored perspectives that consolidate and interpret data from the source models, converting raw information into insights.

The openCAESAR analysis pipeline, utilized to produce these reports, consists of three key steps: querying, reduction, and rendering. This process is exemplified through the Kepler16b running example.

Querying, or executing a query, entails retrieving data from the model in a structured and systematic manner using a formal language — specifically, SPARQL in this context. Section 4.1.1.2 provides an in-depth explanation of how an OML source model is queried. For the purpose of this section, however, SPARQL queries can be viewed as a systematic, programmatic approach to answering questions about the system, as opposed to a manual process.

Executing a viewpoint involves three key stages: retrieving the most up-to-date information from the source model (querying), transforming the retrieved data (reduction), and presenting the results as a view or report (rendering).

3.2.1.1 Querying

The starting point of the views and reports are SPARQL queries. Essentially, they serve to retrieve information from the OML models, as elaborated in detail in Section 4.1.1.2. Listing 3.3 shows a query that aims to retrieve components and their mass, its results are shown in Listing 3.4. Notably, the query demonstrates a high level of complexity, incorporating two `OPTIONAL` clauses. These relational operations enable the query to return data from the model where applicable, accommodating scenarios where certain information may not always be available.

```
PREFIX base:        <http://example.com/tutorial2/vocabulary/base#>
PREFIX mission:     <http://example.com/tutorial2/vocabulary/mission#>
PREFIX vim4:                <http://bipm.org/jcgm/vim4#>

SELECT DISTINCT ?c1_id ?c1_name ?c1_mass ?c2_id ?c2_name
WHERE {
        ?c1 a mission:Component;
                base:hasIdentifier ?c1_id;
                base:hasCanonicalName ?c1_name .
        OPTIONAL {
                ?c1 base:isContainedIn ?c2 .
                ?c2 base:hasIdentifier ?c2_id;
                        base:hasCanonicalName ?c2_name .
        }
        OPTIONAL {
                ?c1_mass_mag vim4:characterizes ?c1;
                        vim4:hasDoubleNumber ?c1_mass .
        }
}
ORDER BY ?c1_id ?c2_id
```

Listing 3.3: Kepler16b components query [3].

3.2.1.2 Reduction

Reductions transform the data into a format closer to the view. Much can be done through querying using available transformations, for example, relational operations such as `FILTER` or aggregation and solution modifiers such as `GROUP BY` and `COUNT` [46]. Still, various reasons make the reduction step important including limitations of the

```
{ "head": {
                "vars": [ "c1_id" , "c1_name" , "c1_mass" , "c2_id" , "c2_name" ]
        },
        "results": {
                "bindings": [
                {
                        "c1_id": { "type": "literal" , "value": "C.02.01" } ,
                        "c1_name": { "type": "literal" , "value": "Orbiter Power
                         ↪  Subsystem" } ,
                        "c1_mass": { "type": "literal" , "datatype":
                         ↪  "http://www.w3.org/2001/XMLSchema#double" , "value":
                         ↪  "297.0" } ,
                        "c2_id": { "type": "literal" , "value": "C.02" } ,
                        "c2_name": { "type": "literal" , "value": "Orbiter Spacecraft"
                         ↪  }
                },
                // Etc.
                ]
        }
}
```

Listing 3.4: Kepler16b components query results [3].

query language, and difficulty in expressing the desired manipulations relative to a more high-level programming language.

3.2.1.3 Rendering

Finally, the data can be presented to stakeholders through static documents or interactive viewers, often published to a repository for easy access.

For example, Figure 3.3 illustrates a mass roll-up of the Orbiter Mission, detailing the mass of all the spacecraft's components and their subcomponents. A more comprehensive demonstration is provided in Figure 2–Figure 4, which showcase a complete report.

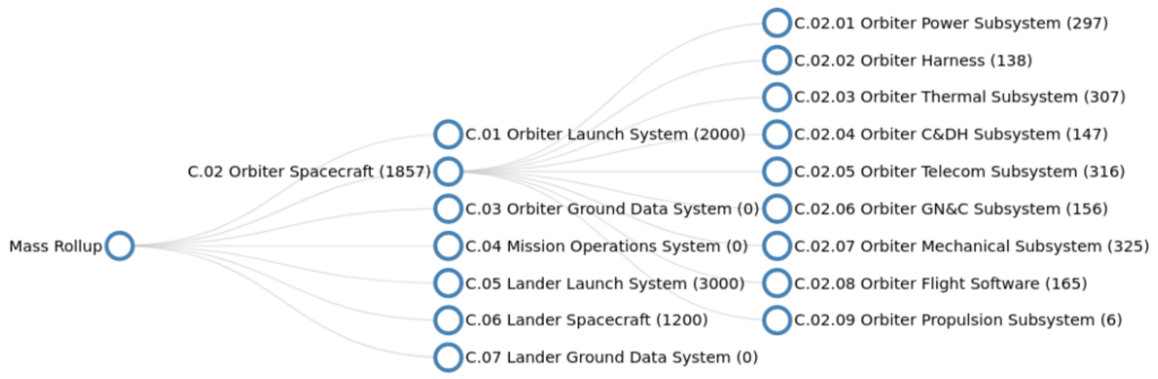

Mass Rollup
C.02 Orbiter Spacecraft (1857)
C.01 Orbiter Launch System (2000)
C.03 Orbiter Ground Data System (0)
C.04 Mission Operations System (0)
C.05 Lander Launch System (3000)
C.06 Lander Spacecraft (1200)
C.07 Lander Ground Data System (0)
C.02.01 Orbiter Power Subsystem (297)
C.02.02 Orbiter Harness (138)
C.02.03 Orbiter Thermal Subsystem (307)
C.02.04 Orbiter C&DH Subsystem (147)
C.02.05 Orbiter Telecom Subsystem (316)
C.02.06 Orbiter GN&C Subsystem (156)
C.02.07 Orbiter Mechanical Subsystem (325)
C.02.08 Orbiter Flight Software (165)
C.02.09 Orbiter Propulsion Subsystem (6)

Figure 3.3: Interactive mass roll-up visualization of the Orbiter Mission [3].

3.2.1.4 Gap

Some key inflexible aspects of the openCAESAR analysis pipeline have now become clear:

1. **Querying:** Writing non-trivial SPARQL queries requires technical expertise.

2. **Reduction:** To handle the query results, data engineering skills are necessary.

3. **Rendering:** Appropriate visualization is possible through data visualization skills.

3.2.2 Paradigm

In contrast to current practices, model reporting is here envisioned as an AI-assisted process centered around a notebook environment. This approach empowers stakeholders to create reports through natural language interactions with an intelligent agent. The agent acts as a mediator, bridging the gap between the user and the system by handling technical complexities.

This paradigm aims to enhance flexibility in reporting by freeing stakeholders from rigid, pre-defined templates. The agent responds to stakeholder queries with a variety of outputs, including textual explanations, tables, schematics, figures, and more. Users can reorganize and adapt these outputs as needed, leveraging the notebook environment's text-processing features, such as markup, to customize and structure their reports.

Rather than requiring stakeholders to construct formal language queries or possess specialized knowledge about report generation in conventional tools and platforms, this framework shifts the focus to a conversational process. Through natural language prompts, the notebook environment facilitates seamless interaction with the AI-agent.

The ideal implementation enables the agent to engage in dynamic conversations, accommodate specific requirements for outputs, and shield stakeholders from technical intricacies, fostering an intuitive and efficient reporting experience.

3.2.3 Implementation

This section gives a summary introduction on the implementation further elaborated upon in Chapter 4.

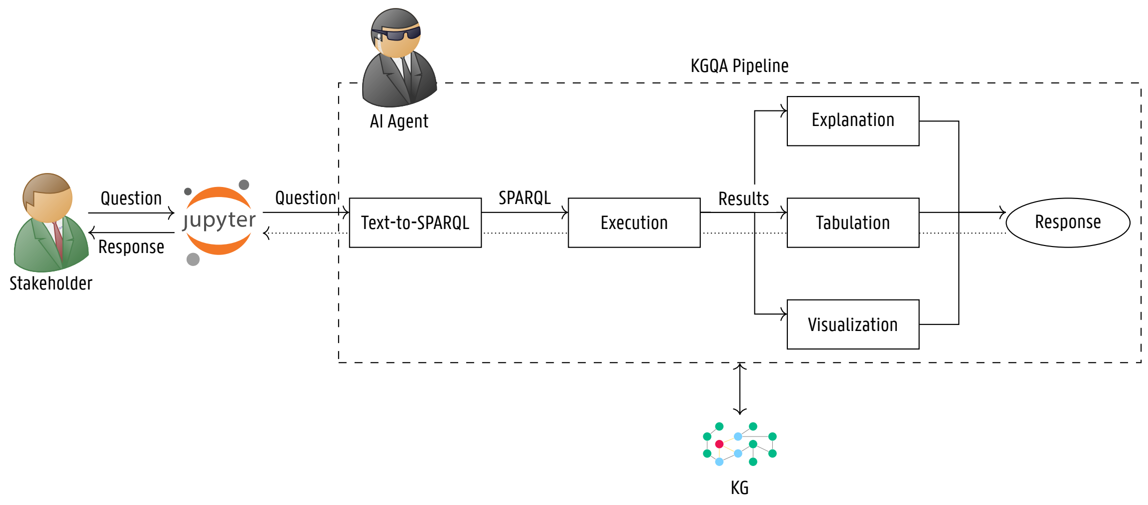


Figure 3.4: Proof-of-concept implementation of the model reporting paradigm.

Due to the inherent complexity of realizing the proposed paradigm, the initial version simplifies the agent's functionality. This proof-of-concept implements the agent as a straightforward and representative KGQA pipeline, as depicted in Figure 3.4. The process follows these sequential steps:

1. A stakeholder submits a natural language inquiry.
2. The inquiry is translated into SPARQL and executed against the KG.
3. Post-processing is applied, limited to one of the following actions:
    - Execute the SPARQL query and return tabulated results.
    - Ask a PLLM, such as ChatGPT, for an explanation based on the question and query execution results.
    - Pass the query to an available visualization engine from the literature [47], which can infer and produce the most suitable type of figure.
4. The final results are returned to the stakeholder's working environment, such as their notebook.

Figure 3.5 shows an excerpt of a model report with an example of each post-processing option.

```
Question:
What evaluation metrics are commonly used when benchmarking models on the FSNS - Test
 ↪  dataset?

Natural Language Explanation:
> The evaluation metric commonly used for benchmarking models on the FSNS - Test
 ↪  dataset is sequence error, which measures the fraction of predictions where the
 ↪  entire predicted sequence does not exactly match the ground truth. It focuses on
 ↪  the accuracy of the full output sequence, making it particularly suited for tasks
 ↪  like text recognition, where exact matches are crucial.

Question:
Provide a list of research paper titles and IDs that have benchmarked models on the
 ↪  Penn Treebank dataset?

Tabulation:
```

Paper	Paper Label
http://orkg.org/orkg/resource/R129331	Generalizing Natural Language Analysis through Span-relation Representations
http://orkg.org/orkg/resource/R130817	Direct Output Connection for a High-Rank Language Model

```
Question:
Provide a histogram showing the mean installed capacity grouped by energy source, with
 ↪  data aggregated into 5-year intervals?

Visualization:
```

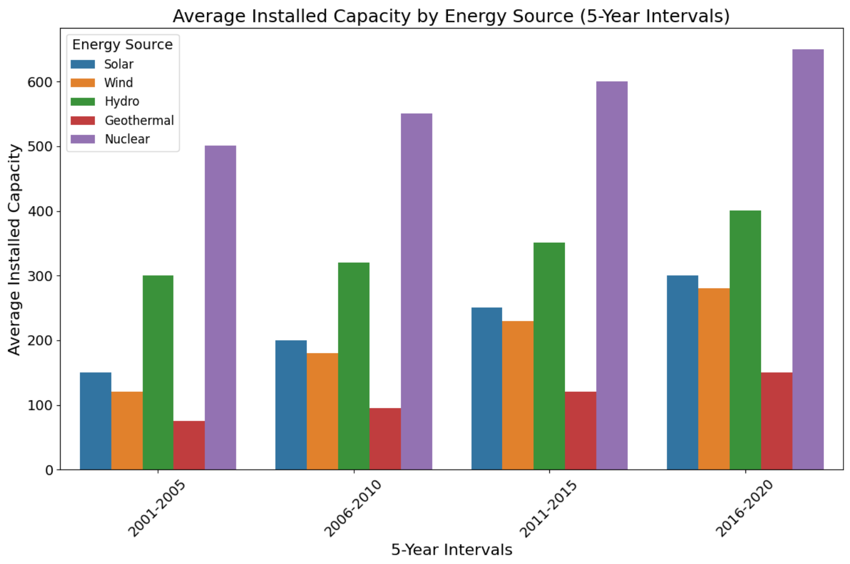


Figure 3.5: An example of a model report including three questions about a scientific domain.

The most significant challenge in the proof-of-concept implementation was the text-to-SPARQL component (see Figure 3.4). As a result, this component serves as the primary focus of this work. Secondary considerations include components post-processing the results of executed SPARQL queries, for which some initial solutions, such as visualization, are proposed. The goal is to assess the feasibility of the proposed paradigm and its effectiveness in reducing technical barriers for stakeholders (assuming a data-scarce scenario such that the assessment is as representative as possible). The following chapter elaborates upon the implementation, going into depth on the notebook environment and the KGQA pipeline.

3.2.3.1 Shift

The proof-of-concept illustrates an alternative analysis workflow, addressing the inflexibility of the openCAESAR pipeline:

1. **Querying:**

   The introduction of a text-to-SPARQL approach eliminates the need for technical expertise. The AI-agent processes natural language prompts from users, seamlessly translating them into queries.

2. **Reduction:**

   Data manipulation is fully automated, relieving users of the burden of manually processing data.

3. **Rendering:**

   Visualization is handled automatically by an external library that intuitively determines the most appropriate format (e.g., pie charts, bar graphs). Users retain control over report formatting, ensuring the presentation aligns with their specific goals.

3.3 Requirements

This section discusses the goals that have guided the architectural and developmental choices for the proof-of-concept implementation of the proposed paradigm presented previously (Section 3.2).

3.3.1 Minimize Data Requirements

MBSE is often concerned with specialized domains and the models in question might be private. Thus, as indicated in Section 2.3, access to for example question-query pairs is not guaranteed. In the absence of domain specific labeled data, many approaches proposed in the literature are not applicable. Although a state-of-the-art approach might be performant on the domain it was trained for, generalization to other domains is generally weak, as assessed by [48], because of unseen question-query templates, unseen Uniform Resource Identifiers (URIs), etc.

Hence, the system must be designed to work for domains for which there is potentially limited or no data available. This scenario is denoted as data scarcity or limited resource conditions.

3.3.2 Maintain Simplicity

As touched upon in the introduction, the target users are non-technical stakeholders with no specifically assumed level of technical expertise. Hence, the general paradigm should be easily usable and the approach should require not require user intervention of a highly technical level, e.g., direct editing of SPARQL queries.

3.3.3 Facilitate Comprehensive Report Creation

To support the creation of comprehensive reports, a wide range of downstream processing capabilities should be enabled, including the generation of explanations, tables, figures, and more. This goal is best achieved through a QA system incorporating a Semantic Parsing step, where queries are explicitly generated. Contrast this with a naive QA approach where, given a user question, Information Retrieval (IR) might be employed to retrieve relevant information, which is then appended to a prompt alongside the original question and passed to a Pre-trained Large Language Model (PLLM), such as ChatGPT, for an answer. This approach raises the critical question of how essential elements of model reporting — such as visualizations, e.g., the mass roll-up of the Orbiter Mission (see Figure 3.3) — can be achieved. Achieving this would necessitate the PLLM to generate results in a consistent, predictable format to enable subsequent reduction and rendering. However, the generative nature of LLMs introduces inherent unpredictability, which can pose significant challenges. Additionally, the output token limits of LLMs may act as a restricting factor. For example, the components query for the Orbiter Mission (see Listing 3.3) retrieves fewer than twenty components, as reflected in the mass roll-up visualization. However, if the mission had significantly more components, relying on a LLM to generate all the results would be inefficient in terms of time, cost, and computational resources — particularly when a straightforward query could achieve the same outcome. In conclusion, while explicit query construction might be unnecessary when a QA system is designed to provide natural language responses alone, it becomes indispensable for effective model reporting.

3.3.4 Enable Human Oversight

An AI-assisted approach is chosen over a fully automated solution for generating reports. First, the output of generative models, such as LMs used for text-to-SPARQL generation, cannot be guaranteed to adhere to certain requirements [39]. Second, full automation often comes at the expense of user control. In the context of reporting, this could result in the inability to customize the structure of the generated report. For instance, a stakeholder might prefer a strict format in which all tabular results are preceded by a concise natural language explanation, as shown in Figure 3.5. When such preferences are not accommodated, the stakeholder must extensively revise the report's structure, leading to inefficiencies. Fully automated systems typically introduce these types of challenges by prioritizing automation over user control [49]. Given these downsides, a human-in-the-loop approach that balances automation and human intervention is more suited [50].

3.4 Validation

This chapter concludes by demonstrating how the proof-of-concept implementation satisfies the stated requirements, setting the stage for the detailed methodology outlined in the next chapter.

First, the requirement for minimal data needs is addressed through the use of domain adaptation and synthetic data generation techniques, ensuring the system operates effectively with limited resources. Second, the implementation maintains accessibility by leveraging the familiar Jupyter Notebook environment requiring no specialized skills beyond familiarity with its interface. Third, the implementation incorporates three key post-processing features: natural language explanation, tabulation, and visualization. These are facilitated by the pipeline's explicit query construction. Finally, stakeholders are empowered to structure and format reports freely using the built-in features of Jupyter Notebook, enhancing customization.

4

Methodology

This chapter details the methodology adopted to develop and validate the proof-of-concept implementation for the proposed model reporting paradigm. It begins with an in-depth technical discussion of the approach taken, followed by detailing how it was implemented, and concludes with an explanation of the evaluation strategy used to assess the approach.

Although the model reporting paradigm is proposed for all of MBSE, the implementation is scoped to the openCAESAR framework. Hence, certain specifics of the approach are tailored to its nature, see Section 4.1.1.2.

4.1 Approach

The following section will first detail the architecture, afterwards the choices that determined its design are justified. Lastly, the contributions of the approach are put forward.

4.1.1 System Architecture

Before delving into the specifics of the architecture, a high-level overview of the pipeline and its constituents is given. Each component is then explored in detail, with a running example introduced at the start of each section when applicable, to illustrate its functionality in context. The running example is rooted in a specific domain, namely Open Research Knowledge Graph (ORKG), that treats scientific papers and authors. Further details about ORKG are provided in Section 5.1.1, since it will also be used for the experimental evaluation, see Chapter 5. The question starting the running example is given by Listing 4.1.

```
Question:
Provide a list of papers that have utilized the Depth DDPPO model and include the
 ↪  links to their code?
```

Listing 4.1: Running example: question.

4.1.1.1 High Level Overview

Circling back to the model reporting paradigm (Figure 3.5), three aspects are essential: the notebook environment, the agent that interacts with the system, and finally the bridge between the two. This section details how all three were implemented in practice as a Jupyter Notebook, a KGQA pipeline and a Python package.

Notebook Environment Jupyter was selected as the notebook environment, given its ubiquity, and the existence of similar AI extensions to the platform.[1] Besides serving as the editing environment, it also renders the returned results from the pipeline and is the source of the final report.

Pipeline The KGQA pipeline was introduced in Section 3.2.3 as part of the proof-of-concept implementation, see Figure 3.4. A more detailed view of the text-to-SPARQL component of the pipeline is illustrated in Figure 4.1, which includes its subcomponents: a domain expert, a translator, and a linker.

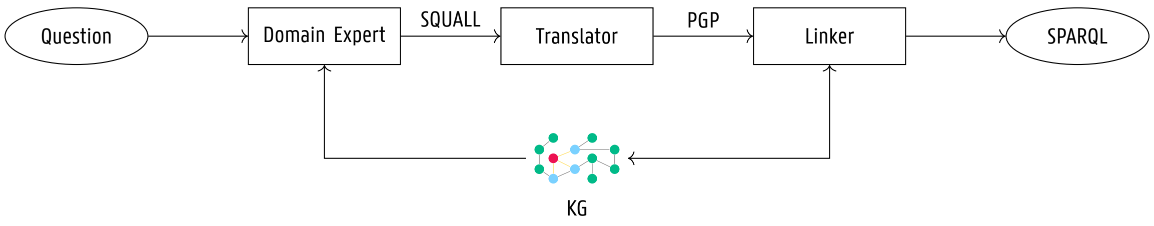


Figure 4.1: Text-to-SPARQL component.

Common approaches for the text-to-SPARQL task frequently train an LLM on a sizeable high-quality dataset consisting of question-SPARQL pairs for the target domain. Due to the specialized domains in MBSE and associated data scarcity, this is not realistically feasible. Hence, the text-to-SPARQL task is deconstructed in three of the five stated components: the domain expert, translator and linker components; combined they are responsible for the task. The domain expert is tailored to a specialized domain and can generate a query for an input question. Instead of SPARQL, the target language is SQUALL, which was introduced in Section 2.1.2. For further justifications behind choosing this CNL, refer to Section 4.1.2.3. However, translation from SQUALL to SPARQL is needed in order to query the system. Furthermore, the domain expert's output does not contain explicit identifiers from the target domain, relying on textual placeholders instead. For example, where a normal query targeted at Wikidata might contain an identifier such as `wd:Q30642`, output of the domain expert would contain the following placeholder: `natural language processing`. The linker component stands in for associating the placeholders to identifiers. To recap, for any given question the domain expert generates equivalent SQUALL containing placeholders, which is mapped to SPARQL by the translator and finally associated to the target KG by the linker, see Figure 4.1. Hereafter, assuming semantic correctness, a query can be executed, and its results handled by the ultimate component. Practically, in case a user chose for visualization the execution of the query is also handled by the visualization engine. For

[1]Jupyter AI is an example of such an AI extension, available at https://github.com/jupyterlab/jupyter-ai.

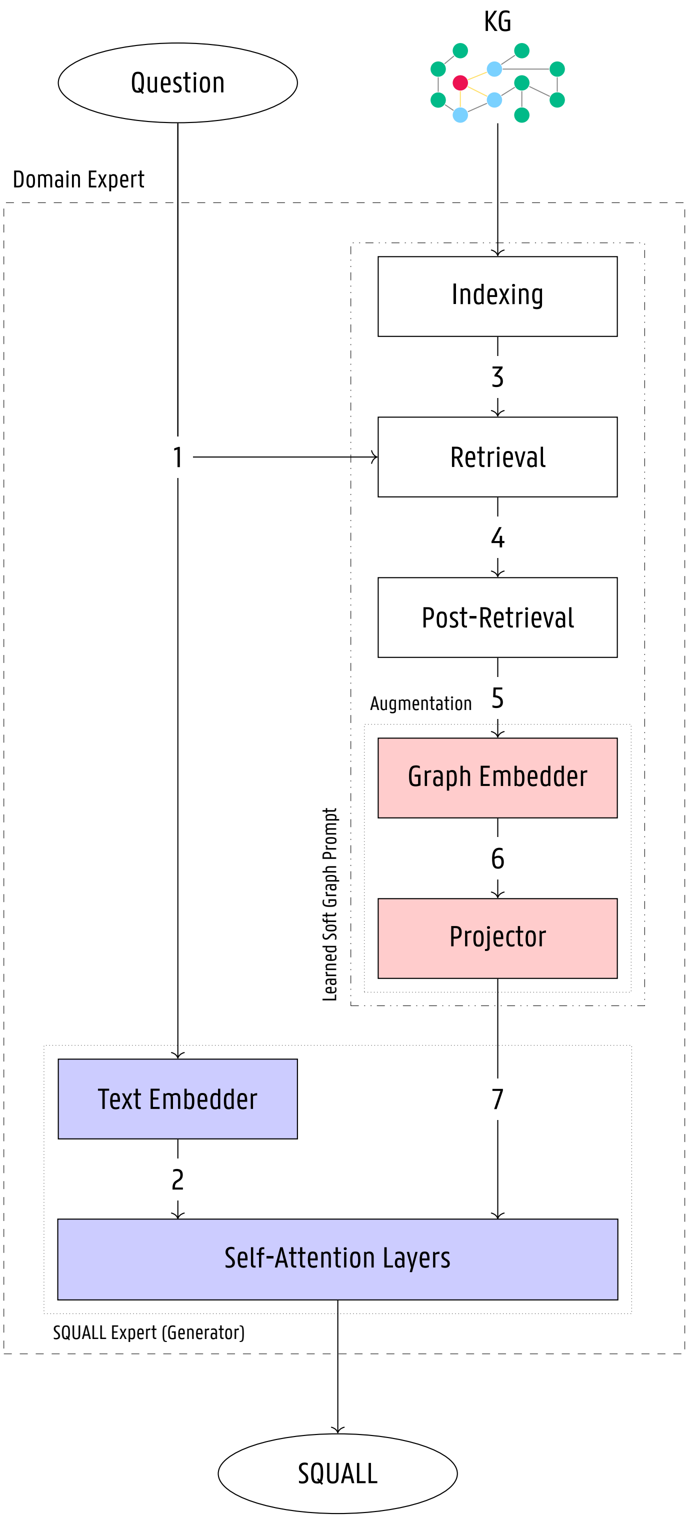
tabular results only execution is necessary. A natural language answer can be generated by prompting a PLLM to answer the input question given the query results. To that end, the prompt is constructed from the input question, task description and limited query results based on context window size.

The order of the translator and linker as well as the choice of using placeholders instead of identifiers are explained in Section 4.1.2.4 and Section 4.1.2.2, respectively.

Domain Expert The domain expert's role involves generating SQUALL queries that are specifically "tailored" to the target domain. While many syntactically correct queries can be equivalent in some sense for a given query language and input question, only those appropriately tailored to the target domain would be relevant, matching the structure of the target KG, i.e., those queries that are semantically correct. This is essential in order to have executable queries. The core issue of the data-scarce scenario then becomes clear: it is not possible to reuse a model-centric and data-driven approach tailored to an original domain for a distinct new one without any alterations, generated queries would not be semantically correct. The idea presented here to overcome this problem is a type of transfer learning where knowledge gained in the original domain is transferred to the target domain. Achieving this tailoring to the target domain and overcoming the data-scarce scenario. An LLM is trained in the original domain on text-to-SQUALL task, from this point onward it is referred to as the "SQUALL expert". The knowledge it gained is leveraged by incorporating it in a RAG model as the generator component. This final model is named the domain expert, since it has access to non-parametric domain information. The training regime can be described as sequential training [51], where the generator is first fine-tuned on a specific task, then frozen for further training of the RAG model. Freezing the SQUALL expert means preventing catastrophic forgetting of its knowledge. The retriever approach is straightforward, and post-retrieval processing [32] is static, neither require training. The augmentation process does retrieval only once, supplying structured data to the generator component (SQUALL expert) [32]. Currently, only the augmentation process stands in for adaption to the target domain during the second phase of sequential training. In conclusion, the domain expert is a RAG architecture consisting of a generator separately fine-tuned during an initial training phase — dubbed the "SQUALL expert" — a static retriever with post-retrieval processing, and an augmentation method optimized during the second round of sequential training, see Figure 4.2.

Retrieval Static retrieval consist of finding the most appropriate vertices and edges of the KG for a given question. This is achieved through textual similarity between the question and `rdfs:label` of all vertices and edges.

Post-Retrieval Processing The retrieved information is used to find a question-relevant subgraph of the KG (structured data). First, the sets of appropriate vertices and edges are ranked based on the similarity score. Then they are assigned a certain value, called prize, based on their rank. Using the Prize-Collecting Steiner Tree (PCST) algorithm, a connected subgraph of the KG can be found that both maximizes the total prize, while minimizing the total cost, which is related to its size.



1	Question
2	Text tokens
3	Vertex and edge embeddings
4	Top-k vertices and edges
5	Minimum connected subgraph
6	Graph embedding
7	Aligned graph token
●	First phase of sequential training: The SQUALL expert is independently fine-tuned on the source domain.
●	Second phase of sequential training: The domain expert is trained on the target domain, only the augmentation process is tuned, the SQUALL expert remains frozen, and other components are static.

Figure 4.2: Domain Expert.

Augmentation The augmentation process first embeds the graph resulting from post-retrieval processing into an embedding followed by "aligning" it with the SQUALL expert, i.e., with the LLM's text embedding space.

SQUALL Expert The augmentation process employed to support the SQUALL expert is atypical in its approach, as it integrates with the intermediate layers of the model [51]. Specifically, retrieved data is utilized at the self-attention layers of the LLM, rather than at the input or output layers (i.e., input-layer or output-layer integration, respectively) [32]. As a concrete example, a graph resulting from post-retrieval processing can contain over 300 000 tokens. Furthermore, since embeddings instead of text are used, other techniques can be leveraged. If input-layer integration was used, the question-relevant graph would need to be linearized, e.g., using a tabular overview of its vertices and edges. The idea to use intermediate-layer integration for graph learning tasks was recently put forward [52]. This leads to an approach dubbed soft graph prompting, and has the advantage of having the ability to pass the retrieved graph through a graph embedder and directly using that advantageous representation instead of a linearized representation, which results in loss of the graph's topological information. The hypothesis that soft graph prompting can aid LLM generation for graph learning tasks was further investigated by [43] on a QA task. A notable distinction between the prior works [43], [52] and the approach presented here lies in the following: the SQUALL expert undergoing soft graph prompting has already been fine-tuned for a specific task. It is important to note that the level of access required for implementing soft graph prompting is unavailable for most LLMs accessed via inference APIs [51]. These models operate largely as black boxes: users provide input and receive output, but the internal processes remain inaccessible. Prominent examples, such as ChatGPT, Gemini, and Claude, do not support this type of prompting.

4.1.1.2 Particulars of the Proof-of-Concept

First, a short note on the particulars of the proof-of-concept. As briefly touched upon in Section 3.1, the proof-of-concept implementation of the model reporting paradigm is tailored to the openCAESAR framework, a representative MBSE approach. Like other MBSE approaches, the openCAESAR framework uses models to represent systems. Distinct to openCAESAR, these models are defined using the Ontological Modeling Language, a formal language inspired by Web Ontology Language 2 (OWL2) and Semantic Web Rule Language (SWRL).

Looking again at Kepler16b, an example of how a relation between its missions and their objectives could be modeled in OML is given by Listing 4.2.

Like OWL2, OML is used for authoring ontologies, but it is specifically tailored for SE. An ontology is essentially a formal representation of knowledge about some domain, in this context a system. It is through OML that openCAESAR introduces the ontological approach to MBSE. This has many benefits for model authoring, e.g., enabling the use of a logical reasoner that can check whether there are any logical contradictions present in the OML model [3].

Answering questions or retrieving information about the model necessitates a different method: querying. OML

```
@rdfs:label "Mission"
concept Mission :> base:IdentifiedElement

@rdfs:label "Objective"
concept Objective :> base:IdentifiedElement

@rdfs:label "Pursues"
@dc:^description "A Mission pursues >=0 Objectives."
relation entity Pursues [
         from Mission
         to Objective
         forward pursues
         reverse isPursuedBy
         asymmetric
         irreflexive
]
```

Listing 4.2: Kepler16b vocabulary model excerpt [3].

constructs (such as `relation`) can be translated into patterns represented within subsets of OWL2 and SWRL. Consequently, OML ontologies can effectively be converted into OWL ontologies.

Various tools can be used to query OWL ontologies in SPARQL such as the OWL ontology editor Protégé. Alternatively, a so-called SPARQL endpoint can be set up using platforms such as Apache Jena, which only take the ontology and then allow users to query it using SPARQL.

Regardless of the approach chosen, the ontology is typically stored in the RDF format (discussed further below). Internally, what is queried against is not a set of textual documents wherein the ontology is defined, but rather a graph constructed from the RDF format, i.e., the RDF graph.

Thus, after model authoring the openCAESAR framework makes the model queryable by mapping it to an OWL ontology and subsequently setting up a SPARQL endpoint. Note that it is necessary to repeat the process if the model is changed in order for the endpoint to be up-to-date.

Although in this proof-of-concept a model is ultimately made queryable through an RDF graph, this is just one type of graph data format and alternatives exist. Neo4j, for example, uses property graphs instead of RDF graphs. Generally, this concept of using a graph of data to represent knowledge is called a Knowledge Graph (KG) [53].

Ontologies and Statements in Knowledge Bases The statements in any KB/KG can be categorized as either belonging to its Terminological Box (TBox) or Assertional Box (ABox). The TBox contains statements describing the domain of interest, while the ABox contains ground statements, i.e., facts associated with, and compliant to, the TBox. For example, in the Kepler16b context, a TBox statement might be, `"Every objective is pursued by` `a mission,"` while a corresponding ABox statement could be, `"The Lander Mission is a mission."` If, for example, the TBox statements are related with object-oriented classes, the ABox statements would be instances of those classes. Given the above highlighted relation between an ontology and a KB, this distinction can also be made in the ontology, both OWL and OML ontologies do this. However, the distinction is not set in stone, different ontologies categorize differently. In openCAESAR, the TBox and ABox are referred to as the vocabulary and description models, and they are similar to OWL's TBox and ABox, respectively [3]. The Kepler16b system design example of Listing 3.1 showed a part of the description model, see Listing 4.2 for its related vocabulary model.

4.1.1.3 Formalization

The formalization presented in this section establishes the foundation for the subsequent discussions. As outlined in the introduction, the proposed approach is specifically tailored to models that can be represented as KGs, implemented as RDF graphs. The graph formalisms introduced here are particularly relevant to Section 4.1.1.5, which describes an algorithm designed to identify subgraphs relevant to specific questions. The aim is to use standardized terminology, fostering consistency and clarity across the methodology. Established concepts from the literature are leveraged to ensure alignment with existing conventions and to improve comprehensibility. In particular, the definitions provided by [54] and the notations introduced by [55] are adopted.

First, the RDF format is first introduced in order to gain insight on how a model in openCAESAR is actually represented when it is queried, which is crucial for retrieval and post-retrieval, see Section 4.1.1.4 and Section 4.1.1.5, respectively. Basic concepts from the RDF framework are introduced, for example, URIs, Blank Nodes (BNodes) and literals, which are important for linking (Section 4.1.1.9).

Beginning then with the RDF framework which represents data as semantic triples, such as *"model reporting is flexible."* Each triple adheres to the structure *"subject predicate object"*, where:

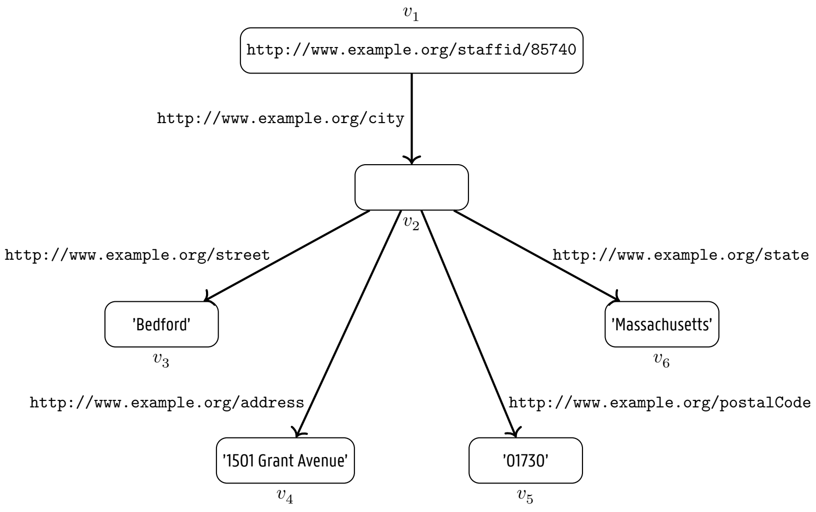
- **Subject** identifies the resource being described (*model reporting*).
- **Predicate** specifies the property or relationship (*is*).
- **Object** provides the value or linked resource (*flexible*).

These triples collectively form a graph, as illustrated in Figure 4.3.[2]

The figure highlights three fundamental RDF terms:

- **URI**: A globally unique identifier representing a resource.
- **BNode**: An unnamed resource or existential variable.

[2]Example from a W3C working draft about RDF available at https://www.w3.org/2001/sw/RDFCore/TR/WD-rdf-concepts-20030117.



v_1
http://www.example.org/staffid/85740
http://www.example.org/city
v_2
http://www.example.org/street
http://www.example.org/state
'Bedford'
v_3
'Massachusetts'
v_6
http://www.example.org/address
http://www.example.org/postalCode
'1501 Grant Avenue'
v_4
'01730'
v_5

Figure 4.3: Example RDF graph, representing a staff member and relationships to its associated properties such as city, address, street, postal code, and state.

- **Literal**: A value such as a string, number, or date.

In RDF, resources can act as classes, which define categories or types; entities, which are specific instances of those categories; or properties, which describe relationships or attributes connecting entities or values.

Formally these notions can be defined as follows.

Definition 1. An *RDF dataset* is a set $\mathscr{D}$ of triples $t = (\text{subject}, \text{property}, \text{object}) \in (I \cup B) \times I \times (I \cup B \cup L)$.[3] The pairwise disjoint infinite sets I, B and L indicate URIs, Blank Nodes (BNodes) (or vertices) and literals.

Before querying an OML ontology is mapped to an OWL ontology and stored in an RDF dataset.

Definition 2. An *RDF graph* is a graph $G = (V, L_V, E, L_E)$ where

- $V = V_c \cup V_e \cup V_b \cup V_l$ is a set of vertices consisting of all subjects and objects in the RDF data, with V_c, V_e, V_b and V_l being the set of class, entity, blank and literal vertices, respectively.
- L_V is a set of vertex labels, for a vertex $v \in V_c \cup V_e$, $v \in V_b$ and $v \in V_l$ their labels correspond to URIs, `NULL` values and literals, respectively.
- $E \subseteq V \times V$ is a set of directed edges, and
- L_E is a set of edge labels, for an edge $e \in E$, its label is its property.

Applying to the example graph (Figure 4.3) with vertices $V = V_c \cup V_e \cup V_b \cup V_l$:

[3]Triples are sequences of three elements.

$V_c = \emptyset, \quad V_e = \{ v_1 \}, \quad V_b = \{ v_2 \}$, and $\quad V_l = \{ v_3, v_4, v_5, v_6 \}$.

Querying is done against the RDF graph representation of an ontology, though any user is naturally shielded from any graph-related concepts and technical details, as these are abstracted away (see Section 4.1.1.2).

However, a foundational understanding of elementary graph theory is essential to clearly explain the proposed approach in the sections that follow. Therefore, these concepts are introduced upfront. While the following formalizations might often be avoided, they are considered to serve an important role in aligning with a theoretical framework that promotes academic rigor and, crucially, ensures consistency. For instance, the domain expert could be introduced solely in terms of generated queries, but this approach would not seamlessly align with the other components. See, for example, the discussion of the post-retrieval component in Section 4.1.1.5, it outputs graphs. However, since queries are inherently related to graphs, as will be elaborated upon, structuring the domain expert's discussion around graphs ensures a cohesive and unified narrative.

Definition 3. A *subgraph* H of a graph G is defined as a graph such that:

$$H = (V_H, E_H), \quad \text{where } V_H \subseteq V \text{ and } E_H \subseteq E.$$

In this notation, V_H and E_H represent the sets of vertices and edges of the subgraph H, respectively.

Definition 4. Two vertices v and w in a graph G are said to be *adjacent* if there is an edge connecting them:

$$v \text{ and } w \text{ are adjacent} \iff (v, w) \in E \text{ or } (w, v) \in E.$$

Definition 5. A *path* p in a graph G is a finite sequence of vertices:

$$\begin{aligned} & p = \{ v_i \}_{i=0}^{n} \in E^*, \quad n < \infty \\ \text{s.t.} \quad & \forall i, j \in \{ 0, 1, \dots, n \} : i \neq j \land v_i \neq v_j \implies v_i \text{ and } v_j \text{ are adjacent,} \end{aligned}$$

where * is the Kleene star operation.[4]

Definition 6. A graph G is said to be *connected* when there exists a path in between every pair of its vertices joining the pair:

$$G \text{ is connected} \iff \forall v, w \in V,\ v \neq w,\ \exists\, p \in E^* : v, w \in p$$

Connected Subgraphs Without delving further into graph theory, a shorthand notation can be introduced to represent all possible connected subgraphs of a graph G using the concept of the hyperspace graph of connected subgraphs $\mathscr{C}(G)$, which is defined for any connected graph [56], [57]. Relevant here is that any connected subgraph H of G is an element of the vertex set of $\mathscr{C}(G)$ by definition:

$$V_{\mathscr{C}(G)} = \{H; H \in \mathscr{C} \land H \subseteq G\},$$

where $\mathscr{C}$ is the set of all connected graphs.

[4]The Kleene star operation * constructs the free monoid $E^* = \bigcup_{n=0}^{\infty} E^n$, where $E^0 = \{ \varepsilon \}$ and ε is the identity (or empty) element [55].

Vertex and Edge Similarity The similarity between a string and a vertex or edge of a graph G is defined using an arbitrary encoder LM λ:

$$\lambda : \Sigma^* \to X$$

where Σ^* is the set of all strings over some alphabet Σ, and X is an arbitrary set, e.g., $\mathbb{R}^n$. By a slight abuse of notation, vertices and edges can also be used as input to λ, with the implicit assumption that with each vertex or edge a string can be associated. In the case that G is a KG, that string is a vertex's or edge's metadata property `rdfs:label`, this attribute being the human-readable name of the resource. A set of graph embeddings is referred to as an index $\mathscr{I}$. If all embeddings result from one graph G, e.g., a KG, the graph's index is defined as follows:

$$\mathscr{I}_G = \{\, \lambda(x) \mid x \in V \cup E \,\}\,. \tag{4.1}$$

A function σ is used to assess the similarity between embeddings, denoted by $\sigma(x, y)$. For convenience, this notation is also extended to allow strings, vertices, and edges as inputs, with the understanding that the similarity function implicitly operates on their embeddings. Specifically, define:

$$\sigma(s, x) := \sigma(\lambda(s), \lambda(x)), \quad s \in \Sigma^*,\; x \in V \cup E, \tag{4.2}$$

The specific similarity function σ used in the experiments is detailed in Section 5.1.2.

Definition 7. A *Basic Graph Pattern (BGP)* is defined as graph $Q = (V_Q, E_Q) \in \mathscr{C}$ s.t.:

- $V_Q \subseteq I \cup L \cup V_{\text{var}}$ is a set of vertices, with I, L and V_{var} being the of URIs, literals and variables, respectively.
- $E_Q \subseteq V_Q \times V_Q$ is a set of edges
- An edge in E_Q either has an edge label in I, i.e., property, or is a variable.

An example of a Basic Graph Pattern (BGP) is illustrated by Figure 4.4.

```
{
  ?x <likes> ?y1 .
  ?x <likes> ?y2 .
  ?x <follows> ?y3 .
}
```

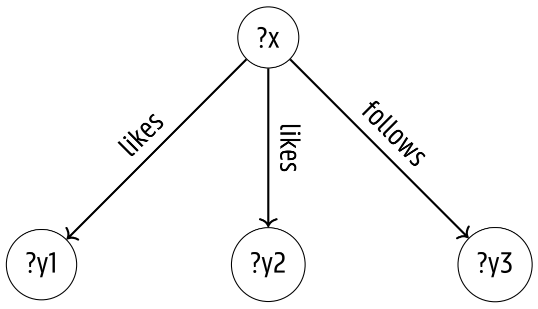


Figure 4.4: Example of Basic Graph Pattern with a star shape [58].

BGPs are the starting point of SPARQL queries which are used to inquire into RDF graphs. To formalize the entirety of the SPARQL language Complex Graph Patterns (CGPs) are needed, which are in essence BGPs with additional operations including projections and unions, e.g., the `FILTER` [55]. Hereafter, they are both mentioned as Graph Patterns (GPs).

Presented here is the Placeholder Graph Pattern (PGP), a graph similar to the Graph Pattern (GP). However, it uses string literals, referred to as placeholders, instead of URIs.

The BGP example shown in Figure 4.4 is in fact a Placeholder Graph Pattern (PGP), given that instead of URIs, strings are used for the edges. In contrast to the previous example of Figure 4.3.

Definition 8. A *Placeholder Graph Pattern (PGP)* is defined as a graph $\hat{Q} = (V_{\hat{Q}}, E_{\hat{Q}}) \in \mathscr{C}$ s.t.:

- $V_{\hat{Q}} \subseteq P \cup L \cup V_{var}$ is a set of vertices, with P, L and V_{var} being the set of placeholders, literals and variables, respectively.,
- $E_{\hat{Q}} \subseteq V_{\hat{Q}} \times V_{\hat{Q}}$ is a set of edges,
- An edge in $E_{\hat{Q}}$ either has an edge label in P, i.e., property, or is a variable,

where $P \subseteq \Sigma^*$.

Reviewing the pipeline, a PGP essentially serves as a precursor to a GP:

1. The domain expert generates a SQUALL expression containing placeholders, see Section 4.1.1.4–Section 4.1.1.7.
2. Translation maps this expression to a PGP, i.e., a SPARQL query where the placeholders have not yet been filled in with actual URIs, see Section 4.1.1.8.
3. Linking maps the PGP to a GP in SPARQL, i.e., an executable query, see Section 4.1.1.9.

See Figure 4.1 for the pipeline.[5]

Finally, given a set of graphs $\mathscr{D}$, the average number of vertices and edges is explicitly defined for clarity as follows:

$$\bar{V}_{\mathscr{D}} = \frac{1}{|\mathscr{D}|} \sum_{G \in \mathscr{D}} |V_G| \tag{4.3}$$

$$\bar{E}_{\mathscr{D}} = \frac{1}{|\mathscr{D}|} \sum_{G \in \mathscr{D}} |E_G|. \tag{4.4}$$

Section 4.1.1.4 uses Equation (4.3) and Equation (4.4) to compare datasets containing graphs.

[5]For clarity the schematics state SPARQL instead of PGP.

4.1.1.4 Retrieval

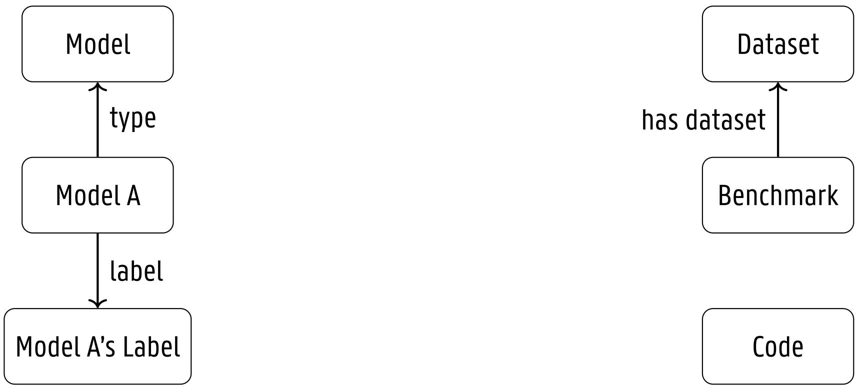
Given a question targeted at a KG = (V, E), retrieval consists of finding relevant information from its index $\mathscr{I}_{\text{KG}}$, see Figure 4.5.

```
Question:
Provide a list of papers that have utilized the Depth DDPPO model and include the
↪  links to their code?
```

```
Vertices:
1, model A
2, Model
3, model A's label
4, benchmark
5, dataset

Edges:
1, type, 2
1, type, 3
4, has dataset, 5
```

(a) Retrieved vertices and edges.



(b) Graph representation.

Figure 4.5: Running example: retrieval.

As stated, $\mathscr{I}_{\text{KG}}$ is created by encoding all its vertices and edges using an LM. The subset of k most similar vertices and edges for a given string s is defined as:

$$V_{s,k} = \text{arg top}\, k_{v \in V} \sigma(s, v), \tag{4.5}$$

$$E_{s,k} = \text{arg top}\, k_{e \in E}\, \sigma(s, e), \tag{4.6}$$

respectively, where arg top $k_{x \in X}\, f(x)$ essentially returns the k elements from the set X that maximize the function f stated as its argument. The similarity function σ was defined in Equation (4.2). For an input strings $s \in \Sigma^*$, the relevant information is defined as the subsets $V_{s,k} \subseteq V$and $E_{s,k} \subseteq E$.

Nature of the Index The process of indexing — i.e., creating a set of graph embeddings $\mathscr{I}$ (see Equation (4.1)) — and retrieval differs significantly from the implementation in [43]. In their approach, a specific graph H is associated with each question, resulting in multiple graph indices $\mathscr{I}_H$ rather than a unified index. This design allows for retrieval within smaller, question-specific indices, simplifying the search process compared to a comprehensive combined index. However, the use of question-specific graphs assumes preprocessing steps if the starting point is a KG. In contrast, this thesis employs a single KG as the foundation for all questions, requiring retrieval from one unified, large-scale index $\mathscr{I}_{KG}$. This approach leads to a significant disparity in graph sizes used as input for the PCST algorithm. Moreover, the KGs used in MBSE are generally much larger than the graphs analyzed in [43] (see Section 5.1.1.1). Of the three datasets utilized in the original work, only WebQSP includes an underlying KG.

The datasets used in [43] are summarized as follows:

- **ExplaGraphs** [59]: Explanatory graphs, supporting and countering arguments are generated given a belief and an argument.
- **SceneGraphs** [60]: Questions are generated from a scene graph structure.
- **WebQSP** [61]: Questions with subgraphs extracted from Freebase by extracting all triples that include entities related to the question within the maximum reasoning hops specified in WebQSP [62].

For a detailed comparison of the datasets and their associated KGs, refer to Table 4.1 and Table 4.2, respectively.

Dataset	# Questions	# Graphs	Average # Vertices	Average # Edges	Source
ExplaGraphs	2 766	2 766	5	4	[43]
SceneGraphs	10 000	10 000	19	68	[43]
WebQSP	4 737	4 737	1 371	4 252	[43], [62]
SciQA	2 565	1	180 184	6 888	[10], [63]

Table 4.1: Summary of datasets with key statistics including the number of questions, graphs, vertices, edges, and their sources.

Dataset	Knowledge Graph	# Triples	# Vertices	# Edges	Source
ExplaGraphs	N/A	N/A	N/A	N/A	[43]
SceneGraphs	N/A	N/A	N/A	N/A	[43]
WebQSP	Freebase	126M	88M	20K	[43], [62]
SciQA	ORKG	1 133 217	180 184	6 888	[10], [63]

Table 4.2: Overview of datasets and their associated (Knowledge Graphs), including triples, vertices, edges, and source references. ORKG version: February 14, 2023.

4.1.1.5 Post-Retrieval

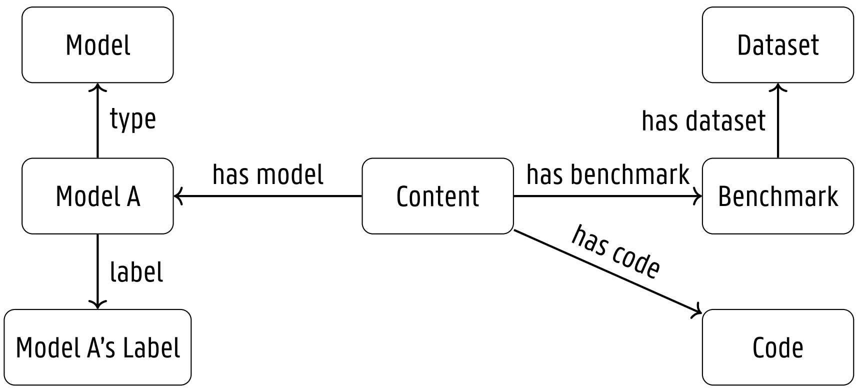
After retrieval, one set of vertices and one set of edges are obtained. Together they do not necessarily form a connected subgraph. Crucial vertices and edges required to construct a semantically correct query, including triples that may not be easily inferable from the question, could also be absent. These challenges are addressed through post-retrieval processing, which produces a connected subgraph ideally encompassing all necessary triples. Refer to Figure 4.6, which represents the post-processed version of Figure 4.5. It includes those additional vertices and edges that may not be as straightforward to infer as others, yet are essential for accurately parsing the question, e.g., `content`.

```
Question:
Provide a list of papers that have utilized the Depth DDPPO model and include the
↪  links to their code?
```

```
Vertices:
1, model A
2, Model
3, model A's label
4, benchmark
5, dataset
6, content
7, model
8, code

Edges:
1, type, 2
1, type, 3
4, has dataset, 5
6, has benchmark, 4
6, has model, 1
6, has code, 8
```

(a) Linearized connected graph.



(b) Connected graph.

Figure 4.6: Running example: post-retrieval.

Post-retrieval starts from the subsets of information and constructs a graph $H \in V_{\mathscr{C}(G)}$ that is relevant to the input string and constricted in size. This is done by posing the task as a Prize-Collecting Steiner Tree (PCST) optimization problem [43]. For a graph with vertices and edges assigned prizes and costs — essentially rewards and penalties — the PCST algorithm aims to identify a connected subgraph that maximizes the total prize while minimizing the total cost.

This approach aligns with the intuition that a question-relevant graph should include as many relevant vertices and edges as possible (maximizing prizes), but only up to the point where additional information becomes less relevant or detrimental (minimizing cost by limiting subgraph size). The algorithm thus seeks to strike a balance between poor-retrieval and over-retrieval, optimizing the relevance of the subgraph.

The prizes are defined as follows:

$$p_{s,k}(v) = \mathbb{1}_{v \in V_{s,k}}(k - r(v)), \qquad v \in V, \tag{4.7}$$

$$p_{s,k}(e) = \mathbb{1}_{e \in E_{s,k}}(k - r(e)), \qquad e \in E, \tag{4.8}$$

where $\mathbb{1}_{P(x)}(f(x))$ is the indicator function for some arbitrary predicate P and function f

$$\mathbb{1}_{P(x)}(f(x)) = \begin{cases} f(x) & P(x), \\ 0 & \neg P(x). \end{cases}$$

and $r_{\sigma,s}(v)$ and $r_{\sigma,s}(e)$ are the rank of a vertex and edge, respectively, according to the similarity function σ compared to a string s. The more similar a vertex or edge is to the string, the lower its rank and thus the higher its prize. The cost c of a graph $H = (V_H, E_H)$ is defined as follows

$$c_C(H) = |V_H| \cdot C \tag{4.9}$$

where C indicates the cost per edge, meant to give control over the subgraph size. The function of c_C is to restrict the size of the subgraph such that the algorithm scales better w.r.t. the size of the graph in question. Finally, the objective is to find $H = (V_H, E_H) \in V_{\mathscr{C}(G)}$, that maximizes:

$$\sum_{v \in V_H} p_{s,k}(v) + \sum_{e \in E_H} p_{s,k}(e) - c_C(H) \tag{4.10}$$

The practical computation of this objective is detailed in previous literature [43]. Note that neither retrieval nor post-retrieval processing require any training on labeled KG-specific data.

For an example of (the linearized version of) a constructed subgraph, see Figure 4.5a. The PCST algorithm has three parameters, Chapter 5 goes into depth on initializing the retrieval component along with the performance impact.

4.1.1.6 Augmentation

Augmentation, initially discussed in Section 2.1.3, aims to optimize generation by enhancing retrieval. While not always explicitly considered a distinct component, the augmentation process outlined here is notably more intricate. This contrasts sharply with the Naive RAG paradigm, which gained prominence with the widespread adoption of ChatGPT [32]. In that paradigm, the primary step involves synthesizing the retrieved context with the posed query into a single prompt, which is then provided as input to the LLM [32]. The complexity of the current approach, however, necessitates a more thorough explanation.

First, the augmentation process is achieved through soft graph prompting [52], which adapts previous work on soft prompt to graph learning tasks. Specifically, it involves integrating the augmentation process at an intermediate layer of a RAG's generator. This is essential, as otherwise the graph would need to be linearized, resulting in the loss of crucial structural information [43], as previously discussed (see Section 4.1.1.1).

The subgraph obtained of post-retrieval processing is embedded using a graph embedder which in turn is projected to the latent space of the generator's text embedder using an Multilayer Perceptron (MLP), resulting in an "aligned" graph token. Afterwards, concatenated graph token and text tokens are inputted to the generator's self-attention layers and pass through the remainder of the LLM, i.e., SQUALL expert as normal. Both the graph embedder and projector are jointly trained during the second phase of sequential training in order to be effective at representing and aligning the graph, respectively. The construction of the dataset and training method are detailed in Section 4.2.1 and Section 4.2.3, respectively.

Graph Embedding Graph embedding maps a graph H to embeddings using a model γ, such as a Graph Neural Network (GNN) or a Graph Attention Network (GAT), followed by a mean pooling operation ρ:

$$h_{\rho\circ\gamma} = \rho(\gamma(H)) \in \mathbb{R}^{d_g}, \quad H \in V_{\mathscr{C}(G)},$$

where $\mathbb{R}^{d_g}$ is the hidden dimension of the graph embedder.

Essentially, γ generates vector representations, which are then aggregated by ρ into a single embedding of size $\mathbb{R}^{d_g}$, effectively summarizing the graph's structural and feature information.

Projection Due to the fact that the text and graph embedding spaces are different, the introduction of another Artificial Neural Network (ANN) that functions as a learned mapping from graph to text embedding space improves performance, as empirically shown by [43]. The ANN responsible for this function is conceptualized as a projector π mapping input graph embeddings to the text embedding space, or in other words it aligns graph tokens with the SQUALL expert:

$$h_\pi = \pi(h) \in \mathbb{R}^{d_t}, \quad h \in \mathbb{R}^{d_g},$$

where $\mathbb{R}^{d_t}$ is the hidden dimension of the text embedder.

4.1.1.7 Generation

Given a question and its associated soft graph prompt, a SQUALL expression is generated, see Figure 4.7.

The text embedder λ of the SQUALL expert is applied to an input string:

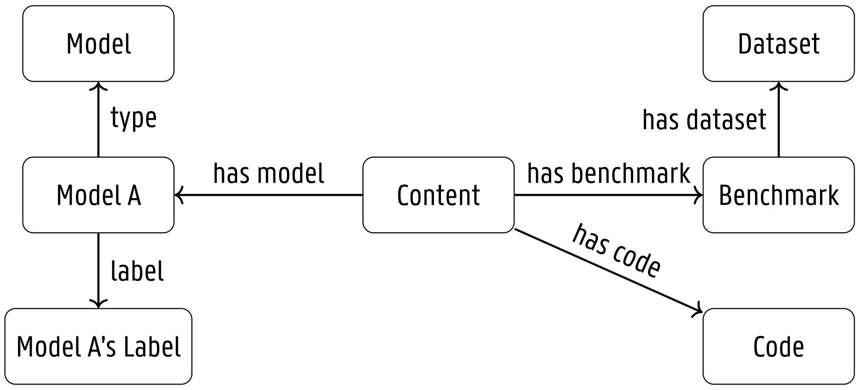
$$h_\lambda = \lambda(s) \in \mathbb{R}^{N\times d_t}, \quad s \in \Sigma^*,$$

where N is the amount of tokens. Afterwards, the concatenation of the string and graph embedding are inputted to the self-attention layers of the SQUALL expert and pass through the model as normal:

$$\hat{s} = \delta(h_\pi h_\lambda) \in \Sigma^*,$$

where h_π is the soft graph prompt.

```
Question:
Provide a list of papers that have utilized the Depth DDPPO model and include the
 ↪  links to their code?
```



(a) Connected graph from post-retrieval used to create the soft graph prompt.

```
SQUALL:
What is a <has_source_code> of a thing that has <has_model> a <Model> X and has a
 ↪  <has_benchmark> that has a <has_dataset> and X has a rdfs:label whose string
 ↪  matches 'Depth DDPPO'?
```

(b) Generated SQUALL expression.

Figure 4.7: Running example: generation.

4.1.1.8 SQUALL-to-SPARQL

The SQUALL language [13] has an accompanying tool, called the squall2sparql translator [64], which can map any syntactically correct SQUALL to SPARQL (S2S), and in a more limited capacity, SPARQL to SQUALL. Thus, assuming proper generation, a string S in SQUALL can be extracted from the model's output $\hat{s}$ and mapped to SPARQL:

$$f_{S2S}: \quad S \mapsto \hat{Q},$$

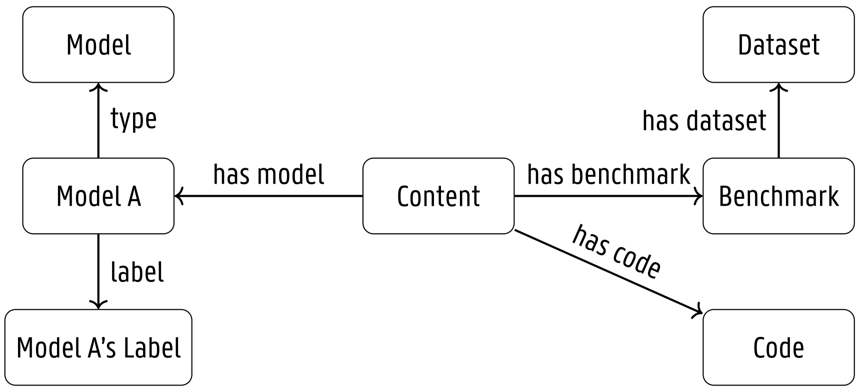
Note that S and $\hat{Q}$ contain placeholders and not URIs, the latter is thus a PGP, see Figure 4.8 for the example.

4.1.1.9 Linking

Given an RDF graph $G = (V, E)$, here, linking is conceptualized as the task of mapping a PGP to a GP (in SPARQL), i.e., associating vertices and edges to placeholders, see Listing 4.3.

Many possible approaches have been proposed that focus specifically on the linking task. The focus of this work

```
Question:
Provide a list of papers that have utilized the Depth DDPPO model and include the
↪  links to their code?
```



(a) Connected graph from post-retrieval used to create the soft graph prompt.

```
SQUALL:
What is a <has_source_code> of a thing that has <has_model> a <Model> X and has a
↪  <has_benchmark> that has a <has_dataset> and X has a rdfs:label whose string
↪  matches 'Depth DDPPO'?
```

(b) Generated SQUALL expression.

```
PGP:
SELECT DISTINCT ?code WHERE {
        ?model a <Model>;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark <has_dataset> ?dataset .
        ?cont <has_benchmark> ?benchmark .
        ?cont <has_model> ?model;
                <has_source_code> ?code .
}
```

(c) PGP.

Figure 4.8: Running example: mapping a SQUALL expression to a Placeholder Graph Pattern.

being on different parts of the text-to-SPARQL puzzle, a linker from the literature was adopted that agrees with the data scarcity initially supposed. It is a dynamic approach part of a proposed platform [25], that requires no training nor data, instead relying on Just-In-Time (JIT)-linking. Although significant changes were made to

```
Question:
Provide a list of papers that have utilized the Depth DDPPO model and include the
 ↪  links to their code?

PGP:
SELECT DISTINCT ?code WHERE {
        ?model a <Model>;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark <has_dataset> ?dataset .
        ?cont <has_benchmark> ?benchmark .
        ?cont <has_model> ?model;
                <has_source_code> ?code .
}

Placeholders
Model, rdfs:label, has dataset, has benchmark, has model, has source code

Potential URIs
orkgc:Model, rdfs:label, orkgp:HAS_DATASET, orkgp:HAS_BENCHMARK, orkgp:HAS_MODEL,
 ↪  orkgp:HAS_SOURCE_CODE

GP:
SELECT DISTINCT ?code WHERE {
        ?model a orkgc:Model;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark orkgp:HAS_DATASET ?dataset .
        ?cont orkgp:HAS_BENCHMARK ?benchmark .
        ?cont orkgp:HAS_MODEL ?model;
                orkgp:HAS_SOURCE_CODE ?code .
}
```

Listing 4.3: Running example: linking a Placeholder Graph Pattern to a Graph Pattern.

the framework itself, it proved to be a valuable starting point. Parts of the codebase dealing with server and KG interactions, including requests and querying, have for example been retained. Hereafter, a summary of the original JIT-linking procedure is given, while the remainder of this section deals with the formalization of the linking approach as it was ultimately implemented.

Assuming a list of triples, where each triple consists of the placeholders (s, p, o), linking [25] involves the following steps. First, exploratory queries are utilized to retrieve potential vertices and edges for each placeholder

— hereafter, they are referred to as probing queries. They query for vertices or edges whose `rdfs:label` contains the individual words of the generated placeholder in question. Then, the results are scored and ranked using a function that evaluates the semantic similarity between the `rdfs:label`s of the retrieved vertices/edges and their corresponding placeholders.

Owing to how previous components were implemented, here, the linker is the final step before result handling. In contrast, the original approach [25] requires some further steps. Adaptations to their linking approach are explained and justified in Section 4.1.2.5.

The implemented linking procedure can be formally stated as follows. Given a PGP ($\hat{Q}$), the goal is to map it to a set of possible GPs, achieved by interacting with the RDF graph $G = (V, E)$ through probing queries

$$\nu : \Sigma^* \to (V \cup E) \times \Sigma^*.$$

For any arbitrary placeholder, a probing query retrieves all potential vertices/edges with their respective `rdfs:label`:

$$I_p = \nu(p), \quad p \in P$$

The correspondence of URI and vertex or edge is implicitly assumed here. Doing this for all placeholders and scoring them:

$$X_p = \left\{ (x, y) \;\middle|\; \begin{array}{l} (x, s) \in I_p \\ \wedge\, y = \sigma(p, s), \end{array} \right\} \quad p \in P$$
$$\mathscr{X} = \{ X_p \}_{p \in P}$$

where σ is a similarity function whose implementation is detailed in Section 5.1.2.

Resulting in a set of scored vertex/edge-`rdfs:label` pairs for each placeholder.[6] Through $\mathscr{X}$ a set of mappings $\mathscr{F}$ is defined:[7]

$$\mathscr{F} = \left\{ f \;\middle|\; \begin{array}{l} f : P \to (V \cup E) \times [0,1] \\ \wedge\, f(p) = X_p^{(k_p)} \\ \wedge\, k_p \in \{1, \dots, |X_p|\} \\ \wedge\, p \in P \end{array} \right\}$$

The mappings are ranked using a scoring functional ϕ defined as follows:[8]

$$\phi : \quad \mathscr{F} \to \mathbb{R}^+, \quad f \mapsto \sum_{p \in P} f(p)_2,$$

[6]The sets of tuples are preordered: $(x_1, y_1) \lesssim (x_2, y_2) \iff y_1 \geq y_2$.

[7]The code implementation of $\mathscr{F}$ is a generator over the Cartesian product $\bigtimes_{X \in \mathscr{X}} X$.

[8]A preorder $\lesssim$ can be defined on $\mathscr{F}$ using ϕ, leading to the preordered set $(\mathscr{F}, \lesssim)$. Specifically, the preorder $\lesssim$ is defined by: $(f_1, f_2) \in \lesssim \iff \phi(f_1) \geq \phi(f_2)$ where $\lesssim \subseteq (\mathscr{F} \times \mathscr{F})$.

Finally, the set of possible GPs is built:

$$\mathscr{Q} = \left\{ Q \;\middle|\; f \in \mathscr{F} \wedge Q = (V_{Q,f}, E_{Q,f}) \right\}$$

where

$$\begin{aligned} V_{Q,f} &= \{ v \mid p \in P \wedge v = f(p)_1 \wedge v \in V \} \cup \left\{ v \;\middle|\; v \in V_{\hat{Q}} \wedge v \notin P \right\} \\ E_{Q,f} &= \{ e \mid p \in P \wedge e = f(p)_1 \wedge e \in E \} \cup \left\{ e \;\middle|\; e \in E_{\hat{Q}} \wedge e \notin P \right\} \end{aligned}$$

The implementation of ν is slightly different depending on the RDF engine used in practice, and whether the input is a vertex or edge. Furthermore, placeholder strings are split up into individual words: $s = \{ w_1 w_2 \dots \}$. An example of a vertex and edge probing query is illustrated in Listing 4.4. For practical reasons, the amount of results is capped.

```
QUERY: potential_vertices(n_max_vertices, w_1, w_2, ...)
SELECT DISTINCT ?subject ?label
WHERE {
        ?subject ?predicate ?label .
        FILTER(
                REGEX(?label, 'w_1', 'i')
                || REGEX(?label, 'w_2', 'i')
                || ...
        )
}
LIMIT n_max_vertices

QUERY: potential_edges(n_max_edges, w_1, w_2, ...)
SELECT DISTINCT ?predicate ?label
WHERE {
        ?subject ?predicate ?object .
        ?predicate rdfs:label ?label .
        FILTER(
                REGEX(?label, 'w_1', 'i')
                || REGEX(?label, 'w_2', 'i')
                || ...
        )
}
LIMIT n_max_edges
```

Listing 4.4: Vertex (above) and edge (below) probing query.

4.1.1.10 Result Handling

Three straightforward post-processing steps are implemented for this proof-of-concept: natural language explanation, tabulation and visualization.

Natural Language Explanation A natural language explanation is straightforwardly achieved through prompting a PLLM with the original stakeholder question and the results of the executed query along with the task to answer and explain if possible.

Tabulation Tabulation consists of a simple formatting function that takes as input the resulting data from an executed query and returns a table to be displayed in the notebook.

Visualization Due to scope limitations, an exemplary implementation from the literature [47] was integrated into the approach. The VizKG framework provides multiple visualization options for query results over KGs, along with a visualization recommendation system. This system was used to automatically generate appropriate visualizations without requiring user input.

For an example see Listing 4.5.

```
Question:
Provide a list of papers that have utilized the Depth DDPPO model and include the
↪  links to their code?

Natural Language Explanation:
The papers that utilized the Depth DDPPO model and include links to their code are as
↪  follows:
1. [Efficient Exploration with Depth DDPPO](https://github.com/repo1)
2. [Optimizing Depth Models](https://github.com/repo2)
```

Listing 4.5: Running example: result handling.

4.1.2 Architectural Choices

Having provided a detailed explanation of the architecture, this section focuses on the key design decisions made. It elaborates on the rationale behind each choice, highlighting how these decisions contribute to the effectiveness of the architecture in enabling model reporting for MBSE.

4.1.2.1 Fine-Tuning over Prompting

Two primary categories of models emerge when considering a neural semantic parser: fine-tuned LLMs and PLLMs with a suitable prompting strategy. The choice between the two is mainly informed by syntactic and semantic correctness.

Syntactic correctness pertains to the formal structure of the output logical forms, while semantic correctness is concerned with the domain-specific understanding for which the parser is trained. Intuitively, a PLLM like ChatGPT might excel in syntactic correctness, assuming it has encountered the logical forms during its extensive pre-training. On the other hand, a smaller, fine-tuned LM could perform better in terms of semantic correctness, given its tailored adaptation to the domain.

Syntactic Correctness The original paper [12], which proposed using LMs as Controlled Natural Language (CNL) semantic parsers, used fine-tuning. As previously discussed, CNL data is typically rare in the pre-training datasets of LMs, suggesting that fine-tuning was likely necessary instead of relying solely on prompting a PLLM. Although this rationale is not explicitly stated, it is considered reasonable and is thus adopted.

Semantic Correctness Furthermore, more recent work [11], which established baselines for the benchmark used to evaluate this approach (see Section 5.7.3.2), demonstrated superior performance a fine-tuned LLM compared to prompted PLLMs. While the focus was on generating SPARQL rather than SQUALL — two formal languages with distinctly different syntaxes — the choice between fine-tuning and prompting remains relevant to semantic correctness. These baselines indicate that fine-tuning likely enhances semantic correctness compared to prompting.

Feasibility of RAG Additionally, the RAG paradigm outlined in Section 4.1.1 is generally infeasible with PLLMs (e.g., ChatGPT) because such models do not provide access to their intermediate layers, a requirement for implementing soft graph prompting. Consequently, fine-tuning serves a dual purpose: it teaches the LLM to generate SQUALL expressions effectively and incorporates non-parametric memory through graph-based representations.

4.1.2.2 Placeholders

The proposal to employ placeholders instead of identifiers in the generator output is based on three key thoughts: identifier hallucination, placeholder hallucination and domain adaptation. Each is explained followed by a demonstrative example.

Identifier Hallucinations By relying on placeholders, identifier hallucination can be avoided. If the SQUALL expert were fine-tuned to generate identifiers, it is reasonable to assume that its integration into the domain expert would often result in erroneous identifiers. The generation of faulty identifiers would likely be influenced by those encountered during the fine-tuning of the SQUALL expert. Clearly, this outcome is highly undesirable. This intuition is corroborated by recent results pertaining to a similar situation. A pre-trained LLM, that was fine-tuned to generate queries for a specialized domain, frequently hallucinated predicates akin to those found in popular

KGs such as Wikidata, likely due to the presence of those domains in its pre-training data [11]. An illustrative example is given by Listing 4.6, it shows a predicate that is nonexistent in the fine-tuning data (SciQA dataset for ORKG domain, see Section 5.1.1), but does exist in the pre-training data. Although unfreezing the generator component

```
SELECT ?quantity WHERE {
        ?crater rdf:type orkgc:Crater .
        ?crater rdfs:label 'Elorza crater' .
        ?crater orkgp:P31 ?iron_oxide_discoveries .
        ?iron_oxide_discoveries orkgp:P2067 ?quantity .
}
```

Listing 4.6: Representative example of identifier hallucination by GPT-3.5 [11]. P2067 is an existing Wikidata property.

during the second round of training likely reduce this problem, it would most definitely increase the data requirements, due to the higher number of parameters, which is challenging given the practical data limitations in MBSE. While other studies address this issue by augmenting datasets to achieve better KB coverage [65], such extensive data engineering was deemed out of scope for this work. Finally, reasoning from the similarity of the sequential training strategy with continual learning, the fear of catastrophic forgetting is introduced [15]: the general "knowledge" the SQUALL expert learned during the initial training phase would deteriorate. Assuming the generator component is frozen, harmonizing the SQUALL expert with the new domain is likely to remain problematic in terms of identifier hallucination, as a substantial portion of identifiers will continue to be Out-of-Vocabulary (OOV), i.e., unseen during the second round of training. This is the case even in high-quality datasets, for example, LC-QuAD 2.0 [30]. The incorporation of non-parametric memory via RAG is unlikely to fully resolve this issue. Two key challenges have been identified that diminish the quality of retrieval, thereby increasing the risk of hallucination [66]. The subgraph produced by the retrieval (Section 4.1.1.4) and post-retrieval (Section 4.1.1.5) components may:

- Fail to include all necessary vertices and edges.
- Contain multiple distinct vertices or edges with identical `rdfs:label`, leading to the potential generation of incorrect URIs for vertices or edges.

These challenges are common in IR and are often attributable to noise in retrieval results. The former represents an issue of poor retrieval (low recall), while the latter, referred to as over-retrieval, can confuse the model [66]. Both issues are inherent limitations of the RAG paradigm. Moreover, it is unclear how these methods would perform with a heterogeneous dataset, composed of multiple question-answering datasets targeting different KGs. At face value, using a heterogeneous dataset to train the proposed SQUALL expert does not pose issues, given the use of placeholders.

Placeholder Hallucination Previous work shows how LLM hallucination of plausible relation placeholders can be leveraged if appropriate post-hoc adjustment is used, in the context of SQUALL generation [12]. In this work, the placeholder hallucination is extended to entities as well, and post-processing is performed by the linker. It is assumed that more plausible hallucinations — i.e., placeholders that are more similar to the expected labels — would improve linking too, since placeholders are more easily associated.

Domain Adaptation A key novel insight of this work is that the use of placeholders can effectively facilitate domain adaptation, further elaborated upon in Section 4.1.3. The SQUALL expert's task is to generate queries with likely placeholders, a role that remains consistent across different domains, with only the domain itself changing when used as generator in the domain expert. Through retrieval augmentation, these placeholders are made more plausible within the context of the target domain. By ensuring that the SQUALL expert has extensive knowledge of SQUALL, it can be reused in another domain, transferring its capabilities effectively. This approach allows the SQUALL expert to be applied to any target domain, ensuring that the significant efforts invested in data collection and training to enhance its SQUALL and SPARQL coverage are not wasted.

Example The previous points are illustrated by revisiting an example. The SQUALL expert trained on Wikidata learns that for a question relating to the NLP domain, the `natural language processing` placeholder is expected, which corresponds to the `wd:Q30642` URI in the Wikidata KG. Now the jump must be made to another domain, for example ORKG, a KG including scientific papers. A stakeholder might be interested in finding papers with the `natural language processing` keyword, this corresponds to the `orkgr:R51020` URI. Thankfully, the SQUALL expert has no knowledge of identifiers, instead it generates a likely placeholder — ideally `natural language processing` — where its generation is augmented by extra context from the retriever. To illustrate the transfer learning capabilities of the proposed approach, consider the queries presented in Listing 4.7.

4.1.2.3 SQUALL over SPARQL

The decision to use SQUALL rather than SPARQL for query generation in this work is rooted in one critical advantage: SQUALL is a CNL. Its similarity to natural language reduces the complexity of generating logical forms lowering training data requirements [12], as previously mentioned (Section 2.1.2). Moreover, SQUALL supports all SPARQL constructs, including many from SPARQL 1.1, and allows for unambiguous mapping to SPARQL. These properties make SQUALL a practical choice for tasks such as query generation and KGQA, particularly in the context of limited data availability.

4.1.2.4 Order Translator and Linker

Although the order of translator and linker could be switched, it is more practical to convert to SPARQL first, because existing parsing libraries for the language prevent further unnecessarily complicating linking. For example,

```
SELECT DISTINCT ?obj WHERE {
        <Delta_Air_Lines> <mouthpiece> ?obj .
        ?obj <instance_of> <periodical_literature> .
}
---
SELECT DISTINCT ?code WHERE {
        ?model a <Model>;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark <has_dataset> ?dataset .
        ?cont <has_benchmark> ?benchmark .
        ?cont <has_model> ?model;
                <has_source_code> ?code .
}
```

Listing 4.7: Example query from the source domain (above) and the target domain (below).

SPARQL parsing allows the disambiguation of subjects and objects on the one hand from predicates on the other. No such parsing libraries for SQUALL were found to exist at the time.

4.1.2.5 Adaptation of the Linker

The linker as originally implemented [25] requires some additional steps before SPARQL queries are obtained. Their generative model has a somewhat different target than the presented domain expert, although placeholders are also used, a list of triples instead of SQUALL is generated. No other additional information is generated, more specifically, the structure of the query is not part of generation. They address this through heuristics, for example, an `ASK` query is assumed if a question starts with "is", which is then built after linking the generated triples.

The linker presented here is more streamlined since the need for query construction is entirely eliminated, along with its resulting limitations. Instead, the query structure emerges directly from the generation process, removing the necessity for heuristics. Consequently, the coverage of the set of all possible SPARQL queries depends on the models (i.e., the SQUALL and domain expert) and their training data ($\mathscr{D}_S$ and $\mathscr{D}_T$). It targets SQUALL, which encompasses the full scope of SPARQL 1.1 [13], as opposed to the more restricted subset achievable through manual query construction. Therefore, practical coverage is constrained by the used datasets (see Section 4.2.1). By leveraging an LLM for generating query structures, this approach has the potential to achieve broader coverage, limited by data collection and engineering rather than the a priori limits of heuristic methods.

4.1.2.6 Differing Retrieval and Linking Strategies

While retrieval and linking share similarities — both aim to relate text to a domain — their differences remain significant. Retrieval, as an IR task, focuses on finding relevant information from the domain given an input string, whereas linking maps an input string to a specific instance within the domain.

For instance, the outcome of retrieval (or post-retrieval processing) provides valuable information, i.e., vertices and edges of the KG, which augment the generation process leading to the PGP with its placeholders. A natural question arises: why is this retrieved information not reused during linking? The challenge lies in disambiguation — selecting the correct vertex from the retrieved data is far from trivial.

As outlined in the following section (Section 4.1.3), the primary technical focus of this work is to address the data scarcity issue. Moreover, both retrieval and linking have been extensively explored in the literature. For example, ReFinED [67] is a specialized entity linker. Consequently, further investigation into harmonizing the retrieval and linking components was deemed out-of-scope for this work.

4.1.3 Contributions

Now that the architecture of the implementation has been elaborated, the technical contributions of this work can be summarized appropriately. The non-technical contributions outlined in the introduction (Chapter 1) include the model reporting paradigm and Mo-Lab, a proof-of-concept implementation of this paradigm, which leverages the reusable domain expert model. Those related to the KGQA pipeline and text-to-SPARQL task now follow.

Domain adaptation is central to this work and its contributions. Here, the adaptation is from the source domain $\mathscr{D}_S$ of the SQUALL expert (see Section 4.2.1.1) to the target domain $\mathscr{D}_T$ of the domain expert (see Section 4.2.1.2 and Section 4.2.1.3 for the ground truth and synthetic case, respectively). The learning task is to generate question-equivalent SQUALL. Conditions for the target domain are further relaxed [22], currently $\mathscr{D}_T$ consists of a sequence of questions $\mathscr{S}$ and a limited labeled support set $\mathscr{D}_{\text{sup}}$ for the target domain.

While domain adaptation through RAG is not a novel concept [44], no prior work has been identified in the literature that applies this approach to the query generation task. In this regard, the combination of components introduced in the proposed architecture is, to the best of the author's knowledge, unique. Although some reliance on labeled data remains, this is deemed acceptable given the minimal amount required and the relative ease of acquisition compared to alternatives. For instance, engineering a complete question-query dataset for a specialized domain from scratch [10], demands significantly more effort.

The key technical contribution of this work is identified as domain adaptation, and represents a significant step forward in integrating contemporary NLP techniques into MBSE. It is leveraged to realize a KGQA pipeline in limited data scenarios. Essential insights were the usage of placeholders (Section 4.1.2.2) and the sequential training approach (Section 4.2.3).

4.2 Implementation Details

This section explains how the datasets used for the approach were created, how the retriever component (Section 4.1.1.4) was initialized and evaluated, and finally how fine-tuning and training was done.

4.2.1 Datasets

This section describes the creation process for the datasets that were employed. The first is used to fine-tune the LLM of the generator on the text-to-SQUALL task, the second for the training of the retrieval, and the final is a replication of the latter through synthetic data generation. Common steps include replacing URIs with placeholders, which results in a PGP given a GP, and SPARQL-to-SQUALL mapping. A dataset $\mathscr{D}$ is formalized as a set of samples: $\mathscr{D} = \{ d_i \mid i \in \mathcal{I} \}$, over some index set $\mathcal{I}$, where each sample is a sequence: $d = (d_n)_{n \in \mathbb{N}}$.

4.2.1.1 Source Domain Data

The source domain dataset $\mathscr{D}_S$ used for fine-tuning the SQUALL expert (see Section 4.2.3.2) includes natural language questions and SQUALL expressions.

It was constructed, starting from LC-QuAD 2.0 [68], which contains 30 000 natural language questions for both the Wikidata and DBpedia 2018, along with the corresponding queries. The dataset contains 27 question and 33 query templates [30]. LC-QuAD 2.0 itself was created by starting from these templates, selecting vertices and appropriate edges, and generating SPARQL. Afterwards, these queries are transformed to Normalized Natural Question Templates (NNQTs), situated between the original formal language and target natural language, and containing textual placeholders in place of the original vertices and edges surrounded by brackets. Through human verbalization they are turned into true natural language questions. To increase variety, a final human paraphrasing step is performed on the verbalized questions, with the explicit intention of combating over-fitting of downstream KGQA models on a particular question syntax [68].

The following steps are required to construct $\mathscr{D}_S$ from LC-QuAD 2.0. Consider a sample with this question:

```
What periodical literature does Delta Air Lines use as a mouthpiece?
```

1. From the sample's Normalized Natural Question Template (NNQT) the placeholders between curly brackets are extracted, see Listing 4.8.

```
NNQT:
What is the {periodical literature} for {mouthpiece} of {Delta Air Lines}?

Placeholders:
["periodical_literature", "mouthpiece", "Delta_Air_Lines"]
```

Listing 4.8: Source domain: extracting placeholders from a Normalized Natural Question Template.

2. Then, from the sample's SPARQL query the URIs are extracted, see Listing 4.9.

```
SPARQL:
SELECT DISTINCT ?obj WHERE {
        wd:Q188920 wdt:P2813 ?obj .
        ?obj wdt:P31 wd:Q1002697 .
}

URIs:
["wd:Q188920", "wdt:P2813", "wdt:P31", "wd:Q1002697"]
```

Listing 4.9: Source domain: extracting URIs from a SPARQL query.

3. Using the sample's query-template, i.e., `<S P ?O ; ?O instanceOf Type>`, URIs can be associated with placeholders. E.g., `Type` is the *object* of the second triple (`?O instanceOf Type`) corresponding to the `wd:Q1002697` identifier of the query. For every sample from LC-QuAD that follows the same template, this `Type` will correspond to the first placeholder of the Normalized Natural Question Template (NNQT). Since this is the case for all placeholders, a URI-to-placeholder mapping can be constructed, see Listing 4.10.

```
{
        "wd:Q188920": "Delta_Air_Lines",
        "wdt:P2813": "mouthpiece",
        "wdt:P31": "instance_of",
        "wd:Q1002697": "periodical_literature"
}
```

Listing 4.10: Source domain: URI to placeholder mapping.

4. Afterwards, the SPARQL query can be transformed to a PGP using the mapping, see Listing 4.11.

```
SELECT DISTINCT ?obj WHERE {
        <Delta_Air_Lines> <mouthpiece> ?obj .
        ?obj <instance_of> <periodical_literature> .
}
```

Listing 4.11: Source domain: Placeholder Graph Pattern mapped from SPARQL.

5. Finally, using the SQUALL-to-SPARQL translator, the PGP is converted to a SQUALL expression:

```
What is a <mouthpiece> of <Delta_Air_Lines> and has <instance_of>
 ↪ <periodical_literature>?
```

Limited data engineering was needed to correct some queries, while the implementation of the numerous template mappings used to create the PGPs was more involved. See Section 5.18 for an overview of the different templates in SciQa and their corresponding PGPs.

4.2.1.2 Ground Truth Target Domain Data

The ground truth target domain dataset ($\mathscr{D}_{\text{gt}}$) used for training the domain expert (see Section 4.2.3.3) includes natural language questions and SQUALL expressions.

SciQA dataset [10] served as the foundation for $\mathscr{D}_{\text{gt}}$, and is publicly available on Zenodo [63]. The dataset comprises 2 565 question-query pairs tailored to the Open Research Knowledge Graph, organized into eight distinct query templates. The creation of SciQA involves a two-phase workflow. In the initial phase, a core set of 100 questions is manually crafted and translated into corresponding SPARQL queries. This set forms the basis for generating the remaining 2 465 questions. The second phase involves creating question and SPARQL query templates with placeholders, which are populated using all possible entities from the ORKG. This structured approach allows for efficient query generation while ensuring variety and relevance. A key distinction between SciQA and LC-QuAD 2 lies in their question generation methods: while SciQA primarily includes questions generated by an LLM, LC-QuAD 2 relies on explicit human verbalization for its questions.

The process to create $\mathscr{D}_{\text{gt}}$ from SciQA is similar to the process of creating $\mathscr{D}_S$ from LC-QuAD, although an additional subgraph construction step is required, and the placeholders cannot be retrieved from a NNQT. Consider a sample with this question:

```
Provide a list of papers that have utilized the Depth DDPPO model and include the
 ↪ links to their code?
```

1. From the sample's SPARQL query the URIs are extracted, see Listing 4.12.

```
SPARQL:
SELECT DISTINCT ?code WHERE {
        ?model a orkgc:Model;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark orkgp:HAS_DATASET ?dataset .
        ?cont orkgp:HAS_BENCHMARK ?benchmark .
        ?cont orkgp:HAS_MODEL ?model;
                orkgp:HAS_SOURCE_CODE ?code .
}

URIs:
["orkgc:Model", "rdfs:label", "orkgp:HAS_DATASET", "orkgp:HAS_BENCHMARK",
 ↪  "orkgp:HAS_MODEL", "orkgp:HAS_SOURCE_CODE"]
```

Listing 4.12: Target domain: extracting URIs from a SPARQL query.

2. For each URI, an `rdfs:label` is queried from the KG such that a URI-to-placeholder mapping can be constructed, see Listing 4.13.

```
{
        "orkgc:Model": "Model",
        "rdfs:label": "rdfs:label",
        "orkgp:HAS_DATASET": "has_dataset",
        "orkgp:HAS_BENCHMARK": "has_benchmark",
        "orkgp:HAS_MODEL": "has_model",
        "orkgp:HAS_SOURCE_CODE": "has_source_code"
}
```

Listing 4.13: Target domain: URI to placeholder mapping.

3. Then, the SPARQL query can be transformed to a PGP using the mapping, see Listing 4.14.

```
SELECT DISTINCT ?code WHERE {
        ?model a <Model>;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark <has_dataset> ?dataset .
        ?cont <has_benchmark> ?benchmark .
        ?cont <has_model> ?model;
                <has_source_code> ?code .
}
```

Listing 4.14: Target domain: Placeholder Graph Pattern mapped from SPARQL.

4. Finally, using the SQUALL-to-SPARQL (S2S) translator, the PGP is converted to a SQUALL expression:

```
What is a <has_source_code> of a thing that has <has_model> a <Model> X and has a
 ↪  <has_benchmark> that has a <has_dataset> and X has a rdfs:label whose string
 ↪  matches 'Depth DDPPO'?
```

5. Construction of the question-relevant subgraph is done using the retrieval and post-retrieval components, as explained in Section 4.1.1.4 and Section 4.1.1.5, respectively.

For an overview of the different types of queries present in SciQA, Section 5.18 can be consulted; the running example specifically is listed under Listing 11.

Due to the complexity of many queries in the SciQA dataset and current limitations of the translator tool — the inverse mapping from SPARQL to SQUALL is not yet complete — many could not be converted to SQUALL automatically. Therefore, the equivalent SQUALL templates for these queries were created manually. To facilitate the process, some projection variables for certain templates, whose inclusion was deemed superfluous, were removed — e.g., a second projection variable that represented the `rdfs:label` of the first.

The dataset is formalized as:

$$\mathscr{D}_{\text{gt}} = \left\{ (s_i, H_i, Q_i, \hat{Q}_i, S_i) \;\middle|\; i \in \mathcal{I}, s_i \in \Sigma^*, H_i \in V_{\mathscr{C}(G)} \right\},$$

where $\mathcal{I}$ is a countable index set. Thus, each sample consists of a question, a constructed subgraph, a GP, a PGP, and a SQUALL expression.

4.2.1.3 Synthetic Target Domain Data

The synthetic dataset $\mathscr{D}_{\text{syn}}$ is constructed in the same way as $\mathscr{D}_{\text{gt}}$, except that its SQUALL queries are not deterministically obtained through the procedure laid out in Section 4.2.1.2. Instead, they are generated using a PLLM, which is prompted to generate equivalent SQUALL given a question. All else being equal, only the method to obtain synthetic SQUALL is elaborated further upon in this section. An example of the resulting prompt template is given by Listing 4.15.

```
Given a Question, generate equivalent SQUALL.
Follow the requirement to only return SQUALL.

Here are some examples:

### Demonstration
Question: Could you provide a list of models that have been tested on the iNaturalist
 ↪  2018 benchmark dataset?
SQUALL: A <Dataset> is a <has_dataset> of a thing X and has a rdfs:label whose string
 ↪  matches "iNaturalist 2018" and X is a <has_benchmark> of a thing Y and a thing is
 ↪  a <has_evaluation> of X and if defined, what is a <has_model> of Y and has a
 ↪  rdfs:label?

// Etc.

### Task
Question: Provide a list of papers that have utilized the Depth DDPPO model and
 ↪  include the links to their code?
SQUALL: // To be generated by the model.
```

Listing 4.15: Instantiation of the generation task used for synthetic data generation. One in-context demonstration is included.

The remainder of this section formalizes the approach in accordance with a recent survey on LLM-driven synthetic dataset generation [69]. While this formalization does not offer substantial additional practical utility, it serves an important role in aligning with a theoretical framework that promotes academic rigor and, crucially, ensures consistency.

The synthetic data generation task is formulated as follows. Seed samples from the ground truth dataset are provided as supporting information consisting of questions and their respective SQUALL mappings:

$$\mathscr{D}_{\text{sup}} = \left\{ (s_i, S_i) \;\middle|\; i \in \mathcal{I}_{\text{sup}} \subseteq \mathcal{I} \right\},$$

The generation task is then noted as:

$$\mathscr{S}_{\text{syn}} \leftarrow \mathcal{M}_p(\mathcal{S}, \mathscr{D}_{\text{sup}}),$$

where $\mathcal{M}$, p and $\mathcal{S}$ are the PLLM, prompt function and sequence of questions from $\mathscr{D}_{\text{gt}}$, respectively. Ideally $\mathscr{S}_{\text{syn}}$ is as similar to $\mathscr{S}_{\text{gt}}$ as possible. The data generation workflow is straightforward and restricted to prompt engineering, whose usual key elements are instantiated as:

- **Task Specification**: $e_{\text{task}} =$ `Given a Question, generate equivalent SQUALL.`

- **Generation Condition**: $e_{\text{condition}} = \texttt{Follow the requirement to only return SQUALL.}$
- **In-Context Demonstrations**: $e_{\text{demo}} = \bigodot_{(s,S)\in\mathscr{D}_{\text{sup}}} \texttt{Question: } s \quad \texttt{SQUALL: } S,$

where $\bigodot$ is the large operator for string concatenation. They are wrapped together through a prompt template function:

$$E : (\Sigma^*)^3 \to \Sigma^*, (s_1, s_2, s_3) \mapsto E(s_1, s_2, s_3).$$

The prompt function is noted as:

$$p(s, \mathscr{D}_{\text{sup}}) \leftarrow E(e_{\text{task}}, e_{\text{condition}}, e_{\text{demo}}), \quad s \in \mathcal{S},$$

which given a sample question s and the support set, maps to an instantiation of the generation task, i.e., the prompt specific to s used to generate its, ideally, equivalent SQUALL. The prompt previously given was an example of such an instantiation (see Listing 4.15).

Supports set are generally included to increase the faithfulness of the generated data, since they can be effectively used by the PLLM due to its In-Context Learning (ICL) capabilities [69].

4.2.2 Retriever Initialization and Evaluation

An ideal retriever, i.e., perfect PCST parameters were implicitly assumed in Section 4.1.1.4, but in practice, it's important to define what constitutes good performance for the retriever, and to establish a method for evaluating this performance. This allows the parameters to be adjusted for better results.

The problem of subgraph retrieval has been previously discussed by [70] in the context of multi-hop KGQA. Certain results were empirically shown that still hold despite SQUALL generation being a distinct task. First, reasoning over a question-relevant subgraph instead of the entire KB is more performant due to limiting the reasoning space. Second, a subgraph that is too small for QA may fail to contain the answer, whereas an overly large graph introduces excessive noise, which can degrade performance. The informativeness of retrieved subgraphs was assessed using the answer coverage rate, an evaluation function that measures the proportion of retrieved content containing the correct answer. In essence, this metric provides a specific way of evaluating recall. Section 4.3.4 elaborates on recall in the context of the text-to-SPARQL task.

The initialization of the PCST parameters in the context KG-subgraph construction, cannot be decoupled from the KG itself. Therefore, their impact is empirically evaluated, see Section 5.2.2.

4.2.3 Learning

This section provides a detailed explanation of the sequential training strategy, thoroughly outlining its two distinct steps.

4.2.3.1 Sequential Training

The sequential training strategy consists of two key steps: fine-tuning the SQUALL expert and training the domain expert, illustrated in Figure 4.2. In the first step, the SQUALL expert — an LLM — is fine-tuned to generate PGPs for questions within a source domain. This tailored fine-tuning equips the model with specialized knowledge for generating queries in SQUALL. Subsequently, the fine-tuned SQUALL expert is integrated into the domain expert — a RAG model — as the generator component. The domain expert is then trained to perform the same task, but in a different target domain. Crucially, during this second training phase, the SQUALL expert remains frozen, preserving its acquired knowledge of SQUALL and avoiding catastrophic forgetting.

The ensuing sections provide details into the specifics of each training step.

4.2.3.2 Fine-Tuning of the SQUALL Expert

The LC-QuAD 2.0 dataset was used as the basis for the source domain dataset ($\mathscr{D}_S$). First, a validation set was created by splitting 20% from its training set using stratified sampling based on query templates to maintain a template distribution consistent with the original training set. The resulting split was: 15 434, 3 859 and 4 781.

Then, the template distribution was optimized by grouping samples with similar query templates, such as the nearly identical templates:

```
?E is_a Type. ?E pred Obj. ?E-secondClause value. MIN (value)
?E is_a Type. ?E pred Obj. ?E-secondClause value. MAX (value)
```

The data was then resampled to achieve a more balanced distribution of query templates. This step aimed to mitigate overfitting to specific query types and enhance the model's adaptability to different domains. By balancing the dataset, the model is less likely to overfit to query structures it encounters frequently during training.

This approach is supported by recent findings [30], which highlight that LLMs often associate specific query structures with types of question structures during training. When confronted with a similar question structure, LLMs tend to generate the query structure they have most often encountered with that input. Balancing the query template distribution reduces this bias, promoting a more generalized and flexible generation process.

The instruction passed to the LLM during fine-tuning follows the Alpaca Prompt Template, see Listing 4.16.[9]

4.2.3.3 Training of the Domain Expert

The SciQA dataset was used as the basis for the target domain dataset ($\mathscr{D}_T$). Its train, validation and test splits contain 1 795, 257, and 513 samples, respectively. These splits were consistently applied across all experiments. Where necessary and feasible, minor data engineering was carried out to rectify errors; otherwise, problematic

[9] Alpaca Prompt Template from https://github.com/tatsu-lab/stanford_alpaca.

```
Below is an instruction that describes a task, paired with an input that provides
 ↪  further context.
Write a response that appropriately completes the request.

### Instruction:
Given the following sentence, your job is to generate SQUALL for it in the json
 ↪  format.

### Input:
What is Delta Air Line's periodical literature mouthpiece?

### Response:
What is a <mouthpiece> of <Delta_Air_Lines> and has <instance_of>
 ↪  <periodical_literature>?
```

Listing 4.16: Fine-tuning example following the Alpaca Prompt Template.

samples were withheld. Finally, the queries from the SciQA dataset were executed on a local endpoint of the associated KG (see Section 5.1.1.1), and those without response were removed. After cleaning SciQA, the distribution of templates in its test set is approximately identical to the train and validation template distributions, see Table 4.3.

Template	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Percentage of Samples	13.01	13.21	14.84	16.06	19.31	3.05	20.33	0.19

Table 4.3: SciQA test split query template distribution.

In order to realize the training of the graph embedder and projector under the assumption of data scarcity, inspiration was taken from related work [71], that proposes training a retriever solely relying on questions.[10] While distinct in approach, this work demonstrates the potential of starting with just a set of questions. It served as the motivation to generate the synthetic dataset $\mathscr{D}_{\text{syn}}$.

Finally, training of the domain expert was based on previous open-sourced code [43]. Changes include the use of the fine-tuned SQUALL expert and the use of one graph for all question, i.e., the Open Research Knowledge Graph

[10] Questions that in turn might be generated themselves.

(ORKG).

4.3 Evaluation

Validating the system requirements (Section 3.4) and answering the research question (Chapter 1) require critical evaluation of the approach, both are first restated here.

System Requirements

SR1. Minimize data requirements.

SR2. Maintain simplicity.

SR3. Facilitate comprehensive report creation.

SR4. Enable human oversight.

Research Questions

RQ1. How does the performance of the proposed domain expert compare to other approaches across various data availability scenarios?

RQ2. Does the domain expert effectively utilize the retrieved information with which it is soft-prompted, and what factors influence the effectiveness of these prompts?

RQ3. What is the impact of using synthetic data, as opposed to ground truth data, on domain adaptation performance during the second learning phase of sequential training?

Given that the last three system requirements are of a boolean nature, and that their satisfaction is dependent on the text-to-SPARQL performance of the system, emphasis is further put on the primary system requirement and the research questions. They can be quantitatively evaluated using four typical performance indicators: Bilingual Evaluation Understudy (BLEU)-score, execution (or query) accuracy and F_1 score [30] — the former originating from Neural Machine Translation (NMT) and the latter two from QA — as well as recall — a classic IR statistic. Two additional evaluation functions were considered but are not discussed further due to their unsuitability.

BLEU was employed for most evaluations, including assessing the quality of $\mathscr{D}_{\text{syn}}$, the performance of the domain expert, the overall text-to-SPARQL component, and comparisons to related work. Execution accuracy and F_1 score were used exclusively to evaluate the performance of the text-to-SPARQL component and to facilitate comparisons with related work. Lastly, recall was solely utilized to analyze the retriever's performance.

4.3.1 BLEU

The corpus-level BLEU score from NLTK is used to evaluate the quality of machine translations between natural languages, though it is also commonly applied to assess translations from natural to formal languages.[11] In this context, BLEU serves to evaluate the generated queries against a baseline prior to the linking step. Since linking is inherently imperfect, isolating the system's performance up to that stage provides valuable insights.

4.3.2 Execution Accuracy

Execution accuracy [72] is the ratio of queries with correct results when executed N_c to the total number of queries N, i.e., N_c/N. The relevancy of this ratio deems from the no-code perspective introduced in Chapter 1. A shortcoming is the possibility of false positives: a set of queries might return identical results. To address this, the F_1-score is also looked at, it offers a more nuanced evaluation by considering both precision and recall.

4.3.3 F_1 Score

SPARQL queries regularly return multiple results. Precision p is defined as the ratio of relevant retrieved results to the total number of retrieved instances, while recall r is the ratio of relevant retrieved results to the total number of relevant instances. In this context, retrieved instances are those returned by the generated SPARQL query, and relevant instances are those present in the ground truth. The F_1 score, calculated as:

$$F_1 = \frac{2pr}{p+r},$$

is computed for each question and then averaged across the entire dataset, resulting in the average F_1 score [61].

4.3.4 Recall

Recall was employed to provide insight into the constructed subgraphs, it essentially evaluates how well a retrieved subgraph corresponds to its query. In general IR terms it is the ratio of relevant retrieved instances to all relevant instances. Here, the instances are vertices and edges, and they are separately evaluated. The ratios are denoted by vertex and edge recall, and respectively measure the fraction of vertices and edges from a SPARQL query that appear in the subgraph. This evaluation confirms the intuitive notion that more generous PCST parameter settings lead to more comprehensive subgraphs (i.e., those that cover a larger portion of the graph's vertices and edges). However, while this approach can indicate the extent of coverage, it does not guarantee increased informativeness due to the growing size of subgraphs and the inherent limitations of LLMs.

[11]NLTK: nltk.org.

Definition 9. *Vertex Recall* and *Edge Recall* are defined as:

$$R_V(Q,H) = \frac{|\{\, v \mid v \in V_Q \wedge v \in V_H \,\}|}{|V_Q|}, \tag{4.11}$$

$$R_E(Q,H) = \frac{|\{\, e \mid e \in E_Q \wedge e \in E_H \,\}|}{|E_Q|}, \tag{4.12}$$

where Q is a GP and $H \in V_{\mathscr{C}(G)}$ with $G = (V_G, E_G)$ the KG. Given a dataset $\mathscr{D}$ consisting of samples that at least include a PGP and subgraph, the average recall is:

$$\overline{R}_{V,\mathscr{D}} = \frac{1}{|\mathscr{D}|} \sum_{(Q,H)\in\mathscr{D}} R_V(Q,H), \tag{4.13}$$

$$\overline{R}_{E,\mathscr{D}} = \frac{1}{|\mathscr{D}|} \sum_{(Q,H)\in\mathscr{D}} R_E(Q,H). \tag{4.14}$$

4.3.5 Graph Similarity

While vertex and edge recall provide an initial measure of the quality of retrieved information, they are limited in scope. Specifically, these evaluation functions do not consider the size of the constructed subgraphs, nor do they address reasoning and prompt size limitations inherent to LLMs. Intuitively, semantically accurate queries should match with the Knowledge Graph making graph similarity a more comprehensive evaluation approach.[12] The relationship between queries and KG subgraphs has been previously studied [54], [73], [74]. This section examines the Graph Edit Distance (GED) and Maximum Common Subgraph (MCS) graph similarity functions; however, experimental results revealed significant limitations, rendering them unsuitable for this task.

Graph Edit Distance The Graph Edit Distance (GED) is defined as the minimum cost required to transform a graph G_1 into an isomorphic graph G_2 through a series of vertex and edge edit operations. This similarity function accounts for the size difference between the query graph and the subgraph, operates as a non-learning algorithm, and allows for configurable edit operation costs. However, the size of the constructed subgraphs, ranging from approximately 16 to 9 000 vertices, and the significant size discrepancy compared to the query-derived graphs (i.e., Graph Patterns), which consist of only 9 to 17 vertices (refer to the GPs in Section 5.18), make GED computationally infeasible. For comparison, experiments with the reference implementation in the NetworkX library [75] were limited to graphs with an average of 5 to 25 vertices. [13] NetworkX's approximate GED implementation also proved insufficient for addressing the scalability challenges.

[12] Indeed, graph databases perform Graph Pattern Matching (GPM) — the task of matching a Graph Pattern against a graph — to query, for example, RDF graphs [55].

[13] NetworkX: networkx.org

Maximum Common Subgraph Maximum Common Subgraph (MCS) identifies the largest subgraph common to two graphs It does not account for differences in graph size. Consequently, it exhibits a tendency to favor the quantity of information, similar to recall metrics. For instance, the most similar subgraph under MCS will invariably be the entire KG itself, regardless of query specificity. Furthermore, the computational intractability of exact MCS [76] further limits its applicability, particularly for large-scale graphs.

4.4 Conclusion

This methodology chapter has outlined the development and validation process of the proposed model reporting paradigm. This involved a detailed examination of the system architecture, design decisions, and the evaluation strategy tailored to the openCAESAR framework. The technical depth provided serves as the foundation for comprehensively assessing the paradigm's effectiveness and applicability to MBSE.

The upcoming experimental chapter provides a thorough empirical evaluation of the proposed methodology, offering critical insights into its validity and practical applicability. This analysis not only assesses the methodology's contributions but also contextualizes them within the broader research landscape.

5

Experiments

This chapter presents the experiments conducted to validate the requirements outlined in Section 3.4. It begins with a detailed explanation of the experimental setup, followed by five key experiments and their results. The approach is then benchmarked against related work from the literature. The chapter concludes with a discussion of the findings and their broader implications.

5.1 Experimental Setup

This section elaborates on the configuration of the Knowledge Graph, the specific models utilized along with their respective settings, and the software and hardware resources employed during the experiments.

5.1.1 Knowledge Graph and Benchmark

ORKG was chosen to evaluate the proposed approach, it is a scientific knowledge graph consisting of papers and authors.[1] As of writing the KG consists of 904 212 unique entities, although for this work the choice was made to work with an older version — because the SciQA benchmark, introduced in Section 4.2.1.2, that targets ORKG is outdated — consisting of 180 184 and 6 888 unique entities and relations, respectively.

5.1.1.1 Motivation

The subsequent paragraphs explain why the choice was made to work with this KG and its associated benchmark.

Domain ORKG and SciQA are specialized, i.e., tailored to a specific instead of a general domain. This is in line with the ontologies that occur in MBSE. A KG with no particular focus would be less appropriate.

[1]ORKG: orkg.org.

Semantics As discussed in Section 4.2.3, the SQUALL expert was fine-tuned using the LC-QuAD 2.0 dataset, which is designed for the Wikidata KG. Despite substituting all URIs with placeholders — mapping GPs to PGPs — before training, the semantic information of the original KG remains embedded in the queries. This persistence occurs because the original queries are inherently structured to align with the KG; otherwise, they would yield no results upon execution.[2] Overlap between datasets can occur; for instance, both DBpedia and Wikidata include the `marriage` relation between two `human` entities. To mitigate such similarities, the approach is evaluated on a specialized domain. If evaluation were instead conducted on DBpedia while the SQUALL expert was trained on Wikidata, the results could appear overly optimistic.

Version SciQA is out of date with the current version of ORKG: only 10 out of 2 565 queries returned results. Hence, the version of February 14, 2023 was used — RDF dump made available at [63] — which returns query results in ≈92% of cases, and contains 181 147 entities and 6 888 properties. It was made accessible through a local SPARQL endpoint using Apache Jena Fuseki.

Size The survey of ontology engineering projects mentioned in the introduction of Chapter 3 also gives insight into the size of real-world ontologies [45], where ontology size is taken to be the number of unique entities. The size of the chosen KG is considered to be sufficiently large to be representative, see Figure 5.1.

5.1.1.2 Templates

To better understand the structure of queries in the SciQA dataset, Listing 5.1 provides two example SPARQL queries, and the equivalent SQUALL expressions. Note that while the questions and SPARQL queries are from SciQA, the SQUALL expressions are constructed as part of this work.

One notable difference between the two examples is the presence of an `OPTIONAL` clause in the bottom one. This difference is also reflected in the corresponding SQUALL expression, where it is represented by `if defined`.

Clearly, these two queries have distinct structures. However, due to the way the SciQA dataset is created, see Section 4.2.1.2, there are only a limited quantity of such unique query structures, called templates, present, namely eight. Note that this is typically the case for these types of datasets, e.g., LC-QuAD. Although queries will vary, for example, in terms of which literals are present, the structure they have will always belong to one of eight templates.

These examples highlight the distinct structures of the queries. However, the SciQA dataset, as explained in Section 4.2.1.2, is constructed with a limited number of unique query structures, referred to as templates. Specifically,

[2] SPARQL is a graph query language where queries are represented as Basic Graph Patterns (BGPs) or Complex Graph Patterns (CGPs) — both of which are ultimately graphs (see Figure 4.4). Executing a query involves matching a GP against a KG, a process known as Graph Pattern Matching (GPM) or subgraph matching. Consequently, any dataset of queries designed for a specific Knowledge Graph will include GPs with topologies that align with the structure of that graph.

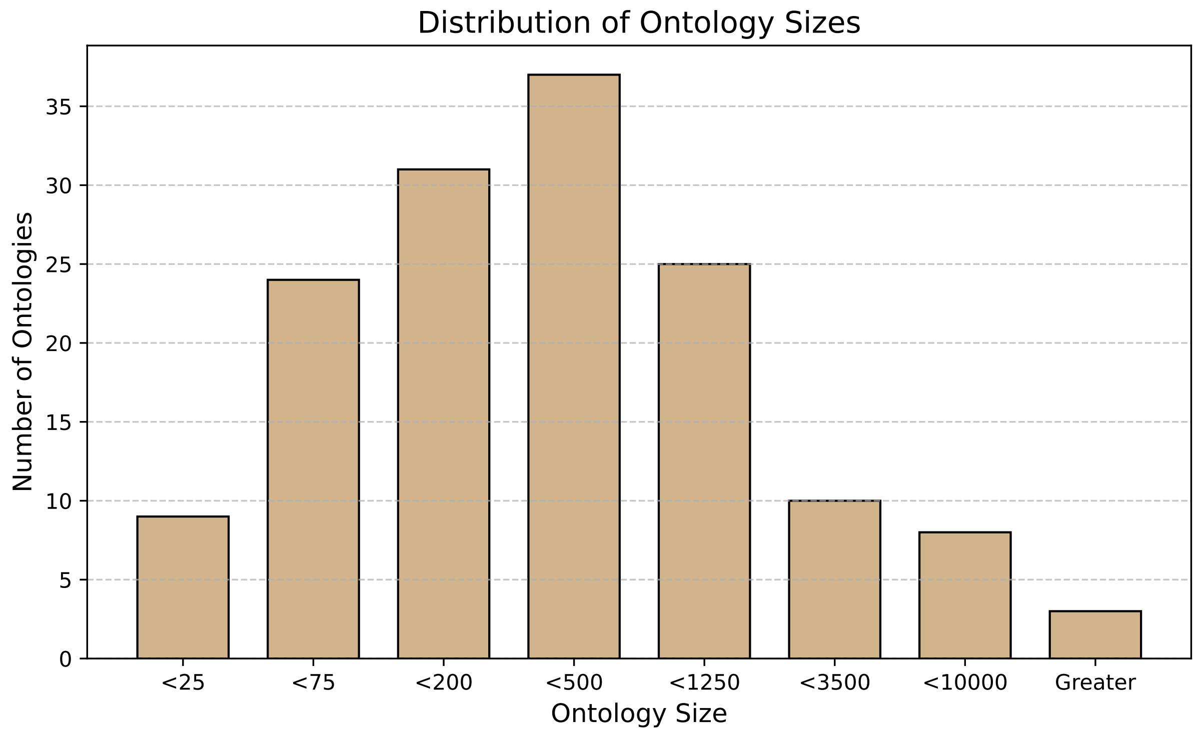


Figure 5.1: Distribution of ontology sizes (i.e., number of entities) from a survey [45].

there are eight such templates in total. This is a common characteristic of datasets in this domain, such as LC-QuAD. While the specifics (e.g., literals) of queries in the dataset may vary, their underlying structure always conforms to one of these templates.

The query templates in SciQA are denoted by the set $T = \{1, \dots, 8\}$. When a specific template $x \in T$ is excluded, this is represented as $T \setminus \{x\}$.

For an example of each template, the appendix can be consulted, see Section 5.18.

5.1.2 Model Configurations and Settings

In the paragraphs ahead practical choices related to the implementation and experimentation are summarized.

Retrieval Before training the domain expert, the KG index ($\mathscr{I}_{\text{KG}}$) is first built using Sentence-BERT [77] as embedding model (λ) with the dimension set to 1 024. The similarity function (σ) used for indexing was the cosine similarity.

Augmentation Following [43], the specific graph embedder used for embedding the subgraphs was the Graph Attention Network [78], with 4 layers, a hidden dimension (d_g) of 1 024, 4 heads and a dropout of 0. The projector

```
SPARQL:
SELECT DISTINCT ?code WHERE {
        ?model a <Model>;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark <has_dataset> ?dataset .
        ?cont <has_benchmark> ?benchmark .
        ?cont <has_model> ?model;
                <has_source_code> ?code .
}
SQUALL:
What is a <has_source_code> of a thing that has <has_model> a <Model> X and has a
 ↪  <has_benchmark> that has a <has_dataset> and X has a rdfs:label whose string
 ↪  matches 'Depth DDPPO'?
===
SPARQL:
SELECT DISTINCT ?model ?model_lbl WHERE {
        ?dataset a <Dataset>;
                rdfs:label ?dataset_lbl .
        FILTER(STR(?dataset_lbl) = 'SentEval')
        ?benchmark <has_dataset> ?dataset;
                <Has_evaluation> ?eval .
        ?paper <has_benchmark> ?benchmark .
        OPTIONAL {
                ?paper <has_model> ?model .
                ?model rdfs:label ?model_lbl .
        }
}
SQUALL:
A <Dataset> is a <has_dataset> of a thing X and has a rdfs:label whose string matches
 ↪  'SentEval' and X is a <has_benchmark> of a thing Y and a thing is a
 ↪  <has_evaluation> of X and if defined, what is a <has_model> of Y and has a
 ↪  rdfs:label?
```

Listing 5.1: Example question-query pairs from SciQA with equivalent SQUALL expression.

is implemented as a Multilayer Perceptron with 2 layers, a hidden dimension of 2 048 and the output dimension (d_t) is 4 096, aligned with the hidden dimension of the LLM used in the generator. Some experimentation was done where the entire KG served as graph token, which required reducing the dimensions of the graph embedder, though this line of investigation was not further pursued.

Generation The LLM used for the generator component was Llama2-7b [79]. It served as the SQUALL expert that was frozen after being fine-tuned in all the experiments. More specifically, a LLaMA-Adapter [80] is fine-tuned. The Parameter-Efficient Fine-Tuning (PEFT) technique that was used was Quantized Low Rank Adaption (QLoRA) [81]. Parameter settings were informed by standard fine-tuning approaches for LLaMA v2.[3]

Domain Expert Observing diminishing returns from initial experiments, patience was set to 3, with all other parameters following the settings described by [43].

Linking The similarity function (σ) used during linking was kept from the earlier implementation [25], it is based on the Coherency Factor (CF) [82]. Given a placeholder string s and an `rdfs:label` of a vertex or edge x, they are represented as sequences of words t:

$$t = (w_i)_{i \in I}.$$

The sequences are embedded using an operator T, here defined as:

$$T : t \mapsto \left\{ W_i^{\phi} = \begin{cases} \lambda(w_i) & w_i \in \text{dom}(\lambda) \wedge \phi = \lambda, \\ \delta(w_i) & w_i \notin \text{dom}(\lambda) \wedge \phi = \delta, \end{cases} \middle| \, i \in I \right\}.$$

In practice λ and δ are the fastText [83] and chars2vec model, respectively.[4] The latter serves as a fallback option if the former fails to recognize the input. The similarity is then defined as:

$$\sigma(s, x) = f_{CF}(T(s), T(x)).$$

Let $\mathcal{S} = T(s)$ and $\mathcal{X} = T(x)$, then the Coherency Factor (CF) between sets of embeddings is defined as follows:

$$f_{CF}(\mathcal{S}, \mathcal{X}) = \frac{1}{|\mathcal{S}||\mathcal{X}|} \sum_{(S^{\phi_s}, X^{\phi_x}) \in \mathcal{S} \times \mathcal{X}} \sigma_* \left(S^{\phi_s}, X^{\phi_x} \right),$$

where

$$\sigma_* \left(S^{\phi_s}, X^{\phi_x} \right) = \mathbb{1}_{\{\, \phi_s = \phi_x \,\}} \left(\sigma_{\text{cos}} \left(S^{\phi_s}, X^{\phi_x} \right) \right),$$

using the indicator function to define embeddings originating from different models as perpendicular. Linking back to the similarity function used for indexing, they are evidently quite distinct. The works that proposed them in their original context either did not explore or report any significant performance impact using alternatives. As a comprehensive evaluation of different similarity functions was deemed out of scope, both functions were retained as originally implemented.

Pre-Trained Large Language Model The OpenAI API was used to create the synthetic datasets.

[3]The fine-tuning procedure was guided by the tutorial available at https://www.kdnuggets.com/fine-tuning-llamav2-with-qlora-on-google-colab-for-free.

[4]Chars2vec available at https://github.com/IntuitionEngineeringTeam/chars2vec.

Software and Hardware The implementation uses a server designed to process incoming questions, interact with the SPARQL endpoint of the KG, and return results. This server is constructed using a lightweight Python Flask application and is hosted on Apache with an Amazon Web Services (AWS) environment. Both the server and the query endpoint are deployed on the same AWS EC2 instance.

5.1.3 PCST Parameter Triple

The subsequent sections make use of a shorthand notation to indicate the parameters of the Prize-Collecting Steiner Tree (PCST) algorithm. The algorithm is used to construct relevant graphs for a given input question, and it is implemented as the retrieval and post-retrieval components.

The parameter triple $(k_\text{v}, k_\text{e}, C)$ specifies the top-k retrieved vertices (k_v), the top-k retrieved edges (k_e), and the cost per edge (C), as defined in Equation (4.5), Equation (4.6), and Equation (4.9).

In practice, k_v and k_e are always equal and are collectively referred to as k.

5.1.4 Notebook and KGQA Pipeline Connection

The interaction between the notebook environment and the KGQA pipeline is facilitated through a custom Python package that introduces a user-friendly magic command for posing questions. Developed by François Goybet as part of his bachelor's project in tandem with this master's thesis, this package enhances usability by enabling seamless communication with the KGQA pipeline hosted on an AWS server.[5]

Once the custom magic is loaded in a notebook cell, users can input a natural language question. Upon execution, the question is sent to the pipeline, which automatically handles all intermediate steps — ranging from question parsing to result retrieval. This process eliminates the need for stakeholders to interact directly with the underlying pipeline, providing an intuitive interface for querying the system.

This extension, tailored to the requirements of this work, was independently developed but designed to integrate seamlessly into the research setup. The connection allows users to leverage the pipeline without requiring extensive technical knowledge. This integration not only simplifies stakeholder interaction with the KGQA pipeline but also significantly enhances the accessibility and practicality of the system.

5.2 Retrieval and Post-Retrieval Analysis

The first experiment investigates the retrieval and post-retrieval components applied to ORKG, introduced in Section 4.1.1.4 and Section 4.1.1.5, respectively.

[5]Python package available at https://pypi.org/project/molab-ext.

These two components aim to construct a relevant subgraph of the KG for a given question that can be used to improve the performance of the domain expert. The running example from Chapter 4 is reintroduced here, see Figure 4.6. Consider the following question: `Provide a list of papers that have utilized the Depth DDPPO model and include the links to their code?` When inputted to the retriever it will return relevant vertices and edges of the KG, for example, there might be a vertex `Model` and an edge `HAS_SOURCE_CODE`. The post-retrieval component is then responsible for creating a connected graph out of these looses vertices and edges.

The primary objective of retrieval is to be as informative as possible. However, two inherent limitations of any RAG paradigm are poor and over-retrieval, see Section 4.1.2.2, both negatively impacting generation. This implies that a poor parameter choice likely reduces the performance of the domain expert. Ideally, the subgraphs at the end of post-retrieval capture all the necessary vertices and edges needed for the domain expert to correctly generate the expected SQUALL, while remaining as compact as possible. Hence, it is important to have some notion of how the parameters of the retrieval and post-retrieval components influence the constructed subgraphs.

What is being evaluated is essentially how informative the subgraphs are in function of the PCST parameters.

5.2.1 Hypothesis

Increasing the top-k parameter is expected to result in larger subgraphs with higher recall for both entities and relations. The cost per edge is anticipated to provide fine control over subgraph size, for example, increasing the cost per edge will likely contribute to noise reduction by steering the algorithm towards smaller subgraphs.

The retriever's parameters are presumed to have a direct impact on the quality of the constructed subgraphs. Larger subgraphs are likely to capture more of the relevant vertices and edges. However, excessive size may introduce unnecessary noise. A higher cost per edge is expected to mitigate noise, focusing the subgraphs on the most informative components while maintaining high recall.

5.2.2 Setup

Evaluation is performed by varying the PCST parameters and investigating how it impacts the subgraphs, outputted by the post-retrieval component, using average vertex/edge recall and average number of vertices/edges, defined in Equation (4.13) and Equation (4.14), respectively.

5.2.3 Results

The results, see Table 5.1, indicate that both the size of the constructed graphs and the recall increase with top-k, confirming the straightforward intuition that more lenience in terms of which vertices and edges are considered relevant, results in bigger constructed subgraphs containing at least as many of the expected vertices and edges.

Parameters	Average # Vertices	Average # Edges	Average Vertex Recall	Average Edge Recall
(5, 5, 0.1)	26	28	0.00	3.57
(5, 5, 0.5)	16	15	0.00	3.57
(10, 10, 0.1)	49	56	15.98	26.36
(10, 10, 0.5)	35	37	3.57	16.07
(30, 30, 0.1)	197	255	10.71	42.86
(30, 30, 0.5)	104	115	27.82	44.19
(100, 100, 0.1)	739	1 150	21.43	78.57
(100, 100, 0.5)	386	477	42.72	65.08
(1000, 1000, 0.1)	7 316	11 341	90.41	95.63
(1000, 1000, 0.5)	3 126	5 166	88.75	95.23

Table 5.1: The average number of vertices and edges, and the average vertex and edge recall (expressed as percentages), in function of PCST parameters.

Evidently, if the entire graph would be returned from the retriever, recall would be equal to 1. As for the cost per edge, increasing it reliably results in a reduced subgraph size, all else being equal. The parameter seems especially valuable for higher top-k values, since the loss in recall reduces (to zero), while resulting in about half the amount of vertices/edges.

Notably, the execution time of the PCST algorithm remains stable even as the top-k vertices and edges increase and the cost per edge decreases, with only a slight increase in running time.

5.3 Synthetic Dataset Quality

This experiment gauges the quality of the synthetic dataset generated using the rudimentary strategy, as explained in Section 4.2.1.3, and used to train the domain expert, see Section 4.2.3.3.

A high-quality training dataset containing question-query pairs for a specific domain is often unavailable in practice due to data scarcity. This work addresses this challenge primarily through Domain Adaptation (DA) and the use of the Controlled Natural Language SQUALL. However, synthetic data can also play a vital role in mitigating data scarcity. To explore this, the performance of the domain expert is compared when trained on ground truth data versus synthetic data, as discussed in Section 5.5.

A basic synthetic data generation approach was deliberately chosen — advanced strategies are considered out of scope — nevertheless, evaluating the quality of the generated synthetic data remains essential for subsequent experiments, as it provides critical insights when interpreting the results.

Assessing the quality of generated synthetic data is straightforward when ground truths are available, as is the case here, direct evaluation can be performed which boils down to evaluating how faithful the generated samples are to their references [69], ideally they are identical. See Listing 5.2 for an example.

```
Ground Truth SQUALL:
What is a <has_source_code> of a thing that has <has_model> a <Model> X and has a
 ↪  <has_benchmark> that has a <has_dataset> and X has a rdfs:label whose string
 ↪  matches 'Depth DDPPO'?
---
High quality synthetic SQUALL:
What is a <has_code_link> of a thing that has <has_model> a <Model> X and has a
 ↪  <has_benchmark> that has a <has_dataset> and X has a rdfs:label whose string
 ↪  matches 'Depth DDPPO'?
> BLEU score: 90
> One placeholder not faithful to reference:
> <has code link> instead of <has source code>.
---
Low quality synthetic SQUALL:
A <Model> is a <has_model> of a thing X and has a rdfs:label whose string matches
 ↪  'Depth DDPPO' and X is a <has_paper> of a thing Y and a thing is a <has_code_link>
 ↪  of Y and if defined, what is the <has_code_link> of Y and has a rdfs:label?
> BLEU score: 23
> Wrong demonstration followed from prompt.
```

Listing 5.2: Comparison of ground truth and synthetic SQUALL for a question, highlighting high-quality and low-quality examples, along with their BLEU scores.

5.3.1 Hypothesis

The higher the Demonstration Per Template (DPT) and the more templates are included in the prompt likely improves performance up to a certain point.

Given that only eight templates are present in the SciQA benchmark, it seems improbable that the inclusion of a template would confuse the PLLM. Furthermore, increasing the DPT gives more examples to the model, which should intuitively increase faithfulness.

5.3.2 Setup

Faithfulness between generated and reference SQUALL queries is evaluated using the BLEU score. Elementary prompt engineering experiments are conducted to examine how variations in the support set ($\mathscr{D}_{\text{sup}}$) affect data quality.

Two engineering dimensions are explored:

- **Template Variations**:

  As outlined in Section 5.1.1.2, SciQA queries belong to one of eight templates. This axis examines the effect of excluding certain templates from the support set. Three configurations are tested:

  - All templates included: T.
  - Excluding template 2: $T \setminus \{2\}$.
  - Excluding template 6: $T \setminus \{6\}$.

  These configurations enable evaluation of the faithfulness of generated samples when a template is absent from the prompt.

  Template 2 was excluded due to its significant dissimilarity to the other query templates, making it a strong candidate for assessing the impact of exclusion on quality. In contrast, templates 4 and 5 were the most similar to each other, it is thus likely that excluding one or the other would result in minimal deterioration of the synthetic data quality. Template 6 was excluded for a similar reason to template 2, as it also exhibited notable dissimilarity.

  The dissimilarity was calculated by grouping the queries from the ground truth dataset by template, extracting all tokens associated with each template, and computing TF-IDF scores. Pairwise cosine similarity was then calculated for all template combinations, with templates ranked from most similar to most dissimilar. Templates 6 and 2 appeared as the least similar templates in the ranking, being the penultimate and last templates, respectively, to form a pair.

  While the ideal scenario would involve testing one setup for each excluded template, practical constraints necessitated focusing on this subset. This selection was deemed most suitable for evaluating generalization to unseen and, likely, significantly (more) dissimilar templates in real-world applications.

- **Number of Demonstration Per Templates (DPTs)**: The second axis explores the impact of varying the number of DPTs. Two setups are considered:

  - One DPT.
  - Three DPTs.

In total, these variations result in six setups with each a unique configuration of the support set ($\mathscr{D}_{\text{sup}}$). For each configuration, corpus-level BLEU scores are computed across all samples and on a per-template basis. The ground truth dataset ($\mathscr{D}_{\text{gt}}$) serves as the reference for evaluating all generated synthetic datasets.

The only factor differentiating the six setups is the composition of $\mathscr{D}_{\text{sup}}$, see Section 4.2.1.3.

Templates	1 Demonstration per Template			3 Demonstrations per Template		
	T	$T \setminus \{2\}$	$T \setminus \{6\}$	T	$T \setminus \{2\}$	$T \setminus \{6\}$
T	74	73	74	93	88	92
1	65	71	69	98	99	97
2	81	<u>38</u>	81	94	<u>37</u>	96
3	89	89	91	98	96	98
4	76	74	79	93	91	94
5	67	69	63	91	94	90
6	79	75	<u>51</u>	97	93	<u>56</u>
7	71	72	71	87	88	86
8	72	82	83	100	100	95

Table 5.2: Quality assessment of the synthetic dataset using BLEU scores expressed as percentages. This evaluation compares the generated SQUALL queries to their corresponding ground truths, including an analysis of queries generated with certain demonstrations excluded from the prompt's support set.

5.3.3 Results

The results of the experiment are given in Table 5.2.

What is being evaluated here is in essence the PLLM's ability to annotate data [84]. As expected, it is unequivocal that increasing the DPT from one to three improved the faithfulness of the data, although no further conclusion can be made with this limited setup.

Moreover, the removal of a template from the prompt clearly has a substantial impact on the data quality for that specific label, but this adverse effect is not equal across all templates, e.g., the quality of the generated samples following query template 2 in the $T \setminus \{2\}$ case is notably worse than the quality of the generated samples following query template 6 in the $T \setminus \{6\}$ case. This could be related to the fact that template 2 is more dissimilar to the remaining demonstrations than template 6 is, see Section 5.3.2. Finally, the removal of one template from the prompt has no significant impact on the quality of generated samples belonging to the remaining templates.

No further conclusions can be made from this limited setup.

5.4 Text-to-SQUALL

This section introduces the first evaluation of the domain expert, particularly its ability to adapt to new a domain that differs from the source domain used during fine-tuning of the SQUALL expert.

The domain expert, see Figure 4.2, is the RAG model that generates SQUALL given a question. Note that these SQUALL expressions do not yet contain any URIs of the target KG. In order to obtain an executable SPARQL query, a SQUALL expression must still pass through the remainder of the text-to-SPARQL pipeline consisting of translation and linking, see Figure 4.1.

The objective of this experiment is to evaluate the performance of the domain expert in isolation, separate from the rest of the pipeline. This approach minimizes confounding factors, such as inaccuracies introduced by an imperfect linking procedure. Additionally, the evaluation focuses solely on the domain expert's capabilities before incorporating synthetic data; the model is trained exclusively on the ground truth dataset ($\mathscr{D}_{\text{gt}}$).

The evaluation assesses the model's ability to generate SQUALL queries with placeholders, comparing its outputs against ground truths for various PCST parameter configurations. While the first experiment (see Section 5.2.2) provides preliminary insights into the influence of these parameters, this experiment extends those findings by directly examining their impact on the quality of the generated queries, as discussed in Section 4.2.2.

5.4.1 Hypothesis

Intuitively, more informative subgraph should enhance performance, particularly when the retrieval and post-retrieval components achieve higher average recall and maintain smaller graph sizes on average.

This assumption is grounded in the expectation that increasing relevant information and decreasing distracting noise improve the generation process.

5.4.2 Setup

Evaluation is done using BLEU scores, comparing generated to reference SQUALL.

5.4.2.1 Reducing the Number of PCST Parameter Configurations

Some parameter configurations were discarded due to practical considerations, such as running time, as informed by the retrieval and post-retrieval analysis experiment (see Section 5.2.3) and a heuristic assessment.

For each pair of PCST parameter triples where k is identical but the cost-per-edge differs, one configuration was excluded from further investigation. Consider the following pair from Table 5.1: (1000, 1000, 0.1) and (1000, 1000, 0.5). Experimental results indicate that while the former achieves slightly higher vertex and edge recall, it also produces subgraphs that are, on average, more than twice the size in terms of vertices and edges. This is a direct consequence of its lower cost-per-edge parameter, and demonstrates a case of diminishing returns.

The heuristic for deciding which parameter triple from a pair (p_1, p_2) to keep, is defined as follows:

$$p^* = \begin{cases} p_1 & \text{if } \overline{R}_{V,\mathscr{D}_1} \gg \overline{R}_{V,\mathscr{D}_2}, \\ p_2 & \text{if } \overline{R}_{V,\mathscr{D}_2} \gg \overline{R}_{V,\mathscr{D}_1}, \\ p_1 & \text{if } \overline{V}_{\mathscr{D}_1} < \overline{V}_{\mathscr{D}_2}, \\ p_2 & \text{if } \overline{V}_{\mathscr{D}_2} \leq \overline{V}_{\mathscr{D}_2}. \end{cases}$$

Here, p^* denotes the retained parameter tuple. $\overline{R}_{V,\mathscr{D}_i}$ and $\overline{V}_{\mathscr{D}_i}$ represent the average vertex recall, see Equation (4.13) and the average number of vertices, see Equation (4.3), respectively, for the dataset resulting from retrieval and post-retrieval when the parameters from p_i were used.

In practical terms, a difference in recall was considered significant if it was at least 10. For the previous example, the difference in recall was only 1.66. As a result, the parameter triple (1000, 1000, 0.5) was selected, as it yields more compact subgraphs.

Finally, parameter triples with $k = 5$ were excluded, as they resulted in an average vertex recall of zero. Instead, a baseline approach was employed.

5.4.2.2 Baseline

The post-retrieval component outputs a graph, that helps the domain expert during generation through the augmentation process, see Figure 4.2. To assess the impact of different PCST parameters on the generation process, a baseline is established.

The baseline was determined based on two key observations. First, the output of the post-retrieval component is a graph. Second, the largest and smallest graphs that the post-retrieval component could theoretically produce are the entire KG and the zero-order graph K_0, respectively. Here, K_0 refers to a graph containing neither vertices nor edges. While both extremes were tested, using the entire KG proved unfeasible due to computational constraints.

This experiment used the baseline by providing K_0 as input to the augmentation process of the domain expert during inference.

Templates	Baseline (K_0)	(10, 10, 0.1)	(30, 30, 0.5)	(100, 100, 0.5)	(1000, 1000, 0.5)
T	98	94	82	91	95
1	99	93	74	84	95
2	100	99	85	99	98
3	100	84	61	70	98
4	99	91	84	94	90
5	100	97	79	95	99
6	$\underline{40}$	53	56	87	50
7	100	100	99	99	94
8	N/A	N/A	N/A	N/A	N/A

Table 5.3: Evaluation of the domain expert's performance on the text-to-SQUALL task using the BLEU score, expressed as precentages, in function of different PCST parameters.

5.4.2.3 Internal Evaluation

It is important to emphasize that the BLEU scores obtained from evaluating the SQUALL expressions generated by the domain expert are meaningful only within the scope of this work. Typically, in Semantic Parsing, BLEU scores are calculated between reference and generated SPARQL queries. However, the SQUALL expressions must still pass through the remainder of the text-to-SPARQL pipeline before becoming SPARQL queries, as illustrated in Figure 4.1.

Nevertheless, BLEU scores based on SQUALL expressions are employed as they enable an effective evaluation of the domain expert.

Consequently, evaluations based on SQUALL expressions are consistently employed, but strictly for internal evaluation purposes.

5.4.3 Results

Given that no previous results were found that detail the impact of input graph size or recall on graph prompting multiple models were trained with differing PCST parameters to test their impact on the text-to-SQUALL task. The results are shown in Table 5.3.

Performance in the K_0 scenario is notably and unexpectedly high. The fact that the best performance is achieved for K_0 is likely due to the relatively homogeneous nature of the SciQA dataset. This homogeneity is characterized by recurring entities and relationships within the SPARQL queries, thus somewhat nullifying the potential benefit of retrieval.

To concretize this, SciQA is compared to three other benchmarks in terms of its vocabulary. The vocabulary of a KB, denoted by K, includes URIs and literals [30]. In the context of RDF datasets: $K = I \cup L$, see Definition 1. A comparison of the datasets and their vocabularies is given by Table 5.4 and Table 5.5, respectively.

Dataset	Total	Train	Val	Test	Source
LC-QuAD	5 000	4 000	500	500	[30]
LC-QuAD 2.0	30 225	21 761	2 418	6 046	[30]
DBNQA	382 794	306 236	38 279	38 279	[30]
SciQA	2 565	1 795	257	513	This work

Table 5.4: Number of samples of four datasets, broken down into training, validation, and test splits, along with the overall totals.

Dataset	Total	Train	Val	Test	OOV	Source
LC-QuAD	4 751	4 150	1 068	1 065	318	[30]
LC-QuAD 2.0	34 801	27 949	5 413	10 993	7 523	[30]
DBNQA	157 672	142 985	36 845	37 130	7 998	[30]
SciQA	1 347	1 077	297	483	182	This work
SciQA, URIs	188	135	50	69	37	This work

Table 5.5: Knowledge Base vocabulary sizes.

Dataset	Total	Train	Val	Test	OOV
LC-QuAD	0.950	1.038	2.136	2.130	0.071
LC-QuAD 2.0	1.151	1.284	2.239	1.818	0.311
DBNQA	0.412	0.467	0.963	0.970	0.023
SciQA	0.525	0.600	1.156	0.942	0.089
SciQA, URIs	0.073	0.075	0.195	0.135	0.018

Table 5.6: Knowledge Base vocabulary size to number of dataset samples.

Clearly, SciQA is a relatively small dataset with a correspondingly limited vocabulary. However, when adjusting for dataset size by examining the ratio of vocabulary to dataset size (as shown in Table 5.6), a clearer picture emerges. From this perspective, SciQA exhibits a relatively poor vocabulary density: only the much larger DBNQA dataset has comparable ratios. This characteristic of SciQA likely makes memorization more effective, offering a partial explanation for the high performance of the K_0 baseline.

Secondly, while SciQA's Out-of-Vocabulary (OOV) ratio is higher than two other datasets, suggesting a more challenging test split, this observation requires further context. The vocabularies reported in previous tables include both URIs and literals. In SciQA, however, literals significantly outweigh URIs in terms of occurrence, thus Table 5.5 and Table 5.6 also give the vocabulary size and its ratio to the number of samples when only URIs are considered for SciQA. The number of unique URIs is therefore included in parentheses for clarity where applicable.

Moreover, literals frequently appear verbatim in the original questions, including those in the test set. Specifically, literals occur in approximately 90% of the samples. Recalling the example from Section 4.2.1.2, the literal `Depth DDPPO` appears exactly in the question, see Listing 5.3.

```
SQUALL:
Provide a list of papers that have utilized the Depth DDPPO model and include the
↪  links to their code?

SPARQL:
SELECT DISTINCT ?code WHERE {
        ?model a orkgc:Model;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark orkgp:HAS_DATASET ?dataset .
        ?cont orkgp:HAS_BENCHMARK ?benchmark .
        ?cont orkgp:HAS_MODEL ?model;
                orkgp:HAS_SOURCE_CODE ?code .
}
```

Listing 5.3: String literal present in both the question and its equivalent query.

This heavy reliance on literals further reduces the vocabulary's expressiveness and may contribute to the observed performance patterns.

Despite these limitations, incorporating more informative subgraphs during graph prompting — particularly those with higher vertex and edge recall — demonstrates a measurable improvement in performance. While the (1000, 1000, 0.5) parameter configuration does not surpass the K_0 baseline, it still outperforms all other tested configurations. Consequently, this combination was selected for further evaluation of the model's generalization capabilities.

Finally, it is hypothesized that the proposed RAG paradigm would show greater benefits and that the baseline would not remain the best-performing setup if a dataset with a richer and more diverse vocabulary were used. However, the current findings remain inconclusive due to the limitations of the dataset. These limitations neither confirm nor entirely refute the hypothesis, underscoring the need for further investigation.

In summary, while these results provide insights, additional research with richer datasets and varied experimental setups is essential to validate the hypothesis and fully realize the potential of the proposed approach, as outlined in Section 5.8.

5.5 Generalization Capabilities

The practical applicability of the domain expert is now assessed.

In practice, there are typically no ground truth datasets available for systems in MBSE comparable to the SciQA dataset. However, a data generation strategy could be used to create a synthetic dataset instead. Furthermore, no matter the dataset used to train the domain expert, there will always question-query templates it has not seen during its training. For example, a question that needs to be answered by a query with a structure totally different from the seen templates.

The aim of this experiment is to understand how well the domain expert generalizes to realistic, data-scarce, scenarios. Due to the scarcity of data in MBSE, it is crucial to evaluate whether the proposed domain expert performs effectively when trained on synthetic data or when encountering unseen question-query templates.

5.5.1 Hypothesis

It was hypothesized that reducing data quality — either through the use of synthetic data or the exclusion of templates during training — would negatively impact performance. While a basic synthetic data generation strategy was employed, it was not expected to significantly degrade the domain expert. In contrast, template exclusion was anticipated to have a pronounced impact, as supported by prior work.

The hypothesis is that reducing data quality — either through the use of synthetic data or the exclusion of templates during training — would negatively impact performance. While a basic synthetic data generation strategy was applied, it was not anticipated to significantly degrade the domain expert's performance. In contrast, template exclusion was expected to have a more pronounced impact, as suggested by prior work.

These hypotheses rest on two key assumptions.

First, the SQUALL expert — used as the generator component of the domain expert, see Figure 4.2 — is presumed to possess sufficient knowledge of SQUALL. Before training, the domain expert is generally expected to generate syntactically correct SQUALL in most cases, due to the SQUALL expert. The domain expert's training is primarily done to align the domain expert with the target domain such that it can generate SQUALL that is also semantically correct.

Second, LLMs are known to struggle with generalizing to unseen question-query templates [48]. Consequently, the proposed approach is not expected to be immune to this limitation.

In essence, training the domain expert means that it learns how to generate particular query structures from questions: it learns the semantics of the domain. It is unreasonable to think the domain expert would be able to generate a semantically correct query for a question if it has not seen that particular type of question-query structure before.

Given that the synthetic data was found to be of reasonable quality — when examples for all question-query templates were included in the prompt — its impact on performance was not expected to be substantial, see Table 5.2.

5.5.2 Setup

Performance is assessed using BLEU scores, comparing generated SQUALL expressions against their references. There are again two evaluation dimensions:

- **Training Data Type**: Ground truth versus synthetic data.
- **Template Familiarity**: Performance on all templates when template is seen during training versus left out.

Combining these dimensions results in four distinct evaluation setups.

A note regarding the setup involving synthetic data where queries from template 2 were excluded from the training data: the prompt used to generate this synthetic data also omitted any examples from template 2. If examples from template 2 had been included in the prompt, the PLLM could have inadvertently generated SQUALL expressions conforming to the template. This would have resulted in samples with template 2 leaking into the training data of the domain expert.

To reduce the number of experiments, the PCST parameters for all setups are set to (1000, 1000, 0.5). This parameter triple resulted in the best results besides the baseline, see Section 5.4.3.

5.5.3 Results

The experimental results indicate performance degradation when reducing training data quality, see Table 5.7.

For the ground truth case, when template 2 is unseen during training the domain expert expectedly does a much poorer job, underscoring the common challenge of generalizing to unseen question-query templates in text-to-query models. The limited size of the training dataset may be insufficient for generalizing to new questions targeting ORKG [85]. Despite extensive fine-tuning of the SQUALL expert, it remains essential that the training dataset for the domain expert is representative of the target domain.

Templates	**Ground Truth Target Domain Data**		**Synthetic Target Domain Data**	
	T	$T \setminus \{2\}$	T	$T \setminus \{2\}$
T	95	90	78	77
1	95	92	93	85
2	98	<u>40</u>	42	36
3	98	91	85	85
4	90	96	47	65
5	99	99	89	93
6	50	76	35	40
7	94	95	81	64
8	N/A	N/A	N/A	N/A

Table 5.7: Evaluation of the domain expert's ability to generalize on the text-to-SQUALL task using the BLEU score, expressed as percentages, for different training datasets.

The performance of the domain expert when trained on generated data is evidently affected by the relatively lower quality of these synthetic samples.

However, some observations are unexpected:

- **Template 8**: Only one test sample was available, which limits the ability to draw significant conclusions from its performance.
- **Template 6**: The poor performance of template 6 can likely be attributed to the limited number of training samples following this template — 48, which constitutes less than 3% of the total training data. Given that models tend to favor the most frequent query templates in the training data [30], it is reasonable to assume that the infrequency of this template may result in its questions being incorrectly categorized as belonging to other, more common templates.
- **Template 4 for Synthetic Target Domain Data**: The poor performance of template 4 remains unclear. Although the quality of its synthetic data is slightly below average, as shown in Table 5.2, and it scores somewhat worse than average for the ground truth case, it seems unlikely that these factors alone would account for such a significant decline in performance. It is possible that the combination of these issues confuses the domain expert in a manner similar to template 6. To investigate this further, all questions were combined on a per-template basis into a single text, which was then transformed into a vector representation using TF-IDF. Subsequently, the templates were compared pairwise using cosine similarity. Interestingly, the most similar pair of templates was template 4 and template 5. Furthermore, template 5 contains slightly more questions than template 4 — approximately 19% compared to 14%. All these factors

combined may explain why the model occasionally generates queries that follow template 5 when they should actually follow template 4.

- **Template 2 for Synthetic Target Domain Data on T** : The same analysis that was applied to template 4 did not reveal any clear reasons for template 2's poor performance. This suggests that a deeper investigation is required to identify the underlying cause.

5.6 Text-to-SPARQL

The penultimate experiment evaluates the text-to-SPARQL component on the eponymous task.

As illustrated in Figure 4.1, this component takes text as input and outputs SPARQL. It is composed of three subcomponents:

1. The domain expert, which generates an equivalent SQUALL expression of a given question.
2. The translator, which transforms SQUALL to SPARQL.
3. The linker, which maps placeholders to URIs.

It is important to reiterate that the translator component produces SPARQL queries that still contain placeholders. These outputs previously referred to as Placeholder Graph Patterns. Only after all placeholders are replaced with URIs do these PGPs become executable queries, referred to as Graph Patterns.

Although evaluation based solely on the domain expert's output is sufficient for internal comparisons, as discussed in Section 5.4.2.3 — for instance, it allows for assessing the performance of the domain expert under various parameter configurations, such as in the text-to-SQUALL experiment detailed in Section 5.4.3 — translation and linking are vital in order to compare the performance of the proposed approach to that of existing methods in the literature.

The evaluation focuses on the text-to-SPARQL component of the proposed KGQA pipeline, as illustrated in Figure 3.4. Two configurations of the domain expert, identified as the best-performing in the text-to-SQUALL evaluation, are included alongside the baseline (see Table 5.3): K_0, (100, 100, 0.5), and (1000, 1000, 0.5). All models were trained on ground truth data ($\mathscr{D}_{\text{gt}}$).

5.6.1 Hypothesis

No significant differences with the results of the text-to-SQUALL experiment were anticipated. Likely the same, decreasing, order in terms of performances: K_0, (1000, 1000, 0.5), and (100, 100, 0.5).

The linking strategy was implemented from the literature, where it was tested and showed competitive performance [25], hence, there was no reason to expect major differences.

5.6.2 Setup

The text-to-SPARQL component is evaluated using the BLEU score, execution accuracy and F_1 score.

5.6.2.1 Addressing Discrepancies in BLEU Scoring

An important consideration is the choice of references for calculating the BLEU score, as illustrated in Listing 5.4.

This example compares the original Placeholder Graph Pattern (PGP) of a SciQA question to an equivalent SQUALL expression. When the SQUALL expression is translated back into SPARQL, the resulting PGP differs slightly from the original. For instance, a `FILTER` statement may be replaced with an equivalent `BIND` clause. While these differences do not alter the query results, they negatively impact the computed BLEU score.

In the provided example, the BLEU score between the original and the translated PGP is only 73, despite the functional equivalence of the queries.

5.6.2.2 Mitigating the Scoring Discrepancy

Since every output from the text-to-SPARQL component is processed through the translator, comparing it to the original PGPs would systematically penalize the evaluation. This issue is highlighted in Listing 5.4, which also provides an example of a generated SQUALL expression and its translated PGP. Depending on the reference used, the BLEU score can vary significantly — 54 when compared to the original PGP versus 79 when compared to the translated PGP. Notably, the only discrepancy in this example is the generated `contribution` placeholder.

This issue persists even when URIs are used instead of placeholders, as linking does not resolve these discrepancies. Therefore, to ensure a fair evaluation of the text-to-SPARQL component, this effect must be avoided.

5.6.2.3 Creating New References

To address this issue, new reference queries are created using the following process:

1. Create equivalent SQUALL expressions for each SPARQL query.
2. Translate the equivalent SQUALL expressions back to SPARQL using the translator.

This approach ensures that the reference queries have also passed through the translator, just like the outputs of the text-to-SPARQL component. Consequently, the discrepancies described earlier are avoided.

```
Original PGP:
SELECT DISTINCT ?x1 WHERE {
        ?x1 <has_contribution> ?x2 .
        ?x2 rdfs:label ?x3 .
        FILTER(STR(?x3) = 'Test')
}
===
Equivalent SQUALL:
What has a <has_contribution> that has a rdfs:label that is 'Test'?
---
Translated PGP:
SELECT DISTINCT ?x1 WHERE {
        ?x1 <has_contribution> ?x2 .
        ?x2 rdfs:label ?x3 .
        BIND ('Test' AS ?x3)
}
===
Generated SQUALL:
What has a <contribution> that has a rdfs:label that is 'Test'?
---
Generated PGP (i.e., mapped from generated SQUALL):
SELECT DISTINCT ?x1 WHERE {
        ?x1 <contribution> ?x2 .
        ?x2 rdfs:label ?x3 .
        BIND ('Test' AS ?x3)
}
```

Listing 5.4: An example question with its original Placeholder Graph Pattern, as well as an equivalent SQUALL expression, and the Graph Pattern the SQUALL expression has been mapped to by the translator.

The detailed process for creating these reference queries is discussed in Section 4.2.1.2. This method provides a fair and consistent basis for evaluation.

5.6.3 Results

The results of the experiment are presented in Table 5.8. These results confirm the hypothesis: the baseline setup remains the best-performing configuration, as discussed in Section 5.4.3, followed closely by the setup using parameter triple (1000, 1000, 0.5).

Samples for which no SPARQL query was ultimately obtained are excluded from the computation of scores. Instead, these cases are represented by their failure percentages. Failures arise due to several reasons:

- General LLM generation issues, such as repeating tokens excessively or producing incomplete outputs.

Approach	BLEU	Accuracy	F1	Failure
Baseline (K_0)	94.34	86.52	86.52	6.83%
(100, 100, 0.5)	92.76	83.57	83.72	21.81%
(1000, 1000, 0.5)	93.95	85.77	85.78	14.98%

Table 5.8: BLEU, execution accuracy and F_1 scores of different setups on the text-to-SPARQL task. All domains experts were trained using the ground truth domain dataset.

- Inability to extract SQUALL expressions from the generated output, e.g., because the LLM does not follow the expected output format.
- Syntax errors in the extracted SQUALL, rendering it unmappable to SPARQL.
- Partial or complete failure in linking placeholders to URIs, e.g., not finding a URI for the `contribution` placeholder.
- Semantic errors in the linked SPARQL query, which can occur due to:
  - Incorrect linking, i.e., mapping placeholders to erroneous URIs.
  - Incorrect semantics in the generated SQUALL, for example, instead of generating

    `?dataset rdf:type <Dataset>`,

    the LLM might produce

    `<Dataset> rdf:type ?dataset`.

Failures are not included in the computation of scores due to the stringent requirements of the translator, which requires inputs to be syntactically correct, otherwise an error is elicited. This contrasts with approaches that generate SPARQL queries directly, which may contain minor syntax errors, but still be readily usable for evaluation. Since the text-to-SPARQL component produces no output in these cases, alternatives were looked at for evaluation purposes, for example, using a generic fallback query. Ultimately, this avenue was not pursued further, because it was deemed to make the results less interpretable.

Consider the baseline, the results now indicate that for over 90% of the inputted questions the text-to-SPARQL component can output a syntactically correct query. Furthermore, among this successful subset of samples, a

BLEU score of 94.34 and an F_1 score of 86.52 are achieved.

5.7 Comparative Analysis

The results of the proposed approach are compared against a relevant selection from the literature to contextualize its performance.

While previous experiments have demonstrated the effectiveness of the proposed approach, a comparative evaluation is necessary to assess its relative performance.

Such comparisons are critical for validating the architecture's effectiveness under limited data requirements and assessing how it performs compared to approaches operating under ideal data availability conditions.

Two scenarios are considered for this analysis:

Realistic Scenario The proposed approach is compared to Knowledge Graph Question Answering platform (KGQAn) [25], a KGQA platform designed to operate without domain-specific training data. This comparison is particularly relevant, as it represents the most challenging version of the text-to-SPARQL task.

Ideal Scenario The performance of the proposed approach is also contrasted with a recent benchmark on the SciQA dataset [11]. This benchmark evaluated a range of LLM setups, including fine-tuning and prompting strategies. This scenario reflects the easiest version of the text-to-SPARQL task, where models are specifically tailored for the domain using ground truth data.

This dual comparison highlights the strengths and limitations of the proposed approach across diverse conditions. By contrasting it with KGQAn for the most challenging scenario and with state-of-the-art LLM setups for the ideal case, a comprehensive evaluation is achieved.

5.7.1 Hypothesis

First, the KGQAn platform was selected for its alignment with the study's objectives. It is hypothesized that KGQAn would perform competitively on the benchmark, providing a robust point of comparison. However, KGQAn is expected to represent a lower bound on performance for this work, as it does not utilize any domain-specific data.

This expectation is grounded in KGQAn's proven adaptability across a wide range of domains, including general-purpose domains and more specialized ones like DBLP [25].

Second, the SciQA benchmark results [11] provide a complementary perspective. The best results were achieved by fine-tuned LLMs, which the proposed approach is hypothesized not to surpass. These benchmarks are expected to establish an upper bound for performance on SciQA, as they only use domain-specific data.

This hypothesis is grounded in the expectation that specialized training confers a significant advantage for the text-to-SPARQL task, particularly on a dataset like SciQA, which is relatively homogeneous, even across splits (see Section 5.4.3). Additionally, it is possible that LLMs do not truly learn sentence semantics but instead rely on mapping questions to learned query structures [48]. Building on this premise, training an LLM on question-query structures it will not encounter during evaluation could result in greater confusion and, consequently, degraded performance.

5.7.2 Setup

First the approaches are summarized and compared to this work, afterwards the setups are detailed.

5.7.2.1 KGQAn

KGQAn [25] was first introduced in Section 2.1.1, because it seemed an interesting work to compare to due to its similar goals as a framework to assist non-technical stakeholders access KG information. It aims to be universal, and capable of forming queries for any KG, without KG specific training data nor prior information.

It attempts to achieve this by splitting up the text-to-SPARQL task in two steps: a question understanding step and a linking step. The first step is aimed at sentence semantics: given an input question a list of semantic triples is generated — e.g., `(model reporting, is, flexible)` — using a trained LLM. The model was previously trained on a QA dataset similar SciQA, however, one made domain agnostic by replacing URIs with placeholders. The second step then attempts to construct a query from the generated list using heuristics, for example, building an `ASK` query if the question starts with `is`. Subsequently, the placeholders are mapped to URIs from the target KG, using a dynamic Just-In-Time (JIT) linker detailed in the linking section (see Section 4.1.1.9). Essentially, it queries the KG using string matching based on the placeholders.

A key distinction between KGQAn and the proposed approach lies in the method of query structure generation. Unlike this work, which leverages an LLM to generate query structures — i.e., the domain expert — KGQAn employs heuristics to construct them from a list of semantic triples. So, while both approaches utilize an intermediary representation, they are distinctly different, namely SQUALL expressions in this work versus semantic triples in KGQAn) Furthermore, the assumptions about data availability differ. KGQAn operates under stricter data constraints, as it does not utilize any domain-specific data, in contrast to the data-scarce setting assumed for this study.

Still, KGQAn remains a viable benchmark due to the significant overlap in the intended audience and the relatively similar data constraints when compared to other methods in the literature.

The parameters for the linker in both the proposed architecture and KGQAn were configured as follows:

- The maximum number of vertices and edges returned from a probing query was set to 50 and 1 000, respectively.
- After scoring, only the top 10 vertices or edges were retained.
- Only the most likely final answer was selected.

5.7.2.2 Benchmarks

Recent work [11] has established the standard for the text-to-SPARQL task on the SciQA dataset using a diverse array of LLMs (e.g., T5, GPT-3.5-turbo) and strategies such as ZSL, Few-Shot Learning (FSL), and fine-tuning.

The primary aim of [11] was to demonstrate how LLM fine-tuning and FSL approaches can achieve strong performance on the challenging SciQA benchmark. However, this work was not conducted in the context of data scarcity or with a focus on generalizability. The methodology employed a straightforward supervised training approach for fine-tuning and relied on prompt optimization techniques for FSL.

In contrast, the approach in this study addresses data scarcity through techniques such as incorporating non-parametric memory within the domain expert. Additionally, generalizability is emphasized by fine-tuning the SQUALL expert specifically for the text-to-SPARQL task and utilizing it as a generator for the domain expert.

Various prompt engineering strategies were explored, with the most effective being a similarity-based sampling approach. This method involved selecting the seven most semantically similar questions as examples and including them in the prompt alongside their corresponding SPARQL queries.

5.7.3 Results

Finally, the results indicate that the proposed approach is competitive especially considering it is built for data-scarce conditions.

5.7.3.1 KGQAn

Contrary to expectations, the platform significantly underperformed. The results, presented here for completeness, indicate that 54% of the test questions failed at generation, i.e., Question Understanding (QU) did not produce a valid list of triples. For the remaining queries, the BLEU score was 4.42, with both accuracy and F_1 effectively zero.

Setup	BLEU
T5-base, fine-tuned	96.31
GPT2-large, fine-tuned	95.04
Dolly-v2-3b, similarity-based FSL	80.15
GPT-3.5-turbo, similarity-based FSL	95.71

Table 5.9: BLEU scores set on the SciQA benchmark by different LLMs [11].

To investigate these unexpected outcomes, the platform's source code and results were examined further. When QU succeeded in generating queries, the outputs were overly generic. While some correct answers were occasionally retrieved during query execution, they were vastly outnumbered by incorrect results. Two critical features were identified as the root causes, severely limiting the platform's generalizability:

- **Question Understanding:**

  Generating triples instead of complete queries using an LLM does not circumvent the template problem [48]. The challenge of producing the correct structure remains significant: outputting a list of triples rather than a complete query does not fundamentally resolve this issue.

- **Query Building:**

  It became clear that its query building approach based on heuristics does not generalize well to questions expected to be answered through CGPs such as those from SciQA, a challenging benchmark [11].

5.7.3.2 Benchmark

The main results are summarized in Table 5.9. For each model, the strategy that yielded the best performance was selected.

Observations on Model Performance The fine-tuned T5 model and the FSL GPT-3.5-turbo configurations, in particular, showcase outstanding performance, setting a high standard for comparison. However, the approach proposed in this work demonstrates results that are only marginally lower than the benchmarks, and are thus considered competitive, see Table 5.8.

Critical Evaluation However, an essential critique arises from [48]:

> The handling of unknown question-query structures is particularly important because it is unclear if models really generate queries based on sentence semantics, or if they simply map them to known query structures.

With enough data, it seems plausible that any sufficiently powerful LLM could generate correct queries for a given question, provided it has encountered/encounters the corresponding question-query template during training/in its prompt.

Additional Insights The analysis in [11] provides several key points:

1. **Entity Hallucination**: Entity hallucination, likely influenced by pre-training data, occurs when models hallucinate entities from external knowledge sources like Wikidata. This aligns with earlier identified concerns in this work (see Section 4.1.2.2).
2. **Syntactical Errors**: The fine-tuned T5 model exhibits a significantly higher rate of syntactical errors (40.0%) compared to FSL GPT-3.5-turbo (5.2%), indicating weaker knowledge of SPARQL. Likely causes include T5's smaller size and differences in pre-training data and tasks.
3. **Semantic Errors**: Frequent errors include misspelled entities and incorrect predicates.

Mitigation in the Proposed Approach The proposed approach preemptively attempted to address these issues:

1. By using placeholders and subsequently linking.
2. By employing the SQUALL expert as the generator for the domain expert, and freezing it during the latter's training phase.
3. By leveraging RAG.

Template Generalization and Prompting Strategies Finally, one of the tested prompting strategies involved randomly sampling examples without regard to query templates. This universally resulted in poorer performance, highlighting a recurring challenge [30]: generalizing to unseen templates. The efficacy of FSL heavily depends on well-chosen examples. It appears unlikely that FSL approaches would succeed with entirely unseen question-query structures [48].

5.8 Discussion

This section provides a comprehensive analysis of the experimental results, offering insights into their broader implications. It connects the findings to existing literature, evaluates their alignment with the stated requirements, and demonstrates how model reporting is implemented.

Throughout this work, numerous decisions were made to address data scarcity, aiming to design an architecture suitable for MBSE. The experiments were structured to explore the potential of the proposed approach in achieving effective model reporting. Key design choices included restating the text-to-SPARQL task as text-to-SQUALL task, fine-tuning a robust LLM to develop a general understanding of SQUALL, integrating this fine-tuned LLM into a trainable RAG architecture, and investigating the model's adaptability to new domains using corresponding (synthetic) data. These choices all had the common goal of realizing robust Domain Adaptation from a data rich source domain to a specialized (MBSE) domain.

5.8.1 Challenges in Generalization to New Questions

Previous work [48] shows that NMT approaches typically struggle with unseen question-query structures. During training, LLMs are often exposed to questions that follow specific structures, a pattern evident in the datasets that are widely used, such as LC-QuAD 2.0, and present in SciQA as well. Consequently, the corresponding queries adhere to certain templates, making it difficult for LLMs to generate correct queries for questions requiring novel query structures.

While the proposed approach is not without limitations, as discussed in Section 5.5, the experiments suggest significant potential for further development. Listing 4.7 illustrates the notable differences between queries in the original and target domains. Enhancing the diversity of training data during the fine-tuning of the SQUALL expert is both feasible and primarily a matter of data engineering. Such improvements could bolster the domain expert's ability to generalize to question-query structures not encountered during the second training phase. Additionally, employing more advanced synthetic data generation techniques could further expand the variety and representativeness of the training dataset, enhancing the model's robustness and adaptability.

5.8.2 Adaptation to New Knowledge Graphs and Domains

The SQUALL expert was fine-tuned on LC-QuAD 2.0, then frozen and integrated into a RAG architecture (see Section 4.1.1), resulting in the domain expert. That model was subsequently trained on SciQA, a dataset tailored for ORKG rather than Wikidata. This training strategy (see Section 4.2.3) on two distinct domains provides valuable insight into the domain adaptation, or more broadly, generalization capabilities of the proposed implementation. As stated above, despite the significant differences between the SPARQL templates used in LC-QuAD 2.0 and SciQA the domain expert exhibited the ability to adapt to the new KG/domain using relatively little synthetic data of a lower than ground truth quality.

These results further corroborate the notion that SQUALL can effectively reduce data requirements. Catastrophic forgetting the SQUALL expert is likely avoided of by keeping it frozen. The use of placeholders enables reusing the SQUALL expert for a new domain, it makes possible this domain adaptation approach . Although deep error analysis is necessary to get good concrete empirical evidence.

5.8.3 Adaptation to New Knowledge Graphs and Domains

The SQUALL expert was fine-tuned on LC-QuAD 2.0, subsequently frozen, and integrated into a RAG architecture (see Section 4.1.1), resulting in the domain expert. This model was then trained on SciQA, a dataset specifically designed for ORKG rather than Wikidata. This dual-domain training strategy (see Section 4.2.3) offers valuable insights into the domain adaptation and generalization capabilities of the proposed implementation.

Despite the significant differences between the SPARQL templates used in LC-QuAD 2.0 and SciQA, the domain expert demonstrated the ability to adapt effectively to the new KG/domain using relatively limited synthetic data, even when the data was of lower quality than ground truth. These results support the potential of SQUALL to reduce data requirements effectively.

Freezing the SQUALL expert likely mitigates catastrophic forgetting, ensuring that the gained SQUALL knowledge is retained. Moreover, the use of placeholders facilitates the reuse of the SQUALL expert across domains, enabling the proposed approach to domain adaptation. While these findings are promising, a detailed error analysis is necessary to provide robust empirical evidence and uncover potential areas for refinement.

5.8.4 Ambiguity Related to RAG

The effectiveness of using RAG in the proposed framework remains inconclusive. While some improvement was observed with higher recall of constructed subgraphs, the baseline approach — where no subgraph is provided during generation — still outperformed the other setups (see Table 5.3). This outcome is likely due to the homogeneous nature of the SciQA dataset (see Section 5.4.3), which may limit the potential advantages of RAG. In such cases, the baseline approach likely leads to memorization — due to the retrieved data being highly similar across samples, — outperforming actual learning to utilize retrieved data, which the other setups do.

Further investigation is required to confirm or rule out the hypothesized benefits of RAG. Future work could explore its performance across more diverse datasets to provide a definitive assessment of its utility within this context.

5.8.5 Feasibility of Using Synthetic Data

It is noteworthy that the majority of SciQA's samples are synthetically generated [10]. Consequently, the experimental results — including those using ground truth data — support the idea that data scarcity can be effectively addressed through the use of synthetic data. Employing advanced techniques to collect and generate question-query templates tailored to specific domains could be critical to model reporting.

5.8.6 Potential for Model Reporting and Data Transformation

SciQA queries are notably complex [11], often surpassing those in datasets like LC-QuAD in several aspects. They frequently involve a high number of triples and true CGPs, which are combinations of BGPs extended with relational operations [46], such as filters and unions. While LC-QuAD queries include features like filters, limits, and basic ordering, they lack the advanced constructs found in SciQA, such as aggregation, grouping, optionals, and nested queries.

Despite this complexity, the domain expert has demonstrated the ability to effectively handle CGPs. Current transformations present in the SPARQL queries include solution modifiers and grouping (see Listing 8). Importantly, SQUALL's comprehensive coverage of the SPARQL 1.1 standard, including graph updates [13], offers significant potential for extending these capabilities. This expansion could support and automate more phases of the model reporting pipeline, including advanced transformations.

Achieving this advancement would require high-quality data for fine-tuning, along with significantly more diverse (synthetic) training data. Such improvements represent a critical step toward enabling more robust transformations within the model reporting pipeline. An overview of the complex patterns achievable by the current approach is provided in Section 5.18, showcasing examples of optional clauses, filters, solution modifiers (e.g., max, limit, order by) aggregation, and grouping.

Conclusion

5.9 Introduction

The primary goal of this work was to present and realize a more flexible approach to reporting in MBSE. To this end, a novel model reporting paradigm was conceptualized. This approach was realized through the development of a proof-of-concept implementation, Mo-Lab, which leverages an innovative architecture and training techniques, addressing the principal challenge, namely data scarcity, that hinders the use of neural models for specialized domains such as those found in MBSE. The insights from this work could guide future efforts to integrate contemporary NLP techniques into MBSE workflows.

5.10 Research Questions and Answers

The research questions outlined in Chapter 1 are reiterated and addressed below.

RQ1: How does the performance of the proposed domain expert compare to other approaches across various data availability scenarios?

In data-rich scenarios, the domain expert, in its baseline setup, performs comparably to state-of-the-art models specifically trained on the benchmark, with only slight performance differences. In data-poor scenarios, the domain expert significantly outperforms KGQAn, a comparable approach. Although direct results for text-to-SPARQL using synthetic data are not available, it can be inferred that the performance would surpass KGQAn, given the minimal performance degradation observed in the text-to-SQUALL task, — when going from ground truth to synthetic data, — and the linking step, — when associating placeholders with URIs.

RQ2: Does the domain expert effectively utilize the retrieved information with which it is soft-prompted, and what factors influence the effectiveness of these prompts?

Due to benchmark limitations, it is inconclusive whether the domain expert fully leverages the soft prompts. However, assuming their efficacy, more relevant constructed graphs with higher vertex and edge recall are likely to enhance prompt effectiveness, and improve generation.

RQ3: What is the impact of using synthetic data, as opposed to ground truth data, on domain adaptation performance during the second learning phase?

Synthetic data negatively impacts performance due to its lower quality and lack of representativeness. For instance, performance is poor on templates not included in the synthetic data. This challenge is consistent with similar approaches, as performance on unseen templates remains an open problem in the field. However, it should be noted that the ground truth data, albeit of a much higher quality than the synthetic data, was essentially also generated, indicating the importance of the quality of the generated data used during the second learning phase.

5.11 Synthesis and Significance

The central motivation of this research stems from the desire to make reporting in MBSE more flexible than is currently possible due to the challenge of data scarcity, which poses a significant barrier to the adoption of contemporary NLP techniques in MBSE. Addressing this challenge required overcoming the reliance on extensive, high-quality datasets or the presupposition of domain familiarity by PLLMs. This work contributes to the field by:

1. Introducing a novel paradigm for model reporting in MBSE.
2. Developing an implementation that addresses data scarcity through innovative architectural and training solutions.
3. Presenting the domain expert, a RAG model whose architecture and training procedures could inspire future research in applying NLP techniques to MBSE.

These contributions highlight the potential of NLP advancements in empowering practitioners in MBSE, demonstrating practical applications of cutting-edge technology.

5.12 Limitations

Despite its contributions, this work evidently has certain limitations:

- The agent is scoped to a Question Answering (QA) pipeline, limiting its generality.
- Results are restricted to illustrative post-processing, such as natural language explanations, tabulations, and visualizations via an external engine.
- Poor performance on unseen question-query templates, a challenge shared by similar approaches, limits its practical applicability.
- No conclusive evidence was gathered on the efficacy of the RAG mechanism.

Section 5.13 outlines how these limitations can be addressed in future research, offering multiple interesting avenues.

5.13 Closing Thoughts

This work aspires to inspire practitioners in MBSE by showcasing the potential of recent advancements in the NLP field. It is hoped that the insights, methodologies, and findings presented here will serve as a foundation for further exploration, contributing to the evolution of reporting in MBSE towards more flexibility and interactivity through AI.

Future Work

This chapter outlines potential future research directions inspired by this work. These avenues are categorized into five key areas: AI agent, Domain Adaptation, Retrieval-Augmented Generation, synthetic data generation, and linking.

5.14 AI Agent

Mo-Lab, as a proof-of-concept implementation of the proposed model reporting paradigm, is currently limited in its capabilities. Future research could explore incorporating the domain expert into a fully-fledged AI agent, enabling interactive features such as dynamic feedback through a dialog between the stakeholder and the agent. This enhancement could facilitate iterative refinement of results, improving both usability and accuracy. Additionally, such an agent could respond to stakeholder requests in terms of analysis preferences, visualization formats, and other personalized requirements.

The proposed text-to-SPARQL pipeline, which includes the domain expert, could serve as a specialized subcomponent of the agent, focusing on text-to-SPARQL tasks. Meanwhile, the general capabilities of a PLLM, such as ChatGPT, could be retained to handle broader interactions, ensuring a versatile and comprehensive AI agent.

5.15 Domain Adaptation

A promising direction for future research is enriching the source domain data used to train the SQUALL expert. Since DA can be applied to a mixed source domain [86], incorporating multiple QA datasets (e.g., LC-QuAD 2.0, DBLP-QuAD, QALD-9-Plus, etc.) could enhance generalizability, improve practical applicability, and reduce training data requirements for the domain expert in the target domain.

5.16 Retrieval Augmented Generation

Evaluating the proposed approach on a diverse set of more varies and less homogeneous benchmarks is necessary, as the current experimentation could not yield conclusive results. For instance, replacing DBLP and DBLP-QuAD with ORKG and SciQA could provide valuable insights. While DBLP was initially considered but abandoned due to its size, revisiting it now can be justified given the promising results achieved.

5.17 Synthetic Data Generation

Further exploration of synthetic data generation techniques presents another exciting research avenue. Investigating methods to produce data that is more faithful, extensive, and varied across the domain could significantly enhance domain expert training. Additionally, this work focused on generating queries while assuming the availability of questions. Future research could explore techniques for simultaneously generating both queries and questions. Developing a framework for directly generating data from an ontology or KG of a domain could also be valuable, particularly if it is practically applicable across real-world MBSE projects.

5.18 Linking

The linking component utilized in this work is a component separate from the remainder of the domain expert, and it is non-neural. Exploring more advanced linkers that could leverage the constructed subgraph output during post-retrieval, might offer significant improvements. Neural approaches applicable in data-scarce scenarios, potentially through prompt engineering or DA, represent another promising direction for future work.

References

[1] M. Mitchell, S. Wu, A. Zaldivar, *et al.*, "Model Cards for Model Reporting," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, Atlanta GA USA: ACM, Jan. 2019, pp. 220–229, ISBN: 978-1-4503-6125-5. DOI: 10.1145/3287560.3287596.

[2] T. Gebru, J. Morgenstern, B. Vecchione, *et al.*, "Datasheets for datasets," *Communications of the ACM*, vol. 64, no. 12, pp. 86–92, Dec. 2021, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3458723.

[3] M. Elaasar, N. Rouquette, D. Wagner, B. J. Oakes, A. Hamou-Lhadj, and M. Hamdaqa, "openCAESAR: Balancing Agility and Rigor in Model-Based Systems Engineering," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Västerås, Sweden: IEEE, Oct. 2023, pp. 221–230, ISBN: 9798350324983. DOI: 10.1109/MODELS-C59198.2023.00051.

[4] D. Wagner, S. Y. Kim-Castet, A. Jimenez, M. Elaasar, N. Rouquette, and S. Jenkins, "CAESAR Model-Based Approach to Harness Design," in *2020 IEEE Aerospace Conference*, Big Sky, MT, USA: IEEE, Mar. 2020, pp. 1–13, ISBN: 978-1-72812-734-7. DOI: 10.1109/AERO47225.2020.9172630.

[5] K. Henderson and A. Salado, "Value and benefits of model-based systems engineering (MBSE): Evidence from the literature," *Systems Engineering*, vol. 24, no. 1, pp. 51–66, 2021, ISSN: 1520-6858. DOI: 10.1002/sys.21566.

[6] J. Höfgen, B. Vogel-Heuser, A. Vicaria, F. Pouzolz, and C. Kurzhals, "Enhancing Model-Based System Architecting Through Formalized Decision Management," in *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*, Aug. 2024, pp. 1053–1060. DOI: 10.1109/CASE59546.2024.10711329.

[7] M. Lutfi and R. Valerdi, "Integration of SysML and Virtual Reality Environment: A Ground Based Telescope System Example," *Systems*, vol. 11, no. 4, p. 189, Apr. 2023, ISSN: 2079-8954. DOI: 10.3390/systems11040189.

[8] K. Trase and E. Fink, "A model-driven visualization tool for use with Model-Based Systems Engineering projects," in *2014 IEEE Aerospace Conference*, Mar. 2014, pp. 1–10. DOI: 10.1109/AERO.2014.6836268.

[9] M. Bialy, V. Pantelic, J. Jaskolka, *et al.*, "Software Engineering for Model-Based Development by Domain Experts," in *Handbook of System Safety and Security*, Elsevier, 2017, pp. 39–64, ISBN: 978-0-12-803773-7. DOI: 10.1016/B978-0-12-803773-7.00003-6.

[10] S. Auer, D. A. C. Barone, C. Bartz, *et al.*, "The SciQA Scientific Question Answering Benchmark for Scholarly Knowledge," *Scientific Reports*, vol. 13, no. 1, p. 7240, May 2023, ISSN: 2045-2322. DOI: 10.1038/s41598-023-33607-z.

[11] J. Lehmann, A. Meloni, E. Motta, *et al.*, "Large Language Models for Scientific Question Answering: An Extensive Analysis of the SciQA Benchmark," in *The Semantic Web*, A. Meroño Peñuela, A. Dimou, R. Troncy, *et al.*, Eds., vol. 14664, Cham: Springer Nature Switzerland, 2024, pp. 199–217, ISBN: 978-3-031-60626-7. DOI: 10.1007/978-3-031-60626-7_11.

[12] J. Lehmann, P. Gattogi, D. Bhandiwad, S. Ferré, and S. Vahdati, "Language Models as Controlled Natural Language Semantic Parsers for Knowledge Graph Question Answering," in *Frontiers in Artificial Intelligence*

and Applications, K. Gal, A. Nowé, G. J. Nalepa, R. Fairstein, and R. Rădulescu, Eds., Krakow (Cracovie), Poland: IOS Press, Sep. 2023, pp. 1348–1356, ISBN: 978-1-64368-437-6. DOI: 10.3233/FAIA230411.

[13] S. Ferré, “SQUALL: The expressiveness of SPARQL 1.1 made available as a controlled natural language,” *Data & Knowledge Engineering*, vol. 94, pp. 163–188, Nov. 2014, ISSN: 0169023X. DOI: 10.1016/j.datak.2014.07.010.

[14] C. Delp, D. Lam, E. Fosse, and Cin-Young Lee, “Model based document and report generation for systems engineering,” in *2013 IEEE Aerospace Conference*, Big Sky, MT: IEEE, Mar. 2013, pp. 1–11, ISBN: 978-1-4673-1813-6. DOI: 10.1109/AERO.2013.6496926.

[15] Z. Li, “Semantic Parsing in Limited Resource Conditions,” Ph.D. dissertation, Sep. 2023. DOI: 10 . 26180 / 24083265.V1.

[16] H. Abdel-Nabi, A. Awajan, and M. Z. Ali, “Deep learning-based question answering: A survey,” *Knowledge and Information Systems*, vol. 65, no. 4, pp. 1399–1485, Apr. 2023, ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-022-01783-5.

[17] W.-t. Yih, M.-W. Chang, X. He, and J. Gao, “Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base,” in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong and M. Strube, Eds., Beijing, China: Association for Computational Linguistics, 2015, pp. 1321–1331. DOI: 10.3115/v1/P15-1128.

[18] S. Reddy, M. Lapata, and M. Steedman, “Large-scale Semantic Parsing without Question-Answer Pairs,” *Transactions of the Association for Computational Linguistics*, vol. 2, D. Lin, M. Collins, and L. Lee, Eds., pp. 377–392, 2014. DOI: 10.1162/tacl_a_00190.

[19] Y. W. Wong and R. Mooney, “Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus,” in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, A. Zaenen and A. van den Bosch, Eds., Prague, Czech Republic: Association for Computational Linguistics, Jun. 2007, pp. 960–967.

[20] Y. Sha, Y. Feng, M. He, S. Liu, and Y. Ji, “Retrieval-Augmented Knowledge Graph Reasoning for Commonsense Question Answering,” *Mathematics*, vol. 11, no. 15, p. 3269, Jul. 2023, ISSN: 2227-7390. DOI: 10.3390/math11153269.

[21] X. Chen, P. Gao, J. Song, and X. Tan, *HiQA: A Hierarchical Contextual Augmentation RAG for Multi-Documents QA*, Sep. 2024. DOI: 10.48550/arXiv.2402.01767.

[22] S. J. Pan and Q. Yang, “A Survey on Transfer Learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010, ISSN: 1558-2191. DOI: 10.1109/TKDE.2009.191.

[23] Z. Li, Y. Lai, Y. Feng, and D. Zhao, “Domain Adaptation for Semantic Parsing,” in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, Yokohama, Japan: International Joint Conferences on Artificial Intelligence Organization, Jul. 2020, pp. 3723–3729, ISBN: 978-0-9992411-6-5. DOI: 10.24963/ijcai.2020/515.

[24] P. Jiang and X. Cai, “A Survey of Semantic Parsing Techniques,” *Symmetry*, vol. 16, no. 9, p. 1201, Sep. 2024, ISSN: 2073-8994. DOI: 10.3390/sym16091201.

[25] R. Omar, I. Dhall, P. Kalnis, and E. Mansour, "A Universal Question-Answering Platform for Knowledge Graphs," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–25, May 2023, ISSN: 2836-6573. DOI: 10.1145/3588911.

[26] Y. Su and X. Yan, "Cross-domain Semantic Parsing via Paraphrasing," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, Copenhagen, Denmark: Association for Computational Linguistics, 2017, pp. 1235–1246. DOI: 10.18653/v1/D17-1127.

[27] A. Ray, Y. Shen, and H. Jin, "Fast Domain Adaptation of Semantic Parsers via Paraphrase Attention," in *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, C. Cherry, G. Durrett, G. Foster, *et al.*, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 94–103. DOI: 10.18653/v1/D19-6111.

[28] Y. Wang, J. Berant, and P. Liang, "Building a Semantic Parser Overnight," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Beijing, China: Association for Computational Linguistics, 2015, pp. 1332–1342. DOI: 10.3115/v1/P15-1129.

[29] T. Kuhn, "A Survey and Classification of Controlled Natural Languages," *Computational Linguistics*, vol. 40, no. 1, pp. 121–170, Mar. 2014, ISSN: 0891-2017, 1530-9312. DOI: 10.1162/COLI_a_00168.

[30] P. A. K. K. Diallo, S. Reyd, and A. Zouaq, "A Comprehensive Evaluation of Neural SPARQL Query Generation From Natural Language Questions," *IEEE Access*, vol. 12, pp. 125 057–125 078, Sep. 2024, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3453215.

[31] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, Red Hook, NY, USA: Curran Associates Inc., Dec. 2020, pp. 9459–9474, ISBN: 978-1-71382-954-6.

[32] Y. Gao, Y. Xiong, X. Gao, *et al.*, *Retrieval-Augmented Generation for Large Language Models: A Survey*, Jan. 2024. DOI: 10.48550/arXiv.2312.10997.

[33] L. Huang, W. Yu, W. Ma, *et al.*, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," *ACM Transactions on Information Systems*, p. 3 703 155, Nov. 2024, ISSN: 1046-8188, 1558-2868. DOI: 10.1145/3703155.

[34] H. Li, Y. Su, D. Cai, Y. Wang, and L. Liu, *A Survey on Retrieval-Augmented Text Generation*, Feb. 2022. DOI: 10.48550/arXiv.2202.01110.

[35] V. Ravi, S. Amrouni, A. Stefanucci, A. Nourbakhsh, P. Reddy, and M. Veloso, *DocuBot : Generating financial reports using natural language interactions*, Feb. 2021. DOI: 10.48550/arXiv.2010.01169.

[36] S. Biswal, C. Xiao, L. M. Glass, M. B. Westover, and J. Sun, *CLARA: Clinical Report Auto-completion*, Mar. 2020. DOI: 10.48550/arXiv.2002.11701.

[37] P. Messina, P. Pino, D. Parra, *et al.*, "A Survey on Deep Learning and Explainability for Automatic Report Generation from Medical Images," *ACM Computing Surveys*, vol. 54, no. 10s, pp. 1–40, Jan. 2022, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3522747.

[38] M. Agarwal, T. Chakraborti, Q. Fu, *et al.*, "NeurIPS 2020 NLC2CMD Competition: Translating Natural Language to Bash Commands," in *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, PMLR, Aug. 2021, pp. 302–324.

[39] F. Wang, X. Liu, O. Liu, *et al.*, "Slide4N: Creating Presentation Slides from Computational Notebooks with Human-AI Collaboration," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, Hamburg Germany: ACM, Apr. 2023, pp. 1–18, ISBN: 978-1-4503-9421-5. DOI: 10.1145/3544548.3580753.

[40] S. R. Joshi, B. Venkatesh, D. Thomas, Y. Jiao, and S. Roy, "A Natural Language and Interactive End-to-End Querying and Reporting System," in *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, Hyderabad India: ACM, Jan. 2020, pp. 261–267, ISBN: 978-1-4503-7738-6. DOI: 10.1145/3371158.3371198.

[41] C. Zhang, Y. Mao, Y. Fan, *et al.*, *FinSQL: Model-Agnostic LLMs-based Text-to-SQL Framework for Financial Analysis*, Jan. 2024. DOI: 10.48550/arXiv.2401.10506.

[42] Y. Peng, S. Lin, Q. Chen, *et al.*, *ChatGraph: Chat with Your Graphs*, Jan. 2024. DOI: 10.48550/arXiv.2401.12672.

[43] X. He, Yijun Tian, Yifei Sun, *et al.*, *G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering*, Feb. 2024. DOI: 10.48550/arXiv.2402.07630.

[44] S. Siriwardhana, R. Weerasekera, E. Wen, T. Kaluarachchi, R. Rana, and S. Nanayakkara, "Improving the Domain Adaptation of Retrieval Augmented Generation (RAG) Models for Open Domain Question Answering," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1–17, Jan. 2023, ISSN: 2307-387X. DOI: 10.1162/tacl_a_00530.

[45] E. Simperl, M. Mochol, T. Bürger, and I. O. Popov, "Achieving Maturity: The State of Practice in Ontology Engineering in 2009," in *On the Move to Meaningful Internet Systems: OTM 2009*, D. Hutchison, T. Kanade, J. Kittler, *et al.*, Eds., vol. 5871, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 983–991, ISBN: 978-3-642-05151-7. DOI: 10.1007/978-3-642-05151-7_17.

[46] R. Angles, M. Arenas, P. Barcelo, A. Hogan, J. Reutter, and D. Vrgoc, *Foundations of Modern Query Languages for Graph Databases*, Jun. 2017. DOI: 10.48550/arXiv.1610.06264.

[47] H. Raissya, F. Darari, and F. J. Ekaputra, "VizKG: A Framework for Visualizing SPARQL Query Results over Knowledge Graphs," *CEUR Workshop Proceedings*, vol. 3023, pp. 95–102, 2021, ISSN: 1613-0073.

[48] S. Reyd and A. Zouaq, "Assessing the Generalization Capabilities of Neural Machine Translation Models for SPARQL Query Generation," in *The Semantic Web – ISWC 2023: 22nd International Semantic Web Conference, Athens, Greece, November 6–10, 2023, Proceedings, Part I*, Berlin, Heidelberg: Springer-Verlag, Nov. 2023, pp. 484–501, ISBN: 978-3-031-47239-8. DOI: 10.1007/978-3-031-47240-4_26.

[49] C. Zheng, D. Wang, A. Y. Wang, and X. Ma, "Telling Stories from Computational Notebooks: AI-Assisted Presentation Slides Creation for Presenting Data Science Work," in *CHI Conference on Human Factors in Computing Systems*, Apr. 2022, pp. 1–20. DOI: 10.1145/3491102.3517615.

[50] J. Heer, "Agency plus automation: Designing artificial intelligence into interactive systems," *Proceedings of the National Academy of Sciences*, vol. 116, no. 6, pp. 1844–1850, Feb. 2019, ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1807184115.

[51] W. Fan, Y. Ding, L. Ning, *et al.*, *A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models*, Jun. 2024. DOI: 10.48550/arXiv.2405.06211.

[52] Z. Liu, X. He, Y. Tian, and N. V. Chawla, “Can we Soft Prompt LLMs for Graph Learning Tasks?” In *Companion Proceedings of the ACM on Web Conference 2024*, May 2024, pp. 481–484. DOI: 10.1145/3589335.3651476.

[53] A. Hogan, E. Blomqvist, M. Cochez, *et al.*, “Knowledge Graphs,” *ACM Computing Surveys*, vol. 54, no. 4, pp. 1–37, May 2022, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3447772.

[54] L. Zou and M. T. Özsu, “Graph-Based RDF Data Management,” *Data Science and Engineering*, vol. 2, pp. 1–15, Mar. 2017. DOI: 10.1007/s41019-016-0029-6.

[55] H. Thakkar, D. Punjani, S. Auer, and M.-E. Vidal, *Towards an Integrated Graph Algebra for Graph Pattern Matching with Gremlin (Extended Version)*, Sep. 2019. DOI: 10.48550/arXiv.1908.06265.

[56] Likin C Simon Romero, “Uniqueness of the Hyperspace Graph of Connected Subgraphs,” *Topology Proceedings*, vol. 31, pp. 283–294, 2007.

[57] Likin C Simon Romero, “The hyperspace graph of connected subgraphs,” Ph.D. dissertation, West Virginia University Libraries, Aug. 2005. DOI: 10.33915/etd.2321.

[58] A. Schätzle, M. Przyjaciel-Zablocki, S. Skilevic, and G. Lausen, *S2RDF: RDF Querying with SPARQL on Spark*, Jan. 2016. DOI: 10.48550/arXiv.1512.07021.

[59] S. Saha, P. Yadav, L. Bauer, and M. Bansal, “ExplaGraphs: An Explanation Graph Generation Task for Structured Commonsense Reasoning,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021, pp. 7716–7740. DOI: 10.18653/v1/2021.emnlp-main.609.

[60] D. A. Hudson and C. D. Manning, *GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering*, May 2019. DOI: 10.48550/arXiv.1902.09506.

[61] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang, and J. Suh, “The Value of Semantic Parse Labeling for Knowledge Base Question Answering,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, K. Erk and N. A. Smith, Eds., Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 201–206. DOI: 10.18653/v1/P16-2033.

[62] L. Luo, Y.-F. Li, G. Haffari, and S. Pan, *Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning*, Feb. 2024. DOI: 10.48550/arXiv.2310.01061.

[63] S. Auer, D. A. C. Barone, C. Bartz, *et al.*, *SciQA benchmark: Dataset and RDF dump*, Mar. 2023. DOI: 10.5281/ZENODO.7729047.

[64] S. Ferré, “Squall2sparql: A Translator from Controlled English to Full SPARQL 1.1,” in *CEUR Workshop Proceedings*, vol. 1179, Valencia, Spain, Sep. 2013.

[65] J. C. Rangel, Mendes de Farias, A. C. Sima, and N. Kobayashi, *SPARQL Generation: An analysis on fine-tuning OpenLLaMA for Question Answering over a Life Science Knowledge Graph*, Feb. 2024. DOI: 10.48550/arXiv.2402.04627.

[66] P. Zhao, H. Zhang, Q. Yu, *et al.*, *Retrieval-Augmented Generation for AI-Generated Content: A Survey*, Jun. 2024. DOI: 10.48550/arXiv.2402.19473.

[67] T. Ayoola, S. Tyagi, J. Fisher, C. Christodoulopoulos, and A. Pierleoni, *ReFinED: An Efficient Zero-shot-capable Approach to End-to-End Entity Linking*, Jul. 2022. DOI: 10.48550/arXiv.2207.04108.

[68] M. Dubey, D. Banerjee, A. Abdelkawi, and J. Lehmann, "LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia," in *The Semantic Web – ISWC 2019*, C. Ghidini, O. Hartig, M. Maleshkova, *et al.*, Eds., vol. 11779, Cham: Springer International Publishing, 2019, pp. 69–78, ISBN: 978-3-030-30796-7. DOI: 10.1007/978-3-030-30796-7_5.

[69] L. Long, R. Wang, R. Xiao, *et al.*, *On LLMs-Driven Synthetic Data Generation, Curation, and Evaluation: A Survey*, Jun. 2024. DOI: 10.48550/arXiv.2406.15126.

[70] J. Zhang, X. Zhang, J. Yu, *et al.*, "Subgraph Retrieval Enhanced Model for Multi-hop Knowledge Base Question Answering," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 5773–5784. DOI: 10.18653/v1/2022.acl-long.396.

[71] D. S. Sachan, M. Lewis, D. Yogatama, L. Zettlemoyer, J. Pineau, and M. Zaheer, "Questions Are All You Need to Train a Dense Passage Retriever," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 600–616, Jun. 2023, ISSN: 2307-387X. DOI: 10.1162/tacl_a_00564.

[72] V. Zhong, C. Xiong, and R. Socher, *Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning*, Nov. 2017. DOI: 10.48550/arXiv.1709.00103.

[73] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao, "gStore: Answering SPARQL queries via subgraph matching," *Proc. VLDB Endow.*, vol. 4, no. 8, pp. 482–493, May 2011, ISSN: 2150-8097. DOI: 10.14778/2002974.2002976.

[74] S. Han, L. Zou, J. X. Yu, and D. Zhao, *Keyword Search on RDF Graphs - A Query Graph Assembly Approach*, Aug. 2017. DOI: 10.48550/arXiv.1704.00205.

[75] Z. Abu-Aisheh, R. Raveaux, J.-Y. Ramel, and P. Martineau, "An Exact Graph Edit Distance Algorithm for Solving Pattern Recognition Problems," in *4th International Conference on Pattern Recognition Applications and Methods 2015*, Lisbon, Portugal, Jan. 2015. DOI: 10.5220/0005209202710278.

[76] I. Roy, S. Chakrabarti, and A. De, *Maximum Common Subgraph Guided Graph Retrieval: Late and Early Interaction Networks*, Oct. 2022. DOI: 10.48550/arXiv.2210.11020.

[77] N. Reimers and I. Gurevych, *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*, Aug. 2019. DOI: 10.48550/arXiv.1908.10084.

[78] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph Attention Networks*, Feb. 2018. DOI: 10.48550/arXiv.1710.10903.

[79] H. Touvron, L. Martin, K. Stone, *et al.*, *Llama 2: Open Foundation and Fine-Tuned Chat Models*, Jul. 2023. DOI: 10.48550/arXiv.2307.09288.

[80] R. Zhang, J. Han, C. Liu, *et al.*, *LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention*, Jun. 2023. DOI: 10.48550/arXiv.2303.16199.

[81] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *QLoRA: Efficient Finetuning of Quantized LLMs*, May 2023. DOI: 10.48550/arXiv.2305.14314.

[82] R. Castro Fernandez, E. Mansour, A. A. Qahtan, *et al.*, "Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, Paris: IEEE, Apr. 2018, pp. 989–1000, ISBN: 978-1-5386-5520-7. DOI: 10.1109/ICDE.2018.00093.

[83] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, Dec. 2017, ISSN: 2307-387X. DOI: 10.1162/tacl_a_00051.

[84] Y. Zhu, P. Zhang, E.-U. Haq, P. Hui, and G. Tyson, *Can ChatGPT Reproduce Human-Generated Labels? A Study of Social Computing Tasks*, Apr. 2023. DOI: 10.48550/arXiv.2304.10145.

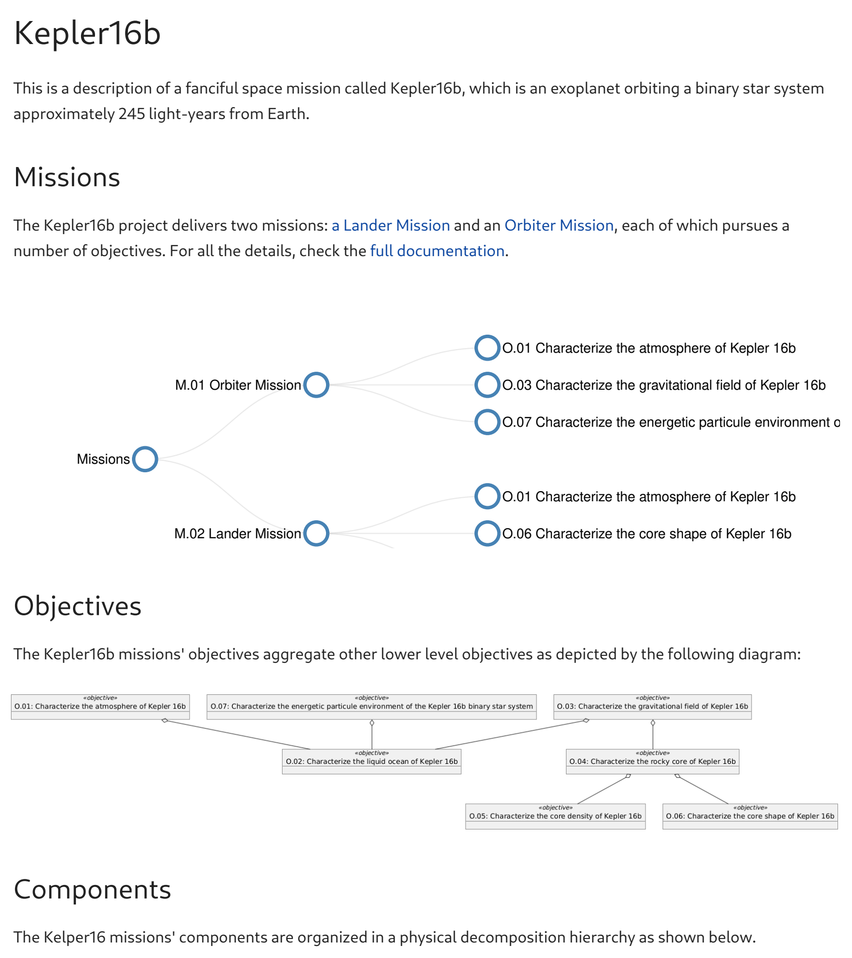
[85] X. Yin, D. Gromann, and S. Rudolph, "Neural machine translating from natural language to SPARQL," *Future Generation Computer Systems*, vol. 117, pp. 510–519, Apr. 2021, ISSN: 0167-739X. DOI: 10.1016/j.future.2020.12.013.

[86] Y. Mansour, M. Mohri, and A. Rostamizadeh, "Domain Adaptation with Multiple Sources," in *Advances in Neural Information Processing Systems*, vol. 21, Curran Associates, Inc., 2008.

Appendices

Kepler16b Report

This appendix presents an example of a report made using openCAESAR.[6][7]



Kepler16b

This is a description of a fanciful space mission called Kepler16b, which is an exoplanet orbiting a binary star system approximately 245 light-years from Earth.

Missions

The Kepler16b project delivers two missions: a Lander Mission and an Orbiter Mission, each of which pursues a number of objectives. For all the details, check the full documentation.

Missions
M.01 Orbiter Mission
O.01 Characterize the atmosphere of Kepler 16b
O.03 Characterize the gravitational field of Kepler 16b
O.07 Characterize the energetic particule environment o
M.02 Lander Mission
O.01 Characterize the atmosphere of Kepler 16b
O.06 Characterize the core shape of Kepler 16b

Objectives

The Kepler16b missions' objectives aggregate other lower level objectives as depicted by the following diagram:

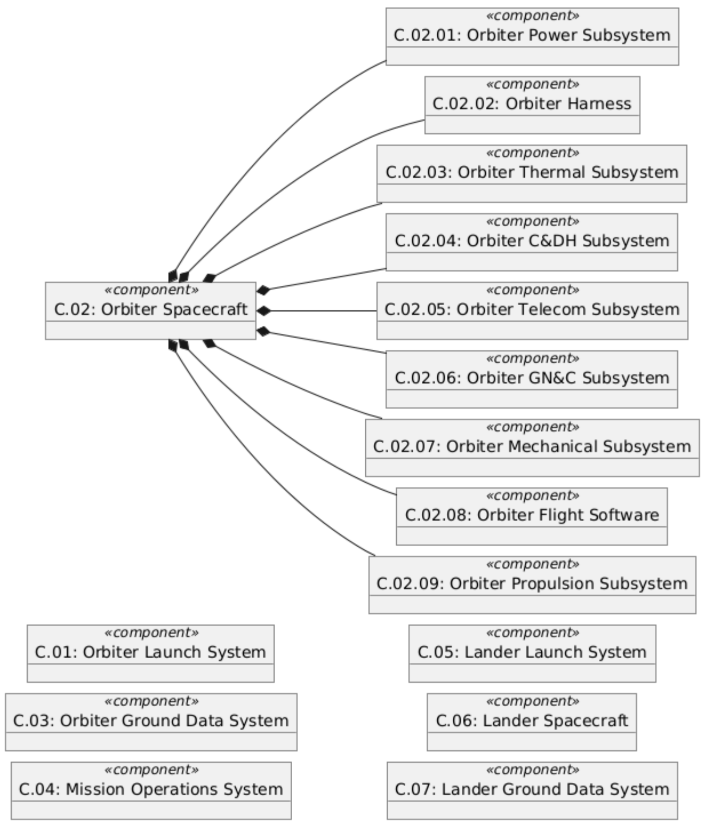
«objective» O.01: Characterize the atmosphere of Kepler 16b
«objective» O.07: Characterize the energetic particule environment of the Kepler 16b binary star system
«objective» O.03: Characterize the gravitational field of Kepler 16b
«objective» O.02: Characterize the liquid ocean of Kepler 16b
«objective» O.04: Characterize the rocky core of Kepler 16b
«objective» O.05: Characterize the core density of Kepler 16b
«objective» O.06: Characterize the core shape of Kepler 16b

Components

The Kelper16 missions' components are organized in a physical decomposition hierarchy as shown below.

Figure 2: Kepler16b report example, page 1.

[6]Kepler16 demo project available at https://github.com/opencaesar/kepler16b-example.
[7]Kepler16b demo report available at https://www.opencaesar.io/kepler16b-example.

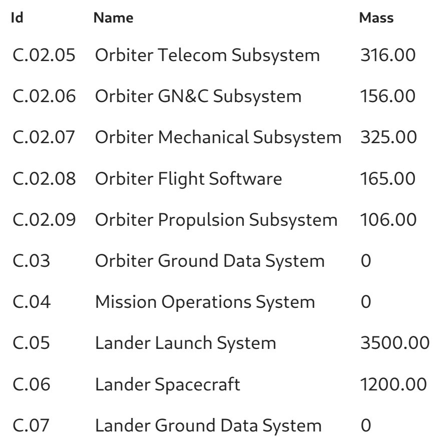

«component»
C.02.01: Orbiter Power Subsystem
«component»
C.02.02: Orbiter Harness
«component»
C.02.03: Orbiter Thermal Subsystem
«component»
C.02.04: Orbiter C&DH Subsystem
«component»
C.02: Orbiter Spacecraft
«component»
C.02.05: Orbiter Telecom Subsystem
«component»
C.02.06: Orbiter GN&C Subsystem
«component»
C.02.07: Orbiter Mechanical Subsystem
«component»
C.02.08: Orbiter Flight Software
«component»
C.02.09: Orbiter Propulsion Subsystem
«component»
C.01: Orbiter Launch System
«component»
C.05: Lander Launch System
«component»
C.03: Orbiter Ground Data System
«component»
C.06: Lander Spacecraft
«component»
C.04: Mission Operations System
«component»
C.07: Lander Ground Data System

Mass Rollup

The Kelper16 missions' components are characterized by their masses. Those masses are rolled up the physical decomposition hierarchy as shown below.

Id	Name	Mass
C.01	Orbiter Launch System	2000.00
C.02	Orbiter Spacecraft	1957.00
C.02.01	Orbiter Power Subsystem	297.00
C.02.02	Orbiter Harness	138.00
C.02.03	Orbiter Thermal Subsystem	307.00
C.02.04	Orbiter C&DH Subsystem	147.00

Figure 3: Kepler16b report example, page 2.



Id	Name	Mass
C.02.05	Orbiter Telecom Subsystem	316.00
C.02.06	Orbiter GN&C Subsystem	156.00
C.02.07	Orbiter Mechanical Subsystem	325.00
C.02.08	Orbiter Flight Software	165.00
C.02.09	Orbiter Propulsion Subsystem	106.00
C.03	Orbiter Ground Data System	0
C.04	Mission Operations System	0
C.05	Lander Launch System	3500.00
C.06	Lander Spacecraft	1200.00
C.07	Lander Ground Data System	0

Figure 4: Kepler16b report example, page 3.

SciQA Templates

This appendix presents an example query (more specifically, a PGP) for each template available in the SciQA dataset. The corresponding SQUALL expressions, which are (nearly) equivalent, are also provided, along with details on the method of acquisition.

In some cases, a second projection variable — representing the `rdfs:label` of the first — was omitted to simplify the manual construction of the queries. Prior to handcrafting the SQUALL queries, the SPARQL queries were further simplified while preserving their equivalence to the originals. For example, nested queries were flattened, see Listing 9.

```
SPARQL:
SELECT DISTINCT ?model ?model_lbl WHERE {
        ?dataset a <Dataset>;
                rdfs:label ?dataset_lbl .
        FILTER(STR(?dataset_lbl) = 'SentEval')
        ?benchmark <has_dataset> ?dataset;
                <Has_evaluation> ?eval .
        ?paper <has_benchmark> ?benchmark .
        OPTIONAL {
                ?paper <has_model> ?model .
                ?model rdfs:label ?model_lbl .
        }
}
---
SQUALL:
A <Dataset> is a <has_dataset> of a thing X and has a rdfs:label whose string matches
 ↪  'SentEval' and X is a <has_benchmark> of a thing Y and a thing is a
 ↪  <has_evaluation> of X and if defined, what is a <has_model> of Y and has a
 ↪  rdfs:label?
---
# Handcrafted SQUALL: 'optional' not supported by SPARQL2SQUALL.
```

Listing 5: SciQA example for template 1.

```
SPARQL:
SELECT DISTINCT ?paper ?paper_lbl WHERE {
        ?dataset a <Dataset>;
                rdfs:label ?dataset_lbl .
        FILTER(STR(?dataset_lbl) = 'DDI extraction 2013 corpus')
        ?benchmark <has_dataset> ?dataset .
        ?cont <has_benchmark> ?benchmark.
        ?paper <has_contribution> ?cont;
                rdfs:label ?paper_lbl .
}
---
SQUALL:
What has a rdfs:label and has a <has_contribution> that has a <has_benchmark> that has
 ↪  <has_dataset> a <Dataset> that has a rdfs:label that is 'DDI extraction 2013
 ↪  corpus'?
---
/* Mapped SPARQL to SQUALL using SPARQL2SQUALL. */
```

Listing 6: SciQA example for template 2.

```
SPARQL:
SELECT DISTINCT ?metric ?metric_lbl WHERE {
        ?dataset a <Dataset>;
                rdfs:label ?dataset_lbl .
        FILTER(STR(?dataset_lbl) = 'Habitat 2020 Point Nav test-std')
        ?benchmark <has_dataset> ?dataset;
                <Has_evaluation> ?eval .
        OPTIONAL {
                ?eval <Has_metric> ?metric .
                ?metric rdfs:label ?metric_lbl.
        }
}
---
SQUALL:
A <Dataset> is a <has_dataset> of a thing X and has a rdfs:label whose string matches
 ↪  'Habitat 2020 Point Nav test-std' and a thing Y is a <has_evaluation> of X and if
 ↪  defined, what is a <has_metric> of Y and has a rdfs:label?
---
# Handcrafted SQUALL: 'optional' not supported by SPARQL2SQUALL.
```

Listing 7: SciQA example for template 3.

```
SPARQL:
SELECT DISTINCT ?metric ?metric_lbl (max(?value) as ?score) WHERE {
        {
                SELECT ?metric ?metric_lbl ?value WHERE {
                        ?dataset a <Dataset>;
                                rdfs:label ?dataset_lbl .
                        FILTER(STR(?dataset_lbl) = 'Walker, walk (DMControl100k)')
                        ?benchmark <has_dataset> ?dataset;
                                <Has_evaluation> ?eval .
                        ?eval <Has_value> ?value .
                        OPTIONAL {
                                ?eval <Has_metric> ?metric .
                                ?metric rdfs:label ?metric_lbl .
                        }
                        ?cont <has_benchmark> ?benchmark .
                        OPTIONAL {
                                ?cont <has_model> ?model .
                                ?model rdfs:label ?model_lbl .
                        }
                }
                ORDER BY DESC(?value)
        }
}
GROUP BY ?metric ?metric_lbl
---
SQUALL:
What is the max of the <Has_value> of a thing X that is a <Has_evaluation> of a thing
 ↪  that (has <has_dataset> a <Dataset> that has rdfs:label 'Walker, walk
 ↪  (DMControl100k)') and is a <has_benchmark> of a thing that if defined, has a
 ↪  <has_model> and if defined, has a rdfs:label per the if defined, <Has_metric> Y of
 ↪  X and per the if defined, rdfs:label of Y?
---
# Handcrafted SQUALL: Verbalization of aggregates not supported by SPARQL2SQUALL.
```

Listing 8: SciQA example for template 4.

```
SPARQL:
SELECT DISTINCT ?model ?model_lbl WHERE {
        ?metric a <Metric>;
                rdfs:label ?metric_lbl .
        FILTER(STR(?metric_lbl) = 'Score') {
                SELECT ?model ?model_lbl WHERE {
                        ?dataset a <Dataset>;
                                rdfs:label ?dataset_lbl .
                        FILTER(STR(?dataset_lbl) = 'Atari 2600 Defender')
                        ?benchmark <has_dataset> ?dataset;
                                <Has_evaluation> ?eval .
                        ?eval <Has_value> ?value;
                                <Has_metric> ?metric .
                        ?cont <has_benchmark> ?benchmark;
                                <has_model> ?model .
                        ?model rdfs:label ?model_lbl .
                }
                ORDER BY DESC(?value)
                limit 1
        }
}
---
SQUALL:
What has a rdfs:label and is a <has_model> of a thing that has a <has_benchmark> that
 ↪  has a <has_dataset> X and has a <has_eval> that has the 1 lowest <has_value> and
 ↪  has a <has_metric> Y and X is a <Dataset> and has a rdfs:label whose string
 ↪  matches 'Atari 2600 Defender' and Y is a <Metric> and has a rdfs:label whose
 ↪  string matches 'Score'?
---
# Handcrafted SQUALL: Verbalization of sub-queries not supported by SPARQL2SQUALL.
```

Listing 9: SciQA example for template 5.

```
SPARQL:
SELECT DISTINCT ?dataset ?dataset_lbl WHERE {
        ?problem a <Problem>;
                rdfs:label ?problem_lbl.
        FILTER(STR(?problem_lbl) = 'Image Generation')
        ?dataset a <Dataset>;
                rdfs:label ?dataset_lbl .
        ?benchmark <has_dataset> ?dataset .
        ?cont <has_benchmark> ?benchmark;
                <has_research problem> ?problem .
}
---
SQUALL:
Which <Dataset> has a rdfs:label and is a <has_dataset> of a <has_benchmark> of a
 ↪  thing that has <has_research problem> a <Problem> that has a rdfs:label that is
 ↪  'Image Generation'?
---
/* Mapped SPARQL to SQUALL using SPARQL2SQUALL. */
```

Listing 10: SciQA example for template 6.

```
SPARQL:
SELECT DISTINCT ?code WHERE {
        ?model a <Model>;
                rdfs:label ?model_lbl .
        FILTER(STR(?model_lbl) = 'Depth DDPPO')
        ?benchmark <has_dataset> ?dataset .
        ?cont <has_benchmark> ?benchmark .
        ?cont <has_model> ?model;
                <has_source code> ?code .
}
---
SQUALL:
What is a <has_source code> of a thing that has <has_model> a <Model> X and has a
 ↪  <has_benchmark> that has a <has_dataset> and X has a rdfs:label whose string
 ↪  matches 'Depth DDPPO'?
---
# Handcrafted SQUALL: Verbalization cannot be made non-ambiguous without brackets by
 ↪  SPARQL2SQUALL.
```

Listing 11: SciQA example for template 7.

```
SPARQL:
SELECT DISTINCT ?problem ?problem_lbl WHERE {
        ?rf a <ResearchField>;
                rdfs:label ?rf_label .
        FILTER(STR(?rf_label) = 'Natural Language Processing')
        ?paper <has_research field> ?rf;
                <has_contribution> ?cont .
        ?cont <has_benchmark> ?benchmark;
                <has_research problem> ?problem .
        ?problem rdfs:label ?problem_lbl .
}
---
SQUALL:
What has a rdfs:label and is a <has_research problem> of a thing that has a
 ↪  <has_benchmark> and is a <has_contribution> of a thing that has <has_research
 ↪  field> a <ResearchField> that has a rdfs:label that is 'Natural Language
 ↪  Processing'?
---
/* Mapped SPARQL to SQUALL using SPARQL2SQUALL. */
```

Listing 12: SciQA example for template 8.

Responsible Use of Generative Artificial Intelligence

The use of generative AI in this thesis has been carefully restricted to two specific use cases: serving as a writing assistant and a productivity enhancer. In particular, ChatGPT was utilized to refine the flow of pre-written passages, summarize chapters for use in introductions and conclusions, and expedite the creation of TikZ schematics and graphs. At no point was AI used to generate content from scratch.

Societal Reflection

Upon reflection, it became clear that certain Sustainable Development Goals were inherently aligned with the research process, even though they were not consciously adopted as guiding principles from the outset.

Great care was taken to ensure responsible production throughout the methodology and experimentation phases. This was demonstrated by the judicious use of computational resources, highlighted by the selection of a comparatively smaller domain-specific model, i.e., the domain expert, which consumed significantly less computational power than, for example, a ChatGPT-based prompting approach.

Transparency and accessibility have also been central to this work. Every step of the research has been explicitly documented and explained to ensure that the findings are accessible and reproducible, fostering inclusivity and openness in scientific research.

Finally, this thesis originated at Polytechnique Montréal and was further developed upon my return to Ghent University. I hope that this work not only contributes to advancing academic knowledge but also fosters stronger collaborative ties between these two institutions. I will enthusiastically encourage and recommend similar exchange opportunities.