FEDERAL UNIVERSITY OF AMAZONAS - UFAM

INSTITUTE OF COMPUTING - ICOMP

GRADUATE PROGRAM IN INFORMATICS - PPGI

Bayesian and Neural Ranking Approaches for Supporting Schema References in Keyword Queries over Relational Databases

Paulo Rodrigo Oliveira Martins

Manaus, AM, Brazil

July 2024

Paulo Rodrigo Oliveira Martins

Bayesian and Neural Ranking Approaches for Supporting Schema References in Keyword Queries over Relational Databases

Thesis submitted for evaluation as a final requirement for obtaining the degree of Doctor in Informatics from the Graduate Program in Informatics, Institute of Computing.

Advisor

Altigran Soares da Silva

Federal University of Amazonas - UFAM

Institute of Computing - IComp

Manaus, AM, Brazil

July 2024

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Martins, Paulo Rodrigo Oliveira
M386b Bayesian and neural ranking approaches for supporting schema references in keyword queries over relational databases / Paulo Rodrigo Oliveira Martins . 2024
121 f.: il. color; 31 cm.

Orientador: Altigran Soares da Silva
Tese (Doutorado em Informática) - Universidade Federal do Amazonas.

1. Keyword Search. 2. Database Systems. 3. Keyword Search over Relation Databases. 4. Bayesian and Neural approaches. 5. Sentence Transformers. I. Silva, Altigran Soares da. II. Universidade Federal do Amazonas III. Título

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Informática

FOLHA DE APROVAÇÃO

"BAYESIAN AND NEURAL RANKING APPROACHES FOR SUPPORTING SCHEMA REFERENCES IN KEYWORD QUERIES OVER RELATIONAL DATABASES"

PAULO RODRIGO OLIVEIRA MARTINS

Tese de Doutorado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Dr. Altigran Soares da Silva - **Presidente**

Prof. Dr. Eduardo Cunha de Almeida - **Membro Externo**

Prof. Dr. Edleno Silva Moura - **Membro Interno**

Prof. Dr. João Marcos Bastos Cavalcanti - **Membro Interno**

Dr. Johny Moreira da Silva - **Membro Externo**

Manaus, 16 de agosto de 2024.

Documento assinado eletronicamente por **Altigran Soares da Silva**, **Professor do Magistério Superior**, em 19/08/2024, às 17:05, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Eduardo Cunha de Almeida**, **Usuário Externo**, em 20/08/2024, às 15:51, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Johny Moreira da Silva**, **Usuário Externo**, em 21/08/2024, às 16:11, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Edleno Silva de Moura**, **Professor do Magistério Superior**, em 22/08/2024, às 10:14, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **João Marcos Bastos Cavalcanti**, **Professor do Magistério Superior**, em 22/08/2024, às 10:44, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Maria do Perpétuo Socorro Vasconcelos Palheta**, **Secretária**, em 22/08/2024, às 12:08, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufam.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2196169** e o código CRC **55CAF68B**.

Avenida General Rodrigo Octávio, 6200 - Bairro Coroado I Campus Universitário Senador Arthur Virgílio Filho, Setor Norte - Telefone: (92) 3305-1181 / Ramal 1193
CEP 69080-900, Manaus/AM, coordenadorppgi@icomp.ufam.edu.br

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This work was partially supported by Amazonas State Research Support Foundation - FAPEAM - through the POSGRAD project.

"Why is programming fun? What delights may its practitioner expect as his reward? First is the sheer joy of making things. As the child delights in his mud pie, so the adult enjoys building things, especially things of his own design. I think this delight must be an image of God's delight in making things, a delight shown in the distinctness and newness of each leaf and each snowflake."

(Fred Brooks)

Abstract

Relational Keyword Search (R-KwS) systems enable naive/informal users to explore and retrieve information from relational databases without knowing schema details or query languages. These systems take the keywords from the input query, locate the elements of the target database that correspond to these keywords, and look for ways to "connect" these elements using information on referential integrity constraints, i.e., key/foreign key pairs. Although several such systems have been proposed in the literature, most of them only support queries whose keywords refer to the contents of the target database. Very few support queries in which keywords refer to elements of the database schema. In this work, we propose Lathe, a novel R-KwS designed to support such queries. To this end, we first generalize the well-known concepts of Query Matches (QMs) and Candidate Joining Networks (CJNs) to handle keywords referring to schema elements and propose new algorithms to generate them. Then, we introduce an approach to automatically select the CJNs that are more likely to represent the user intent when issuing a keyword query. Our key contributions are a novel Bayesian-based QM ranking algorithm that prioritizes relevant QMs, avoiding the processing of less likely answers, an effective Bayesian CJN ranking algorithm leveraging QM rankings to prioritize and evaluate relevant CJNs, an eager CJN evaluation strategy that discards spurious CJNs early, and a novel transformer-based neural approach for QM ranking and CJN ranking, leading to improved results on metrics such as recall and $R@k$. We present a comprehensive set of experiments performed with query sets and datasets previously used in experiments with state-of-the-art R-KwS systems and methods. Our results indicate that Lathe can handle a wider variety of keyword queries while remaining highly effective, even for large databases with intricate schemas. Additionally, we developed PyLatheDB, a Python library for Relational Keyword Search that implements Lathe.

Resumo

Sistemas de Busca por Palavra-Chave em Banco de Dados Relacional (R-KwS) permitem que usuários leigos ou informais explorem e recuperem informações de bancos de dados relacionais sem precisar conhecer detalhes do esquema ou linguagens de consulta. Esses sistemas utilizam as palavras-chave da consulta de entrada, localizam os elementos do banco de dados que correspondem a essas palavras-chave e buscam maneiras de "conectar" esses elementos usando informações sobre restrições de integridade referencial, isto é, o par chave/chave estrangeira. Embora vários desses sistemas tenham sido propostos na literatura, a maioria deles suporta apenas consultas cujas palavras-chave se referem ao conteúdo do banco de dados. Poucos sistemas oferecem suporte a consultas em que as palavras-chave se referem a elementos do esquema do banco de dados. Neste trabalho, propomos o Lathe, um novo R-KwS projetado para suportar esse tipo de consulta. Para isso, primeiro generalizamos os conceitos conhecidos de *Query Matches* (QMs) e *Candidate Joining Networks* (CJNs) para lidar com palavras-chave que se referem a elementos do esquema e propomos novos algoritmos para gerá-los. Em seguida, introduzimos uma abordagem para selecionar automaticamente as CJNs que têm maior probabilidade de representar a intenção do usuário ao fazer uma consulta por palavras-chave. Nossas principais contribuições incluem um novo algoritmo de ranqueamento de QMs bayesiano, que prioriza QMs relevantes, evitando o processamento de respostas menos prováveis; um algoritmo de ranqueamento de CJNs também bayesiano que utiliza o ranqueamento de QMs para priorizar e avaliar CJNs relevantes; uma estratégia de *eager evaluation* que descarta CJNs espúrias logo que são criadas; e uma nova abordagem neural baseada em transformers para ranqueamento de QMs e CJNs, resultando em melhorias em métricas como recall e $R@k$. Apresentamos um conjunto abrangente de experimentos realizados com conjuntos de consultas e dados previamente utilizados em experimentos com sistemas e métodos de R-KwS de última geração. Nossos resultados indicam que o Lathe é capaz de lidar com uma variedade maior de consultas por palavras-chave, mantendo-se altamente eficaz, mesmo para grandes bancos de dados com esquemas complexos. Além disso, desenvolvemos o PyLatheDB, uma biblioteca Python para Busca por Palavra-Chave em Banco de Dados Relacional que implementa o Lathe.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Keyword Search over Relational Databases (R-KwS) enables naive/informal users to retrieve information from relational databases (DBs) without any knowledge about schema details or query languages. The success of search engines shows that untrained users are at ease using keyword search to find information of interest. However, enabling users to search relational DBs using keyword queries is a challenging task, because the information sought frequently spans multiple relations and attributes, depending on the schema design of the underlying DB. As a result, R-KwS systems face the challenge of automatically determining which pieces of information to retrieve from the database and how to connect them to provide a relevant answer to the user.

In the last two decades, the R-KwS was extensively studied in academia, which lead to several improvements in performance and effectiveness. A well-known approach for R-KwS is to generate Candidate Joining Networks (CJNs), which are networks of joined database relations that are translated into SQL queries whose results provide an answer to the input keyword query. The first algorithm for CJN generation is CNGen, which was first presented in the DISCOVER system [Hristidis and Papakonstantinou, 2002], but was later adopted by most R-KwS systems [Agrawal et al., 2002, Hristidis et al., 2003, Luo et al., 2007, Coffman and Weaver, 2010b]. Despite the possible large number of CJNs, most works in the literature focused on improving CJN evaluation and ranking the returned results from the database, which can be seen as Joining Networks of Tuples (JNTs), instead. Specifically, DISCOVER II [Hristidis et al., 2003], SPARK [Luo et al., 2007], and CD [Coffman and Weaver, 2010b] used information retrieval (IR) style score functions to rank the top-K JNTs. KwS-F [Baid et al., 2010] imposed a time limit for the evaluation of CJNs, returning potentially partial results, as well as a summary of CJNs that have yet to be evaluated. Later, CNRank [Oliveira et al., 2015] introduced a CJN ranking, requiring only the top-ranked CJNs to be evaluated. MatCNGen [Oliveira et al., 2018, de Oliveira et al., 2020] proposed a

novel method for generating CJNs that efficiently enumerated the possible matches for the query in the DB. These *Query Matches* (QMs) are then used to guide the CJN generation process, greatly decreasing the number of CJNs generated and improving the performance of the CJN evaluation.

1.1 Motivation

In general, the keywords from a query may refer to both database values, such as tuples containing these keywords, and schema elements, such as relation and attribute names. For instance, consider the query "$will\ smith\ films$" over a database on movies. Keywords "$will$" and "$smith$" may refer to values of person names. The keyword "$films$" on the other hand, is more likely to refer to a schema element, the name of the relation about movies. Although a significant number of query keywords correspond to schema references [Bergamaschi et al., 2011a], the majority of previous work on R-KwS systems in the literature does not support references to schema information such as the one in the above query. As a result, given a query, they will search for attributes whose tuples include the keyword "$films$", which is unlikely to yield a useful answer for the user.

Another important issue is that there may exist several void CJNs, that is, CJNs whose execution against the database return empty results. Therefore, match-based approaches [Oliveira et al., 2018, de Oliveira et al., 2020], which consider only a number of CJNs for each QM, may not generate any useful CJN for the relevant QM. As a result, the quality of results of such approaches is decreased, especially for databases with intricate schema.

1.2 Methodology

In this work, we study new techniques to support schema references in keyword queries over relational databases. Specifically, we propose Lathe[1], a new R-KwS system to generate a suitable SQL query from a keyword query, considering that keywords may refer either to instance values or to elements of the database schema, that is, relations and attributes. Lathe follows the *Schema Graph* approach for R-KwS systems [Oliveira et al., 2018, Coffman and Weaver, 2010a]. Given a keyword query, this approach consists of generating relational algebra expressions called *Candidate Joining Networks*[2] (CJNs), which are likely to express user intent when formulating the original query. The

[1]The name Lathe refers to the fact that our system assigns a structure or form to an unstructured keyword-based query

[2]Most of the previous work uses the term *Candidate Networks* instead. Here, we use *Candidate Joining Networks* because we consider it more meaningful.

generated CJNs are *evaluated*, that is, they are translated into SQL queries and executed by a DBMS, resulting in several *Joining Networks of Tuples* (JNTs) that are collected and supplied to the user.

Among the methods based on the Schema Graph approach, Lathe is, to the best of our knowledge, the first method to address the problem of generating and ranking CJNs considering queries with keywords that can refer to either schema elements or attribute values. We revisited and generalized concepts introduced in previous approaches [Hristidis and Papakonstantinou, 2002, Oliveira et al., 2015, Oliveira et al., 2018, de Oliveira et al., 2020], such as tuples-sets, QMs, and the CJNs themselves, to enable schema references.

In addition, we proposed a more effective approach to CJN generation that four major novel contributions a Bayesian QM ranking, a Bayesian CJN Ranking, an Eager CJN Evaluation technique, and a Neural approach for QM Ranking and CJN Ranking. Lathe roughly matches keywords to the values of attributes or to schema elements such as names of attributes and relations. Next, the system combines the keyword matches into QMs that cover all the keywords from the query. The QMs are ranked, using either a Bayesian or a neural approach, so that only the most relevant ones are used to generate CJNs. The CJN generation explores the primary key/foreign key relationships to connect all the elements of the QMs. In addition, Lathe employs an eager CJN evaluation strategy, which ensures that all CJNs generated will yield nonempty results when evaluated. The CJNs are then ranked using either a Bayesian or a neural approach, and evaluated. Finally, the CJN evaluation results are delivered to the user. Unlike the previous methods, Lathe provides the user with the most relevant answer without relying on JNTs rankings. This is due to the effective rankings of QMs and CJNs that we propose, which are absent from the majority of previous work.

1.3 Hypotheses and Research Questions

Hypothesis 1. *The support of schema references leads to better recall while still maintaining the effectiveness of the system.*

Q 1.1. How to extend existing algorithms in R-KwS to support schema references?

Q 1.2. Which similarity function can be used to match keywords to schema elements?

Q 1.3. What is the impact of matching keywords to attributes instead of relations, as it was done with tuple-sets?

Hypothesis 2. *Advancing part of the CJN Ranking and applying it to ranking QMs decrease the number of CJNs generated, while maintaining the effectiveness of CJN Ranking.*

Q 2.1. How does QM Ranking affect the final results returned to the user?

Hypothesis 3. *An eager evaluation strategy for CJN Generation improves the quality of CJN Ranking by pruning spurious CJNs that do not yield any information to the user.*

Q 3.1. What is the trade-off of adopting a CJN evaluation strategy?

Q 3.2. How to implement this strategy efficiently?

Q 3.3. Is eager CJN evaluation more effective for databases with intricate schema?

Hypothesis 4. *A neural approach based on sentence-transformer models provides superior QM Ranking and CJN Ranking in comparison to the traditional Bayesian approach, improving the quality of results.*

Q 4.1. What models are adequate for ranking QMs and CJNs?

Q 4.2. How to represent QMs and CJNs as sentences for the transformer-based models?

Q 4.3. How to fine-tune the models for QM and CJN ranking?

Q 4.4. Can we implement a data augmentation strategy to minimize the need for manually labeling training data for fine-tuning?

1.4 Key Contributions

Our key contributions are: (i) a novel method for generating and ranking CJNs that supports keywords referring to schema elements and instance values; (ii) a novel Bayesian-based QM ranking algorithm that prioritizes relevant QMs, avoiding the processing of less likely answers; (iii) an effective Bayesian CJN ranking algorithm leveraging QM rankings to prioritize and evaluate relevant CJNs; (iv) an eager CJN evaluation strategy that discards spurious CJNs early; (v) a novel transformer-based neural approach for QM ranking and CJN ranking, leading to improved results on metrics such as Recall and $R@k$.

We performed several experiments to assess the effectiveness and efficiency of Lathe. Initially, we compared the quality of its results with those obtained from several previous R-KwS systems, including the state-of-the-art QUEST [Bergamaschi et al., 2013] system, using a benchmark proposed by Coffman & Weaver [Coffman and Weaver, 2010a]. We then evaluated the quality of our novel QM ranking and assessed the CJN ranking by comparing different configurations in terms of the number of QMs, the number of CJNs generated per QM, and the use of the eager evaluation strategy. Additionally, we assessed the speed of each phase of Lathe, as well as the trade-off between quality and speed for various system configurations. Moreover, we performed experiments comparing the Bayesian approach and the Neural approach, considering several transformer-based models. Overall, Lathe achieved

better results than all R-KwS systems tested in our experiments. Our results indicate that the novel QM ranking and eager CJN evaluation greatly improved the quality of CJN generation. Finally, we also developed PyLatheDB [Martins et al., 2023a], a Python library for Relational Keyword Search that implements Lathe. The library source code and demonstration are available at `https://github.com/bdri-ufam/PyLatheDB`.

1.5 Document Outline

The outline of the remainder of this thesis is as follows. Chapter 2 reviews the related literature on relational keyword search systems based on schema graphs and support for schema references. Chapter 3 summarizes all phases of our method, which are discussed in detail in Chapter 4-8. Chapter 9 summarizes the findings of the experiments we conducted. Chapter 10 presents PyLatheDB, a Python library for R-KwS, which implements Lathe. Chapter 11 summarizes additional developments, systems, and research projects derived from this research. Chapter 12 concludes the thesis, providing a final reflection on the research outcomes, key findings, and plans for future work.

Chapter 2

Background and Related Work

In this chapter, we discuss the background and related work on keyword search systems over relational databases, supporting schema references in such systems, and deep neural networks (DNNs) applied to tabular data. For a more comprehensive view of the state-of-the-art in keyword-based and natural language queries over databases, we refer the interested reader to a survey on this matter [Affolter et al., 2019].

2.1 Relational Keyword Search Systems

Current R-KwS systems fall in one of two distinct categories: systems based on *Schema Graphs* and systems based on *Instance Graphs*. Systems in the first category are based on the concept of *Candidate Joining Networks* (CJNs), which are networks of joined relations that are used to generate SQL queries and whose evaluation return several *Joining Networks of Tuples* (JNTs) which are collected and supplied to the user. This method was proposed in DISCOVER [Hristidis and Papakonstantinou, 2002] and DBXplorer [Agrawal et al., 2002], and it was later adopted by several other systems, including DISCOVER-II [Hristidis et al., 2003], SPARK [Luo et al., 2007], CD [Coffman and Weaver, 2010b], KwS-F [Baid et al., 2010], CNRank [Oliveira et al., 2015], and MatCNGen [Oliveira et al., 2018, de Oliveira et al., 2020]. Systems in this category make use of the underlying basic functionality of the RDBMS by generating appropriate SQL queries to retrieve answers to keyword queries posed by users.

Systems in the second category are based on a structure called *Instance Graph*, whose nodes represent tuples associated with the keywords they contain, and the edges connect these tuples based on referential integrity constraints. BANKS [Aditya et al., 2002], BANKS-II [Kacholia et al., 2005], BLINKS [He et al., 2007] and, Effective [Liu et al., 2006] use this approach to compute keyword queries results by finding subtrees in a data graph that minimizes

the distance between nodes matching the given keywords. These systems typically generate the query answer in a single phase that combines the tuple retrieval task and the answer schema extraction. However, the Instance Graph approach requires a materialization of the DB and requests a higher computational cost to deliver answers to the user. Furthermore, the important structural information provided by the database schema is ignored, once the data graph has been built.

2.2 R-KwS Systems based on Schema Graphs

In our research, we focus on systems based on Schema Graphs, since we assume that the data we want to query are stored in a relational database and we want to use an RDBMS capable of processing SQL queries. Also, our work expands on the concepts and terminology introduced in DISCOVER [Hristidis and Papakonstantinou, 2002, Hristidis et al., 2003], and expanded in CNRank [Oliveira et al., 2015] and MatCNGen [Oliveira et al., 2018, de Oliveira et al., 2020]. We expanded this formal framework to handle keyword queries that may refer to attribute values or to database schema elements. As a result, it inherits and maintains all guarantees regarding the generation of complete, sound, and meaningful CJNs.

The best-known algorithm for CJN Generation is CNGen, which was introduced in DISCOVER [Hristidis and Papakonstantinou, 2002], and was later adopted as a default in most of the R-KwS systems proposed in the literature [Agrawal et al., 2002, Hristidis et al., 2003, Luo et al., 2007, Coffman and Weaver, 2010b]. This algorithm employs a Breadth-First Search approach [Cormen et al., 2009] to generate a complete, non-redundant set of CJNs. As a result, CNGen frequently generates a large number of CJNs, resulting in a costly CJN generation and evaluation process.

Initially, most of the subsequent work focused solely on the CJN evaluation, which may generate a large number of JNTs. DISCOVER-II [Hristidis et al., 2003], SPARK [Luo et al., 2007], and CD [Coffman and Weaver, 2010b] introduced algorithms for ranking JNTs using IR style score functions.

KwS-F [Baid et al., 2010] addressed the efficiency and scalability problems in CJN evaluation in a different way. Their approach consists of two steps. First, a limit is imposed on the time the system spends evaluating CJNs. After this limit is reached, the system must return the (possibly partial) top-K JNTs. Second, if there are any CJNs that have yet to be evaluated, they are presented to the user in the form of query forms, from which the user can choose one and the system will evaluate the corresponding CJN.

CNRank [Oliveira et al., 2015] proposed a method for lowering the cost of CJN evaluation by ranking them based on the likelihood that they will provide relevant answers to

the user. Specifically, CNRank presented a probabilistic ranking model that uses a *Bayesian Belief Network* [de Cristo et al., 2003] to estimate the relevance of a CJN given the current state of the underlying database. The model assigns a score to each generated CJN, so that only a few CJNs with the highest scores need to be evaluated.

MatCNGen [Oliveira et al., 2018, de Oliveira et al., 2020] introduced a match-based approach for generating CJNs. The system enumerates the possible ways which the query keywords can be matched in the DB beforehand, to generate query answers. MatCNGen then generates a single CJN, for each of these query matches, drastically reducing the time required to generate CJNs. Furthermore, because the system assumes that answers must contain all of the query keywords, each keyword must appear in at least one element of a CJN. As a result of the generation process avoiding generating too many keyword occurrence combinations, a smaller but better set of CJNs is generated.

Lastly, Coffman & Weaver [Coffman and Weaver, 2010a] proposed a framework for evaluating R-KwS systems and reported experimental results over three representative standardized datasets they built, namely MONDIAL, IMDb, and Wikipedia, along with respective query workloads. The authors compare nine R-KwS systems, assessing their effectiveness and performance in a variety of ways. The resources of this framework were also used in the experiments of several other studies on R-KwS systems [Luo et al., 2007, Oliveira et al., 2015, Oliveira et al., 2018, de Oliveira et al., 2020, Coffman and Weaver, 2012].

2.3 Support to Schema References in R-KwS

Overall there are few systems in the literature that support schema references in keywords queries. One of the first such systems was BANKS [Bhalotia et al., 2002], a R-KwS system based on Instance Graphs. However, hence the query evaluation with keywords matching metadata can be relatively slow, since a large number of tuples may be defined to be relevant to the keyword. Keymantic [Bergamaschi et al., 2011a], KEYRY [Bergamaschi et al., 2011b], and QUEST [Bergamaschi et al., 2013] also addressed the support for schema references in keyword queries. However, despite these systems being classified as schema-based since they aim at generating a suitable SQL query given an input keyword query, they do not rely on the concept of CJNs, as Lathe, the system we propose, and all DISCOVER-based systems do. Keymantic and KEYRY consider a scenario where data instances are not accessible, such as in databases on the hidden web and sources hidden behind wrappers in data integration settings, where typically only metadata is made available. Both systems rely on similarity techniques based on structural and lexical knowledge that can be extracted from the available metadata, e.g., names of attributes and tables, attribute domains, regular expressions, or from

other external sources, such as ontologies, vocabularies, domain terminologies, etc. The two systems mainly differ in the way they rank the possible interpretations they generate for an input query. While Keymantic relies on an extension the authors proposed for the Hungarian algorithm [Bourgeois and Lassalle, 1971], KEYRY is based on the Hidden Markov Model [Bergamaschi et al., 2011b], a probabilistic sequence model, adapted for keyword query modeling. QUEST [Bergamaschi et al., 2013] can be thought of as an extension of KEYRY because it uses a similar strategy to rank the mappings from keywords to database elements. QUEST, on the other hand, considers the database instance to be accessible and includes features derived from it for ranking interpretations, in contrast to KEYRY.

From these systems, QUEST is the one most similar to Lathe. However, it is difficult to draw a direct comparison between the two systems as QUEST does not rely on the formal framework from CJN-related previous work [Hristidis and Papakonstantinou, 2002, Hristidis et al., 2003, Oliveira et al., 2015, Oliveira et al., 2018, de Oliveira et al., 2020] and it also resolves a smaller set of keyword queries then Lathe. QUEST, in particular, does not support keyword queries whose resolution necessitates SQL queries with self-joins. As a result, when comparing QUEST to other approaches, the authors limited the experimentation to 35 queries rather then the 50 included in the original benchmark [Bergamaschi et al., 2013, Coffman and Weaver, 2010a]. Lathe, on the other hand, supports all 50 queries.

Finally, there are systems that propose going beyond the retrieval of tuples that fulfill a query expressed using keywords and try to provide a functionality close to structured query languages. This is the case of SQAK [Tata and Lohman, 2008] that allows users to specify aggregation functions over schema elements. Such an approach was later expanded in systems such as SODA [Blunschi et al., 2012] and SQUIRREL [Ramada et al., 2020], which aim to handle not only aggregation functions, but also keywords that represent predicates, groupings, orderings and so on. To support such features, these systems rely on a variety of resources that are not part of the database schema or instances. Among these are conceptual schemas, generic and domain-specific ontologies, lists of reserved keywords, and user-defined metadata patterns. We see such useful systems as being closer to natural language query systems [Affolter et al., 2019]. In contrast, Lathe, like any typical R-KwS system, aims at retrieving sets of JNTs that fulfill the query, and not computing results with the tuples. In addition, it does not rely on any external resources.

2.4 Deep Neural Networks and Tabular Data

Deep neural networks (DNNs) have revolutionized the field of machine learning, achieving remarkable success in various domains such as images, audio, and text [Devlin et al., 2018].

However, their application to tabular data, which is characterized by a mix of numerical and categorical features, presents unique challenges. The heterogeneous nature of tabular data complicates the direct application of DNNs, prompting the development of specialized approaches to better handle this data type [Shwartz-Ziv and Armon, 2022].

Recent advancements in DNNs for tabular data can be categorized into three main groups: data transformations, specialized architectures, and regularization models [Borisov et al., 2022]. Data transformation techniques convert tabular data into formats more suitable for neural networks, often employing sophisticated encoding strategies for categorical variables. Specialized architectures, including hybrid models and transformer-based approaches, are designed to leverage the unique characteristics of tabular data. Regularization models focus on preventing overfitting and enhancing the generalization capabilities of DNNs.

Among these, specialized architectures form the largest group of approaches. Transformer-based approaches [Vaswani et al., 2017], inspired by their success in text and visual data, have also been adapted for tabular data, utilizing deep attention mechanisms to handle the diverse features present in such datasets [Arik and Pfister, 2021].

An important problem in this domain is table retrieval, which involves identifying the most relevant table from a set of tables given a specific query [Zhang and Balog, 2018, Zhang and Balog, 2021]. This task is crucial for finding tables that can answer a given question or provide relevant information. Considering that Candidate Joining Networks (CJNs) can be interpreted as database views, the ranking of CJNs can be seen as a variant of the table retrieval problem. Both tasks require understanding and matching the structure and content of tables or views to a given query or information need.

Several transformer-based systems have been developed to handle table retrieval effectively [Badaro et al., 2023]. TaBERT [Yin et al., 2020] integrates natural language text with structured tabular data by linearizing table structures and introducing content snapshots. Linearization is necessary because transformer models are traditionally designed to process unstructured data like text, so the structured rows and columns of a table must be converted into a serialized sequence format. StruBERT [Trabelsi et al., 2022] builds on TaBERT by incorporating horizontal self-attention, leading to improvements in table retrieval and similarity tasks. TAPAS [Herzig et al., 2020] extends the BERT model to jointly encode table structures and questions, facilitating various operations directly within the table context.

While table retrieval systems like TaBERT, StruBERT, and TAPAS primarily handle single tables, CJNs span different database tables, making the retrieval and ranking tasks more complex. This complexity highlights the need for specialized approaches to manage multi-table scenarios effectively, pushing the boundaries of what current transformer-based systems can achieve.

More recently, models like Table-GPT, TableLlama, and TableLLM have been proposed

to advance the state of tabular data processing by leveraging large language models (LLMs). Table-GPT [Li et al., 2023] adapts the GPT architecture specifically for table tasks, demonstrating significant improvements in tasks such as table question answering, table-to-text generation, and table-based data augmentation. TableLlama [Zhang et al., 2023] focuses on creating open, large generalist models that handle a wide variety of table-related tasks. TableLLM [Zhang et al., 2024] targets real office usage scenarios, enabling LLMs to perform complex tabular data manipulations, including table joining, filtering, and aggregation.

Despite these advancements, LLMs like Table-GPT, TableLlama, and TableLLM face limitations. One significant drawback is their computational complexity and resource requirements, often requiring substantial data and computational power for training and inference, which can be impractical for many real-world applications. Additionally, while LLMs excel at handling specific tasks they are fine-tuned for, they may struggle with tasks involving intricate relationships across multiple tables, such as those required for CJN ranking. The integration of multiple data sources and the complexity of understanding the combined semantics of these sources remain challenging for current LLMs, highlighting the need for further advancements in this area.

Chapter 3

Lathe Overview

In this chapter we present an overview of Lathe. We begin by presenting a simple example of the task carried out by our system. For this, we illustrate in Figure 3.1 a simplified excerpt from the well-known IMDB[1].

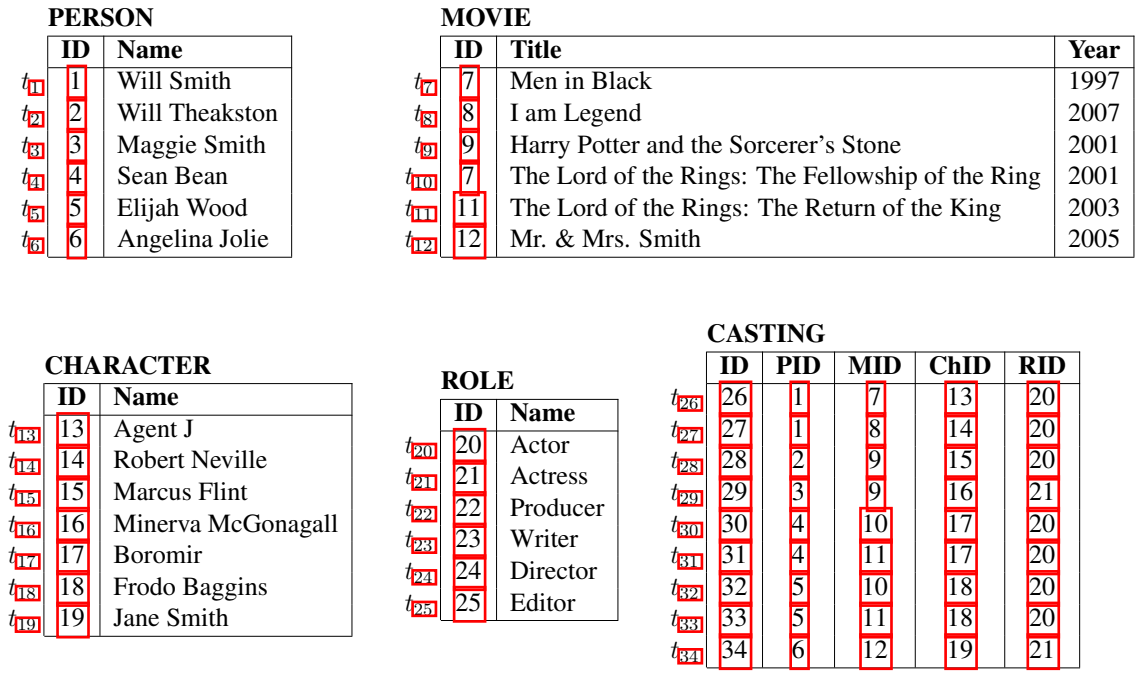


Figure 3.1: A simplified excerpt from IMDB

Consider that a user input the keyword query Q="$will\ smith\ films$", where the user wants the system to list the movies in which Will Smith appears. Notice that, informally, the terms "*will*" and "*smith*" are likely to match the contents of a relation from the DB, while the term "films" is likely to match the name of a relation or attribute.

[1]Internet Movie Database https://www.imdb.com/interfaces/

As other methods previously proposed in the literature, such as CN-Gen [Hristidis and Papakonstantinou, 2002] and MatCNGen [Oliveira et al., 2018, de Oliveira et al., 2020], the main goal of Lathe is, given a query such as Q, generating a SQL query that, when executed, fulfills the information needed for the user. The difference between Lathe and these previous methods is that they are not able to handle references to schema elements, such as "films" in Q.

For query Q, two of the possible SQL queries that would be generated are presented in Figures 3.2 (a) (S_1) and (b) (S_2), whose respective results for the database of Figure 3.1 are presented in Figures 3.2(c) and (d). In the query S_1, the keywords "will" and "smith" match the value of a single tuple of relation PERSON, while the keyword "films" matches the name of the relation MOVIE. As a result, S_1 retrieves the movies which the person Will Smith was in, and thus, satisfies the original user intent. As for query S_2, the keywords "will" and "smith" match values of two different tuples in relation PERSON, that is, they refer to two different persons. The keyword "films" matches the name of the relation MOVIE again. Therefore, S_2 retrieves movies in which two different persons, whose names respectively include the terms "*will*" and "*smith*", participated in. In these case, the persons are Will Theakston and Maggie Smith.

```
SELECT m.title, p.name
FROM person p
JOIN casting c ON p.id=c.person_id
JOIN movie m ON m.id = c.movie_id
WHERE p.name ILIKE '%will%'
AND p.name ILIKE '%smith%';
```

(a)

```
SELECT m.title, p1.name, p2.name
FROM person p1
JOIN casting c1 ON p1.id=c1.person_id
JOIN movie m ON m.id = c1.movie_id
JOIN casting c2 ON m.id = c2.movie_id
JOIN person p2 ON p2.id=c2.person_id
WHERE p1.name ILIKE '%will%'
AND p2.name ILIKE '%smith%'
AND p1.id<>p2.id;
```

(b)

m.title	p.name
Men in Black	Will Smith
I am Legend	Will Smith

(c)

m.title	p1.name	p2.name
Harry Potter and the Sorcerer's Stone	Will Theaskton	Maggie Smith

(d)

Figure 3.2: SQL queries generated for the keyword query "*will smith movies*" and their returned results.

As this example indicates, there may be several plausible SQL queries related to a given keyword query. Therefore, it is necessary to decide which alternative is more likely to fulfill the user intent. This task is also carried out by Lathe.

Next, we present an overview of the components and the functioning of Lathe.

3.1 System Architecture

In this section, we present the overall architecture of Lathe. We base our discussion on Figure 3.3, which illustrates the main phases that comprise the operation of the method.

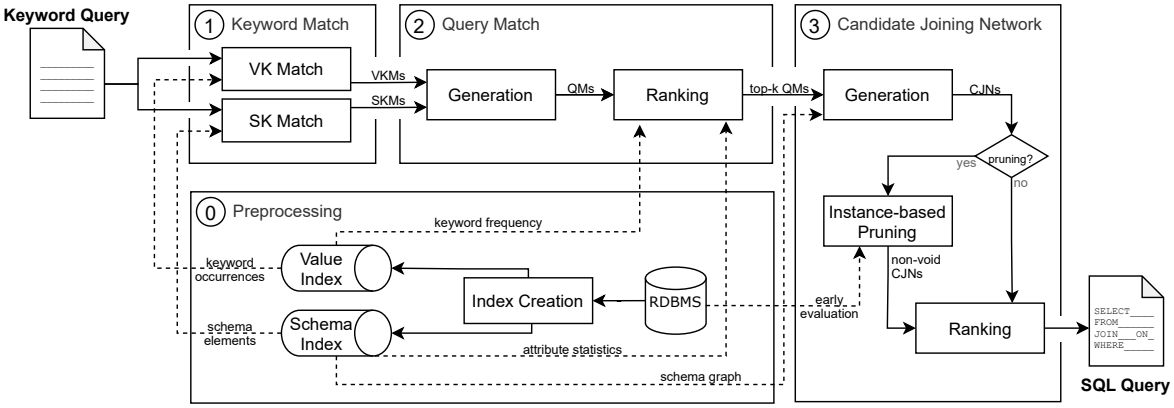


Figure 3.3: Main phases and architecture of Lathe

The process begins with an input keyword query posed by the user. The system then attempts to associate each of the keywords from the query with a database schema element, such as a relation or an attribute. The system relies on the DB schema, i.e., the names of relations and attributes, or on the DB instance, i.e., on the values of the attributes, for this. This phase, called *Keyword Matching* (1), generates sets of *Value-Keyword Matches* (VKMs), which associate keywords with sets of tuples whose attribute values contain these keywords, and *Schema-Keyword Matches* (SKMs), which associate keywords with names of relations or attributes deemed as similar to these keywords.

In Table 3.1 we show possible matches between keywords in the input query and the database elements. For example, the keywords "will smith" are found together in the values of the attribute `name` of the `PERSON` relation. The keyword "will" is also found alone in the values of `PERSON.name`, which is the case of the person Will Theakston present in instance shown in Figure 3.1. The term "*smith*" can refer to either the name of a person, the name of a character or even the title of a movie, in this case "Mr. & Mrs. Smith". Since these keywords are part of attribute values, these matches are considered VKMs. In the case of the keyword "*films*", it actually matches the *name* of the `Movie` relation, which is why in Table 3.1 the keyword "*films*" matches `MOVIE.self`. Thus, this match is considered an SKM. The *Keyword Matching* phase is detailed in Chapter 4.

In the next phase, *Query Matching* (2), Lathe generates combinations of VKMs and SKMs. In these combinations, we consider that all keywords in the query must be matched; in other words, the combination must be *total*. Furthermore, each combination must be a *minimal cover* for the keywords from the query, meaning that all pairs of keywords and attributes are "useful"; that is, if we remove any of the pairs, this would result in a non-total combination. In Figure 3.4 we present all possible QMs of the KMs illustrated in Table 3.1.

Although the Query Matching phase may generate a large number of QMs due to its combinatorial nature, only a few of them are useful in producing plausible answers to the user.

Table 3.1: Keyword matched for the query "*will smith films*"

Keywords	Type	Database Element	Algebra Expression
will smith	value	PERSON.name	$\sigma_{\texttt{name}\supseteq\{will,smith\}}(\texttt{PERSON})$
will	value	PERSON.name	$\sigma_{\texttt{name}\ni will}(\texttt{PERSON})$
smith	value	PERSON.name	$\sigma_{\texttt{name}\ni smith}(\texttt{PERSON})$
	value	CHARACTER.name	$\sigma_{\texttt{name}\ni smith}(\texttt{CHARACTER})$
	value	MOVIE.title	$\sigma_{\texttt{title}\ni smith}(\texttt{MOVIE})$
films	schema	MOVIE.self	MOVIE

will smith→PERSON.name
films→MOVIE
(a)

will→PERSON.name
smith→PERSON.name
films→MOVIE
(b)

will→PERSON.name
smith→CHARACTER.name
films→MOVIE
(c)

will→PERSON.name
smith→MOVIE.title
films→MOVIE
(d)

Figure 3.4: Examples of combinations of keywords matches that comprises a query match.

To address this, we propose the first algorithm for *Ranking Query Matches* in the literature. This ranking assigns a score to QMs based on their likelihood of satisfying the needs of the user when formulating the keyword query, ensuring that only the top-ranked QMs are processed in subsequent phases. By doing so, it avoids having to process less likely QMs. This method effectively filters out less likely QMs, enhancing efficiency.

We propose two distinct approaches for QM Ranking: a Bayesian method based on a Bayesian Belief Network (BBN)[de Cristo et al., 2003] and a neural method leveraging transformer-based models [Yin et al., 2020]. Details of QMs and their generation are presented in Chapter 5, the Bayesian ranking in Chapter 7, and the neural ranking in Chapter 8.

Lastly, in the *Candidate Joining Network Generation* (3) phase, the system searches for interpretations for the keyword query. That is, the system tries to connect all the keyword matches from the QMs through CJNs, which are based on the schema graph. CJNs can be thought as relational algebra joining expressions that can be directly translated into SQL queries. For instance, both the QMs shown in Figure 3.4 (a) and (b) can be connected using the CASTING relation, resulting in CJNs whose SQL translation is presented in Figure 3.2 (a) and (b), respectively.

Also, the system performs a *Candidate Joining Network Ranking* which favors CJNs that are more concise in terms of the number of relations they employ. We propose two distinct strategies for CJN Ranking: a Bayesian approach that benefits from the prior QM ranking and a neural approach that utilizes transformer-based models. Once we have identified the most

likely CJNs, they can be evaluated as SQL queries that are executed by a DBMS to the users. We notice that some of the generated CJNs may return empty results when they are evaluated. Thus, Lathe can use an *eager evaluation strategy* for pruning such void CJNs, which consists of evaluating CJNs before ranking them, and prune the ones that return empty results. We call this process *instance-based* pruning. Details of CJNs and their generation are presented in Chapter 6, the Bayesian ranking in Chapter 7, and the neural ranking in Chapter 8.

During the whole process of generating CJNs, Lathe uses two data structures which are created in a *Preprocessing stage* (0): the *Value Index* and the *Schema Index*.

The Value Index is an inverted index that stores keyword occurrences in the database, indicating the relations, attributes, and tuples where a keyword appears. These occurrences are retrieved to generate VKMs. Furthermore, the Value Index is used to calculate *term frequencies* for the QMs and CJNs Rankings. The Schema Index is an inverted index that stores database schema information, as well as statistics about relations and attributes. While database schema information, such as PK/FK relationships, are used for the generation of CJNs, the statistics about attributes, such as *norm* and *inverted frequency*, are used for rankings of QMs and CJNs.

In the following chapters we present each of the phases of Figure 3.3, describing the steps, definitions, data structures, and algorithms we used.

Chapter 4

Keyword Matching

In this chapter, we present the details on keyword matches and their generation. Their role in our work is to associate each keyword from the query to some attribute or relation in the database schema. Initially, we classify them as either VKMs and SKMs, according to the type of associations they represent. Later, we provide a generalization of the keyword matches and we introduce the concept of *Keyword-Free Matches*, which will be used in the next phases of our method.

4.1 Value-Keyword Matching

We may associate the keywords from the query to some attribute in the database schema based on the values of this attribute in the tuples that contain these keywords using *value-keyword matches*, according to Definition 1.

Definition 1. *Let Q be a keyword query and R be a relation state over the relation schema $R(A_1, \ldots, A_m)$. A **value-keyword match** from R over Q is given by:*

$$R^V[A_1^{K_1}, \ldots, A_m^{K_m}] = \{t | t \in R \wedge \forall A_i : W(t[A_i]) \cap Q = K_i\}$$

where K_i is the set of keywords from Q that are associated to the attribute A_i, $W(t[A_i])$ returns the set of words in t for attribute A_i and V denotes a match of keywords to the database values.

Notice that each tuple from the database can be a member of only one value-keyword match. Therefore, the VKMs of a given query are **disjoint sets of tuples**.

Throughout our discussion, for the sake of compactness in the notation, we often omit mappings of attributes to empty keyword sets in the representation of a VKM. For instance,

we use the notation $R^V[A_1^{K_1}]$ to represent $R^V[A_1^{K_1}, A_2^{\{\}}, \ldots, A_n^{\{\}}]$.

Example 1. *Consider the database instance of Figure 3.1. The following VKMs can be generated for the query "will smith films".*

$$PERSON^V[name^{\{will,smith\}}] = \{t_{11}\}$$
$$PERSON^V[name^{\{will\}}] = \{t_{12}\}$$
$$PERSON^V[name^{\{smith\}}] = \{t_{13}\}$$

VKMs play a similar role to the tuple-sets from related literature [Hristidis and Papakonstantinou, 2002, Oliveira et al., 2018]. They are, however, more expressive because they specify which attribute is associated with each keyword. Previous R-KwS systems based on the DISCOVER system, on the other hand, are unable to create tuple-sets that span multiple attributes [Hristidis and Papakonstantinou, 2002, Hristidis et al., 2003, Oliveira et al., 2015]. Example 2 shows a keyword query that includes more than one attribute.

Example 2. *Consider the query "lord rings 2001" whose intent is to return which Lord of the Rings movie was launched in 2001. We can represent it with the following value-keyword match:*

$$MOVIE^V[title^{\{lord,rings\}}, year^{\{2001\}}] = \{t_{10}\}$$

The generation of VKMs uses a structure we call the *Value Index*. This index stores the occurrences of keywords in the database, indicating the relations and tuples a keyword appears and which attributes are mapped to the keyword. Lathe creates the Value Index during a preprocessing phase that scans all target relations only once. This phase comes before the query processing and it is not expected to be repeated frequently. As a result, without further interaction with the DBMS, answers are generated for each query. The Value Index has following the structure, which is shown in Example 3.

$$I_V = \{term : \{relation : \{attribute : \{tuples\}\}\}\}$$

Example 3. *The VKMs presented in Example 1 are based on the following keyword occurrences:.*

$$I_V[will] = \{PERSON : \{name : \{t_{11}, t_{12}\}\}\}$$
$$I_V[smith] = \{PERSON : \{name : \{t_{11}, t_{13}\}\}\}$$

$$I_V[smith][PERSON] = \{name : \{t_1, t_3\}\}$$
$$I_V[smith][PERSON][name] = \{t_1, t_3\}$$

In Lathe, the generation of VKMs is carried out by the VKMGen algorithm, presented in details in Appendix A.

4.2 Schema-Keyword Matching

We may associate the keywords from the query to some attribute or relation in the database schema based on the name of the attribute or relation using *Schema-Keyword Matches*, according to Definition 2. Specifically, our method matches keywords to the names of relations and attributes using similarity metrics.

Definition 2. *Let $k \in Q$ be a keyword from the query, $R(A_1, \ldots, A_m)$ be a relation schema. A **schema-keyword match** from R over Q is given by:*

$$R^S[A_1^{K_1}, \ldots, A_m^{K_m}] = \{t | t \in R \wedge \forall k \in K_i : sim(A_i, k) \geq \varepsilon\}$$

where $1 \leq i \leq m$, K_i is the set of keywords from Q that are associated with the schema element A_i, $sim(A_i, k)$ gives the similarity between the name of a schema element A_i and the keyword k, which must be above a threshold ε, and S denotes a match of keywords to the database schema.

In this representation, we use the artificial attribute $self$ when we match a keyword to the name of a relation. Example 4 shows an instance of a schema-keyword match wherein the keyword "$films$" is matched to the relation $MOVIE$.

Example 4. *The following schema-based relation matches are created for the query "will smith films", considering a threshold $\varepsilon = 0.6$.*

$$MOVIE^S[self^{\{films\}}] = \{t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$$
$$MOVIE^S[title^{\{will\}}] = \{t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$$
$$PERSON^S[name^{\{smith\}}] = \{t_1, t_2, t_3, t_4, t_5\}$$

where $sim(a, b)$ gives the similarity between the schema element a and the keyword b, $sim(movie, films) = 1.00$, $sim(title, will) = 0.87$ and $sim(name, smith) = 0.63$.

Despite their similarity to VKMs, the schema-keyword matches serve a different purpose

in our method, ensuring that the attributes of a relation appear in the query results. As a result, they do not "filter" any of the tuples from the database, implying that they do not represent any selection operation over database relations.

Similarity Metrics

For the matching of keywords to schema elements, we used two similarity metrics based on the lexical database *WordNet*: the *Path similarity* [Miller, 1998, Pedersen et al., 2004] and the *Wu-Palmer similarity* [Wu and Palmer, 1994, Pedersen et al., 2004]. We introduce the WordNet database and the two similarity metrics below.

WordNet Database WordNet [Miller, 1998] is a large lexical database that resembles a thesaurus, as it groups words based on their meanings. One use of WordNet is to measure similarity between words based on the relatedness of their *senses*, the many different meanings that words can have [Keselj, 2009]. As a result, the word "film" can refer to a movie, as well as the act of recording or the plastic film. Each of these senses have a different relation to the sense of a "show". Wordnet represents sense relationships , such as *synonymy*, *hyponymy*, and *hypernymy*, to measure similarity between words. Synonyms are two word senses that share the same meaning. In addition, we say that the sense c_1 is a hyponym of the sense c_2 if c_1 is more specific, denoting a subclass of c_2. For instance, *"protagonist"* is a hyponym of *"character"*; *"actor"* is a hyponym of *"person"*, and *"movie"* is a hyponym of *"show"*. The hypernymy is the opposite of hyponymy relation. Thus, c_2 us a hypernymy of c_1.

Path Similarity The Path similarity [Miller, 1998, Pedersen et al., 2004] exploits the structure and content of the WordNet database. The relatedness score is inversely proportional to the number of nodes along the shortest path between the senses of two words. If the two senses are synonyms, the path between them has length 1. The relatedness score is calculated as follows:

$$sim_{path}(w_1, w_2) = \max_{\substack{c_1 \in senses(w_1) \\ c_2 \in senses(w_2)}} \left[\frac{1}{|shortest\_path(c_1, c_2)|} \right]$$

Wu-Palmer Similarity The Wu-Palmer measure (WUP) [Wu and Palmer, 1994, Pedersen et al., 2004] calculates relatedness by considering the depths of the two synsets c_1 and c_2 in the WordNet taxonomies, along with the depth of the *Least Common Subsumer* (LCS). The most specific synset c_3 is the LCS, which is the ancestor of both synsets c_1 and c_2. Because the depth of the LCS is never zero, the score can never be zero (the depth of the root of a taxonomy is one). Also, the score is 1 if the two input synsets are the same. The WUP similarity for two words w_1 and w_2 is given by:

$$sim_{wup}(w_1, w_2) = \max_{\substack{c_1 \in senses(w_1) \\ c_2 \in senses(w_2)}} \left[2 \times \frac{depth(lcs(c_1, c_2))}{depth(c_1, c_2)} \right]$$

As in the case of VKMs, we detail the SKMGen algorithm used in Lathe in Appendix B.

4.3 Generalization of Keyword Matches

Initially, we presented Definitions 1 and 2 which, respectively, introduce VKMs and SKMs. We chose to explain the specificity of these concepts separately for didactic purposes. They are, however, both components of a broader concept, *Keyword Match* (KM), which we define in Definition 3. In the following phases, this generalization will be useful when merging VKMs and SKMs.

Definition 3. *Let Q be a keyword query and R be a relation state over the relation schema $R(A_1, \ldots, A_m)$. Let $VKM = R^V[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]$ be a value-keyword match from R over Q. Let $SKM = R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]$ be a schema-keyword match from R over Q. A general* ***keyword match*** *from R over Q is given by:*

$$R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]^V[A_1^{K_1^V}, \ldots, A_m^{K_m^V}] = VKM \cap SKM$$

The representations of VKMs and SKMs in the general notation are given as follows:

$$R^S[A_1^{K_1}, \ldots, A_m^{K_m}] = R^S[A_1^{K_1}, \ldots, A_m^{K_m}]^V[A_1^{\{\}}, \ldots, A_m^{\{\}}]$$

$$R^V[A_1^{K_1}, \ldots, A_m^{K_m}] = R^S[A_1^{\{\}}, \ldots, A_m^{\{\}}]^V[A_1^{K_1}, \ldots, A_m^{K_m}]$$

Another concept in the generation of CJNs is *keyword-free matches*, which we describe in Definition 4. These are KMs that are not associated with any keyword but are used as auxiliary structures, such as intermediate nodes in CJNs.

Definition 4. *We say that a keyword match KM given by:*

$$KM = R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]^V[A_1^{K_1^V}, \ldots, A_m^{K_m^V}]$$

is a ***keyword-free match*** *if, and only if, $\nexists K_i^S \neq \{\} \wedge \nexists K_i^V \neq \{\}$, where $1 \leq i \leq m$.*

For the sake of simplifying the notation, we will represent a keyword-free match as $R^S[\,]^V[\,]$ or simply by R.

Chapter 5

Query Matching

In this chapter, we describe the processes of generating QMs, which are combinations of the keyword matches generated in the previous phases that comprise every keyword from the keyword query.

5.1 Query Matches Generation

We combine the associations present in the KMs to form total and non-redundant answers for the user. In other words, Lathe looks for KM combinations that satisfy two conditions: (i) every keyword from the query must appear in at least one of the KMs and (ii) if any KM is removed from the combination, the remaining combination no longer satisfies the first condition. These combinations, called *Query Matches* (QMs), are described in Definition 5.

Definition 5. *Let Q be a keyword query. Let $M = \{KM_1, \ldots, KM_n\}$ be a set of keyword matches for Q in a certain database instance I, where:*

$$KM_i = R_i^S[A_{i,1}^{K_{i,1}^S}, \ldots, A_{i,m_i}^{K_{i,m_i}^S}]^V[A_{i,1}^{K_{i,1}^V}, \ldots, A_{i,m_i}^{K_{i,m_i}^V}]$$

*Also, let $C_{KM_i} = \bigcup_{\substack{1 \leq j \leq m_i \\ X \in \{S,V\}}} K_{i,j}^X$ and $C_M = \bigcup_{1 \leq i \leq n} C_{KM_i}$ be the sets of all keywords associated with KM_i and with M, respectively. We say that M is a **query match** for Q if, and only if, C_M forms a **minimal set cover** of the keywords in Q. That is, $C_M = Q$ and $C_M \backslash C_{KM_i} \neq Q$, $\forall KM_i \in M$.*

Notice that a QM cannot contain any keyword-free match, as it would not be minimal anymore. Example 5 presents combinations of KMs which are or are not QMs.

Example 5. *Considering the KMs from the Examples 1 and 4, only some of the following sets*

are considered QMs for the query "will smith films":

$$M_1 = \{PERSON^V[name^{\{will,smith\}}], MOVIE^S[self^{\{films\}}]\}$$
$$M_2 = \{PERSON^V[name^{\{will\}}], PERSON^V[name^{\{smith\}}], MOVIE^S[self^{\{films\}}]\}$$
$$M_3 = \{PERSON^V[name^{\{will\}}], PERSON^V[name^{\{smith\}}]\}$$
$$M_4 = \{PERSON^V[name^{\{will,smith\}}], MOVIE^S[self^{\{films\}}], CHARACTER\}$$
$$M_5 = \{PERSON^V[name^{\{will,smith\}}], PERSON^V[name^{\{smith\}}], MOVIE^S[self^{\{films\}}]\}$$

The sets M_1 and M_2 are considered QMs. In contrast, the sets of keyword matches M_3, M_4 and M_5 are not QMs. While M_3 does not include all query keywords, M_4 and M_5 are not minimal, that is, they have unnecessary KMs.

We present the QMGen algorithm for generating QMs in Appendix C.

Chapter 6

Candidate Joining Networks

In this chapter we present the details on our method for generating and pruning Candidate Joining Networks (CJNs), which represent different interpretations of the keyword query. We recall that our definition of CJNs expands on the definition presented in [Hristidis and Papakonstantinou, 2002] to support keywords referring to schema elements.

The generation of CJNs uses a structure we call a *Schema Graph*. In this graph, there is a node representing each relation in the database and the edges correspond to the *referential integrity constraints* (RIC) in the database schema. In practice, this graph is built in a preprocessing phase based on information gathered from the database schema.

Definition 6. *Let $\mathcal{R} = \{R_1, \ldots, R_n\}$ be a set of relation schemas from the database. Let E be a subset of the ordered pairs from $\mathcal{R}^2$ given by:*

$$E = \{\langle R_a, R_b\rangle | \langle R_a, R_b\rangle \in \mathcal{R}^2 \wedge R_a \neq R_b \wedge RIC(R_a, R_b) \geq 1\}$$

*where $RIC(R_a, R_b)$ gives the number of Referential Integrity Constraints from a relation R_a to a relation R_b. We say that a **schema graph** is an ordered pair $G_S = \langle \mathcal{R}, E\rangle$, where $\mathcal{R}$ is the set of vertices (nodes) of G_S, and E is the set of edges of G_S.*

Example 6. *Considering the sample movie database introduced in Figure 3.1, our method generates the schema graph below.*

$$\begin{aligned} G_S =< & \{PERSON, MOVIE, CASTING, CHARACTER, ROLE\}, \\ & \{\langle CASTING, PERSON\rangle, \langle CASTING, MOVIE\rangle, \\ & \langle CASTING, CHARACTER\rangle, \langle CASTING, ROLE\rangle\} > \end{aligned}$$

In Figure 6.1, we represent a graphical illustration of G_S.

$$\begin{array}{ccccc} & & ROLE & & \\ & & \uparrow & & \\ PERSON & \leftarrow & CASTING & \rightarrow & MOVIE \\ & & \downarrow & & \\ & & CHARACTER & & \end{array}$$

Figure 6.1: A schema graph for the sample movie database of Figure 3.1

Once we defined the schema graph, we can introduce an important concept, the *Joining Network of Keyword Matches* (JNKM). Intuitively, a joining network of keyword matches J contains every KM from a query match M. Additionally, J may contain free-keyword matches, comprising the set F, which are necessary to connect all KMs in J for the sake of connectivity. Finally, J is a connected graph that is structured according to the schema graph G_S. The definition of joining network of keyword matches is given as follows:

Definition 7. *Let M be a query match for a keyword query Q. Let G_S be a schema graph. Let F be a set of keyword-free matches from the relations of G_S. Consider a graph of keyword matches $J = \langle \mathcal{V}, E \rangle$, where $\mathcal{V}$ and E are the vertices and edges of J. We say that J is a* ***joining network of keyword matches*** *from M over G_S if the following conditions hold:*

$$\begin{aligned} &i)\ \ \mathcal{V} = M \cup F \\ &ii)\ \forall KM_i \in \mathcal{V} : \exists \langle KM_a, KM_b \rangle \in E |\ i \in \{a, b\} \\ &iii) \forall \langle KM_a, KM_b \rangle \in E \implies \exists \langle R_a, R_b \rangle \in G_S \end{aligned}$$

For the sake of simplifying the notation, we will use a graphical illustration to represent JNKMs, which is shown in Example 7.

Example 7. *Considering the query match M_1 previously generated in Example 5, the following JNKMs can be generated:*

$$J_1 = PERSON^V[name^{\{will,smith\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$$

$$\begin{array}{ccccc} J_2 = PERSON^V[name^{\{will,smith\}}] & \leftarrow & CASTING & \rightarrow & MOVIE^S[self^{\{films\}}] \\ & & \downarrow & & \\ & & CHARACTER & & \end{array}$$

The JNKMs J_1 and J_2 cover the query match M_1. The interpretation of J_1 looks for the movies *of the* person *will smith. J_2 looks for the* movies *of the* person *will smith and which* character *will smith played in these movies.*

Notice that a JNKM might have unnecessary information for the keyword query, which

was the case of J_2 presented in Example 7. One approach to avoid generating unnecessary information is to generate Minimal Joining Networks of Keyword Matches (MJNKM), which are addressed in Definition 8. Roughly, a MJNKM cannot have any keyword-free match as a leaf, that is, a keyword-free match incident to a single edge.

Definition 8. *Let G_S be a schema graph. Let M be a query match for a query Q. We say that $J = \langle \mathcal{V}, E \rangle$ from M over G_S is **minimal joining network of keyword matches** (MJNKM) if, and only if, the following condition holds:*

$$\forall KM_i \in \mathcal{V} \; (\; \exists ! \langle KM_a, KM_b \rangle \in E | i \in \{a, b\} \implies KM_i \neq R_i^S[\,]^V[\,] \;)$$

Example 8. *Considering the query match M_2 previously generated in Example 5, the following MJNKMs can be generated:*

$$J_3 = PERSON^V[name^{\{smith\}}] \leftarrow CASTING \rightarrow PERSON^V[name^{\{will\}}]$$
$$\downarrow$$
$$MOVIE^S[self^{\{films\}}]$$

Another issue that a JNKM might have is representing an inconsistent interpretation. For instance, it is impossible for J_3 presented in Example 8 to return any results from the database. By Definition 1, the VKMs $PERSON^V[name^{\{will\}}]$ and $PERSON^V[name^{\{smith\}}]$ are disjoint. However, a tuple from $CASTING$ cannot refer to two different tuples of $PERSON$. Thus J_3 is inconsistent. We notice that previous work in literature for CJN generation had addressed this kind of inconsistency [Hristidis and Papakonstantinou, 2002, Oliveira et al., 2018]. They did not, however, consider the situation in which there exist more than one RIC from one relation to another. In contrast, based on the theorems and definitions presented in [Hristidis and Papakonstantinou, 2002], Lathe proposes a novel approach for checking consistency in CJNs that support such scenarios. Theorem 1 presents a criterion that determines when a JNKM is *sound*, that is, it can only produce JNTs that do not have more than one occurrences of a tuple. The proof of Theorem 1 is presented in Appendix E.

Theorem 1. *Let $G_S = \langle \mathcal{R}, E_G \rangle$ be a schema graph. Let $J = \langle \mathcal{V}, E_J \rangle$ be a joining network of keyword matches. We say that J is **sound**, that is, it does not have more than one occurrences of the same tuple for every instance of the database if, and only if, the following condition holds $\forall KM_a \in \mathcal{V}, \forall \langle R_a, R_b \rangle \in E_G$:*

$$RIC(R_a, R_b) \geq |\{KM_c | \langle KM_a, KM_c \rangle \in E_J \wedge R_c = R_b\}|$$

where $RIC(R_a, R_b)$ indicates the number of Referential Integrity Constraints from a relation

R_a to a relation R_b.

Example 9 presents a JNKM that is sound, although it would be deemed not sound by previous approaches [Hristidis and Papakonstantinou, 2002, Oliveira et al., 2018].

Example 9. *Consider a simplified excerpt from the MONDIAL database [May, 1999], presented in Figure 6.2. As there exists 2 RICs from the relation $BORDER$ to $COUNTRY$, represented by the attributes* Ctry1_Code *e* Ctry2_Code*, a tuple from $BORDER$ can be joined to at most two distinct tuples from $Country$, which is the case of $t_{35} \bowtie t_{38} \bowtie t_{36}$. Thus, the following MJNKM is sound:*

$$J_4 = COUNTRY^V[name^{\{colombia\}}] \leftarrow BORDER \rightarrow COUNTRY^V[name^{\{brazil\}}]$$

COUNTRY

	Code	Name	Capital_ID
t_{35}	CO	Colombia	1
t_{36}	BR	Brazil	2
t_{37}	PE	Peru	3

BORDER

	Ctry1_Code	Ctry2_Code	Length
t_{38}	CO	BR	1643
t_{39}	PE	BR	1560

CITY

	ID	Name	Population
t_{40}	1	Bogota	1643
t_{41}	2	Brasilia	1560
t_{42}	3	Lima	1560

Figure 6.2: A simplified excerpt from MONDIAL

Finally, Definition 9 describes a Candidate Joining Network (CJN), which is roughly a sound minimal joining network of keyword matches.

Definition 9. *Let M be a query match for the keyword query Q. Let G_S be a schema graph. Let CJN be a joining network of keyword matches from M over G_S given by $CJN = \langle \mathcal{V}, E \rangle$. We say that CJN is a* ***candidate joining network*** *if, and only if, CJN is minimal and sound.*

Example 10. *Considering the query match M_2 previously generated in Example 5, the following CJN can be generated:*

$$CJN_1 = MOVIE^S[self^{\{films\}}] \leftarrow CASTING \rightarrow PERSON^V[name^{\{will\}}]$$
$$\uparrow$$
$$CASTING \rightarrow PERSON^V[name^{\{smith\}}]$$

The candidate joining networks CJN_1 covers the query match M_2. CJN_1 is a minimal and sound JNKM. The interpretation of CJN_1 searches for the movies where both persons "will" (e.g. Will Theakston) and "smith" (e.g. Maggie Smith) participate in. The two keyword-free matches from the $CASTING$ are treated as different nodes in the candidate joining network CJN_1.

The details on how we generate CJNs in Lathe are described by the CNKMGen Algorithm in Appendix F.

6.1 Candidate Joining Network Pruning

In this section we present an *eager evaluation* strategy for pruning CJNs. Even if CJNs contain valid interpretations of the keyword query, some of them may fail to produce any JNTs as a result. Thus, we can improve the results of our CJN generation and ranking by pruning what we call *void* CJNs, which are CJNs with no JNTs in their results.

Example 11. *Considering the database instance of Figure 3.1 and the keyword query "will smith films", the following CJNs can be generated:*

$$CJN_2 = MOVIE^S[self^{\{films\}}]^V[name^{\{smith\}}] \leftarrow CASTING \\ \downarrow \\ PERSON^V[name^{\{will\}}]$$

$$CJN_3 = MOVIE^S[self^{\{films\}}] \leftarrow CASTING \rightarrow PERSON^V[name^{\{will\}}] \\ \uparrow \\ CASTING \rightarrow CHARACTER^V[name^{\{smith\}}]$$

The interpretation of CJN_2 looks for the movies whose name contains the keyword "smith" (e.g. "Mr. & Mrs. Smith") and in which a person whose contains "will" (e.g. "Will Theakston") participate in. The interpretation of CJN_3 looks for the movies where a person whose name contains "will" (e.g. "Will Theakston") played the character "smith" (e.g. "Jane Smith"). Notice that although the candidate joining networks CJN_2 and CJN_3 both provide valid interpretations for the keyword query, they do not produce any tuples as a result in the given database instance.

As most of the previous work does not rank CJNs but only evaluates them and ranks their resulting JNTs instead, the pruning of void CJNs has previously never been addressed. Lathe employs a pruning strategy that evaluates CJNs as soon as they are generated, pruning the void ones. This strategy, as demonstrated in our experiments, can significantly improve the quality of the CJN generation process, particularly in scenarios where the schema graph contains a large number of nodes and edges.

For instance, one of the datasets we use in our experiments, the MONDIAL database, contains a large number of relations and relational integrity constraint (RICs). This results in a schema graph with several nodes and edges, which, intuitively, incur a large number of possible CJNs for a single QM. In contrast, we discovered that such schema graphs are

prone to produce a large number of void CJNs. In particular, while approximately 20% of the keyword queries used in our experiments required us to consider 9 CJNs per QM, the eager evaluation strategy reduced this value to 2 CJNs per QM.

Notice, however, that to find if some CJN is void, we must execute it as an SQL in the DBMS, which incurs an additional cost and an increase in the CJN generation time. Despite that, we notice in our experiments that the eager evaluation strategy does not necessarily hinder the performance of a R-KwS system. In fact, reducing the number of CJNs per QM alone improves the system efficiency because this parameter influences the CJN generation process. Furthermore, the eager evaluation advances the CJN evaluation, which is already a required step in the majority of R-KwS systems in the related work. Lastly, we can set a maximum number of CJNs to probe during the eager evaluation, which limits the increase in CJN generation time.

Chapter 7

Bayesian Ranking

In this chapter, we explore the Bayesian ranking methodology used to assess the relevance of Query Matches (QMs) and Candidate Joining Networks (CJNs) within relational databases. This approach employs a Bayesian Belief Network (BBN) framework, integrating two key scoring mechanisms: the value score and the schema-based score. Traditionally, the Bayesian approach has been the primary method used in our system for ranking QMs and CJNs. However, recognizing the advancements in machine learning, we also investigate an alternative neural ranking approach in Chapter 8, which offers a modern perspective on information retrieval in relational databases.

The Bayesian ranking of CJNs is intricately linked to the ranking of QMs, with additional penalization applied to larger CJNs to balance relevance and complexity. By combining these elements, our ranking methodology aims to provide comprehensive and contextually relevant results to users, addressing the challenges of information retrieval in complex database environments. This chapter provides a detailed examination of the Bayesian ranking process, offering insights into its mechanisms and efficacy.

7.1 Query Matches Ranking

As described in Section 3, Lathe performs a ranking of the QMs generated in the previous step. This ranking is necessary because frequently many QMs are generated, yet only a few of them are useful for producing plausible answers to the user.

Lathe estimates the relevance of QMs based on a *Bayesian Belief Network* (BBN) model [Ribeiro and Muntz, 1996] for the current state of the underlying database. In practice, this model assesses two types of relevance when ranking query matches. The TF-IDF model is used to calculate the *value-based score*, which adapts the traditional Vector space model to the context of relational databases, as done in LABRADOR [Mesquita et al., 2007] and

CNRank [Oliveira et al., 2015]. The *schema-based score*, on the other hand, is calculated by estimating the similarity between keywords and schema elements names.

In Lathe, we consider only the top-k QMs for the succeeding phases. By doing so, we avoid generating CJNs that are less likely to properly interpret the keyword query.

Belief Bayesian Network

We adopted the BBN framework [Ribeiro and Muntz, 1996, de Cristo et al., 2003] for modeling distinct IR problems. This framework is simple and allows for the incorporation of features from distinct models into the same representational scheme. Other keyword search systems, such as LABRADOR [Mesquita et al., 2007] and CNRank [Oliveira et al., 2015], have also used it.

In our model, we interpret the QMs as documents, which are ranked for the keyword query. Figure 7.1 illustrates an example of the adopted Bayesian Network. The nodes that represent the keyword query are located at the top of the network, on the Query Side. The Database Side, located at the bottom of the network, contains the nodes that represent the QM that will be scored. The center of the network is present on both sides and it comprises sets of keywords: the set V of all terms present in the values of the database and the set S of all schema element names.

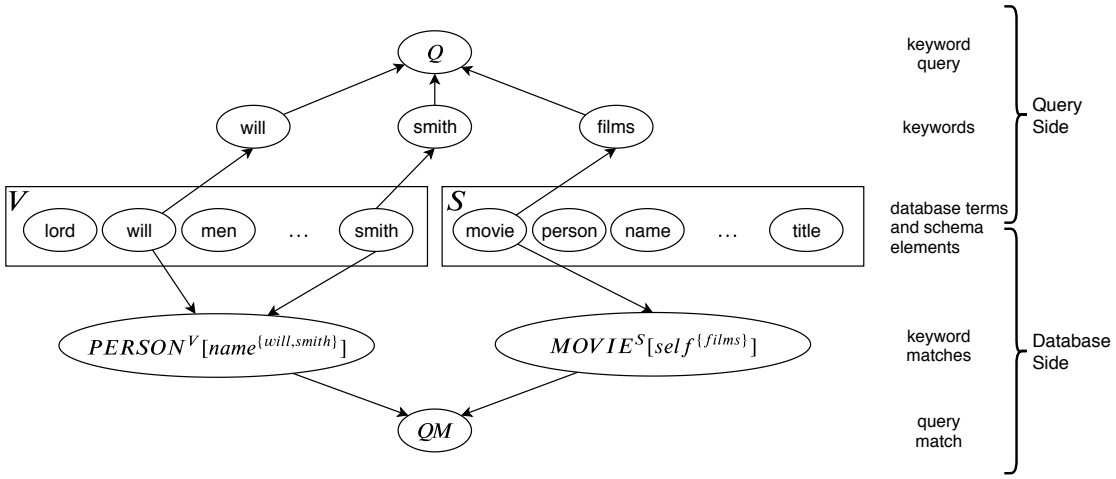


Figure 7.1: Bayesian Network corresponding to the query $Q = \{will, smith, films\}$

In our Bayesian Network, we rank QMs based on their similarities with the keyword query. This similarity is interpreted as the probability of observing a query match QM given the keyword query Q, that is, $P(QM|Q) = \mu P(QM \wedge Q)$, where $\mu = 1/P(Q)$ is a normalizing constant [Pearl, 2014].

Initially, we define a random binary variable associated with each keyword from the sets V and S, which indicates whether the keyword is observed in the keyword query. As

these random variables are the root nodes of our Bayesian Network, all of the probabilities of the other nodes depend on them. Therefore, if we consider $v \subseteq V$ and $s \subseteq S$ as the sets of keywords observed, we can derive the probability of any non-root node x as follows: $P(x) = P(x|v, s) \times P(v) \times P(s)$.

As all the possibilities of v and s are equally likely *a priori*, we can calculate them as $P(v) = (1/2)^{|V|}$ and $P(s) = (1/2)^{|S|}$, respectively.

The instantiation of the root nodes of the network separates the query match nodes from the query nodes, making them mutually independent. Therefore:

$$P(QM \wedge Q) = P(Q|v, s)P(QM|v, s)P(v)P(s)$$

The probability of the keyword query $Q = \{q_1, \ldots, q_{|Q|}\}$ is split between the probability of each of its keywords:

$$P(Q|v, s) = \prod_{1 \leq i \leq |Q|} P(q_i|v, s)$$

A keyword q_i from the query is observed, given the sets s and v, either if q_i occurs in the values of the database or if q_i has a similarity above a threshold ε with a schema element.

$$P(q_i|v, s) = (q_i \in v) \veebar (\exists k \in s : sim(q_i, k) \geq \varepsilon)$$

Similarly, in our network, the probability of a query match QM is split between the probability of each of its KMs.

$$P(QM|v, s) = \prod_{1 \leq i \leq |QM|} P(KM_i|v, s)$$

We compute the probability of KMs using two different metrics: a *schema score* based on the same similarities used in the generation of SKMs; and a *value score* based on a Vector model [Baeza-Yates and Ribeiro-Neto, 2008, Salton and Buckley, 1988] using the cosine similarity.

$$P(KM_i|v, s) = \prod_{\substack{1 \leq j \leq m_i \\ K^V_{i,j} \neq \emptyset}} cos(\overrightarrow{A_{i,j}}, \overrightarrow{v \cap K^V_{i,j}}) \prod_{\substack{1 \leq j \leq m_i \\ K^S_{i,j} \neq \emptyset}} \frac{\sum_{t \in s \cap K^S_{i,j}} sim(A_{i,j}, t)}{|s \cap K^S_{i,j}|}$$

where $KM_i = R^S_i[A^{K^S_{i,1}}_{i,1}, \ldots, A^{K^S_{i,m_i}}_{i,m_i}]^V[A^{K^V_{i,1}}_{i,1}, \ldots, A^{K^V_{i,m_i}}_{i,m_i}]$.

It is important to distinguish the documents from the Bayesian Network model and the documents from the Vector Model. The former are QMs, and the query is the keyword query

itself, whereas the documents from the Vector model are database attributes, and the query is the set of keywords associated with the KM.

Once we know the document and the query of the Vector model, we can calculate the cosine similarity by taking the inner product of the document and the query. The cosine similarity formula is given as follows:

$$cos(\overrightarrow{A_{i,j}}, \overrightarrow{v \cap K_{i,j}^V}) = \frac{\overrightarrow{A_{i,j}^V} \cdot \overrightarrow{v \cap K_{i,j}^V}}{|\overrightarrow{A_{i,j}}| \times |\overrightarrow{v \cap K_{i,j}^V}|} = \alpha \times \frac{\sum_{t \in V} w(\overrightarrow{A_{i,j}}, t) \times w(\overrightarrow{v \cap K_{i,j}^V}, t)}{\sqrt{\sum_{t \in V} w(\overrightarrow{A_{i,j}}, t)^2}}$$

where $\alpha = 1/(\sum_{t \in V} w(\overrightarrow{v \cap K_{i,j}^V}, t)^2)^{1/2}$ is the constant that represents the norm of the query, which is not necessary for the ranking.

The weights for each term are calculated using the TF-IDF measure. This measure is based on the term frequency and specificity in the collection. We use the *raw frequency* and *inverse frequency*, which are the most recommended forms of TF-IDF weights [Baeza-Yates and Ribeiro-Neto, 2008].

$$w(\vec{X}, t) = freq_{X,t} \times \log \frac{N_A}{n_t}$$

where $\vec{X} \in \{\overrightarrow{A_{i,j}}, \overrightarrow{v \cap K_{i,j}^V}\}$ can be either the document or the query, N_A is the number of attributes in the database, and n_t is the number of attributes that are mapped to the occurrences of the term t. In the case of $\vec{X}$ being the query, $freq_{X,t}$ gives the number of occurrences of a term t in the keyword query, which is generally 1. In the case of $\vec{X}$ being an attribute (document), $freq_{X,t}$ gives the occurrences of a term t in an attribute, which is obtained from the Value Index.

We present the bayesian algorithm for ranking QMs in Appendix D.

7.2 Candidate Joining Networks Ranking

In this section, we introduce CJNKMRank, a novel ranking method for CJNs based on the ranking of QMs as detailed in Algorithm 11. This approach is necessary because many CJNs are often generated, yet only a few produce relevant answers.

As described in Section 7.1, our QM ranking advances many of the features present in other systems, such as CNRank [Oliveira et al., 2015]. By leveraging the scores of QMs, CJNKMRank provides a straightforward yet effective ranking of CJNs. This is achieved by applying a penalization for larger CJNs, ensuring a balance between relevance and complexity.

Therefore, the score of a candidate joining network CJN_M from a query match M is calculated as:

$$score(CJN_M) = score(M) \times \frac{1}{|CJN_M|}$$

To maintain the order of CJNs with identical scores, a stable sorting algorithm [Cormen et al., 2009] is used, as described in Line 6 of Algorithm 1.

Algorithm 1: CJNKMRank(QM)

Input: A set of candidate joining networks CJN
Output: The set of candidate networks $RCJN$

```
1 RCJN ← [ ]
2 for C ∈ RCN do
3     let M be the query match used to generate C
4     cjn_score = score(M)/|C|
5     RCJN.append( ⟨cjn_score, C⟩ )
6 Sort RCN in descending order
7 return RCJN
```

Chapter 8

Neural Ranking

As way of improving the results obtained with the Bayesian ranking approach from Chapter 7, in this chapter, we introduce two distinct approaches for ranking QMs and CJNs, leveraging transformer-based models. Our motivation includes exploring the recent successes of these models in various information retrieval tasks [Devlin et al., 2018, Fang et al., 2024, Reimers and Gurevych, 2019] to investigate ways for enhancing the performance of the IR-based approach presented in Chapter 7. The neural ranking offers a modern alternative that could further improve the relevance of the results, specially in consistently identifying the most relevant answers which is a challenge for the Bayesian approach.

In a first approach, we employ pre-trained neural language models to capture the similarity between a keyword query and either a QM or a CJN. Notice that our QMs and CJNs span information from multiple database tables and that Neural Language Models are optimized for processing text inputs. Hence, we must adequately convert the structured data to a suitable textual format allowing the Language Models to perform tabular data understanding. Inspired by previous approaches such as TaBERT [Yin et al., 2020] and StruBERT [Trabelsi et al., 2022], as detailed in Section 8.2 we designed a *linearization* process to translate our QMs and CJNs into sentence structures. However, different from them, our linearization extends beyond individual tables, encompassing views that span information from multiple database tables.

In a second approach, we aim to further improve the effectiveness of the pre-trained models by fine-tuning them for the task of ranking QMs and CJNs for a given keyword query [Reimers and Gurevych, 2019]. We anticipate that the fine-tuned models exhibit enhanced capability in understanding the relations between keyword queries and the tables structures, retrieving a better QM and CJN ranking when compared to the other approaches. As the fine-tuning process requires annotated training data, we also propose a data augmentation technique to provide such data, reducing the reliance on manually annotated

samples[Thakur et al., 2021]. This approach enhances the ability of the model to generalize across diverse QM and CJN scenarios, thereby enhancing keyword search over relational databases.

With the introduction of neural ranking, Lathe can now be executed following different pipelines, each employing different QM and CJN ranking approaches. Figure 8.1 illustrates five pipelines. The first pipeline uses the Bayesian approach for ranking both QMs and CJNs. The second pipeline combines Bayesian QM ranking with a pre-trained neural CJN ranking. The third pipeline uses Bayesian QM ranking and fine-tuned neural CJN ranking. The fourth pipeline integrates fine-tuned neural QM ranking with pre-trained neural CJN ranking. Finally, the fifth pipeline employs fine-tuned neural ranking for both QMs and CJNs.

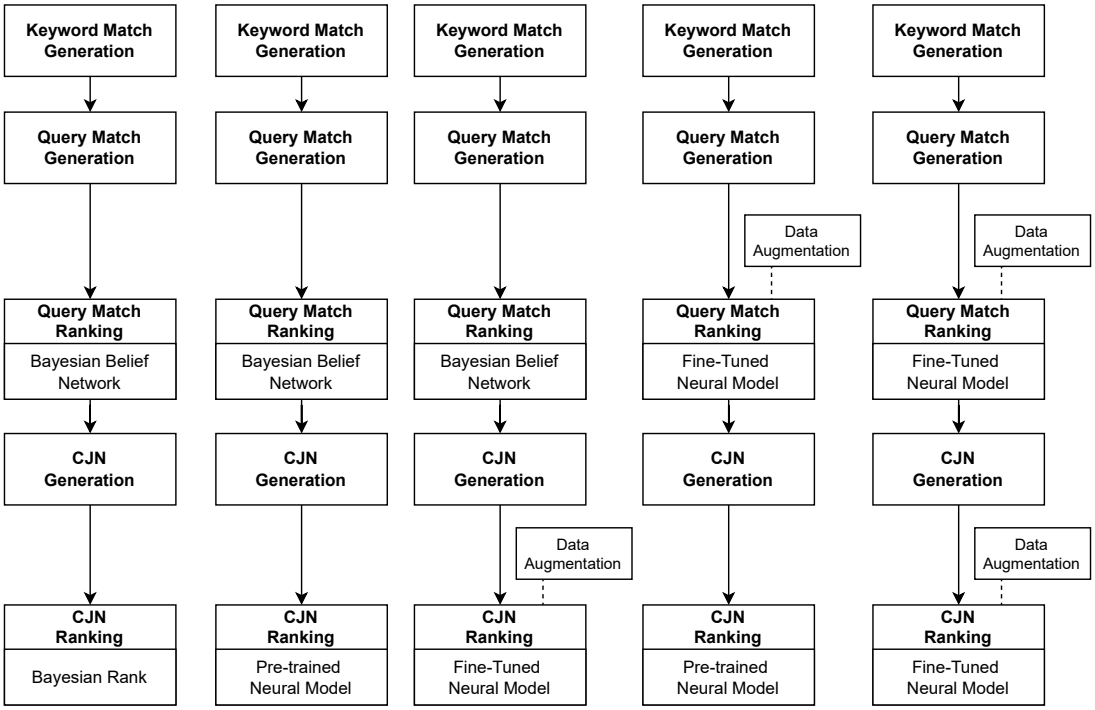


Figure 8.1: Pipelines for Lathe

The structure of this chapter is as follows: Section 8.1 provides an overview of the QM and CJN neural ranking. Section 8.2 details the linearization process for keyword queries, QMs, and CJNs for their use in sentence-transformer models. Subsequently, Section 8.3 outlines the adaptation of pre-trained sentence-transformer models for QM and CJN ranking, utilizing specific training examples to fine-tune the models for improved relevance assessment. Finally, Section 8.4 presents data augmentation techniques for generating synthetic examples to enhance the fine-tuning process.

8.1 Neural Ranking

In our system, we employ similar methodologies for ranking both QMs and CJNs concerning a given keyword query. Although occurring at different stages, the processes share a common foundation, leveraging sentence-transformer models for effective comparison and relevance assessment.

8.1.1 QM Ranking

The QM ranking process utilizes sentence-transformer models to rank QMs for a given keyword query. This systematic process is detailed in Algorithm 2, and it involves linearization, embedding generation, and similarity-based ranking.

Algorithm 2: NeuralQMRank(Q,QM)

Input: A keyword query Q
A set of query matches QM
Output: The set of ranked query matches RQM

1 $RQM \leftarrow [\,]$
2 **let** $Model$ be the sentence-transformer model
3 $S_Q \leftarrow sentence(Q)$
4 $E_Q \leftarrow Model.encode(S_Q)$
5 **for** $M_i \in QM$ **do**
6 $\quad S_{M_i} \leftarrow sentence(M_i)$
7 $\quad E_{M_i} \leftarrow Model.encode(S_{M_i})$
8 $\quad score \leftarrow sim(E_Q, E_{M_i})$
9 $\quad RQM$.append($\langle score, M_i \rangle$)
10 **Sort** RQM in descending order
11 **return** RQM

First, the algorithm linearizes both the keyword query and the QMs into structured sentences (Lines 3 and 6), as described in Section 8.2. This step ensures that the information contained within the keyword query and the QMs is properly structured and represented in a textual format compatible with subsequent analysis.

Next, the algorithm utilizes a sentence-transformer model to generate embeddings for both the linearized keyword query and the linearized QMs (Lines 4 and 7). These embeddings encapsulate the semantic representations of the sentences, facilitating meaningful comparison and analysis. The subsequent step involves computing the similarity between the embedding representing the keyword query and the embeddings representing the QMs to determine the relevance of each QM to the keyword query (Lines 8-9). The similarity measure can be either cosine similarity or dot product similarity, depending on the specific sentence-transformer

model used. Finally, the algorithm sorts the QMs in descending order based on their similarity (Line 10). This ranking ensures that the most semantically aligned QMs with the keyword query are prioritized, thus improving the relevance of the search results returned to the user.

8.1.2 CJN Ranking

The neural CJN ranking is carried out by Algorithm 3. A notable distinction in the ranking process of CJNs lies in the linearization step, which differs from that of QMs. To rank the CJNs for a given keyword query, the algorithm first executes queries generated from them against the database, obtaining the database views they return (Line 6). It then linearizes the rows of these views, transforming the tabular data into structured sentences. Next, the algorithm applies aggregation techniques to generate a single embedding representation from all sentences obtained from the CJN (Lines 7-16). This Linearization process for CJNs is detailed in Section 8.2.2. The embedding encapsulates the semantic representations of the sentences, facilitating meaningful comparison and relevance assessment.

The algorithm then computes the similarity between the embedding representing the keyword query and the aggregated embedding representing the CJN to determine the relevance of each CJN to the keyword query (Lines 17-18). It uses either cosine similarity or dot product similarity, depending on the specific model employed.

Finally, the algorithm sorts the CJNs in descending order based on their similarity scores, ensuring that it prioritizes the most semantically aligned CJNs with the keyword query, thus enhancing the relevance of the search results returned to the user (Line 19). Despite the nuances in linearization, the overall ranking methodology for CJNs remains consistent with that of QMs, underscoring the robustness and adaptability of our approach.

8.2 Linearization

This section details the linearization process for the keyword query, the QMs and CJNs, crucial for facilitating compatibility with language models.

The keyword query linearization consists of translating the keyword query as a sentence "query: Q", where Q is the keyword query. Then, we encode it with the sentence-transformer model to generate embedding for the keyword query. Figure 8.2 exemplifies the sentence for the query "Will Smith films".

Q	Will Smith films
$sentence(Q)$	query: Will Smith films

Figure 8.2: Sentence translation of the keyword query

Algorithm 3: NeuralCJNRank(Q,CJN)

Input: A keyword query Q
A set of candidate joining networks CJN
A literal variable indicating the aggregation approach agg
Output: The set of ranked candidate joining networks $RCJN$

$RCJN \leftarrow [\,]$
let $Model$ be the sentence-transformer model
$S_Q \leftarrow sentence(Q)$
$E_Q \leftarrow Model.encode(S_Q)$
for $C \in CJN$ **do**
  let $view$ be the resulting view when running C against the database.
  if $agg =$ "mean" **then**
    $E_{view} \leftarrow \{\,\}$
    for $row \in view$ **do**
      $S_{row} \leftarrow row\_sentence(row)$
      $E_{row} \leftarrow Model.encode(S)$
      $E_{view} \leftarrow E_{view} \cup E_{row}$
    $E_C \leftarrow mean(E_{view})$
  else
    $S_{view} \leftarrow multivalue\_sentence(view)$
    $E_C \leftarrow Model.encode(S_{view})$
  $score \leftarrow sim(E_Q, E_C)$
  $RCJN$.append($\langle score, C \rangle$)
Sort $RCJN$ in descending order
return $RCJN$

QM linearization involves translating keyword matches from the QMs into structured sentences, followed by encoding using sentence-transformer models.

Since the results of CJNs when executed against a database can be interpreted as database views, we explore techniques designed for tabular data to encode them. Specifically, we translate each row from the resulting view into a sentence. Then, we employ different methods for aggregating these sentences into a single representation, which serves as the CJN representation. These aggregation methods encompass strategies for consolidating the row-level information retrieved from the database, allowing us to capture essential relational nuances embedded within CJNs.

With these encoding methodologies elucidated we establish a foundation for subsequent chapters by leveraging the encoded data for neural network-based ranking. This comprehensive approach promises to optimize keyword search efficiency and accuracy.

8.2.1 QM Linearization

The linearization of QMs involves translating them into sentences and then encoding them with sentence-transformer models. This translation process is straightforward: each keyword mapping from the QM is represented in the format "table.attribute.type: keywords", separated by a pipe symbol. Subsequently, these sentences are encoded using such models. Example 12 presents sentences generated from QMs.

Example 12. *Considering the query matches M_1 and M_2 previously generated in Example 5, their sentences can be generated as shown in Figure 8.3:*

M_1	$\{PERSON^V[name^{\{will,smith\}}], MOVIE^S[self^{\{films\}}]\}$
$sentence(M_1)$	answer: person.name.value: will smith ǀ movie.self.schema: films
M_2	$\{PERSON^V[name^{\{will\}}], PERSON^V[name^{\{smith\}}], MOVIE^S[self^{\{films\}}]\}$
$sentence(M_2)$	answer: person.name.value: will ǀ person.name.value: smith ǀ movie.self.schema: films

Figure 8.3: Sentence translation of query matches.

8.2.2 CJN Linearization

The linearization of CJNs involves translating the rows of database views into individual sentences, capturing the relevant information contained within each row. We employ a snapshot approach similar to that used in TaBERT [Yin et al., 2020] and StruBERT [Trabelsi et al., 2022] to handle large database views efficiently. This technique entails selecting a subset of rows from the database view to ensure it remains within the token capacity constraints of transformer-based models. Once the snapshot of the CJN is obtained, we employ aggregation techniques to generate a single embedding of the CJN. We explore two primary approaches for aggregation: mean and combination.

Mean Approach In the mean approach, we encode the sentence for each row individually and then compute the average embedding of all the row embeddings, resulting in a single embedding that represents the entire CJN Figure 8.4 presents the steps for the mean approach. The sentences generated for CJNs using this approach are shown in the Example 13.

Combination Approach The combination approach involves aggregating the information from multiple rows in the snapshot into a single comprehensive sentence, then encoding this aggregated sentence to represent the CJN. Figure 8.5 presents the steps for the combination approach. This approach aggregates the rows using either the Multivalue technique, which concatenates the values of each attribute across all rows within the snapshot, then we generate a single comprehensive sentence.

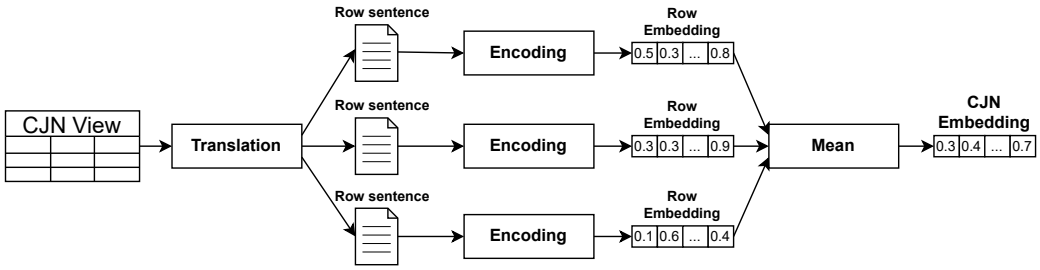


Figure 8.4: Mean Approach for CJN Linearization

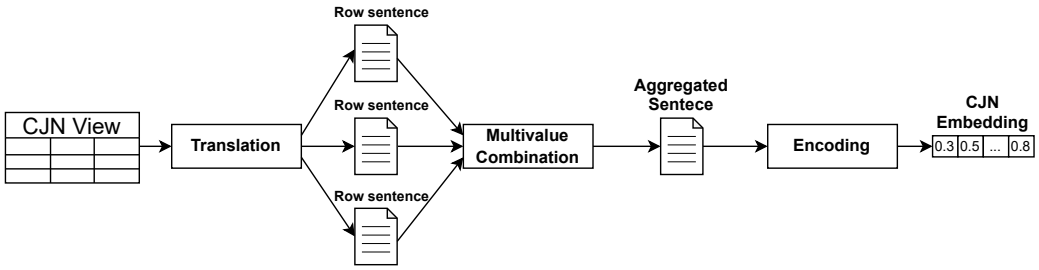


Figure 8.5: Combination Approach for CJN Linearization

The sentences generated for CJNs using this approach are also shown in the Example 13.

Example 13. *Considering the query matches M_1 and M_2 encoded in Example 12, we can generate the candidate joining networks CJN_1 and CJN_2, shown with their respective results in Figure 8.6. Based on a snapshot, with the three first rows of the results, we are able to generate the sentences shown in Figure 8.7.*

$$CJN_1 = PERSON^V[name^{\{will,smith\}}] \uparrow CASTING \downarrow MOVIE^S[self^{\{films\}}]$$

$$CJN_2 = MOVIE^S[self^{\{films\}}] \leftarrow CASTING \rightarrow PERSON^V[name^{\{will\}}] \uparrow CASTING \rightarrow PERSON^V[name^{\{smith\}}]$$

Results for CJN_1	
t3.title	**t1.name**
Bad Boys	Smith, Will
Enemy of the State	Smith, Will
Free Enterprise	Smith, Will
Ali	Smith, Will
A Closer Walk	Smith, Will

Results for CJN_2			
t3.year	**t3.title**	**t1.name**	**t5.name**
1944	The Last Horseman	Wills, Luke	Smith, Tom
1977	Looking Up	Hussing, Will	Smith, Andrew
1977	Who Has Seen the Wind	Woods, Will	Smith, Cedric
1981	Urgh! A Music War	Sergeant, Will	Smith, Barry
1999	The Lost Son	Welch, Will	Smith, Rachel Quigley

Figure 8.6: CJNs and their results returned from the database.

It is noteworthy that while the results for CJN_1 exhibit distinct rows, representing them as sentences reveals some duplicates. This occurrence arises because the keyword "films" refers to the table name rather than its values. This observation underscores the nuanced challenges in accurately representing CJN results through sentence encoding.

Aggr. Approach	Sentences for CJN_1	Sentences for CJN_2
Mean	answer: person.name: Smith, Will | movie: films	answer: person.name: Wills, Luke | movie: films | person.name: Smith, Tom
	answer: person.name: Smith, Will | movie: films	answer: person.name: Hussing, Will | movie: films | person.name: Smith, Andrew
	answer: person.name: Smith, Will | movie: films	answer: person.name: Woods, Will | movie: films | person.name: Smith, Cedric
Combination	answer: person.name: Smith, Will, Smith, Will, Smith, Will | movie: films	answer: person.name: Wills, Luke, Hussing, Will, Woods, Will | movie: films | person.name: Smith, Tom, Smith, Andrew, Smith, Cedric

Figure 8.7: Sentence representations for the candidate joining networks CJN_1 and CJN_2.

8.3 Fine-Tuning

In this section, we present our ranking approach aligned to fine-tuning sentence-transformer models, initially pre-trained on extensive and general corpora. As shown previously, these models discern semantic similarities and allows ranking relevant QMs and CJNs for keyword queries. However, due to their generalized nature these pre-trained models do not hold weights for too specific domains and tasks. State-of-the-art works have shown that the performance of Language Models on specific tasks and domains are improved when these models are adapted with fine-tuning [Reimers and Gurevych, 2019, Devlin et al., 2018]. Hence, we fine-tune them for QM and CJN ranking, enabling the language models to better discern pertinent patterns and relationships within the linearized data that holds structured information. This section outlines the methodology and significance of fine-tuning, elucidating the intricacies of optimizing their performance for targeted tasks.

8.3.1 QM Ranking Fine-tuning

In the fine-tuning process for QM ranking, sentence-transformer models are adapted using specific training examples. These examples consist of tuples comprising the keyword query representation, the QM representation, and the similarity between them.

We rely on the Bayesian model, in Section 7, to compute the relevance score of a QM and then build the training set we use for fine-tuning. A score of 1 indicates relevance to the keyword query, while a sigmoid function of the score in the Bayesian model is applied otherwise. Additionally, a fixed weight of 0.4 is added to ensure negative examples are appropriately accounted for. The calculation of the score follows this formula:

$$Sim(Q, M) = \begin{cases} 1, & \text{if } M \text{ is relevant for } Q \\ \dfrac{1}{1 + e^{-0.4 \times bayesian\_score(Q,M)}}, & \text{otherwise} \end{cases}$$

where $bayesian\_score(Q, M)$ denotes the score of the query match M for query Q using the traditional Bayesian approach.

Example 14. *Considering the sentences for the query matches M_1 and M_2, shown in Figure 8.3. We are able to generate the two train examples shown in Figure 8.8*

Positive Example	("query: Will Smith films", "answer: person.name.value: will smith | movie.self.schema: films", score=1.0)
Negative Example	("query: Will Smith films", "answer: person.name.value: will | person.name.value: smith | movie.self.schema: films", score=0.22)

Figure 8.8: Training examples for the QM ranking fine-tuning

8.3.2 CJN Ranking Fine-tuning

Similarly, in the fine-tuning process for CJN ranking, sentence-transformer models are adapted using specific training examples. However, as CJNs may generate several sentences, there may be several training examples for a single CJN. These examples consist of tuples comprising the keyword query representation, the CJN representation, and the similarity between them.

Again, we rely on the Bayesian model, in Section 7, to compute the relevance score of a CJN and then build the training set we use for fine-tuning. A score of 1 indicates relevance to the keyword query, while a sigmoid function of the score in the Bayesian model is applied otherwise. Additionally, a weight of 0.4 is added to ensure negative examples are appropriately accounted for. The calculation of the score follows this formula:

$$Sim(Q, S) = \begin{cases} 1, & \text{if } CJN \text{ is relevant for } Q \\ \dfrac{1}{1 + e^{-0.4 \times bayesian\_score(Q, CJN)}}, & \text{otherwise} \end{cases}$$

where S is a sentence for the candidate joining network CJN, and $bayesian\_score(Q, M)$ denotes the score of CJN for query Q using the traditional Bayesian approach.

Example 15. *Considering the sentences for the query matches CJN_1 and CJN_2, shown in Figure 8.7. We are able to generate the train examples shown in Figure 8.9*

8.4 Data Augmentation

In order to facilitate the fine-tuning process, which necessitates a robust set of training examples comprising keyword queries, their respective relevant QMs, and CJNs, we employed

Aggr. Approach	Positive Examples	Negative Examples
Mean	("query: Will Smith films", "answer: person.name: Smith, Will | movie: films", score=1.0)	("query: Will Smith films", "answer: person.name: Wills, Luke | movie: films | person.name: Smith, Tom", score=0.044)
	("query: Will Smith films", "answer: person.name: Smith, Will | movie: films", score=1.0)	("query: Will Smith films", "answer: person.name: Hussing, Will | movie: films | person.name: Smith, Andrew", score=0.044)
	("query: Will Smith films", "answer: person.name: Smith, Will | movie: films", score=1.0)	("query: Will Smith films", "answer: person.name: Woods, Will | movie: films | person.name: Smith, Cedric", score=0.044)
Multivalue	("query: Will Smith films", "answer: person.name: Smith, Will, Smith, Will, Smith, Will | movie: films", score=1.0)	("query: Will Smith films", "answer: person.name: Wills, Luke, Hussing, Will, Woods, Will | movie: films | person.name: Smith, Tom, Smith, Andrew, Smith, Cedric", score=0.044)

Figure 8.9: Sentence representations for the candidate joining networks CJN_1 and CJN_2.

a data augmentation strategy. Given the importance of having a diverse and comprehensive dataset for effective model training, data augmentation plays a pivotal role in enriching the available training examples. One of the challenges in this context is the lack of annotated data for training. Thus, one of our goals is to minimize the need for manually labeling data by leveraging data augmentation techniques to generate a wider variety of training examples.

The data augmentation process involve four main steps: (i) Extraction of CJN Templates; (ii) Generation of new keyword queries and CJNs; (iii) Run the keyword search for each query; (iv) Generate sentences for each QM or CJN generated by Lathe for that query.

Given a keyword query and its relevant CJN, we initiate the augmentation process by extracting CJN templates. A template denotes a CJN wherein all keywords are replaced by wildcards, represented by the symbol '?'. This step facilitates the creation of generalized structures that encapsulate the essence of query semantics without being bound to specific keywords.

Example 16. *Considering the candidate joining networks CJN_1, which was generated in Example 13 and it is the relevant CJN for the query "Will Smith films". We can extract the following template T_1, from CJN_1 by replacing its keywords with a wildcard '?'. This template selects information on all the movies for all persons, and its results are shown in Figure 8.10.*

$$CJN_1 = PERSON^V[name^{\{will,smith\}}] \quad T_1 = PERSON^V[name^{\{?\}}]$$
$$\uparrow \qquad\qquad\qquad\qquad \uparrow$$
$$CASTING \qquad\qquad\qquad CASTING$$
$$\downarrow \qquad\qquad\qquad\qquad \downarrow$$
$$MOVIE^S[self^{\{films\}}] \qquad MOVIE^S[self^{\{?\}}]$$

Next, we run the SQL statement derived from T_1 against the database, which returns the rows shown in Figure 8.10. Then, for each row, we are able to generate a new keyword query, and its relevant CJN and QM. First, we use a subset of the row values to generate a keyword query. Second, we generate the relevant CJN by filling the wildcards with keywords

Results for T_1		
t1.name	**t3.title**	**t3.year**
Hues, Jack	The Guardian	1990
Coote, Robert	The House of Fear	1939
O'Halloran, Jack	The Flintstones	1994
Zorn, John	Notes on Marie Menken	2006
Kern, Robert	Plymouth Adventure	1952

Figure 8.10: Results for the template T_1.

from the query. Third, the relevant QM is the set of non-free keyword matches from the CJN. Figure 8.11 shows five keyword queries and their answers, which were generated from template T_1.

Keyword Queries and Answers generated for template T_1		
1	Query	Hues Jack films
	CJN	$PERSON^V[name^{\{hues,jack\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$
	QM	$\{PERSON^V(name^{\{hues,jack\}}), MOVIE^S(self^{\{films\}})\}$
2	Query	Coot Robert films
	CJN	$PERSON^V[name^{\{coot,robert\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$
	QM	$\{PERSON^V(name^{\{coot,robert\}}), MOVIE^S(self^{\{films\}})\}$
3	Query	Halloran Jack films
	CJN	$PERSON^V[name^{\{halloran,jack\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$
	QM	$\{PERSON^V(name^{\{halloran,jack\}}), MOVIE^S(self^{\{films\}})\}$
4	Query	Zorn John films
	CJN	$PERSON^V[name^{\{zorn,john\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$
	QM	$\{PERSON^V(name^{\{zorn,john\}}), MOVIE^S(self^{\{films\}})\}$
5	Query	Kern Robert films
	CJN	$PERSON^V[name^{\{kern,robert\}}] \leftarrow CASTING \rightarrow MOVIE^S[self^{\{films\}}]$
	QM	$\{PERSON^V(name^{\{kern,robert\}}), MOVIE^S(self^{\{films\}})\}$

Figure 8.11: Keyword Queries and and Answers for the template T_1.

We use these augmented data to generate positive examples for the fine-tuning of the QM ranking and CJN ranking. Next, we perform the keyword search for the newly created queries, and use the non-relevant QMs and CJNs returned by Lathe as negative examples.

Chapter 9

Experiments

In this section, we report a set of experiments performed using datasets and query sets previously used in similar experiments reported in the literature. Our goal is to evaluate the CJN Ranking, the QM ranking, and how our Eager Evaluation strategy can improve the CJN Generation. We evaluate both the Bayesian approach (Chapter 7) and the deep learning approach (Chapter 8).

9.1 Experimental Setup

Datasets

For all the experiments, we used three datasets, *IMDb*, *MONDIAL*, and *Yelp*, which were used for the experiments performed with previous R-KwS systems and methods [Coffman and Weaver, 2010a, Coffman and Weaver, 2012, Luo et al., 2007, Oliveira et al., 2015, Oliveira et al., 2018, de Oliveira et al., 2020, Afonso et al., 2021]. The IMDb dataset is a subset of the well-known Internet Movie Database (IMDb)[1], which comprises information related to films, television shows, and home videos – including actors, characters, etc. The MONDIAL dataset [May, 1999] comprises geographical and demographic information from the well-known *CIA World Factbook*[2], the International Atlas, the TERRA database, and other web sources.

The Yelp dataset is a subset of Yelp[3], which comprises information about businesses, reviews, and user data. The three datasets have distinct characteristics. The IMDb dataset has a simple schema, but query keywords often occur in several relations. Although the

[1] https://www.imdb.com/
[2] https://www.cia.gov/library/publications/the-world-factbook/
[3] https://www.yelp.com/dataset

MONDIAL dataset is smaller, its schema is more complex or dense, with more relations and relational integrity constraints (RICs). The Yelp dataset has the highest number of tuples but its schema is simple. Table 9.1 summarizes the details of each dataset.

Table 9.1: Datasets we used in our experiments

Dataset	Size(MB)	Relations	Attributes	RIC	Tuples
IMDb	701	6	33	5	1,673,076
MONDIAL	14	28	48	38	17,115
Yelp	7898	7	24	5	12,856,448

Query Sets

We used the query sets provided by Coffman & Weaver [Coffman and Weaver, 2010a] benchmark for the IMDb and MONDIAL datasets. The query set for Yelp was obtained from SQLizer [Yaghmazadeh et al., 2017] and consists of 28 queries formulated in Natural Language. We adapted all of its queries to our experiments by extracting only their keyword terms.

However, we notice that several queries from IMDb and MONDIAL query sets do not have a clear intent, compromising the proper evaluation of the results, for instance, the ranking of CJNs. To provide a fairer evaluation, we generated an additional query set for each original set. In these new sets, we replaced queries that we consider unclear with equivalent queries that include added schema references. As an example, consider the query "*Saint Kitts Cambodia*" for the MONDIAL dataset, where *Saint Kitts* and *Cambodia* are the names of the two countries. There exist several interpretations of this keyword query, each of them with a distinct way to connect the tuples corresponding to these countries. For example, one might look for shared religions, languages, or ethnic groups between the two countries. While all these interpretations are valid in theory, the relevant interpretation defined by Coffman & Weaver [Coffman and Weaver, 2010a] in their golden standard indicates that the query searches for organizations in which both countries are members. In this case, we replaced the query with "Saint Kitts Cambodia Organizations".

Table 9.2 presents the query sets we used in our experiments, along with some of their features. Query sets whose names include the suffix "-DI" correspond to those in which we have replaced ambiguous queries as explained above. Thus, these queries sets have no ambiguous queries and they have a higher number of schema references.

Table 9.2: Query sets we used in our experiments

Query Set	Target Dataset	Total Queries	Ambiguous Queries	Schema References
IMDb	IMDb	50	5	20
IMDb-DI	IMDb	50	-	25
MONDIAL	MONDIAL	50	7	12
MONDIAL-DI	MONDIAL	50	-	19
Yelp	Yelp	28	-	24

Golden Standards

The benchmark from Coffman & Weaver [Coffman and Weaver, 2010a] provided the relevant interpretation and its relevant SQL results for each query of the IMDb and MONDIAL datasets. In the case of the Yelp dataset, SQLizer [Yaghmazadeh et al., 2017] provided the relevant SQL queries for natural language queries. Since we derived keyword queries from the latter, we also adapted the SQL queries to reflect this change. We then manually generated the golden standards for CJNs and QMs using relevant SQL provided by Coffman & Weaver and in SQLizer.

Metrics

We evaluate the ranking of CJNs and QMs using several metrics: Recall, Recall at ranking position K ($R@K$), Max Recall Position, Precision at ranking position 1 ($P@1$), and Mean Reciprocal Rank (MRR).

Recall is the ratio of relevant results retrieved to the total number of relevant results. Recall at K ($R@K$) is the mean recall across multiple queries considering only the first K results. If fewer than K results are retrieved by a system, we calculate the recall value at the last result. For instance, if the system returns the relevant CJN in at most position 3 of the ranking for 35 out of 50 queries, then the system would obtain an $R@3$ of 0.7.

Given that each keyword query has exactly one relevant QM and one relevant CJN, $R@K$ effectively measures whether the relevant QM or CJN is among the top K results returned by the system. This metric is important for understanding the effectiveness of our retrieval approach in ensuring that the relevant results are generated and returned.

The Max Recall Position indicates the highest rank position K within which the relevant QM is found. A lower Max Recall Position implies that the relevant QM is found within the top K positions, reducing the need to generate CJNs for many QMs.

Precision at 1 ($P@1$) is the ratio of relevant results found in the first position for each query to the number of queries. We use Recall and P@1 to compare Lathe

and other R-KwS systems. These metrics were chosen because they were reported in QUEST [Bergamaschi et al., 2016], ensuring a fair and consistent comparison.

The Mean Reciprocal Rank (MRR) value indicates how close the correct CJN is to the first position of the ranking. Given a keyword query Q, the value of the *reciprocal ranking for* Q is given by $RR_Q = \frac{1}{K}$, where K is the rank position of the relevant result. Then, the MRR obtained for the queries in a query set is the average of RR_Q, for all Q in the query set. This metric is particularly useful in our context because it penalizes systems that place the relevant QM or CJN further down the ranking, thereby highlighting models that not only retrieve the relevant item but do so with higher precision.

Lathe Setup

For the experiments we report here, we set a maximum size for QMs and CJNs of 3 and 5, respectively. Also, we consider three important parameters for running Lathe: N_{QM}, the maximum number of QMs considered from the QM ranking; N_{CJN}, the maximum number of CJNs considered from each QM; and P_{CJN}, the number of CJNs probed per QM by the eager evaluation. In this context, a *setup* for Lathe is a triple $N_{QM}/N_{CJN}/P_{CJN}$. The most common setup we used in our experiments is $8/1/9$, in which we take the top-5 QMs in the ranking, generate and probe up to 9 CJNs for each QM, and take only the first non-empty CJN, if any, from each QM. We call this the *default setup*. Later in this section, we will discuss how these parameters affect the effectiveness and the performance of Lathe, as well as why we use the default configuration.

All the resources, including source code, query sets, datasets and golden standards used in our experiments are available at `https://github.com/bdri-ufam/Lathe`.

Fine-Tuning Setup

In our experiments, we used a set of parameters for configuring the transformer-based models we used as weel as the data augmentation techniques we described in Chapter 8. We utilized a batch size of 128, which was chosen as it was the maximum that did not exceed the GPU memory capacity, ensuring efficient utilization of hardware resources without causing memory overflow. The model was trained over 2 epochs, allowing us to go over the dataset twice to balance between sufficient training time and computational efficiency. We incorporated 100 warmup steps to gradually increase the learning rate, aiding in stable model convergence. Evaluations were conducted every 500 steps to monitor performance and make necessary adjustments during training. We used the CosineSimilarityLoss function for training, ensuring that the model effectively learned to measure similarity between embeddings. This choice of

loss function was crucial for our task, as it directly influenced the quality and accuracy of the learned embeddings.

For fine-tuning, we relied on the golden standards as the validation set, ensuring accurate performance evaluation against a trusted reference. The training and test sets were derived from data augmentation, with a generation ratio of 200. We split the data into 80% for training and 20% for testing, maintaining a balanced approach to model training and evaluation.

Additionally, we utilized view snapshots, comprising the first 3 rows from the database views, to generate sentences for the CJNs. This technique provided structured sentences that encapsulated essential information from the views, facilitating meaningful comparisons and relevance assessments.

System Details

We ran the experiments reported in Sections 9.2 and 9.3 on a Linux machine running Artix Linux (64-bit, 32GB RAM, AMD Ryzen™ 5 5600X CPU @ 3.7GHz). We ran the experiments reported in Section 9.4 on a Linux machine Ubuntu 22.04.4 LTS (64-bit, 32GB RAM, Intel(R) Xeon(R) W-2225 CPU @ 4.10GHz, 2 x 16GiB DIMM DDR4 Synchronous 3200 MHz, NVIDIA Quadro RTX 5000). We used PostgreSQL as the underlying RDBMS with a default configuration. All implementations were made in Python 3.

9.2 Preliminary Results: CJN Generation

We present in this section some statistics about the CJN generation process. Table 9.3 shows the maximum and average numbers of KMs, QMs, and CJNs generated for each query set. The last two columns refer to the ratio of the number of CJNs to the number of QMs. Notice that we removed the maximum caps for the number of CJNs and CJNs per QM in the experiment reported here. However, we maintained the limit sizes of 3 and 5 for the QMs and CJNs, respectively.

Table 9.3: Statistics for the CJN process of each query set.

Query sets	**Num. KMs**		**Num. QMs**		**Num. CJNs**		**CJNs / QMs**	
	Max	Avg	Max	Avg	Max	Avg	Max	Avg
IMDb	47	15.38	702	80.50	656	97.28	0.93	1.21
IMDb-DI	64	16.20	702	85.34	656	103.88	0.93	1.22
MONDIAL	8	3.12	9	2.10	35	9.40	3.89	4.48
MONDIAL-DI	8	3.40	9	2.42	44	11.04	4.89	4.56
Yelp	21	11.82	124	32.39	301	74.07	2.43	2.29

Overall, the query sets for both IMDb and Yelp datasets achieved higher maximum and average numbers of KMs and QMs. This result is due to a higher number of tuples and the keywords being present in multiple relations or combinations. For example, in the IMDb dataset, several persons, characters, and even movies share the same name or part of it. In the case of Yelp, for instance, the keyword *"texas"* can match a state, a restaurant name, or a username. On the other hand, in MONDIAL, the keywords often match a few attributes only. For example, a city name probably does not overlap with the names of countries, continents, etc. Consequently, the system produces a low number of KMs and QMs for the query sets of this dataset.

Regarding the CJN generation, the query sets for IMDb and Yelp achieved high numbers of CJNs because of their already high numbers of QMs, but a low ratio of CJNs per QMs due to their simple schema graphs. As for the query sets for the MONDIAL dataset, they achieved opposite results due to their complex schema graph.

9.3 Experimental Results: Bayesian Ranking

In this section, we present the results of experiments we carried out using Lathe with the Bayesian Ranking described in Chapter 7. This encompasses our initial results with the Lathe system. We compare Lathe with other R-KwS systems, examining its performance and capabilities. Additionally, we analyze various configurations for CJN generation, providing insights into their effectiveness. These results have been previously reported and published [Martins et al., 2023b].

9.3.1 Comparison with other R-KwS systems

In this experiment, we first compare Lathe with QUEST [Bergamaschi et al., 2013], the current state of art R-KwS system with support to schema references and then we also compare Lathe with several other R-KwS systems. Here, we used the default Lathe setup, that is, $8/1/9$. We compare our results to those published by the authors, which refer to the MONDIAL dataset, because we were unable to run QUEST due to the lack of code and enough details for implementing it. It is important to note that QUEST only supports 35 out of the 50 queries in the original query set. Figure 9.1 depicts the results for these 35 supported queries[4]. The graphs show the recall and P@1 values for the raking produced by each system considering the golden standard supplied by Coffman & Weaver [Coffman and Weaver, 2010a].

Both systems achieved perfect recall; that is, all the expected solutions for the given keyword queries were retrieved. Concerning P@1, Lathe obtained better results than QUEST,

[4]Specifically, queries 01-20, 26-35 and 46-50.

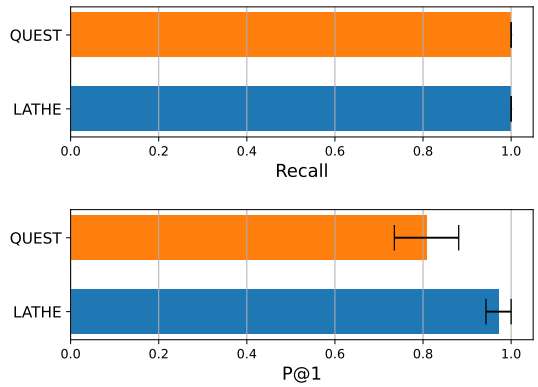


Figure 9.1: Comparison of Lathe with the QUEST system.

with an average of 0.97 with a standard error of 0.03, which indicates that, in most cases, the correct solution was the one corresponding to the CJN ranked as the first by Lathe.

Next, we compare the results obtained for Lathe with those published in the comprehensive evaluation published by Coffman & Weaver [Coffman and Weaver, 2010a] for the systems BANKS [Aditya et al., 2002], DISCOVER [Hristidis and Papakonstantinou, 2002], DISCOVER-II [Hristidis et al., 2003], BANKS-II [Kacholia et al., 2005], DPBF [Ding et al., 2007], BLINKS [He et al., 2007] and STAR [Kasneci et al., 2009]. Because this comparison uses all 50 keyword queries from the MONDIAL dataset, we did not include QUEST in the comparison. Figure 9.2 shows the recall and P@1 values for the raking produced by each system when the golden standard provided by Coffman & Weaver [Coffman and Weaver, 2010a] is taken into account.

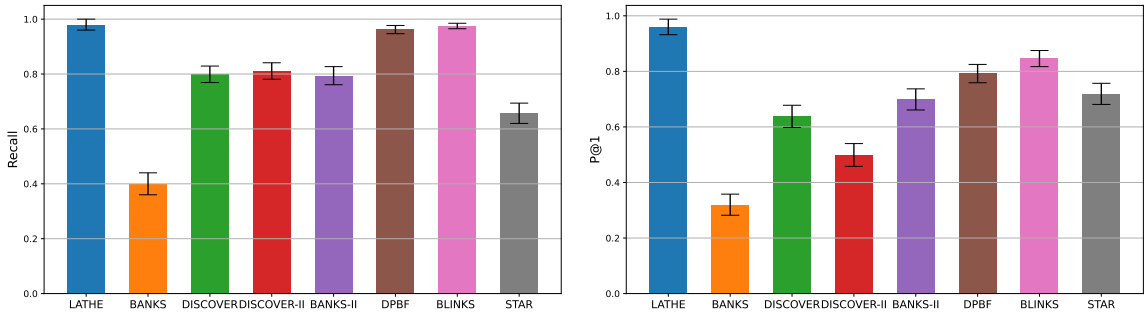


Figure 9.2: Comparison with other approaches using Recall and P@1 metrics

Overall, Lathe achieved the best results in Recall and P@1 value. The only systems achieving a recall value close to Lathe were DPBF and BLINKS, which are based on data graph. Thus, they require a materialization of the database. The difference in the recall values between Lathe, DISCOVER, and DISCOVER-II is mainly due to not supporting schema references. Regarding the P@1, Lathe obtained a value of 0.96 with a standard error of 0.03, which is significantly higher than the results for other systems. Especially when comparing

Lathe with DISCOVER and DISCOVER-II, the reason for Lathe's best performance in P@1 is due to the Lathe's novel ranking of QMs and the improved ranking of CJNs.

9.3.2 Evaluation of Query Matches Ranking

In this experiment, we evaluate the quality of QMs ranking according to the metrics MRR and $R@K$. As shown by the results in Section 9.2, there can be many QMs depending on the query. As a result, we want to verify how effective the QMRank algorithm is at selecting the most likely correct QM from among those generated in this experiment. Figure 9.3 shows the results obtained with $R@K$ up to the tenth ranking position and the MRR metric.

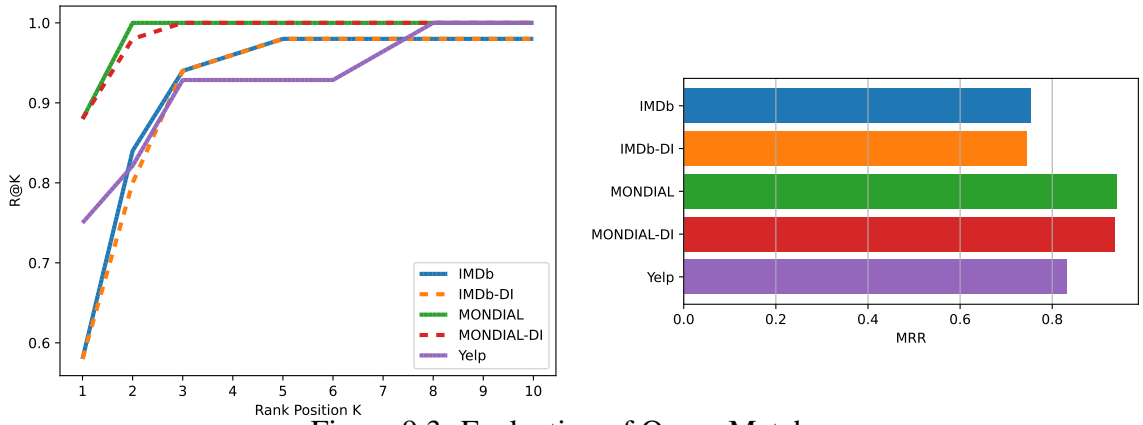


Figure 9.3: Evaluation of Query Matches

As shown by R@K plot in Figure 9.3, the correct QM is found at least in the eighth ranking position for all query sets. However, for MONDIAL and MONDIAL-DI, the relevant QM is at least in the third position for all queries. Yelp obtained an $R@8$ of 1 and $R@3$ of 0.93, which indicates that the system returns the relevant QM by the eighth position, and in most cases, up to the third position. Regarding the IMDb dataset there is one query in it whose relevant QM is not minimal. As QMs must be minimal by Definition 5, Lathe does not support this query. Consequently, the query sets for the IMDb dataset can obtain an $R@K$ value of 0.98 at most. IMDb and IMDb-DI achieved this value at position 5.

Lathe obtained an MRR of 0.75 for both IMDb and IMDb-DI, 0.83 for Yelp, and 0.96 and 0.95 for MONDIAL and MONDIAL-DI, respectively. This result indicates that the relevant QM is often in the top positions of the ranking. Notice that the QM ranking impacts the generation and ranking of CJNs. In practice, a high $R@K$ value with a low K allows us to generate fewer CJNs without compromising the quality of the CJN ranking. Based on the obtained results, we set the parameter N_{QM} to 8, which indicates that Lathe will only generate CJNs for the top-8 query matches.

9.3.3 Evaluation of the Candidate Joining Network Ranking

In this experiment, we evaluate the quality of our approach for CJN generation and ranking. We used the metrics MRR and $R@K$ for K up to the tenth rank position. We tested several different setups but to save space we report here only those with representative distinct results. Specifically, we report the results of four setups without the eager evaluation, that is, $8/1/0$, $8/2/0$, $8/8/0$ and $8/9/0$ and two setups with the eager evaluation, that is $8/1/9$ and $8/2/9$.

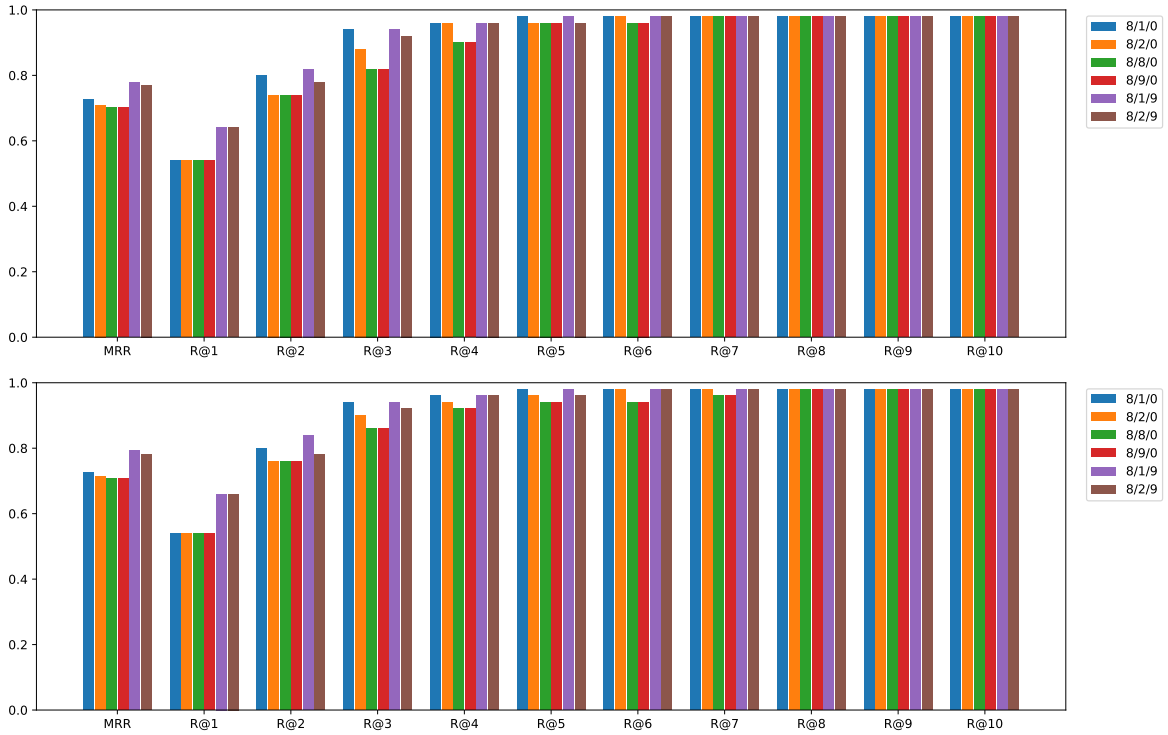


Figure 9.4: Ranking of Candidate Joining Networks - IMDb (top) and IMDb-DI (bottom)

Figure 9.4 shows the results for the IMDb and IMDb-DI query sets. As it can be seen, regardless of the configuration, our method was able to place the relevant CJNs in the top positions in the ranking, and the result is very similar for both IMDb and IMDb-DI query sets. This shows that in these datasets, our method was able to disambiguate the queries properly, even without the addition of schema references. It is worth noting that the values of $R@1$ in both query sets show that the configurations with the eager evaluation achieved better results because they place the relevant CJNs in the first ranking position more frequently. The $R@K$ metric also shows that the quality of the ranking decreases as the number of CJNs per QM increases, especially for K in the range $2 \leq K \leq 6$.

Figure 9.5 shows the results for MONDIAL and MONDIAL-DI. In these query sets, the configurations with the eager evaluation achieved significantly better results. The configurations $8/1/0$ and $8/2/0$ could not generate the relevant CJN for around 20% of the queries due to a low number of CJNs per QM, therefore, their results were capped at an MMR and $R@K$

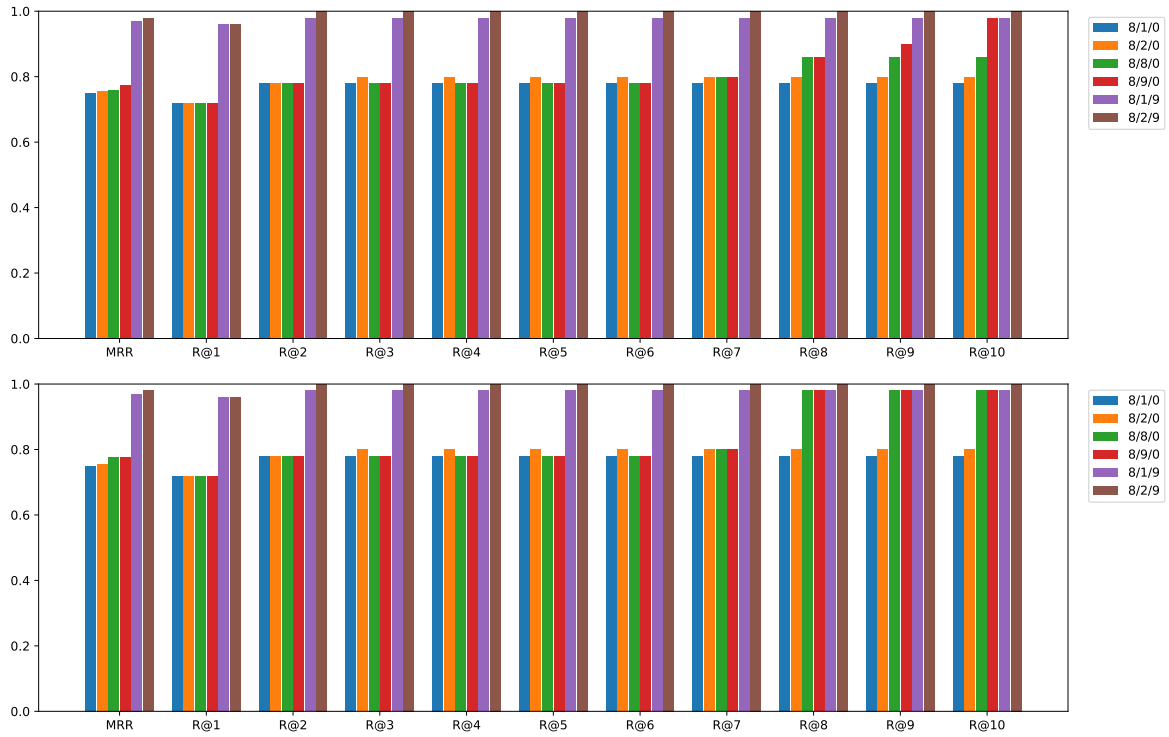


Figure 9.5: Ranking of Candidate Joining Networks - MONDIAL (top) and MONDIAL-DI (bottom)

value of 0.8, approximately. The configurations $8/8/0$ and $8/9/0$ were able to generate the relevant CJN for most of the cases, although the large number of CJNs per QM negatively affected the ranking of CJNs. Finally, the configurations $8/1/9$ and $8/2/9$ produced the best results because the pruning enables us to generate the relevant CJN with a low number of CJNs per QM while also placing the relevant CJN in higher rank positions. Notice that the disambiguation of queries in the MONDIAL-DI query set allowed configurations $8/8/0$ and $8/9/0$ to have better results, especially for the $R@K$ metric for K above 7. The eager evaluation configurations were able to disambiguate the queries without relying on the addition of schema references, therefore, their results were consistent across the MONDIAL and MONDIAL-DI query sets.

Figure 9.6 shows the results for the Yelp query set. Overall, the eager CJN evaluation did not affect the results for this query set, probably because the database schema graph was simple and the ways of connecting the query matches were straightforward. Configurations $8/1/0$ and $8/1/9$ achieved the best results, obtaining a MRR of 0.85 and $R@2$ of 0.92 for the CJN generation. This indicates that the relevant CJNs are often found up to the second ranking position, with exception of two queries, whose relevant CJN were found in positions 5 and 7, respectively. The other configurations obtained slightly worse results, with an MRR of 0.84 and an $R@2$ of 0.89, approximately.

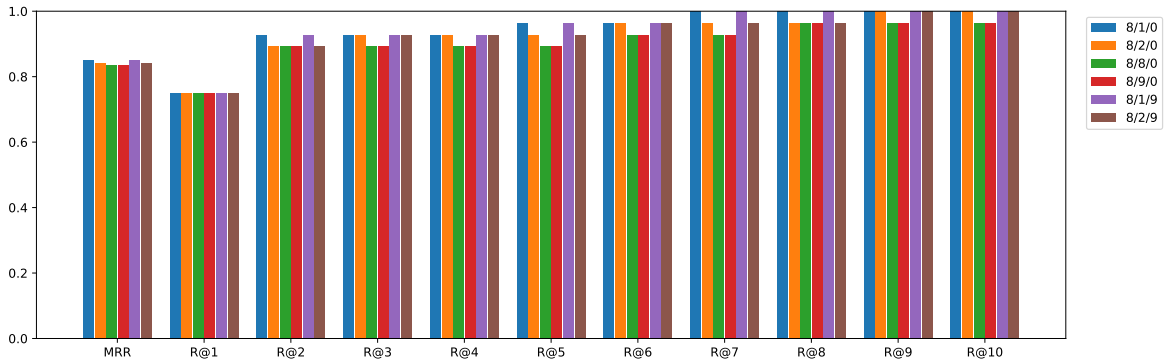


Figure 9.6: Ranking of Candidate Joining Networks - Yelp

Regardless of the datasets and configurations, our method achieved an MRR value above 0.7, which indicates that on average, the relevant CJN is found between the first and the second rank positions. In the IMDb dataset, the decrease of $R@K$ values according to the number of CJNs taken per QM is also reflected on the MRR metric. However, in the MONDIAL dataset, the improvement of the $R@K$ values due to the disambiguation of queries is not reflected on the MRR value, as this improvement only happens in low ranking positions ($K \leq 8$).

The eager CJN evaluation inherently affects the performance of the CJN generation process. Therefore it is important to look at the trade-off between the effectiveness and the efficiency in each configuration. We examine this trade-off in the next section.

9.3.4 Performance Evaluation

In this experiment, we aim at evaluating the time spent for obtaining the CJN given a keyword query, and analyze the trade-offs between efficiency and efficacy of the different configurations used in Lathe.

Lathe obtained better execution times for the IMDb dataset in all configurations. Also, the disambiguate variants of query sets yield slower execution times in comparison with the original counterparts.

Figure 9.7 summarizes the average execution time for each phase of the process: Keyword Matching, Query Matching and the Candidate Joining Network Generation. In this first experiment, we used the configuration $8/1/0$. Lathe obtained better total execution times for the IMDb dataset, followed by the Yelp dataset. In addition, the query sets IMDb-DI and MONDIAL-DI yield slower execution times in comparison with the original counterparts. Also, it is worth noting that the execution times for each query set are related to the number of KMs, QMs, and CJNs in the query sets shown in Table 9.3.

Regarding keyword matching, the Yelp dataset yielded the worst execution times,

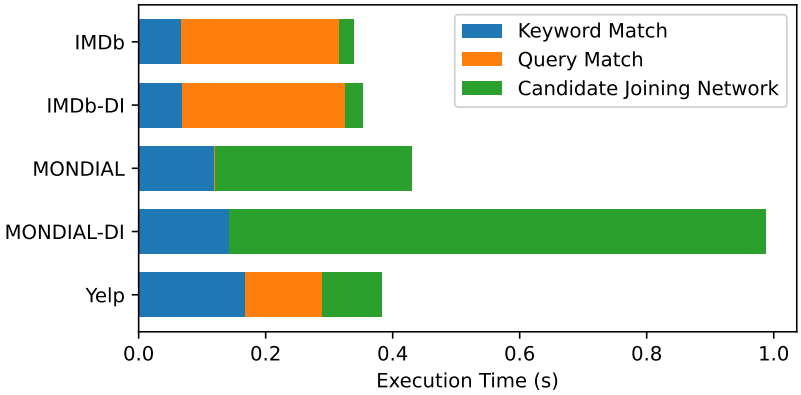


Figure 9.7: Average Execution Times for each phase of Lathe. The QM generation time for the MONDIAL and MONDIAL-DI query sets is in the range of microseconds, therefore this minimal time does not appear prominently in the chart due to the scale.

with 167ms, probably because of its higher number of attributes and tuples. Although the MONDIAL dataset has fewer tuples than IMDb, its higher number of schema elements (28 relations and 48 attributes) results in a higher execution time than IMDb.

Due to the combinatorial nature of QM generation, the execution times for the Query Matching phase are directly related to the number of QMs. While the execution times for the IMDb and IMDb-DI query sets that produced a high number of QMs are 247 and 256 milliseconds, respectively, the results for the MONDIAL and MONDIAL-DI are around 190 and 202 microseconds. The Yelp dataset achieved 121 miliseconds. It is important to note that the QM generation time for the MONDIAL and MONDIAL-DI query sets is in the range of microseconds, orders of magnitude smaller than the times for other query sets. This minimal time does not appear prominently in the chart due to the scale, contributing to the overall efficiency of the system.

Concerning the CJN phase, the execution times for MONDIAL are significantly higher in comparison with the execution times for IMDb and Yelp, despite the lower number of CJNs for the MONDIAL. Because the CJN generation algorithm is based on a Breadth-First Search, the greater the number of vertices and edges in the schema graph of the MONDIAL dataset, the greater the number of iterations and, consequently, the slower the execution times. This behavior persists throughout different configurations, an issue we analyze below.

9.3.5 Quality versus Performance

Figure 9.8 presents an evaluation of the CJN generation performance, comparing the same configurations used in the experiment of Section 9.3.3. We present the results for the IMDB, MONDIAL and Yelp datasets in different scales because they differ by order of magnitude. Overall, execution times increase as the number of CJNs taken per QM increases. This pattern

is more pronounced in the MONDIAL dataset. Also, the eager CJN evaluation incurs an unavoidable increase in the CJN generation time as the system has to probe the CJNs running queries into the database.

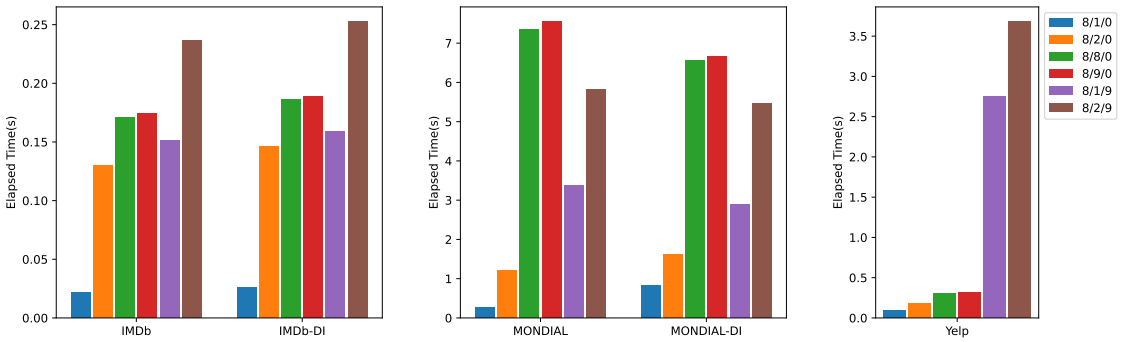


Figure 9.8: Performance Evaluation of the CJN Generating phase

As the configurations have an impact on both the quality of the CJN ranking and the performance, it is important to examine the trade-off between effectiveness and efficiency. Configuration $8/1/0$ and $8/2/0$ achieved the best execution times due to the low number of CJNs per QM and not relying on database accesses. However, these configurations did not achieve the highest values of MRR and R@K for the IMDb and MONDIAL datasets. Therefore, they are recommended if one must prioritize efficiency.

Configuration $8/1/9$ obtained better results than configurations $8/8/0$, $8/9/0$ for the IMDb and MONDIAL datasets and better than $8/2/9$ for all datasets. Although this configuration is slower than $8/1/0$ and $8/2/0$, the significantly better results of MRR and $R@K$ values for MONDIAL and IMDb datasets make the $8/1/9$ configuration an overall recommended option, especially if one must prioritize effectiveness.

We do not recommend the configurations $8/2/0$, $8/8/0$, $8/9/0$ and $8/2/9$ because their MRR and $R@K$ values do not justify the increase in execution times. Although $8/2/9$ obtained the best MRR and $R@K$ values for the MONDIAL dataset, it is 37%-80% slower than $8/1/9$. Configurations $8/8/0$ and $8/9/0$ achieved a slight increase in the $R@K$ metric for the MONDIAL dataset, for $K \leq 8$, however, they obtained lower values of MRR and $R@K$ values for the IMDb and Yelp datasets.

It is interesting noting that the configurations with eager CJN evaluation spend time to probe CJNs while sending queries to the DBMS. However, as they generate a smaller set of CJNs, the overall performance is not hindered in comparison with the configurations without it.

9.4 Experimental Results: Neural Ranking

This section presents a set of experiments comparing the Bayesian approach with the approach we developed with modern neural techniques, specifically transformer-based models as detailed in Chapter 8. We aim to assess the advancements and improvements brought by neural methods in the context of QM and CJN ranking. The experiments provide a comprehensive evaluation, highlighting the strengths and potential limitations of both approaches.

9.4.1 Neural QM Ranking

In this section, we evaluate the performance of different models on the task of QM ranking using the metrics MRR (Mean Reciprocal Rank), Recall, and Max Recall Position. Our main goal is to assess the effectiveness of transformer-based models for ranking QMs and benchmark them against the Bayesian model, which serves as the ranking solution in our Bayesian approach.

A lower Max Recall Position (k) indicates that the relevant QM is found within the top k positions, requiring the CJN generation only for the first k QMs. Nevertheless, a high Max Recall Position generates a major amount of CJNs, this ensures that the relevant QM is generated for most keyword queries, thereby increasing the likelihood of generating the most relevant CJN. Hence, a high MRR indicates good overall ranking performance.

Table 9.4 shows twenty models analyzed for this experiment, detailing their similarity function and abbreviation. The letters B, P and F in the abbreviations respectively identify bayesian, pre-trained and fine-tuned models. The subscript indicator identifies each model, and in the case of transformer-based models it also identifies their respective variations. The Bayesian model (B_{QM}) follows the Bayesian approach and serves as a baseline for comparison against the transformer-based models. The analysis includes both pre-trained and fine-tuned models.

We evaluated the models on three datasets: IMDB, MONDIAL, and Yelp. Below, we present the average results for MRR and Recall, and the maximum result for Max Recall Position across these datasets.

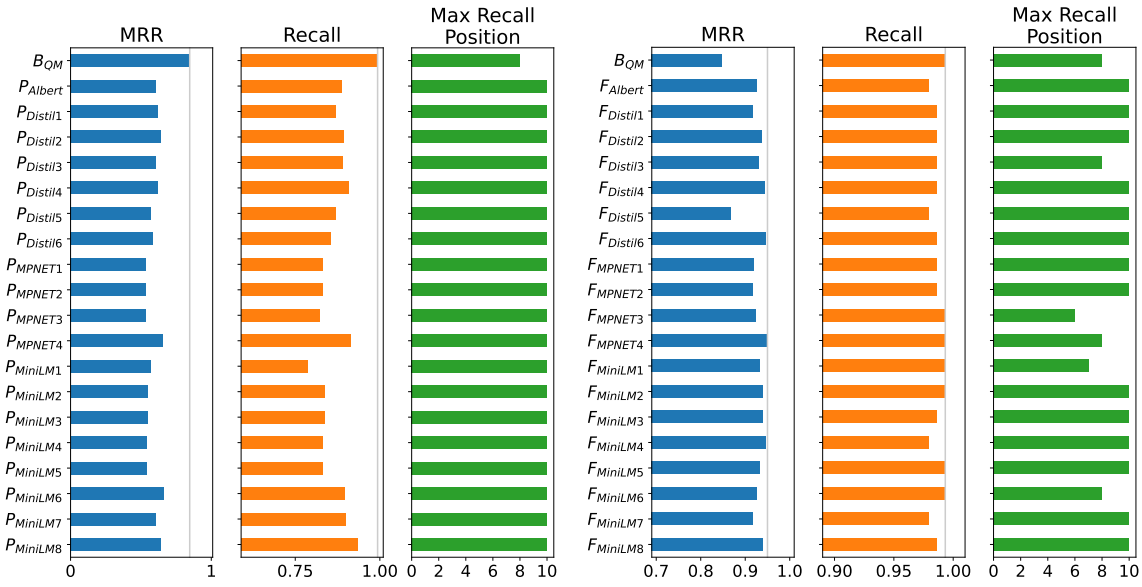
All Datasets Analysis

Figure 9.9 shows the average MRR and Recall across the IMDB, MONDIAL, and Yelp datasets, along with the maximum Max Recall Position for the same datasets.

The Bayesian model (B_{QM}) achieved an MRR of 0.847, Recall of 0.993, and a Max Recall Position of 8. Pre-trained models performed worse than the Bayesian model, which

Table 9.4: Neural QM Ranking Models

Model	**Similarity**	**Abbreviation**	
Bayesian	cos	B_{QM}	
Model	**Similarity**	**Abbreviation**	
		Pre-Trained	**Fine-tuned**
paraphrase-albert-small-v2	cos	P_{Albert}	F_{Albert}
all-distilroberta-v1	cos	$P_{Distil1}$	$F_{Distil1}$
distiluse-base-multilingual-cased-v1	cos	$P_{Distil2}$	$F_{Distil2}$
distiluse-base-multilingual-cased-v1	dot	$P_{Distil3}$	$F_{Distil3}$
distiluse-base-multilingual-cased-v2	cos	$P_{Distil4}$	$F_{Distil4}$
distiluse-base-multilingual-cased-v2	dot	$P_{Distil5}$	$F_{Distil5}$
multi-qa-distilbert-cos-v1	cos	$P_{Distil6}$	$F_{Distil6}$
all-MiniLM-L12-v2	cos	$P_{MiniLM1}$	$F_{MiniLM1}$
all-MiniLM-L6-v2	cos	$P_{MiniLM2}$	$F_{MiniLM2}$
all-MiniLM-L6-v2	dot	$P_{MiniLM3}$	$F_{MiniLM3}$
multi-qa-MiniLM-L6-cos-v1	cos	$P_{MiniLM4}$	$F_{MiniLM4}$
multi-qa-MiniLM-L6-cos-v1	dot	$P_{MiniLM5}$	$F_{MiniLM5}$
paraphrase-MiniLM-L3-v2	cos	$P_{MiniLM6}$	$F_{MiniLM6}$
paraphrase-multilingual-MiniLM-L12-v2	dot	$P_{MiniLM7}$	$F_{MiniLM7}$
paraphrase-multilingual-MiniLM-L12-v2	cos	$P_{MiniLM8}$	$F_{MiniLM8}$
all-mpnet-base-v2	cos	P_{MPNET1}	F_{MPNET1}
all-mpnet-base-v2	dot	P_{MPNET2}	F_{MPNET2}
multi-qa-mpnet-base-dot-v1	dot	P_{MPNET3}	F_{MPNET3}
paraphrase-multilingual-mpnet-base-v2	cos	P_{MPNET4}	F_{MPNET4}



(a) Pre-trained Models (b) Fine-tuned Models

Figure 9.9: Evaluation of the Neural QM ranking on all datasets (Average for MMR and Recall, and Max for Max Recall Position).

indicates they struggled to accurately interpret QM linearization and ranking tasks. However, fine-tuning these models led to substantial improvements.

Fine-tuned models like F_{MPNET4}, $F_{MiniLM4}$, and $F_{Distil6}$ achieved higher MRR scores than the baseline, indicating their superior capability in ranking relevant QMs higher on

average. This highlights the impact of fine-tuning in enhancing the models' understanding and ranking abilities for QM tasks.

Regarding Recall, several fine-tuned models, including F_{MPNET3}, F_{MPNET4}, $F_{MiniLM1}$, $F_{MiniLM2}$, $F_{MiniLM5}$, and $F_{MiniLM6}$, achieved a Recall of 0.993. This performance is on par with the Bayesian baseline, showcasing these models' ability to retrieve all relevant QMs effectively. Such results underscore the models' comprehensive recall capabilities after fine-tuning.

In terms of Max Recall Position, the fine-tuned models also demonstrated improvements. For instance, F_{MPNET3} achieved the best performance by retrieving the relevant QM within the top 6 positions, outperforming the baseline's position 8. Similarly, $F_{MiniLM1}$ and F_{MPNET4} achieved positions 7 and 8, respectively. These results reflect the models' enhanced efficiency in identifying relevant QMs earlier in the ranking list, a crucial factor for user satisfaction in information retrieval tasks.

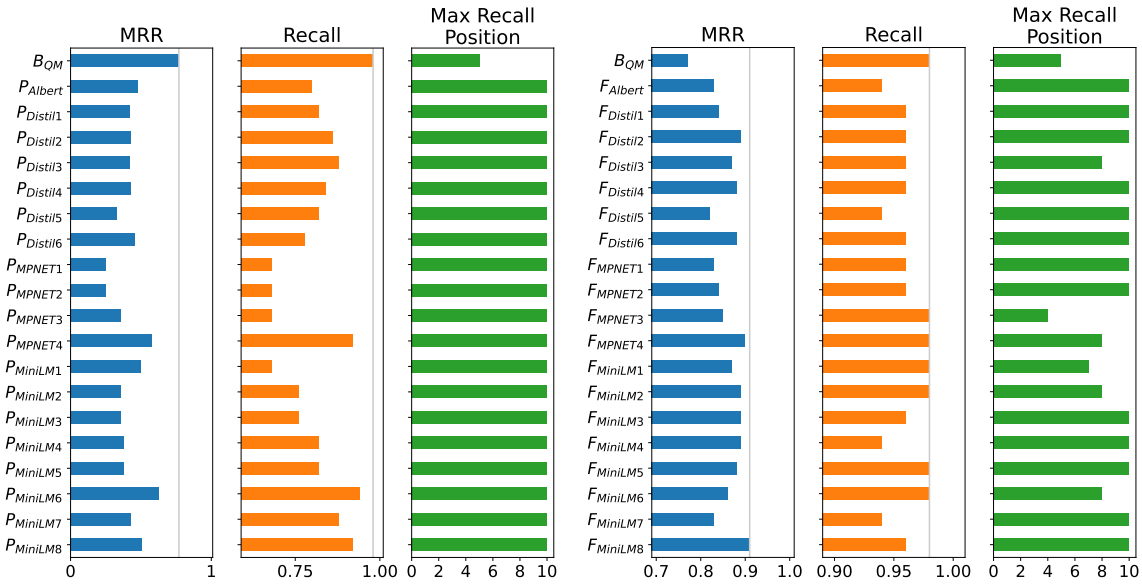
Among the evaluated models, $F_{MiniLM1}$, F_{MPNET3}, and F_{MPNET4} stand out as the best performers overall. These models consistently achieve high MRR, Recall, and low Max Recall Position across all datasets.

Individual Dataset Analysis

The results for individual datasets largely mirror the trends observed in the overall analysis, with fine-tuned models outperforming pre-trained ones and the Bayesian baseline.

Figure 9.10 shows the results for the IMDB dataset. The Bayesian model (B_{QM}) achieved an MRR of 0.77, Recall of 0.98, and a Max Recall Position of 5. Pre-trained models consistently performed worse than the baseline. Fine-tuned models like $F_{MiniLM8}$ and F_{MPNET4} achieved MRR scores of 0.91 and 0.9, respectively, indicating their superior capability in ranking relevant QMs higher on average. Several fine-tuned models, including F_{MPNET3}, F_{MPNET4}, $F_{MiniLM1}$, $F_{MiniLM2}$, $F_{MiniLM5}$, and $F_{MiniLM6}$, achieved a Recall of 0.98, on par with the Bayesian baseline, showcasing these models' ability to retrieve all relevant QMs effectively. In terms of Max Recall Position, F_{MPNET3} achieved the best performance by retrieving the relevant QM within the top 4 positions, outperforming the baseline's position 5. Similarly, $F_{MiniLM1}$ and F_{MPNET4} achieved positions 7 and 8, respectively, reflecting enhanced efficiency in identifying relevant QMs earlier.

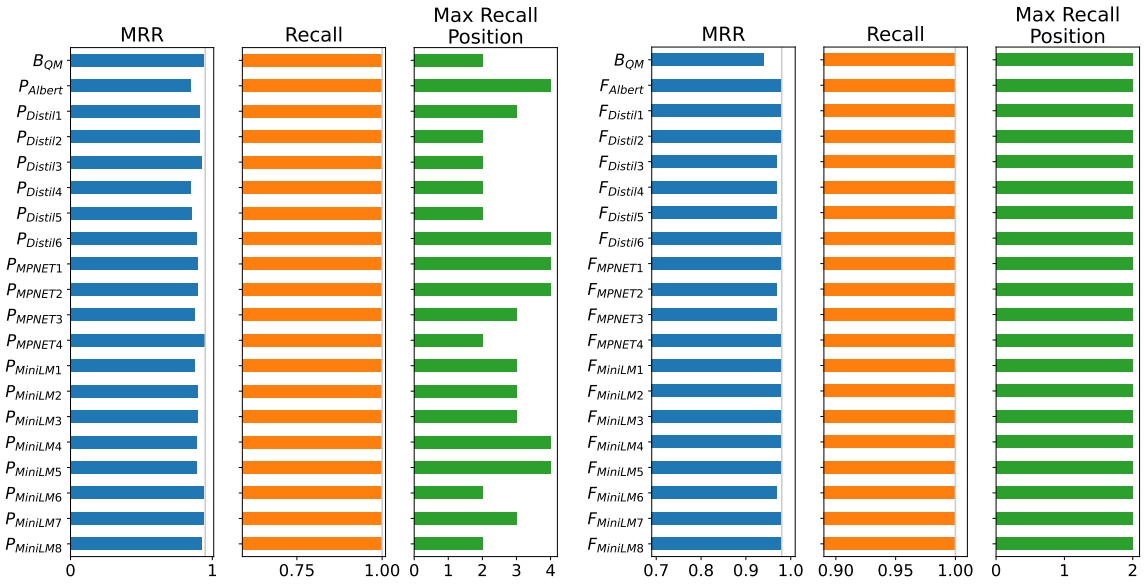
Figure 9.11 shows the results for the MONDIAL dataset. The Bayesian model (B_{QM}) achieved an MRR of 0.94, Recall of 1.0, and a Max Recall Position of 2. Most sentence-transformer models achieved perfect Recall (1.0) and very low Max Recall Positions (2). The pre-trained model P_{MPNET4} obtained a slightly better MRR of 0.95. Fine-tuned models like F_{Albert}, $F_{Distil1}$, F_{MPNET4}, and $F_{MiniLM1}$ all achieved an MRR of 0.98, making them top



(a) Pre-trained Models (b) Fine-tuned Models

Figure 9.10: Evaluation of the Neural QM ranking on the IMDb dataset.

performers for this dataset. Due to MONDIAL having fewer QMs on average, which was observed in Table 9.3 and discussed in Section 9.2, a pre-trained model achieved a better MRR score than the baseline, and models based on the Albert and the distilled version of RoBERTa were among the top performers.

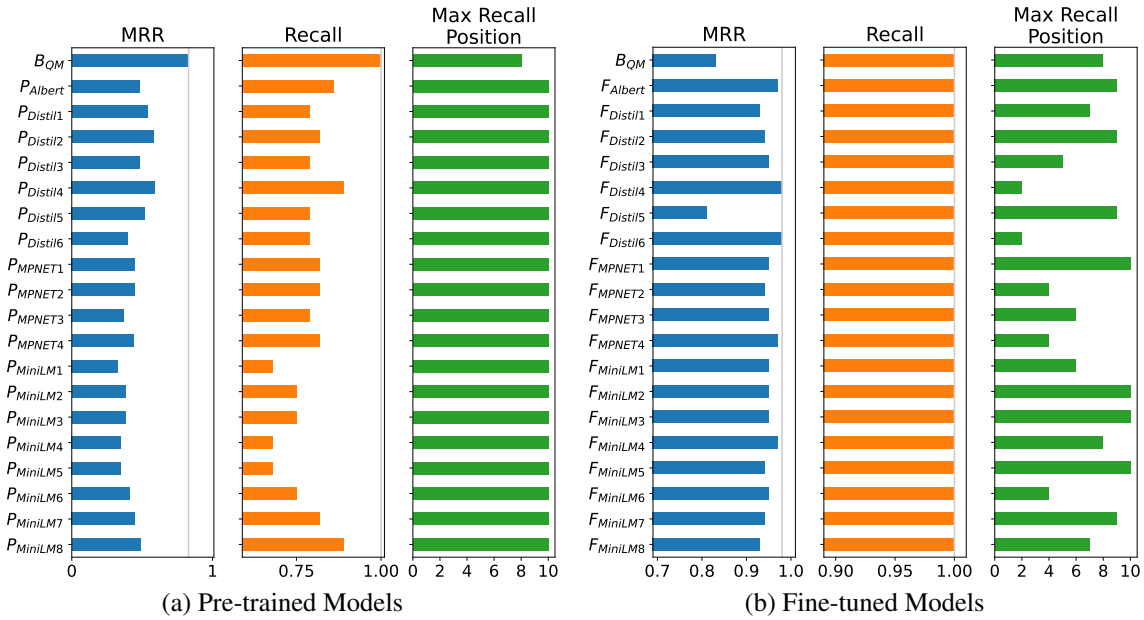


(a) Pre-trained Models (b) Fine-tuned Models

Figure 9.11: Evaluation of the Neural QM ranking on the MONDIAL dataset.

Figure 9.12 shows the results for the Yelp dataset. The Bayesian model (B_{QM}) had an MRR of 0.83, Recall of 1.0, and a Max Recall Position of 8. Pre-trained models consistently performed worse than the baseline. Fine-tuned models like $F_{Distil4}$ and $F_{Distil6}$ achieved the

highest MRR of 0.98 and Recall of 1.0, with a Max Recall Position of 2. However, F_{MPNET4} and $F_{MiniLM1}$ also performed well, obtaining MRR scores of 0.97 and 0.95 respectively, and Max Recall Positions of 4 and 6.



(a) Pre-trained Models (b) Fine-tuned Models

Figure 9.12: Evaluation of the Neural QM ranking on the Yelp dataset.

Discussion

The results indicate that sentence-transformer models significantly outperform the Bayesian baseline in terms of MRR, suggesting that these models provide better overall ranking of QMs. Fine-tuned models, in particular, demonstrate superior performance in the QM ranking.

Models based on MiniLM and MPNET were consistently among the top performers across all datasets, a trend also observed in the experiments shown on the Sentence-Transformers library website[5]. Specifically, the models $F_{MiniLM1}$, F_{MPNET3}, and F_{MPNET4} achieved high MRR, Recall, and low Max Recall Position consistently, making them ideal candidates for practical applications where efficiency in QM ranking is crucial. Consequently, these models were chosen as the best performers for the QM ranking task and were utilized in the Neural CJN ranking experiment.

9.4.2 Neural CJN Ranking

In this section, we evaluate and compare several CJN models using different datasets. Our goal is to demonstrate the effectiveness of transformer-based models, and compare it to Bayesian models.

[5] Sentence Transformers Pretrained Models https://sbert.net/docs/sentence_transformer/pretrained_models.html

Table 9.5 shows twenty-four models analyzed for this experiment, detailing their similarity function and abbreviation. The letters B, P, and F in the abbreviations respectively identify Bayesian, pre-trained, and fine-tuned models. The subscript indicator identifies each model, and in the case of transformer-based models, it also identifies their respective variations. Each abbreviation consists of its QM model abbreviation followed by a dash, and then the CJN abbreviation. The CJN abbreviation may also have trailing characters indicating the aggregation approach used, which can be '*' for the multivalue approach, and '+' for the mean approach. The CJN abbreviation B_{CJN} indicates that a simple CJN ranking was applied, where the CJNs were ranked based on their QM score, divided by their size. The model $B_{QM}-B_{CJN}$ follows the Bayesian approach, and it serves as a baseline for comparison against the transformer-based models. These models were ranked in the top-10 in at least one of the datasets. Table G.1 from Appendix G presents a complete list of the CJN Ranking models.

Table 9.5: CJN Ranking Models

QM Model	CJN Model	Similarity	Agg. Approach	CJN Type	Abbreviation
B_{QM}	Bayesian	cos	—	bayesian	$B_{QM}-B_{CJN}$
B_{QM}	paraphrase-albert-small-v2	cos	Multivalue	fine-tuned	$B_{QM}-F_{Albert*}$
B_{QM}	all-distilroberta-v1	cos	Multivalue	fine-tuned	$B_{QM}-F_{Distil1*}$
B_{QM}	multi-qa-MiniLM-L6-cos-v1	dot	Mean	fine-tuned	$B_{QM}-F_{MiniLM5+}$
B_{QM}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$B_{QM}-F_{MiniLM8*}$
$F_{MiniLM1}$	Bayesian	cos	—	bayesian	$F_{MiniLM1}-B_{CJN}$
$F_{MiniLM1}$	paraphrase-albert-small-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Albert*}$
$F_{MiniLM1}$	all-distilroberta-v1	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Distil1*}$
$F_{MiniLM1}$	all-distilroberta-v1	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{Distil1+}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Distil3*}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM2*}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM3*}$
$F_{MiniLM1}$	paraphrase-MiniLM-L3-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM6+}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM8*}$
F_{MPNET3}	Bayesian	cos	—	bayesian	$F_{MPNET3}-B_{CJN}$
F_{MPNET3}	paraphrase-albert-small-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{Albert*}$
F_{MPNET3}	all-distilroberta-v1	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{Distil1*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{Distil3*}$
F_{MPNET3}	all-MiniLM-L6-v2	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM3*}$
F_{MPNET3}	paraphrase-MiniLM-L3-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM6+}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM8*}$
F_{MPNET4}	Bayesian	cos	—	bayesian	$F_{MPNET4}-B_{CJN}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	dot	Multivalue	fine-tuned	$F_{MPNET4}-F_{Distil3*}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$F_{MPNET4}-F_{MiniLM8*}$

Evaluation on All Datasets

We begin by evaluating the performance of various CJN models on all available query sets. Figure 9.13 presents the Mean Reciprocal Rank (MRR) and Recall@k (R@k) scores for each model.

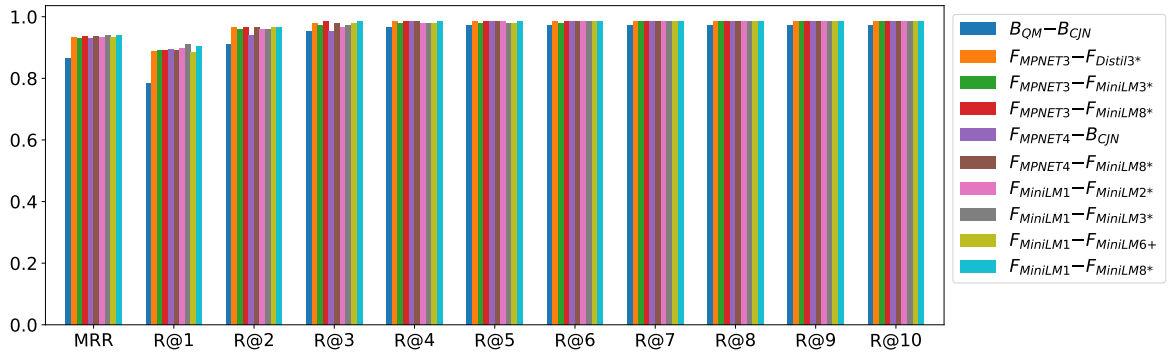


Figure 9.13: Neural CJN Ranking - All Datasets

The baseline model, $B_{QM}-B_{CJN}$, achieves a decent performance with an MRR of 0.867 and high Recall scores, with R@1 at 0.7833, R@2 at 0.91, reaching a recall plateau of 0.9733 at R@6. However, most of the fine-tuned models show substantial improvements over the baseline, with MRR values ranging from 0.93 to 0.94. Recall metrics for these models also indicate enhanced performance, often obtaining a R@1 above 0.883, R@2 above 0.94, and reaching or exceeding 0.98 by R@4.

Specifically, the models $F_{MPNET3}-F_{Distil3*}$, $F_{MPNET3}-F_{MiniLM8*}$, and $F_{MiniLM1}-F_{MiniLM8*}$ are the top performers, with MRR values of 0.9333, 0.9367, and 0.94, respectively. These models demonstrate superior recall performance compared to the baseline. For instance, $F_{MPNET3}-F_{Distil3*}$ achieves an R@1 of 0.8867 and an R@10 of 0.9867, showing consistent improvement over the baseline across all recall points.

Models involving fine-tuning on MiniLM and MPNET consistently outperform others, which aligns with trends observed in the pretraining experiments on the Sentence-Transformers library website[6].

Among the top 10 models, one uses the mean aggregation approach, seven use the multivalue aggregation approach, and one model besides the baseline employs the simple ranking method. This distribution highlights the superiority of the multivalue aggregation approach for generating embeddings for CJNs. Additionally, the inclusion of the $F_{MPNET4}-B_{CJN}$ model among the top performers suggests that the F_{MPNET4} model is so effective in QM ranking that there is no need to rank CJNs using another neural model. Instead, we can simply divide the QM score by the size of the CJN to achieve competitive results.

Furthermore, the combination of models, such as $F_{MPNET3}-F_{MiniLM8*}$, indicates that pairing different fine-tuned models can leverage their strengths, leading to improved ranking accuracy.

Overall, the fine-tuned models, especially those based on MiniLM and MPNET, exhibit substantial improvements in MRR and recall metrics, demonstrating their effectiveness in the

[6]Sentence Transformers Pretrained Models https://sbert.net/docs/sentence_transformer/pretrained_models.html

CJN ranking task. The consistent high performance across various recall points underscores their potential for practical applications in information retrieval systems.

Evaluation on Specific Datasets

We further evaluate the CJN models on specific datasets to assess their performance in domain-specific scenarios. The results for individual datasets largely mirror the trends observed in the overall analysis, with fine-tuned models outperforming pre-trained ones and the Bayesian baseline.

Figure 9.14 shows the results for the IMDb dataset. The baseline model $B_{QM}-B_{CJN}$ achieved an MRR of 0.78, highlighting a significant improvement with transformer-based models. Fine-tuned models demonstrate better performance at lower recall points (R@1 to R@4), indicating their effectiveness in ranking relevant CJNs higher. Most models achieve a high recall plateau at R@6. The fine-tuned models $B_{QM}-F_{MiniLM8*}$, $F_{MPNET3}-F_{MiniLM8*}$, $F_{MPNET4}-F_{MiniLM8*}$, $F_{MiniLM1}-F_{MiniLM8*}$ achieved the highest MRR of 0.9 and consistently high recall scores. This indicates that the $F_{MiniLM8*}$ model is highly effective for the CJN ranking, no independent of which model was used for the QM ranking. Furthermore, all the top 4 performer models used the multivalue (*) approach, indicating its superiority.

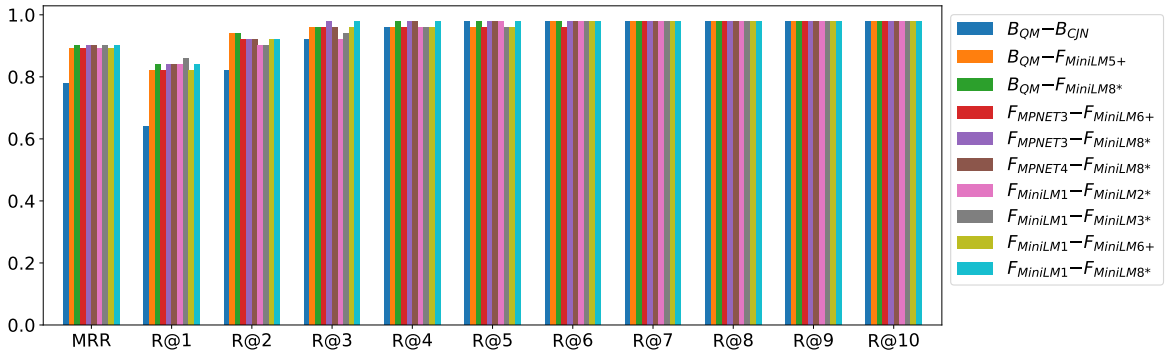


Figure 9.14: Neural CJN Ranking - IMDb

Figure 9.15 shows the results for the MONDIAL dataset. The baseline model $B_{QM}-B_{CJN}$ performs equally well compared to fine-tuned models. This performance equality might be attributed to the dataset's nature, where simpler models can achieve high performance due to less complexity in the data. All models achieved the same MRR of 0.97, and reached a recall plateau at R@2, indicating that there is minimal variation in performance across different CJN models for this dataset.

Figure 9.16 shows the results for the Yelp dataset. The baseline model $B_{QM}-B_{CJN}$ had an MRR of 0.85, significantly lower than the top-performing fine-tuned models, highlighting the efficacy of transformer-based approaches. Several fine-tuned models achieve

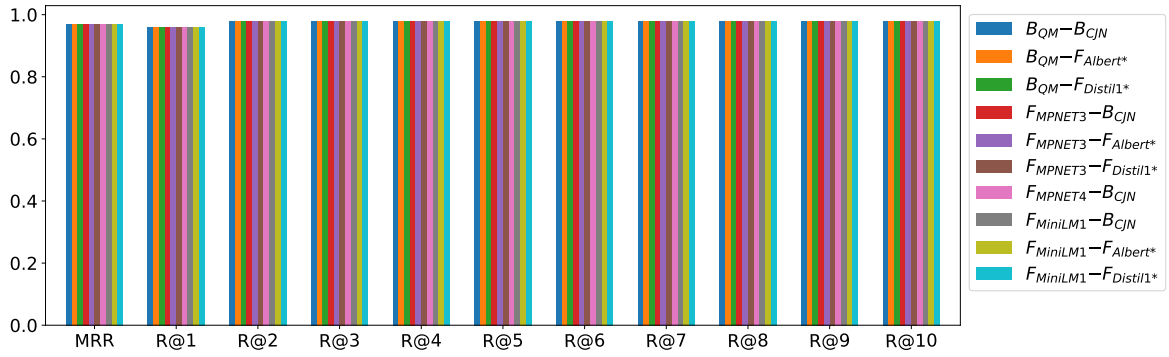


Figure 9.15: Neural CJN Ranking - MONDIAL

perfect recall from R@2 onwards, indicating these models are highly effective at ranking the most relevant CJNs at the top. Models like F_{MPNET3}−$F_{Distil3*}$, F_{MPNET4}−$F_{Distil3*}$, and $F_{MiniLM1}$−$F_{Distil3*}$ achieved an MRR of 0.98, with perfect recall from R@2 onwards, demonstrating outstanding performance in ranking relevant CJNs.

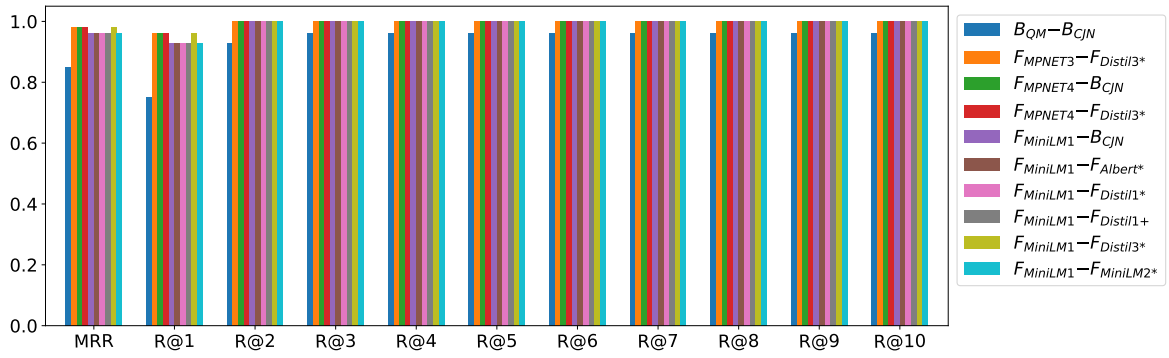


Figure 9.16: Neural CJN Ranking - Yelp

Discussion

The comprehensive evaluation of CJN models across different datasets highlights the superiority of fine-tuned transformer models over traditional Bayesian models. In the IMDB and Yelp datasets, fine-tuned models demonstrated substantial improvements in Mean Reciprocal Rank (MRR) and recall scores, particularly at lower recall points. This underscores the ability of transformer models to effectively rank the most relevant CJNs at the top positions, making them highly suitable for practical applications in information retrieval systems.

In the IMDB dataset, models like B_{QM}−$F_{MiniLM8*}$ and F_{MPNET3}−$F_{MiniLM8*}$ achieved the highest MRR of 0.9, far surpassing the baseline's 0.78. The consistent high performance across various recall points further demonstrates their reliability in identifying relevant context join networks. The use of multivalue aggregation approaches played a crucial role in this success.

For the Yelp dataset, fine-tuned models reached near-perfect or perfect recall scores, with models such as $F_{MPNET3}-F_{Distil3*}$ achieving an MRR of 0.98. These models consistently ranked the most relevant CJNs at the top positions, showcasing the effectiveness of transformer-based approaches over traditional Bayesian methods for high-accuracy relevance ranking.

In contrast, the Mondial dataset revealed that both Bayesian and fine-tuned transformer models performed equally well, with all models achieving an MRR of 0.97. This suggests that for datasets with less complexity, simpler models can be as effective as more complex transformer-based approaches. The minimal performance variation across different CJN models for this dataset highlights the suitability of simpler models in certain contexts, where the added complexity of fine-tuning may not yield significant advantages.

Overall, the findings indicate that fine-tuned transformer models, particularly those involving MiniLM and MPNET, offer substantial improvements in MRR and recall metrics across various datasets. The consistent high performance underscores their potential for enhancing information retrieval systems. However, the effectiveness of simpler models in certain datasets suggests that the choice of model should be context-dependent, balancing complexity and performance based on the dataset characteristics.

9.4.3 Performance Analysis: Neural Models

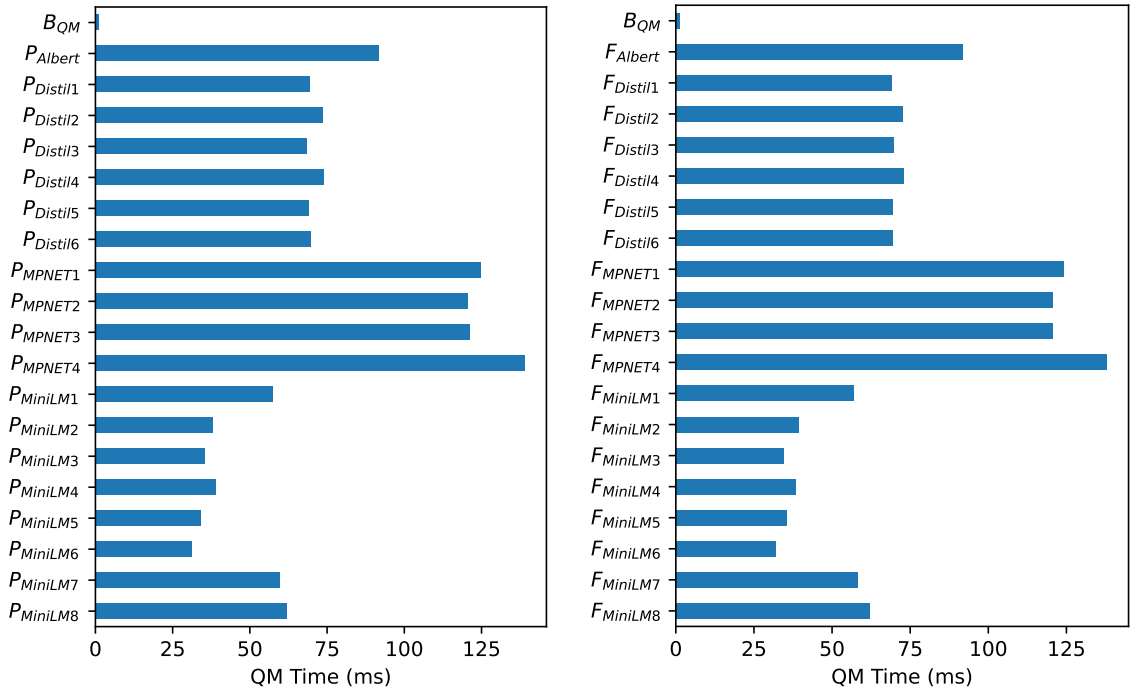
This section presents the time performance analysis of the models for the QM and CJN ranking tasks. We compare the baseline Bayesian model with various transformer-based models to evaluate their efficiency across different datasets.

QM Ranking

In this experiment, we evaluated the time performance of the neural QM ranking models presented in Section 9.4.1. We compared the baseline Bayesian model (B_{QM}) with several pre-trained and fine-tuned transformer-based models.

Figure 9.17 shows the time performance evaluation on the IMDb dataset. The Bayesian model (B_{QM}) exhibited a QM time of 1.13 milliseconds, outperforming all transformer-based models. Among the fine-tuned transformer models, $F_{MiniLM6}$ achieved a QM time of 32.07 milliseconds, $F_{MiniLM1}$ at 56.86 milliseconds, F_{MPNET3} at 120.56 milliseconds, and F_{MPNET4} at 137.99 milliseconds. The pre-trained models demonstrated similar trends, with $P_{MiniLM6}$ at 31.36 milliseconds, $P_{MiniLM1}$ at 57.45 milliseconds, P_{MPNET3} at 121.36 milliseconds, and P_{MPNET4} at 139.06 milliseconds. These results indicate that the Bayesian model is more efficient in terms of time performance for the IMDb dataset.

Figure 9.18 shows the time performance evaluation on the MONDIAL dataset. The Bayesian model exhibited a QM time of 0.03 milliseconds, demonstrating superior efficiency



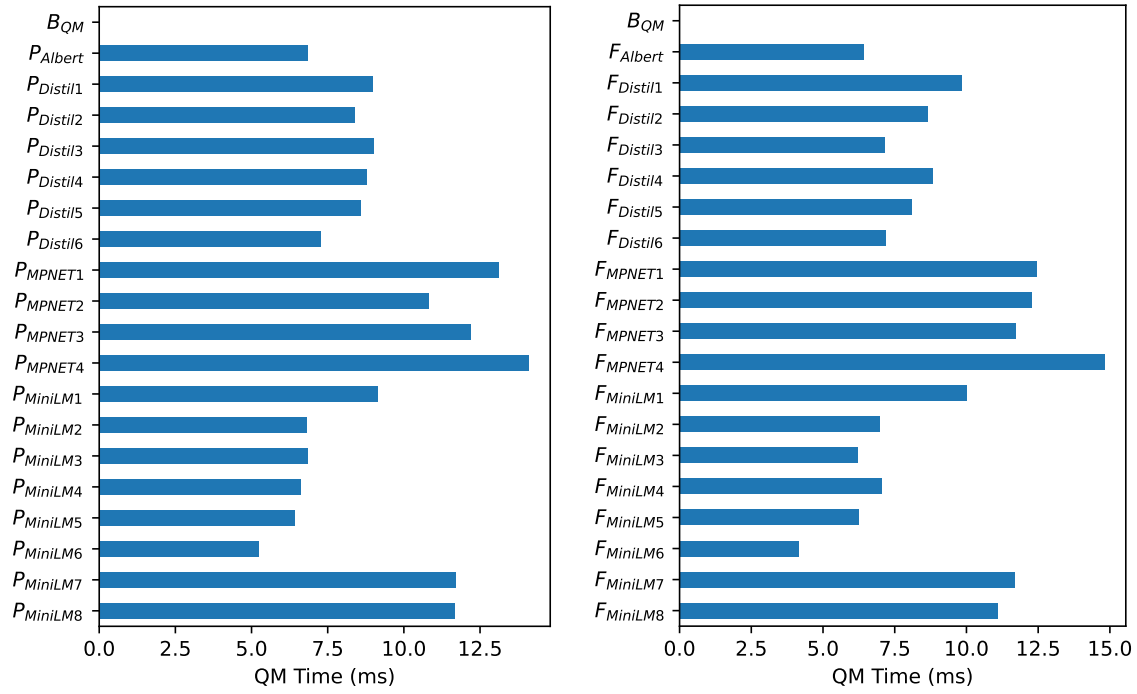
(a) Pre-trained Models (b) Fine-tuned Models

Figure 9.17: Evaluation of the performance of Neural QM models on the IMDb dataset.

compared to transformer models. Among the fine-tuned models, $F_{MiniLM6}$ had a QM time of 4.18 milliseconds, $F_{MiniLM1}$ at 10.02 milliseconds, F_{MPNET3} at 11.72 milliseconds, and F_{MPNET4} at 14.81 milliseconds. Pre-trained models also followed this pattern, with $P_{MiniLM6}$ at 5.22 milliseconds, $P_{MiniLM1}$ at 9.16 milliseconds, P_{MPNET3} at 12.18 milliseconds, and P_{MPNET4} at 14.10 milliseconds. These results highlight the efficiency of the Bayesian model for the MONDIAL dataset.

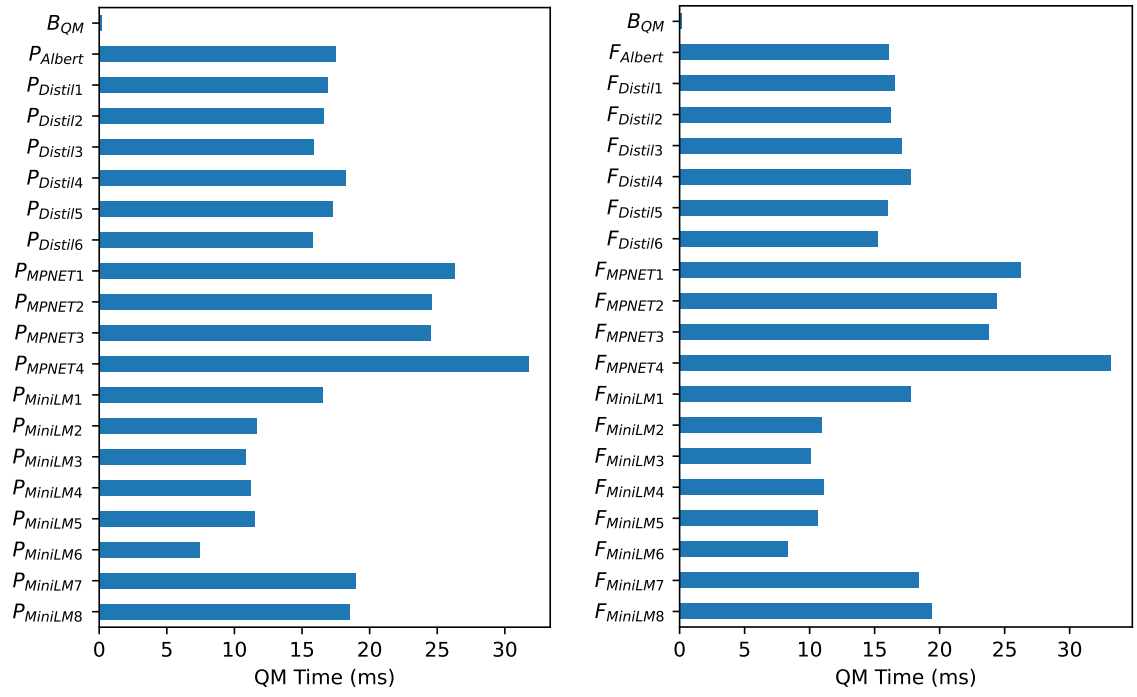
Figure 9.19 shows the time performance evaluation on the Yelp dataset. The Bayesian model showed the fastest QM time of 0.17 milliseconds. Among the fine-tuned transformer models, $F_{MiniLM6}$ achieved a QM time of 8.29 milliseconds, $F_{MiniLM1}$ at 17.80 milliseconds, F_{MPNET3} at 23.82 milliseconds, and F_{MPNET4} at 33.16 milliseconds. The pre-trained versions displayed similar trends, with $P_{MiniLM6}$ at 7.42 milliseconds, $P_{MiniLM1}$ at 16.51 milliseconds, P_{MPNET3} at 24.51 milliseconds, and P_{MPNET4} at 31.77 milliseconds. Again, the Bayesian model outperformed transformer models in terms of time performance for the Yelp dataset.

Overall, the time performance evaluations across the IMDb, MONDIAL, and Yelp datasets indicate that the Bayesian model generally outperforms transformer-based models in QM time. Transformer models showed relatively slower performance due to factors such as computational overhead, the self-attention mechanism's quadratic time complexity, the vast number of parameters, and the need for multiple layers. Specifically, models like $F_{MiniLM1}$,



(a) Pre-trained Models (b) Fine-tuned Models

Figure 9.18: Evaluation of the performance of Neural QM models on the MONDIAL dataset.



(a) Pre-trained Models (b) Fine-tuned Models

Figure 9.19: Evaluation of the performance of Neural QM models on the Yelp dataset.

F_{MPNET3}, and F_{MPNET4}, although not the fastest, performed well. In the experiments detailed in Section 9.4.1, these models also achieved the best results for MRR, recall, and

max recall position.

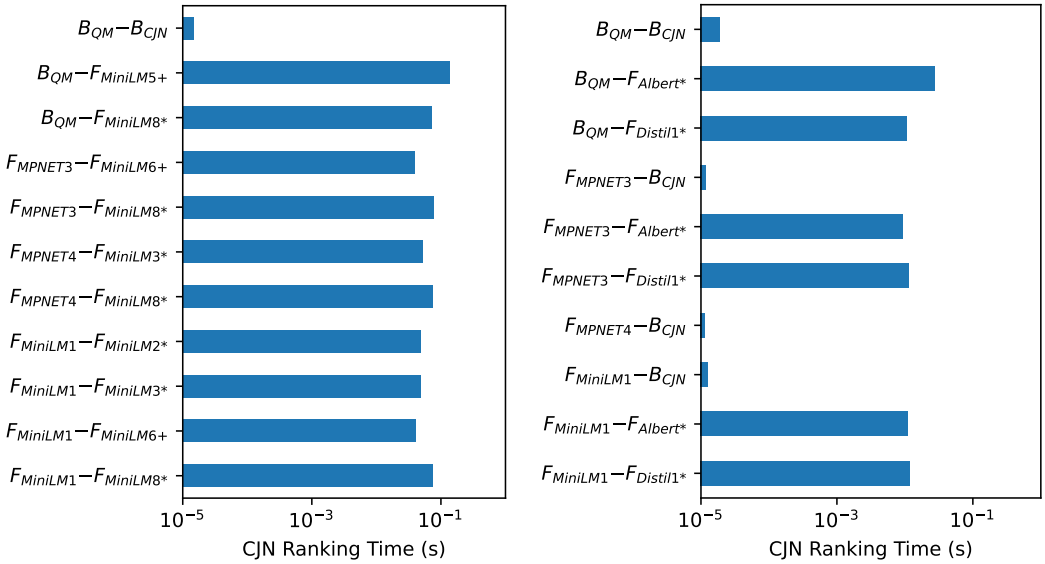
CJN Ranking

In this experiment, we evaluated the time performance of neural CJN ranking models, comparing the baseline Bayesian model ($B_{QM}-B_{CJN}$) with the fine-tuned transformer-based models presented in Section 9.4.2. Notably, we chose not to consider the CJN generation time in this evaluation because it is significantly longer than the ranking time, which would complicate the data analysis on the charts. We anticipated that the Bayesian model would be faster because it leverages QM ranking and penalizes CJNs by their length, which is a straightforward process. In contrast, transformer-based models rely on the CJN linearization process, which involves translating CJNs into SQL queries, executing them against the DBMS, and processing the results into sentences — a time-consuming process.

Figure 9.20a shows the time performance evaluation on the IMDb Dataset. The Bayesian model ($B_{QM}-B_{CJN}$) achieved the fastest time of 0.0000148 milliseconds, significantly outperforming all other models. Fine-tuned models showed varied performance, with $F_{MiniLM1}-F_{MiniLM6+}$ being the quickest at 39.76 milliseconds. Other fine-tuned models, such as $B_{QM}-F_{MiniLM8}$ and $F_{MPNET3}-F_{MiniLM8}$, also performed well but were slower compared to the fastest fine-tuned models. This indicates that while the Bayesian approach is highly efficient for CJN ranking in the IMDb dataset, fine-tuned models, especially those using MiniLM architectures, provide competitive results, although they remain significantly slower than the Bayesian model.

Figure 9.20b shows the time performance evaluation on the MONDIAL dataset. For the MONDIAL dataset, the Bayesian model ($B_{QM}-B_{CJN}$) achieved the fastest time of 0.0000189 milliseconds. Among the fine-tuned models, $B_{QM}-F_{Distil1}$ was the fastest at 10.62 milliseconds, followed closely by $F_{MiniLM1}-F_{MiniLM6+}$ at 8.59 milliseconds. Other fine-tuned models, such as $F_{MPNET3}-F_{MiniLM3}$ and $F_{MPNET4}-F_{MiniLM8*}$, also performed well but remained significantly slower than the Bayesian model. However, it is worth noting that the model $B_{QM}-B_{CJN}$ was slower than $F_{MPNET3}-B_{CJN}$, $F_{MPNET4}-B_{CJN}$, and $F_{MiniLM1}-B_{CJN}$ because these latter models require fewer QMs to be generated, resulting in slightly faster performance. This dataset highlights that while the Bayesian approach is the most efficient for CJN ranking, fine-tuned models, particularly those involving Distil architectures, offer competitive performance, albeit with higher elapsed times.

Figure 9.21 shows the time performance evaluation on the Yelp dataset. The Bayesian model ($B_{QM}-B_{CJN}$) achieved the fastest time of 0.0000166 milliseconds. Among fine-tuned models, $F_{MiniLM1}-F_{Albert}$ was the fastest at 27.30 milliseconds, followed by $F_{MiniLM1}-F_{MiniLM2}$ at 29.17 milliseconds. However, some fine-tuned models, like



(a) IMDb (b) MONDIAL

Figure 9.20: Evaluation of the performance of neural CJN ranking models on datasets IMDb and MONDIAL.

$F_{MiniLM1}-F_{MiniLM6+}$, showed significantly longer times, up to 2.57 seconds. This dataset underscores the superior efficiency of the Bayesian model for CJN ranking while highlighting the variability in fine-tuned model performance.

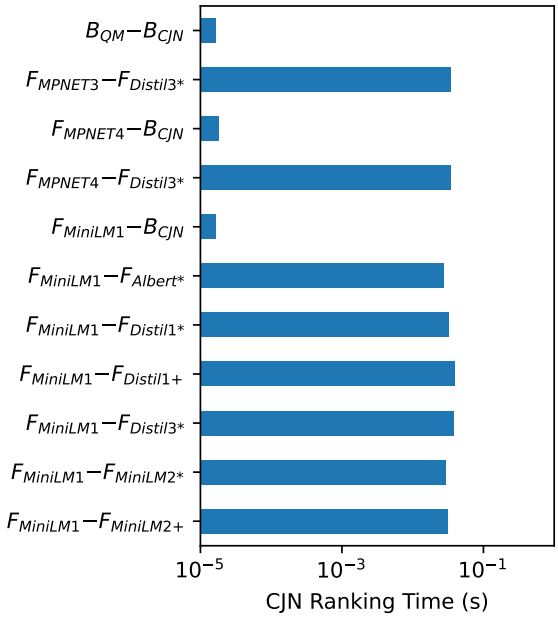


Figure 9.21: Evaluation of the performance of CJN Ranking models on the Yelp dataset.

Across all datasets, the Bayesian model ($B_{QM}-B_{CJN}$) consistently demonstrated superior time performance compared to the fine-tuned transformer-based models. The Bayesian

model's efficient use of QM ranking and straightforward penalization process made it significantly faster. The fine-tuned models, while offering competitive performance, generally exhibited higher elapsed times. This pattern was observed across all datasets, with the Bayesian model achieving the fastest times in every instance. These findings underscore the effectiveness of the Bayesian approach for time-efficient CJN ranking, especially in scenarios where rapid query processing is crucial.

9.4.4 Final Remarks

In the experiments reported, we compared the Bayesian and Neural Ranking approaches for CJN and QM ranking tasks. The Bayesian ranking method is a fully unsupervised approach, simplifying its implementation and deployment. On the other hand, the Neural Ranking approach has shown significant improvements in metrics such as MRR and R@K for CJN ranking and has also enhanced the QM ranking, although it required fine-tuning to achieve these results.

Performance-wise, our experiments revealed that the Bayesian approach is faster for both QM ranking and CJN ranking compared to the transformer-based models. The Bayesian model consistently demonstrated superior time performance across all datasets, while the neural models, despite being slower, showed decent performance in other evaluation metrics like MRR and recall. Nonetheless, with continuous advancements in neural models, we anticipate that the neural approach will not only become more viable but will also yield superior results over time.

It is also important to note that we were unable to compare Lathe's Neural Ranking with other systems due to the limitations of the results reported in QUEST, which were specific to the MONDIAL dataset. In this dataset, the neural ranking showed no improvement but also no setback compared to other methods.

Chapter 10

PyLatheDB

In this chapter, we present *PyLatheDB*, a Python library for Keyword Search over Relational Databases[Martins et al., 2023a]. PyLatheDB implements Lathe, and, for this reason, allows developers to easily run Lathe or incorporate its features, such as keyword matching, into their own applications. The library is written in Python and implements all the steps from Lathe. Table 10.1 presents the main modules available in the library, with a brief description of each function.

Module	**Description**
Database Handler	Interfaces the DB connections, evaluates CJNs, and iterates over the schema and the values of the database
Index Handler	Generates and manages the Schema Index, Value Index and Schema Graph
KM Handler	Generates Keyword Matches and manages the Similarity Functions for the Schema-Keyword Matching
QM Handler	Generates and ranks Query Matches
CJN Handler	Generates CJNs and translates them to SQL queries

Table 10.1: Main Modules Implemented in PyLatheDB.

To demonstrate PyLatheDB, we have created a Jupyter notebook, which is available alongside the library source code[1]. The notebook includes an environment preparation step and the processing of keyword queries against the IMDb and MONDIAL datasets.

For instance, consider the keyword query "*julia roberts films*". Figure 10.1 contains a screenshot from an execution of this notebook that shows the keyword matches (SKMs and VKMs) found in the database for the query keyword, and the query matches (QMs) built from the combinations of the keyword matches. Figure 10.2 presents two possible results generated by PyLatheDB for the given query. Each result comprises a graph representation of the CJN, its translation into an SQL query, and the returned answer from the RDBMS.

[1] https://github.com/bdri-ufam/PyLatheDB

```
results·=·lathe.keyword_search('julia·roberts·films')
results.kms()
results.qms()
```

SKMs:

```
MOVIE.s(*{films})
```

VKMs:

```
MOVIEINFO.v(info{julia})
PERSON.v(name{julia})
CHARACTER.v(name{julia})
MOVIE.v(title{julia})
MOVIEINFO.v(info{roberts})
PERSON.v(name{roberts})
CHARACTER.v(name{roberts})
MOVIE.v(title{roberts})
MOVIEINFO.v(info{films})
CHARACTER.v(name{films})
MOVIE.v(title{films})
MOVIEINFO.v(info{julia,roberts})
MOVIEINFO.v(info{julia,films})
PERSON.v(name{julia,roberts})
MOVIE.v(title{julia,roberts})
```

1st QM:

```
{PERSON.v(name{julia,roberts}), MOVIE.s(*{films})}
```

2nd QM:

```
{MOVIE.s(*{films}).v(title{julia,roberts})}
```

3rd QM:

```
{MOVIEINFO.v(info{julia,roberts}), MOVIE.s(*{films})}
```

4th QM:

```
{PERSON.v(name{roberts}), MOVIE.s(*{films}), PERSON.v(name{julia})}
```

5th QM:

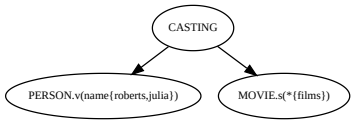
```
{PERSON.v(name{julia,roberts}), MOVIE.v(title{films})}
```

Figure 10.1: KMs and QMs for the query "*julia roberts films*"

The first CJN retrieves the movies in which Julia Roberts starred, thus satisfying the original user intent. On the other hand, the fourth CJN returned by the library retrieves movies in which two different persons, whose names respectively match the keywords "*julia*" and "*roberts*", e.g. Raul Julia and Robert Harvey, participated.

1st CJN:

Graph:



SQL:

```
SELECT
      t3.*,
      t1.name
FROM
      person t1
      JOIN casting t2 ON t2.person_id = t1.id
      JOIN movie t3 ON t2.movie_id = t3.id
WHERE
      t1.name_tsvector @@ to_tsquery('roberts & julia');
```

Results:

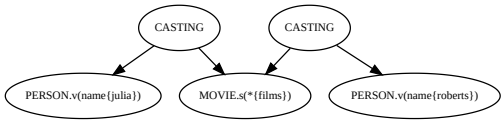
6 to 10 of 75 entries Filter

name	id	title	year
Roberts, Julia	330088	Michael Collins	1996
Roberts, Julia	165628	Everyone Says I Love You	1996
Roberts, Julia	102358	Confessions of a Dangerous Mind	2002
Roberts, Julia	461972	Steel Magnolias	1989
Roberts, Julia	397419	Pretty Woman	1990

Show 5 per page 1 2 3 10 15

4th CJN:

Graph:



SQL:

```
SELECT
      t3.*,
      t5.name,
      t1.name
FROM
      person t1
      JOIN casting t2 ON t2.person_id = t1.id
      JOIN movie t3 ON t2.movie_id = t3.id
      JOIN casting t4 ON t4.movie_id = t3.id
      JOIN person t5 ON t4.person_id = t5.id
WHERE
      t1.ctid <> t5.ctid
      AND t2.ctid <> t4.ctid
      AND t1.name_tsvector @@ to_tsquery('julia')
      AND t5.name_tsvector @@ to_tsquery('roberts');
```

Results:

1 to 5 of 2814 entries Filter

name	name	id	title	year
Dolan, Robert	Vera, Julia	393843	Por vida	2009
Harvey, Robert	Julia, Raul	527311	The Rookie	1990
Dubac, Robert	Julia, Raul	527311	The Rookie	1990
Ellis, Robert	Gordon, Julia Swayne	114233	Dark Secrets	1923
Gaillard, Robert	Gordon, Julia Swayne	285120	Lady Godiva	1911

Show 5 per page 1 2 10 100 500 560 563

Figure 10.2: CJNs for the query "*julia roberts films*"

The library also allows users to tune experimental parameters, such as the maximum number of QMs, the maximum number of CJN to be considered for each QM, and the number of CJN probed per QM by the eager evaluation.

Chapter 11

Further Developments

Building on the foundations of Lathe, this chapter explores two key works: SEREIA and Exverbis. These works extend PyLatheDB's modular architecture to new domains. SEREIA translates unstructured queries into document stores' native languages, enabling efficient keyword search without requiring users to understand the data structure. Exverbis extracts keyword queries from natural language queries, then leverages PyLatheDB's keyword matching and CJN generation modules. Together, these works showcase PyLatheDB's adaptability in addressing diverse data retrieval challenges.

11.1 SEREIA

With the modularization of keyword search features provided by PyLatheDB, we proposed SEREIA [Afonso et al., 2024, Afonso et al., 2021], a system that enables keyword search over document stores. More specifically, SEREIA aims at generating a structured query in the document stores' native language that corresponds to the non-structured query provided by the user. This approach avoids the necessity of users' previous knowledge on the document collections' structure and organization, besides the document store query language syntax and operations.

In practice, SEREIA uses PyLatheDB as an underlying system. It adapts the Database Handler to retrieve information from document stores, such as MongoDB, instead of relational databases. Furthermore, SEREIA uses a Structured Query Generation, which is more complex than the CJN Translation present in PyLatheDB, as the former uses the *Aggregation Framework Pipeline*[1]. SEREIA leverages the Keyword Match, Query Matching and the CJN Generation/Ranking from PyLatheDB.

[1] https://docs.mongodb.com/manual/aggregation/

To illustrate the workflow, consider the simplified excerpt of the *Yelp!* database, represented in Figure 11.1, which identifies documents by an ID in the top right corner and, also, illustrates how documents interconnect.

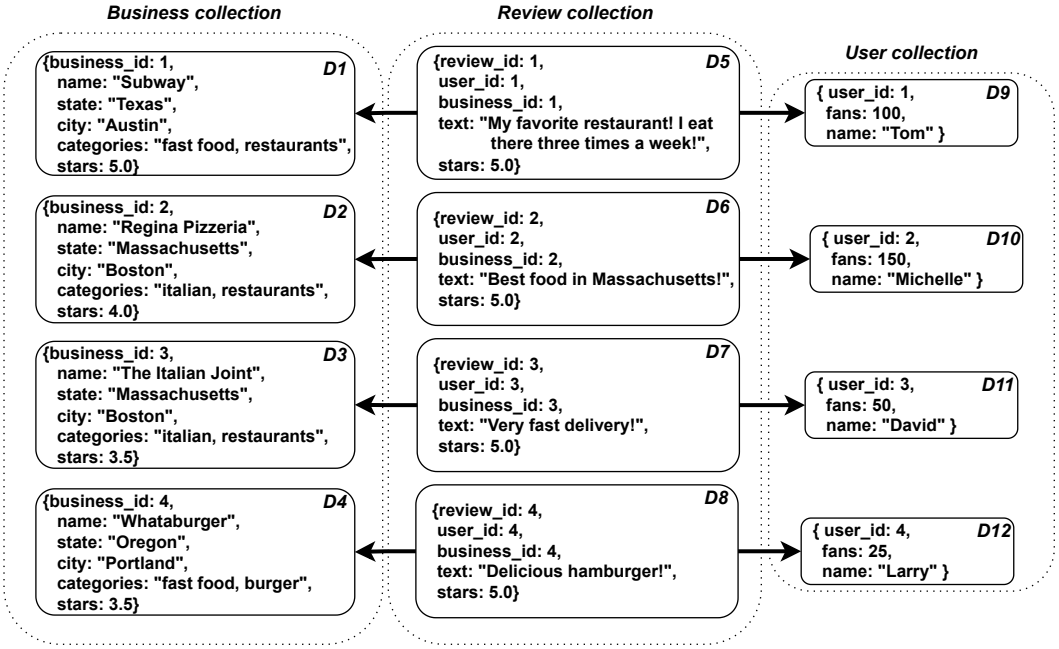


Figure 11.1: An excerpt from collections in the *Yelp!* database

Consider that a user inputs the keyword query "*italian restaurants reviewed michelle*", expecting to retrieve all documents containing data on italian restaurants that were reviewed by user Michelle. Figure 11.2 shows the Candidate Joining Network that satisfies the user intent. In this scenario, the structured query that correctly represents the CJN is shown in Figure 11.3 (left). Then, the expected results are retrieved by issuing the the structured query to the underlying document store.

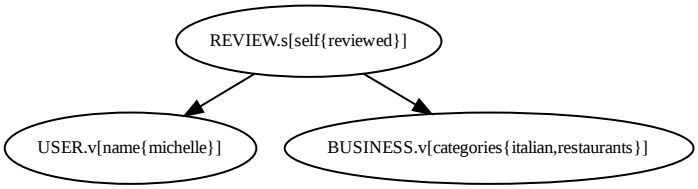


Figure 11.2: CJN for the keyword query

The result that satisfies this keyword query is obtained by joining data from documents $D2$, $D6$ and $D10$ from the excerpt, resulting in the output shown in Figure 11.3 (right).

```
db.user.aggregate([
{"$match": {"$expr": {"$regexMatch": {
   "input": "$user.name", "regex": "michelle",
   "options": "i"}}}},
{"$lookup": {"from": "review", "foreignField": "user_id",
            "localField": "user_id", "as": "review"}},
{"$unwind": "$review" },
{"$lookup": {"from": "business", "foreignField": "business_id",
            "localField": "review.business_id", "as": "business"}},
{"$unwind": "$business" },
{"$match": {"$expr": {"$regexMatch": {
   "input": "$categories", "regex": "restaurants",
   "options": "i"}}},
{"$match": {"$expr": {"$regexMatch": {
   "input": "$categories", "regex": "italian",
   "options": "i"}}}}])
```

```
{business_id: 2,
  name: "Regina Pizzeria",
  state: "Massachusetts",
  categories: "italian, restaurants",
  stars: 4.0,
  review: {
    review_id: 2,
    user_id: 2,
    business_id: 1,
    text: "Best food in Massachusetts!",
    stars: 5.0 },
  user" {
    user_id: 2,
    name: "Michelle",
    fans: 150 }}
```

Figure 11.3: MongoDB Structured Query (left) and its results for the keyword query (right).

11.2 Exverbis

While keyword-based search is adequate for users with exploratory intents, keyword queries are not as comprehensive as natural language queries (NLQ), which allows users issue more complex or specific queries. For this reason, Natural Language Interfaces for Databases (NLIDBs) have also been attracting recent interest in the literature and industry.

The main issues faced by NLIDBs can be summarized into two (1) natural language parsing and (2) keyword to database matching and join path generation in the database. The latter is also present in R-KwS systems, which allow us to take advantage of such systems to improve NLIDBs.

We proposed Exverbis [Citolin, 2021, Ferreira, 2022], a NLIDB system that leverages the keyword matching and the CJN generation from the PyLatheDB library. More specifically, Exverbis parses NLQs and translate them into keyword queries. Then, PyLatheDB is used for the keyword matching and the CJN generation, which is used for the generation of join paths. Finally, Exverbis generates SQL queries using the join paths generated by PyLatheDB.

To exemplify the process, take as example the query in Figure 11.4. The bold words in the query indicate keywords, that is, words which are likely to refer to database elements. The underline texts indicate operations implicitly expressed in the query.

return me the number of **papers** written by **"H. V. Jagadish"** in **VLDB conference** after **2000** with more than 200 **citations**

Figure 11.4: Example of a natural language query

Figure 11.5 shows the SQL query returned by PyLatheDB (left), and the returned SQL query, which was enchanced with implicit operations by Exverbis.

```
SELECT p.title, a.name, c.name,
p.year, p.citation_num
FROM publications p
JOIN writes w ON w.pid=p.pid
JOIN authors a ON a.aid=w.aid
JOIN conferences c ON c.cid=p.cid
WHERE a.name ILIKE '%h.v.jagadish%'
AND c.name ILIKE '%vldb%'
AND p.year = 2000
```

```
SELECT COUNT(DISTINCT p.title)
FROM publications p
JOIN writes w ON w.pid=p.pid
JOIN authors a ON a.aid=w.aid
JOIN conferences c ON c.cid=p.cid
WHERE a.name ILIKE '%h.v.jagadish%'
AND c.name ILIKE '%vldb%'
AND p.year>2000
AND p.citation_num>200
```

Figure 11.5: SQL query returned by PyLatheDB (left), and SQL query enhanced with implicit operations (right).

Chapter 12

Conclusions

In this thesis, we presented Lathe, a new relational keyword search (R-KwS) system for generating suitable SQL queries from keyword queries. Lathe is the first to address the problem of generating and ranking Candidate Joining Networks (CJNs) based on queries with keywords that can refer to either instance values or database schema elements, such as relations and attributes. Additionally, Lathe introduces four key innovations: (i) a novel Bayesian-based QM ranking algorithm that prioritizes relevant QMs, avoiding the processing of less likely answers; (ii) an effective Bayesian CJN ranking algorithm leveraging QM rankings to prioritize and evaluate relevant CJNs; (iii) an eager CJN evaluation strategy that discards spurious CJNs early; and (iv) a novel transformer-based neural approach for QM ranking and CJN ranking, leading to improved results on metrics such as Recall and $R@k$. We also presented a comprehensive set of experiments performed with query sets and datasets previously used in experiments with previous state-of-the-art R-KwS systems and methods. Our experiments indicate that Lathe can handle a wider variety of keyword queries while remaining highly effective, even for large databases with intricate schemas.

Our experience in the development of Lathe raised several ideas to apply its features in other R-KwS scenarios. For this reason, we developed PyLatheDB, an open-source Python library for Keyword Search over Relational Databases. This library allows developers to easily run Lathe or incorporate its features, such as keyword matching, into their own applications. For example, with the modularization of keyword search features provided by PyLatheDB, we proposed SEREIA, a system that enables keyword search over document stores, and Exverbis, a Natural Language Interface for Databases (NLIDB) system that leverages the keyword matching and the CJN generation from the PyLatheDB library.

Despite the significant advancements and positive results achieved with Lathe, there remain several promising directions for future research and development. Building on the foundation laid in this thesis, these directions aim to further enhance the capabilities and

performance of relational keyword search systems.

Future Work

Several possibilities for future research and development can further enhance the capabilities and performance of Lathe, ensuring its continued relevance and effectiveness in various contexts.

- **Improve the Efficiency of Transformer-Based Models:** Implement advanced optimization techniques such as knowledge distillation, pruning, and quantization to reduce the computational load of transformer models without sacrificing effectiveness. Leveraging the latest transformer architectures such as GPT-3 and their successors can enhance Lathe's ability to understand and process complex queries.
- **Run Experiments on More Datasets:** To validate Lathe's robustness and generalizability, experiments should be conducted on datasets from various domains, such as healthcare, finance, and e-commerce. Testing on databases with more intricate schema structures can highlight Lathe's strengths and areas for improvement in handling complex relationships and nested queries.
- **Explore Other Models for QM and CJN Ranking:** Investigate the potential of large language models like GPT-4 and beyond for QM and CJN ranking to offer enhanced contextual understanding and semantic matching capabilities. Combining LLMs with traditional machine learning models could leverage the strengths of both approaches, potentially leading to even better performance in ranking tasks.
- **Apply Keyword Search Over Relational Databases to Data Lakes:** Extend Lathe's keyword search capabilities to data lakes, allowing users to query across a more extensive and diverse set of data sources. Address the scalability challenges of querying vast amounts of unstructured data in data lakes by exploring techniques such as indexing and distributed processing to maintain efficient performance.
- **Use Semantic Labeling to Infer Schema References:** Implement semantic labeling techniques to improve Lathe's ability to infer schema references from keyword queries. Incorporating ontologies and knowledge graphs can provide a richer semantic context, aiding in more accurate mapping of keywords to database schema elements.
- **Use Lathe as an Intermediate System for Text-to-SQL:** Leverage Lathe's capabilities in a Text-to-SQL pipeline to enable users to interact with relational databases using

natural language queries. Develop intermediate representations of queries to bridge the gap between natural language inputs and SQL outputs, making the translation process more efficient and accurate.

- **Extend to Real-Time Querying and Feedback Loops:** Enhance Lathe to support real-time querying, making it more suitable for applications that require instant data retrieval and analysis. Incorporate user feedback loops to continuously refine and improve Lathe's performance.
- **Explore Cross-Lingual and Multi-Lingual Capabilities:** Expand Lathe to support multiple languages, making it more versatile and applicable in global contexts. Develop cross-lingual search capabilities to allow users to query databases in one language and retrieve relevant information from data stored in another language, enhancing the system's utility in multilingual environments.

These future directions aim to build upon the foundation laid by Lathe, pushing the boundaries of relational keyword search systems and enhancing their applicability and performance in real-world scenarios. By addressing these areas, we can continue to improve the effectiveness and efficiency of keyword search over relational databases, making data more accessible and usable for a broader range of users.

Bibliography

[Aditya et al., 2002] Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Sudarshanxe, S., et al. (2002). Banks: Browsing and keyword searching in relational databases. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 1083–1086. Elsevier.

[Affolter et al., 2019] Affolter, K., Stockinger, K., and Bernstein, A. (2019). A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28(5):793–819.

[Afonso et al., 2021] Afonso, A., Martins, P., and da Silva, A. (2021). Sereia-busca por palavras-chave em document stores. In *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados*, pages 133–144. SBC.

[Afonso et al., 2024] Afonso, A., Martins, P., and da Silva, A. (2024). Sereia: document store exploration through keywords. *Knowledge and Information Systems*, pages 1–32.

[Agrawal et al., 2002] Agrawal, S., Chaudhuri, S., and Das, G. (2002). Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings 18th International Conference on Data Engineering*, pages 5–16. IEEE.

[Arik and Pfister, 2021] Arik, S. Ö. and Pfister, T. (2021). Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687.

[Badaro et al., 2023] Badaro, G., Saeed, M., and Papotti, P. (2023). Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics*, 11:227–249.

[Baeza-Yates and Ribeiro-Neto, 2008] Baeza-Yates, R. and Ribeiro-Neto, B. (2008). *Modern Information Retrieval: The Concepts and Technology Behind Search*. Addison-Wesley Publishing Company, USA, 2nd edition.

[Baid et al., 2010] Baid, A., Rae, I., Li, J., Doan, A., and Naughton, J. (2010). Toward scalable keyword search over relational data. *Proceedings of the VLDB Endowment*, 3(1-2):140–149.

[Bergamaschi et al., 2011a] Bergamaschi, S., Domnori, E., Guerra, F., Trillo Lado, R., and Velegrakis, Y. (2011a). Keyword search over relational databases: a metadata approach. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 565–576. ACM.

[Bergamaschi et al., 2013] Bergamaschi, S., Guerra, F., Interlandi, M., Trillo-Lado, R., and Velegrakis, Y. (2013). Quest: A keyword search system for relational data based on semantic and machine learning techniques. *Proc. VLDB Endow.*, 6(12):1222–1225.

[Bergamaschi et al., 2016] Bergamaschi, S., Guerra, F., Interlandi, M., Trillo-Lado, R., and Velegrakis, Y. (2016). Combining user and database perspective for solving keyword queries over relational databases. *Information Systems*, 55:1–19.

[Bergamaschi et al., 2011b] Bergamaschi, S., Guerra, F., Rota, S., and Velegrakis, Y. (2011b). A hidden markov model approach to keyword-based search over relational databases. In *International Conference on Conceptual Modeling*, pages 411–420. Springer.

[Bhalotia et al., 2002] Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. (2002). Keyword searching and browsing in databases using banks. In *Proceedings 18th International Conference on Data Engineering*, pages 431–440. IEEE.

[Blunschi et al., 2012] Blunschi, L., Jossen, C., Kossmann, D., Mori, M., and Stockinger, K. (2012). Soda: Generating sql for business users. *Proceedings of the VLDB Endowment*, 5(10):932–943.

[Borisov et al., 2022] Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., and Kasneci, G. (2022). Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*.

[Bourgeois and Lassalle, 1971] Bourgeois, F. and Lassalle, J.-C. (1971). An extension of the munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM*, 14(12):802–804.

[Citolin, 2021] Citolin, L. (2021). Exverbis: Exploiting latent words dependencies to improve natural language interfaces to databases. Master's thesis, Universidade Federal do Amazonas.

[Coffman and Weaver, 2010a] Coffman, J. and Weaver, A. C. (2010a). A framework for evaluating database keyword search strategies. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 729–738. ACM.

[Coffman and Weaver, 2010b] Coffman, J. and Weaver, A. C. (2010b). Structured data retrieval using cover density ranking. In *Proceedings of the 2nd International Workshop on Keyword Search on Structured Data*, page 1. ACM.

[Coffman and Weaver, 2012] Coffman, J. and Weaver, A. C. (2012). An empirical performance evaluation of relational keyword search techniques. *IEEE Transactions on Knowledge and Data Engineering*, 26(1):30–42.

[Cormen et al., 2009] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

[de Cristo et al., 2003] de Cristo, M. A. P., Calado, P. P., Da Silveira, M. d. L., Silva, I., Muntz, R., and Ribeiro-Neto, B. (2003). Bayesian belief networks for ir. *International Journal of Approximate Reasoning*, 34(2-3):163–179.

[de Oliveira et al., 2020] de Oliveira, P., da Silva, A., Moura, E., and de Freitas, R. (2020). Efficient match-based candidate network generation for keyword queries over relational databases. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1.

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[Ding et al., 2007] Ding, B., Yu, J. X., Wang, S., Qin, L., Zhang, X., and Lin, X. (2007). Finding top-k min-cost connected trees in databases. In *2007 IEEE 23rd international conference on data engineering*, pages 836–845. IEEE.

[Fang et al., 2024] Fang, X., Xu, W., Tan, F. A., Zhang, J., Hu, Z., Qi, Y., Nickleach, S., Socolinsky, D., Sengamedu, S., and Faloutsos, C. (2024). Large language models (llms) on tabular data: Prediction, generation, and understanding-a survey. *arXiv preprint arXiv:2402.17944*.

[Ferreira, 2022] Ferreira, B. C. C. (2022). An ir-based approach to keyword to database mapping in natural language database interfaces. Master's thesis, Universidade Federal do Amazonas.

[He et al., 2007] He, H., Wang, H., Yang, J., and Yu, P. S. (2007). Blinks: ranked keyword searches on graphs. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 305–316. ACM.

[Hearne and Wagner, 1973] Hearne, T. and Wagner, C. (1973). Minimal covers of finite sets. *Discrete Mathematics*, 5(3):247–251.

[Herzig et al., 2020] Herzig, J., Nowak, P. K., Müller, T., Piccinno, F., and Eisenschlos, J. (2020). Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

[Hristidis et al., 2003] Hristidis, V., Gravano, L., and Papakonstantinou, Y. (2003). Efficient ir-style keyword search over relational databases. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 850–861. VLDB Endowment.

[Hristidis and Papakonstantinou, 2002] Hristidis, V. and Papakonstantinou, Y. (2002). Discover: Keyword search in relational databases. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 670–681. Elsevier.

[Kacholia et al., 2005] Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., and Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st international conference on Very large data bases*, pages 505–516. VLDB Endowment.

[Kasneci et al., 2009] Kasneci, G., Ramanath, M., Sozio, M., Suchanek, F. M., and Weikum, G. (2009). Star: Steiner-tree approximation in relationship graphs. In *2009 IEEE 25th International Conference on Data Engineering*, pages 868–879. IEEE.

[Keselj, 2009] Keselj, V. (2009). Speech and language processing daniel jurafsky and james h. martin (stanford university and university of colorado at boulder) pearson prentice hall, isbn 978-0-13-187321-6.

[Li et al., 2023] Li, P., He, Y., Yashar, D., Cui, W., Ge, S., Zhang, H., Fainman, D. R., Zhang, D., and Chaudhuri, S. (2023). Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263*.

[Liu et al., 2006] Liu, F., Yu, C., Meng, W., and Chowdhury, A. (2006). Effective keyword search in relational databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 563–574. ACM.

[Luo et al., 2007] Luo, Y., Lin, X., Wang, W., and Zhou, X. (2007). Spark: top-k keyword query in relational databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 115–126. ACM.

[Martins et al., 2023a] Martins, P., Afonso, A., and Da Silva, A. (2023a). Pylathedb-a library for relational keyword search with support to schema references. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 3627–3630. IEEE.

[Martins et al., 2023b] Martins, P., Da Silva, A., Afonso, A., Cavalcanti, J., and De Moura, E. (2023b). Supporting schema references in keyword queries over relational databases. *IEEE Access*.

[May, 1999] May, W. (1999). Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik. Available from http://dbis.informatik.uni-goettingen.de/Mondial.

[Mesquita et al., 2007] Mesquita, F., da Silva, A. S., de Moura, E. S., Calado, P., and Laender, A. H. (2007). Labrador: Efficiently publishing relational databases on the web by using keyword-based query interfaces. *Information Processing & Management*, 43(4):983–1004.

[Miller, 1998] Miller, G. A. (1998). *WordNet: An electronic lexical database*. MIT press.

[Oliveira et al., 2015] Oliveira, P., da Silva, A., and de Moura, E. (2015). Ranking candidate networks of relations to improve keyword search over relational databases. In *2015 IEEE 31st International Conference on Data Engineering*, pages 399–410. IEEE.

[Oliveira et al., 2018] Oliveira, P., da Silva, A., de Moura, E., and Rodrigues, R. (2018). Match-based candidate network generation for keyword queries over relational databases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1344–1347. IEEE.

[Pearl, 2014] Pearl, J. (2014). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

[Pedersen et al., 2004] Pedersen, T., Patwardhan, S., and Michelizzi, J. (2004). Wordnet:: Similarity: measuring the relatedness of concepts. In *Demonstration papers at HLT-NAACL 2004*, pages 38–41. Association for Computational Linguistics.

[Ramada et al., 2020] Ramada, M. S., da Silva, J. C., and de Sá Leitão-Júnior, P. (2020). From keywords to relational database content: A semantic mapping method. *Information Systems*, 88:101460.

[Reimers and Gurevych, 2019] Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

[Ribeiro and Muntz, 1996] Ribeiro, B. A. and Muntz, R. (1996). A belief network model for ir. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. Citeseer.

[Salton and Buckley, 1988] Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.

[Shwartz-Ziv and Armon, 2022] Shwartz-Ziv, R. and Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90.

[Tata and Lohman, 2008] Tata, S. and Lohman, G. M. (2008). Sqak: doing more with keywords. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 889–902. ACM.

[Thakur et al., 2021] Thakur, N., Reimers, N., Daxenberger, J., and Gurevych, I. (2021). Augmented SBERT: Data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. In Toutanova, K., Rumshisky, A., Zettlemoyer, L., Hakkani-Tur, D., Beltagy, I., Bethard, S., Cotterell, R., Chakraborty, T., and Zhou, Y., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 296–310, Online. Association for Computational Linguistics.

[Trabelsi et al., 2022] Trabelsi, M., Chen, Z., Zhang, S., Davison, B. D., and Heflin, J. (2022). Strubert: Structure-aware bert for table search and matching. In *Proceedings of the ACM Web Conference 2022*, pages 442–451.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Wu and Palmer, 1994] Wu, Z. and Palmer, M. (1994). Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138. Association for Computational Linguistics.

[Yaghmazadeh et al., 2017] Yaghmazadeh, N., Wang, Y., Dillig, I., and Dillig, T. (2017). Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA).

[Yin et al., 2020] Yin, P., Neubig, G., tau Yih, W., and Riedel, S. (2020). TaBERT: Pretraining for joint understanding of textual and tabular data. In *Annual Conference of the Association for Computational Linguistics (ACL)*.

[Zaki, 2000] Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3):372–390.

[Zhang and Balog, 2018] Zhang, S. and Balog, K. (2018). Ad hoc table retrieval using semantic similarity. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 1553–1562, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

[Zhang and Balog, 2021] Zhang, S. and Balog, K. (2021). Semantic table retrieval using keyword and table queries. *ACM Transactions on the Web (TWEB)*, 15(3):1–33.

[Zhang et al., 2023] Zhang, T., Yue, X., Li, Y., and Sun, H. (2023). Tablellama: Towards open large generalist models for tables. *arXiv preprint arXiv:2311.09206*.

[Zhang et al., 2024] Zhang, X., Zhang, J., Ma, Z., Li, Y., Zhang, B., Li, G., Yao, Z., Xu, K., Zhou, J., Zhang-Li, D., et al. (2024). Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. *arXiv preprint arXiv:2403.19318*.

Appendix A

VKMGen Algorithm

As shown in Algorithm 4, Lathe retrieves tuples from the database in which the keywords occur and uses them to generate value-keyword matches. Initially, the VKMGen Algorithm takes the occurrences of each keyword from the *Value Index* and form partial value-keyword matches, which are not guaranteed to be disjoint sets yet (Lines 3-8). The pool of VKMs is represented by the *Hash Table* P, whose keys are KMs and values are sets of tuple IDs.

Algorithm 4: VKMGen(Q)

Input: A keyword query $Q{=}\{k_1, k_2, \ldots, k_m\}$
Output: The set of value-keyword matches VK

```
1  let I_V the Value Index
2  let P be a Hash Table.
3  for keyword k_i ∈ Q do
4      if k_i ∈ I_V then
5          for relation R_j ∈ I_V[k_i] do
6              for attribute A_k ∈ I_V[k_i][R_j] do
7                  let KM be the partial keyword match R_j^V[A_k^{k_i}]
8                  P[KM] ← I_V[k_i][R_j][A_k]
9  P ← VKMIter(P)
10 for value-keyword match KM_u ∈ P do
11     VK ← VK ∪ {KM_u}
12 return VK
```

Next, Lathe ensures that VKMs are disjoint sets through the Algorithm 5, VKMInter, which is based on the ECLAT algorithm [Zaki, 2000] for finding frequent itemsets. VKMInter looks for non-empty intersections of the partial value-keyword matches recursively until all of

them are disjoint sets, and thus, proper VKMs. These intersections are calculated as follows:

$$KM_1 \cap KM_2 = \begin{cases} \emptyset & , \text{if } R_a \neq R_b \\ R^V_{ab}[A^{K_{ab,1}}_{ab,1}, \ldots, A^{K_{ab,m}}_{ab,m}] & , \text{if } R_a = R_b \end{cases}$$

where $KM_x = R^V_x[A^{K_{x,1}}_{x,1}, \ldots, A^{K_{x,m}}_{x,m}]$ for $x \in \{a, b\}$, and $K_{ab,i} = K_{a,i} \cup K_{b,i}$.

Algorithm 5: VKMInter(P)

Input: A Hash Table P whose keys are partial value-keyword matches and values are tuples.
Output: A Hash Table P whose keys are proper value-keyword matches and values are tuples.

1 **let** P_{next} be a Hash Table.
2 **let** R be a Hash Table.
3 **for** value-keyword match $KM_u \in P$ **do**
4 $\quad R[KM_u] \leftarrow \emptyset$
5 **for** pair of keyword matches $\{KM_a, KM_b\} \in \binom{P}{2}$ **do**
6 $\quad KM_{ab} \leftarrow KM_a \cap KM_b$
7 $\quad T_{ab} \leftarrow P[KM_a] \cap P[KM_b]$
8 $\quad$ **if** $T_{ab} \neq \emptyset$ **and** KM_{ab} is valid **then**
9 $\quad\quad P_{next}[KM_{ab}] \leftarrow T_{ab}$
10 $\quad\quad R[KM_a] \leftarrow R[KM_a] \cup T_{ab}$
11 $\quad\quad R[KM_b] \leftarrow R[KM_b] \cup T_{ab}$
12 **for** value-keyword match $KM_u \in R$ **do**
13 $\quad P[KM_u] \leftarrow P[KM_u] - R[KM_u]$
14 $\quad$ **if** $P[KM_u] = \emptyset$ **then**
15 $\quad\quad$ **remove** KM_u from P
16 $\quad\quad P.remove(KM_u)$
17 $P_{next} \leftarrow$ **VKMInter**(P_{next})
18 **update** P **with** P_{next}
19 **return** P

VKMInter uses three hash tables: P, P_{next} and R. The pool P contains the partial VKMs of the current iteration. The pool P_{next} contains the partial VKMs for the next iteration. The pool R stores the tuple IDs to be removed from the VKMs of P at the end of the current iteration, turning the partial VKMs into proper value-keyword matches.

VKMInter first defines the hash tables P_{next} and R, then initializes R with empty sets (Lines 1-4). Next, the algorithm iterates over all pairs $\{KM_a, KM_b\}$ of VKMs in P and tries to create a new keyword match KM_{ab}, which is the intersection of KM_a e KM_b (Lines 5-11). If KM_{ab} is valid, that is, if KM_a e KM_b are VKMs over the same database relation, and the

tuples T_{ab} within KM_{ab} are not empty, then we add KM_{ab} to the next iteration pool P_{next} and add the tuples T_{ab} to R for removal after the iteration (Lines 8-11). After all the possible intersections are processed, VKMInter iterates over R and removes the tuples for each VKM of the pool P, making them proper disjoint keyword matches (Lines 12-16). Lastly, VKMInter recursively process the pool P_{next} for the next iteration, then it updates and returns the current pool P (Lines 18-19).

After the execution of VKMInter, in Line 9 of VKMGen, we obtained the value-keyword matches and their tuples. As the sets of tuples are only required for the generation of VKMs, VKMGen generates and outputs the set of value-keyword matches, ignoring the tuples from P (Lines 10-11).

Appendix B

SKMGen Algorithm

The generation of schema-keyword matches uses a structure we call the *Schema Index*, which is created in a preprocessing phase, alongside with the Value Index. This index stores information about the database schema and statistics about attributes, which are used for the ranking of QMs, which will be explained in Chapter 5. The stored information follows the structure below:

$$I_S = \{relation : \{attribute : \{(norm, maxfrequency)\}\}\}$$

The generation of SKMs is carried out by Algorithm 6, SKMGen. First, the algorithm iterates over the relations and attributes from the Schema Index. Then, SKMGen calculates the similarity between each keyword and schema element. It only considers the pairs with similarity above a defined threshold ε (Line 8), which are used to generate SKMs (Line 3).

Algorithm 6: SKMGen(Q)

Input: A keyword query Q=$\{k_1, k_2, \ldots, k_m\}$, the Schema Index I_S
Output: The set of schema-keyword matches SK

```
1  SK ← {}
2  for keyword k_i ∈ Q do
3      for relation R_j ∈ I_S do
4          if sim(k_i, R_j) ≥ ε then
5              let KM be the schema-keyword match R_j^S[self^{k_i}]
6              SK ← SK ∪ {KM})
7          for attribute A_l ∈ I_S[R_j] do
8              if sim(k_i, A_l) ≥ ε then
9                  let KM be the schema-keyword match R_j^S[A_l^{k_i}]
10                 SK ← SK ∪ {KM}
11 return SK
```

Appendix C

QMGen Algorithm

The generation of query matches is carried out by Algorithm 7, QMGen, which preserves the ideas proposed in MatCNGen [Oliveira et al., 2018], adapt them to keyword matches instead of tuple-sets. Let VK and SK be respectively sets of value-keyword matches, and schema-keyword matches previously generated. The algorithm looks for combinations of keyword matches in $P{=}VK{\cup}SK$ that form minimal covers for the query Q. At a first glance, this statement may suggest that we need to generate the whole power set of P to obtain the complete set of QMs. However, it can be shown that any minimal cover of a set of n elements has at most n subsets [Hearne and Wagner, 1973]. Therefore, no match for a query Q can be formed by more than $|Q|$ keyword matches. Also, as the QM ranking presented in Section 7.1 penalizes QMs with a large number of KMs, we can define a maximum QM size $t{\leq}|Q|$ to prune QMs which are less likely to be relevant. For this reason, QMGen iterates over all the subsets of P whose size is less than or equal to a maximum QM size t, which is at most the size of the query Q (Lines 3-4). Next, QMGen checks whether the combination M of keyword matches form a minimal cover for the query. The evaluation of minimal cover is carried out by Algorithm 8.

The algorithm MinimalCover iterates through the KMs from the combination M, generating a set C_M which comprise all keywords covered by M (Lines 1-5). Next, the algorithm checks whether M is total, that is, whether $C_M{=}Q$. Notice that since KMs can only associate an attribute or relation in the database schema to keyword from the query Q, that is $C_M{\subseteq}Q$, then we can imply that $C_M{=}Q$ if, and only if, $|C_M|{=}|Q|$ (Line 6). Next, MinimalCover checks whether M is minimal, that is, if we remove any keyword match from M it will no longer be total. For this reason, MinimalCover iterates again through the KMs and, for each one, it generates a set C_{KM} which comprise all keywords covered by KM. Then, the algorithm check whether the set difference of $C_M{\setminus}C_{KM}$ is still equal to Q, which can be achieved by comparing $|C_M{\setminus}C_{KM}|{=}|Q|$.

Algorithm 7: QMGen(Q, VK, SK)

Input: A keyword query Q=$\{k_1, k_2, \ldots, k_m\}$
The set of value-keyword matches VK
The set of schema-keyword matches SK
The maximum QM size t
Output: The set of query matches QM

```
1  P = VK ∪ SK
2  QM ← ∅
3  for i ∈ {1, ..., min(|Q|, t)} do
4      for combination of keyword matches M ∈ (P choose i) do
5          if MinimalCover(M, Q) then
6              M ← MergeKeywordMatches(M)
7              QM ← QM ∪ {M}
8  return QM
```

Algorithm 8: MinimalCover(Q, M)

Input: A keyword query Q=$\{k_1, k_2, \ldots, k_m\}$
The set of keyword matches M
Output: If the set of keywords from M forms a minimal cover over Q

```
1  C_M ← ∅
2  for keyword match KM ∈ M do
3      let KM be R^X[A_1^{K_1}, ..., A_m^{K_m}], where X ∈ {S, V}
4      for i ∈ {1, ..., m} do
5          C_M ← C_M ∪ K_i
6  if |C_M| ≠ |Q| then
7      return False
8  for keyword match KM ∈ M do
9      let KM be R^X[A_1^{K_1}, ..., A_m^{K_m}], where X ∈ {S, V}
10     C_KM = ∅
11     for i ∈ {1, ..., m} do
12         C_KM ← C_KM ∪ K_i
13     if |C_M \ C_KM| = Q then
14         return False
15 return True
```

If M forms a minimal cover for Q, then M is considered a query match. However, M may have some keyword matches which can be merged, especially SKMs. The merging of KMs from M is carried out by Algorithm 9. Notice that we cannot merge two VKMs since they are disjoint sets, however we can merge a schema-keyword match with both a SKM or

a VKM. The algorithm MergeKeywordMatches uses the two *hash tables* P_{VK} and P_{SK} to store, respectively, the VKMs and SKMs based on the relation they are built upon (Lines 1-12). Next, the algorithm iterates through the relations present in P_{SK} and tries to merge all possible KMs from that relation, resulting in a keyword match KM_{merged}. KM_{merged} starts as a keyword-free match but it is merged with all existent SKMs (Lines 14-17), then it is merged an arbitrary value-keyword match VKM, if existent (Lines 18-21). Lastly, KM_{merged} and all values-keyword matches except VKM are added to the query match M', which is returned at the end of MergeKeywordMatches (Lines 22-23).

After merging all the possible elements from the query match M, QMGen adds M to the set of query matches QM, which is returned at the end of the algorithm.

Algorithm 9: MergeKeywordMatches(Q, M)

Input: The set of keyword matches M
Output: The set of keyword matches M'

1 **let** P_{VK} be a Hash Table.
2 **let** P_{SK} be a Hash Table.
3 **for** $KM \in M$ **do**
4 **let** KM be $R^X[A_1^{K_1}, \ldots, A_m^{K_m}]$, where $X \in \{S, V\}$
5 $P_{VK}[R] \leftarrow \emptyset$
6 $P_{SK}[R] \leftarrow \emptyset$
7 **for** $KM \in M$ **do**
8 **let** KM be $R^X[A_1^{K_1}, \ldots, A_m^{K_m}]$, where $X \in \{S, V\}$
9 **if** $X = S$ **then**
10 $P_{SK}[R] \leftarrow P_{SK}[R] \cup \{KM\}$
11 **else**
12 $P_{VK}[R] \leftarrow P_{VK}[R] \cup \{KM\}$
13 $M' \leftarrow \emptyset$
14 **for** $R \in P_{SK}$ **do**
15 **let** KM_{merged} be a keyword-free match from R
16 **for** $SKM \in P_{SK}[R]$ **do**
17 $KM_{merged} \leftarrow KM_{merged} \cap SKM$
18 **if** $P_{VK}[R] \neq \emptyset$ **then**
19 **let** VKM be an element from $P_{VK}[R]$
20 $KM_{merged} \leftarrow KM_{merged} \cap VKM$
21 $P_{VK}[R] \leftarrow P_{VK}[R] - \{VKM\}$
22 $M' \leftarrow M' \cup \{KM_{merged}\} \cup P_{VK}[R]$
23 **return** M'

Appendix D

Bayesian QMRank Algorithm

The ranking of Query Matches is carried out by Algorithm 10, QMRank. Notice, that, intuitively, the process of ranking QMs advances part of the relevance assessment of the CJNs, which was first proposed in CNRank [Oliveira et al., 2015]. This yields to an effective ranking of QMs and a simpler ranking of CJNs. QMRank uses a value score and a schema score, which are respectively related to the VKMs and SKMs that compose the QM.

The algorithm first iterates over each query match, assigning 1 to both $value\_score$ and $schema\_score$. Next, QMRank goes through each keyword match from the QM. In the case of a KM matching the values of an attribute, the algorithm updates the $value\_score$ based on the cosine similarity using TF-IDF weights. QMRank retrieves the term frequency and inverted attribute frequency from the Value Index , and the norm of an attribute from the Schema Index, which are all calculated in the preprocessing phase (see Section 3.1). In the case of a KM matching the name of a schema element, the algorithm updates the $schema\_score$ the average similarity of the keywords with the schema elements based on the similarity functions presented in Section 4. Once the algorithm aggregates the scores of KMs to generate the score of QMs, the final step is to sort them in descending order.

Algorithm 10: QMRank(QM)

Input: A set of query matches QM
Output: The set of ranked query matches RQM

$RQM \leftarrow [\,]$
for $M \in QM$ **do**
  $value\_score \leftarrow 1$, $schema\_score \leftarrow 1$
  for $KM \in M$ **do**
    let KM be $R^S[A_1^{K_1^S}, \ldots, A_m^{K_m^S}]^V[A_1^{K_1^V}, \ldots, A_m^{K_m^V}]$
    for $i \in \{1, \ldots, m\}$ **do**
      if $|K_i^V| \geq 1$ **then**
        $weight\_sum \leftarrow 0$
        $norm_{A_i} \leftarrow I_S[R][A_i]$
        for $word \in K_i^V$ **do**
          $tf \leftarrow |I_V[word][R][A_i]|$
          $weight\_sum \leftarrow weight\_sum + tf \times \textbf{iaf}(word)$
        $value\_score \leftarrow value\_score \times weight\_sum / norm_{A_i}$
      if $|K_i^S| \geq 1$ **then**
        $weight\_sum \leftarrow 0$
        for $word \in K_i^S$ **do**
          if $A_j = self$ **then**
            $schema\_element \leftarrow R$
          else
            $schema\_element \leftarrow A_i$
          $weight\_sum \leftarrow weight\_sum + \textbf{sim}(schema\_element, word)$
        $schema\_score \leftarrow schema\_score \times weight\_sum / |K_i^S|$
  $final\_score \leftarrow value\_score \times schema\_score$
  RQM.append($\langle final\_score, M \rangle$)
Sort RQM in descending order
return RQM

Appendix E

Sound Theorem

Theorem 1. *Let $G_S = \langle \mathcal{R}, E_G \rangle$ be a schema graph. Let $J = \langle \mathcal{V}, E_J \rangle$ be a joining network of keyword matches. We say that J is **sound**, that is, it does not have more than one occurrences of the same tuple for every instance of the database if, and only if, the following condition holds $\forall KM_a \in \mathcal{V}, \forall \langle R_a, R_b \rangle \in E_G$:*

$$RIC(R_a, R_b) \geq |\{KM_c | \langle KM_a, KM_c \rangle \in E_J \wedge R_c = R_b\}|$$

where $RIC(R_a, R_b)$ indicates the number of Referential Integrity Constraints from a relation R_a to a relation R_b.

Proof. Let R_a and R_b be database relations so that there exists n *Referential Integrity Constraint* (RICs) from R_a to R_b. Intuitively, a tuple from R_a may refer to at most n tuples from R_b. Consider a joining network of keyword matches J wherein a keyword match over R_a is adjacent to m keyword matches over R_b, that is $J = \langle \mathcal{V}, E \rangle$, where $\mathcal{V} = \{KM_1, \ldots, KM_{m+1}\}$, $E = \{\langle KM_1, KM_i \rangle | 2 \leq i \leq m\}$, and $R_1 = Ra \wedge R_i = R_b, 2 \leq i \leq m$. We can translate J into a relational algebra expression wherein the edges are join operations using RICs and keyword matches are selection operations over relations. For didactic purposes, we assume, without loss of generality, that all the KMs of J are keyword-free matches. Let k_j be a key attribute from R_i and $f_{i,j}$ be the attribute from R_i that references k_j. The SQL translation of J can be represented by T_{m+1}, which expands a join operation in each iteration.

$$\begin{aligned} T_1 &= R_1 \\ T_2 &= T_1 \bowtie_{f_{1,2}=k_2} R_2 \\ T_3 &= T_2 \bowtie_{f_{1,3}=k_3} R_3 \end{aligned}$$

$$T_{n+1} = T_n \bowtie_{f_{1,n+1}=k_{n+1}} R_{n+1}$$
$$T_{n+2} = T_{n+1} \bowtie_{f_{1,x}=k_{n+2}} R_{n+2}, \text{where } x \in \{2, \ldots, n+1\}$$

Notice that by the iteration $n+2$, all RICs from R_a to R_b were already used once. Therefore, this expansion require that we use one of the RICs twice, which would lead to redundancy. For instance, if assume $x = 2$, without loss of generality, then:

$$T_2 = T_1 \bowtie_{f_{1,2}=k_2} R_2$$
$$T_{n+2} = T_{n+1} \bowtie_{f_{1,2}=k_{n+2}} R_{n+2}$$

As the join conditions are stacked in each iteration, we can say that:

$$f_{1,2} = k_2 \wedge f_{1,2} = k_{n+2}$$

which implies that $k_2 = k_{n+2}$ and, thus, all the returning JNTs would have more than one occurrence of the same tuple for every instance of the database.

$$T_{m+1} = T_m \bowtie_{f_{1,x}=k_{m+1}} R_{m+1}$$

□

Appendix F

CJNGen Algorithm

The generation and ranking of CJNs is carried out by Algorithm 11, CJNGen, which uses a *Breadth-First Search* approach [Cormen et al., 2009] to expand JNKMs until they comprehend all elements from a query match.

Despite being based on the MatCNGen Algorithm [Oliveira et al., 2018], CJNGen provides support for generating CJNs wherein there exists more than one RIC between one database relation to another, due to the definition of soundness presented in Theorem 1. Also, CJNGen does not require an intermediate structure such as the *Match Graph* in the MatCNGen system.

We describe CJNGen in Algorithm 11. For each query match, CJNGen generates the candidate joining networks for this query match using an internal algorithm called CJNInter, which we will focus on describing in the remainder of this section.

Algorithm 11: CJNGen(RQM, G_S)

Input: The set of ranked query matches RQM
The schema graph G_S
Output: The set of candidate networks CJN

```
1 CJN = {}
2 for query match M ∈ RQM do
3     CJN_M ← CJNInter(M, G_S)
4     CJN ← CJN ∪ CJN_M
5 return CJN
```

In Algorithm 12, we present CJNInter. This algorithm takes as input a query match M and the schema graph G_S. Next, it chooses a KM from the QM as a starting point, resulting in an unitary graph (Lines 3-4). If the query match M has only one element, we already generated the one possible candidate joining network (Line 6).

Algorithm 12: CJNInter($G_S, M, score_M$)

Input: The query match M; The schema graph G_S
Output: A set CJN of candidate networks for the query match M

```
1  CJN ← [ ]
2  J ← Graph()
3  let KM be an element from M
4  Add KM to J.V
5  if |M| = 1 then
6      return {J}
7  D ← queue()
8  D.enqueue(J)
9  while D ≠ {} do
10     J ← D.dequeue()
11     for KM_u ∈ J.V do
12         let KM_u be R_u^S[A_{u,1}^{K_{u,1}^S}, ..., A_{u,m}^{K_{u,m}^S}]^V[A_{u,1}^{K_{u,1}^V}, ..., A_{u,m_u}^{K_{u,m_u}^V}]
13         let G_S^U be the undirected version of G_S
14         for R_a adjacent to R_u in G_S^U do
15             for KM_v ∈ M\CN.V do
16                 let KM_v be R_v^S[A_{v,1}^{K_{v,1}^S}, ..., A_{v,m}^{K_{v,m}^S}]^V[A_{v,1}^{K_{v,1}^V}, ..., A_{v,m_v}^{K_{v,m_v}^V}]
17                 if R_v = R_a then
18                     J' ← J
19                     Expand J' with KM_v joined to KM_u
20                     if J' ∉ CN and J' is sound then
21                         if J'.V ⊇ M then
22                             CN.append(J')
23                         else
24                             D.enqueue(J')
25             J' ← J
26             Expand J' with R_a^S[ ]^V[ ] joined to KM_u
27             D.enqueue(J')
28 return CJN
```

Next, the CJNInter initializes a queue D, which is used to store the JNKMs which are not CJNs (Lines 7-8). In Loop 9-27, CJNInter takes one JNKM J from the queue and tries to expand it with KMs. N otice that J can be expanded with incoming and outgoing neighbors, therefore it uses an undirected schema graph G_S^U(Line 14). Also, the elements of M can only be added once in a JNKM but keyword-free matches can be added several times.

The expansion of J results in a JNKM J' (Lines 18-19). Then , CJNInter verifies whether J' was already generated and whether it is *sound*, according to Definition 9. If J'

fails to meet these two conditions it is pruned (Line 20).

If J' was not pruned, CJNInter checks whether J' covers the query match M. If it does, J' is a candidate joining network and it will be added to the list CJN. If J' does not cover M, then it will be added to the deque D (Lines 21 -24). At the end of the procedure, CJNInter returns the set CJN of candidate joining networks for the query match M(Line 28).

The CJN generation algorithm also implements some basic CJN pruning strategies, which are based on the following parameters: the top-k CJNs, the top-k CJNs per QM and the maximum CJN size. Also, the algorithm implements a few strategies to prune the JNKMs which are not minimal or not sound, the maximum node Degree, the maximum number of keyword-free matches, and the distinct foreign keys.

F.1 Maximum Node Degree

As the leaves of a CJN must be keyword matches from the query match, then a CJN must have at most $|QM|$ leaves. Also, considering that the maximum node degree in a tree is less or equal to the number of its leaves, we can safely prune the JNKMs that contains a node with a degree greater than $|QM|$.

F.2 Maximum Number of Keyword-free Matches

The size of a CJN is based on the size of the query match and the number of keyword-free matches, that is, the size of a candidate joining network CJN_M for a query match M is given by $|CJN_M|=|M|+|F|$, where F is a set of keyword-free matches. Thus, if we consider a maximum CJN size T_{max}, we can also set a maximum number of keyword-free matches for a CJN, given by $|F|\leq T_{max}-|M|$. Therefore, we can prune all JNKMs that contain more keyword-free matches than this maximum number set.

The number of CJNs generated can be further reduced by the pruning and ranking them. In Section 7.2, we present a ranking of the candidate joining networks returned by CJNGen. In Section 6.1, we present pruning techniques for the generation of the candidate joining networks from CJNGen and CJNInter.

Appendix G

Neural Models Table

Table G.1: CJN Ranking Models

QM Model	CJN Model	Similarity	Agg. Approach	CJN Type	Abbreviation
B_{TFIDF}	TF-IDF	cos		bayesian	$B_{TFIDF}-B_{Simple}$
B_{TFIDF}	paraphrase-albert-small-v2	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{Albert*}$
B_{TFIDF}	paraphrase-albert-small-v2	cos	Mean	fine-tuned	$B_{TFIDF}-F_{Albert+}$
B_{TFIDF}	all-distilroberta-v1	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{Distil1*}$
B_{TFIDF}	all-distilroberta-v1	cos	Mean	fine-tuned	$B_{TFIDF}-F_{Distil1+}$
B_{TFIDF}	distiluse-base-multilingual-cased-v1	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{Distil2*}$
B_{TFIDF}	distiluse-base-multilingual-cased-v1	cos	Mean	fine-tuned	$B_{TFIDF}-F_{Distil2+}$
B_{TFIDF}	distiluse-base-multilingual-cased-v1	dot	Multivalue	fine-tuned	$B_{TFIDF}-F_{Distil3*}$
B_{TFIDF}	distiluse-base-multilingual-cased-v1	dot	Mean	fine-tuned	$B_{TFIDF}-F_{Distil3+}$
B_{TFIDF}	distiluse-base-multilingual-cased-v2	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{Distil4*}$
B_{TFIDF}	distiluse-base-multilingual-cased-v2	cos	Mean	fine-tuned	$B_{TFIDF}-F_{Distil4+}$
B_{TFIDF}	distiluse-base-multilingual-cased-v2	dot	Multivalue	fine-tuned	$B_{TFIDF}-F_{Distil5*}$
B_{TFIDF}	distiluse-base-multilingual-cased-v2	dot	Mean	fine-tuned	$B_{TFIDF}-F_{Distil5+}$
B_{TFIDF}	multi-qa-distilbert-cos-v1	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{Distil6*}$
B_{TFIDF}	multi-qa-distilbert-cos-v1	cos	Mean	fine-tuned	$B_{TFIDF}-F_{Distil6+}$
B_{TFIDF}	all-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{MiniLM1*}$
B_{TFIDF}	all-MiniLM-L12-v2	cos	Mean	fine-tuned	$B_{TFIDF}-F_{MiniLM1+}$
B_{TFIDF}	all-MiniLM-L6-v2	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{MiniLM2*}$
B_{TFIDF}	all-MiniLM-L6-v2	cos	Mean	fine-tuned	$B_{TFIDF}-F_{MiniLM2+}$
B_{TFIDF}	all-MiniLM-L6-v2	dot	Multivalue	fine-tuned	$B_{TFIDF}-F_{MiniLM3*}$
B_{TFIDF}	all-MiniLM-L6-v2	dot	Mean	fine-tuned	$B_{TFIDF}-F_{MiniLM3+}$
B_{TFIDF}	multi-qa-MiniLM-L6-cos-v1	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{MiniLM4*}$
B_{TFIDF}	multi-qa-MiniLM-L6-cos-v1	cos	Mean	fine-tuned	$B_{TFIDF}-F_{MiniLM4+}$
B_{TFIDF}	multi-qa-MiniLM-L6-cos-v1	dot	Multivalue	fine-tuned	$B_{TFIDF}-F_{MiniLM5*}$
B_{TFIDF}	multi-qa-MiniLM-L6-cos-v1	dot	Mean	fine-tuned	$B_{TFIDF}-F_{MiniLM5+}$
B_{TFIDF}	paraphrase-MiniLM-L3-v2	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{MiniLM6*}$
B_{TFIDF}	paraphrase-MiniLM-L3-v2	cos	Mean	fine-tuned	$B_{TFIDF}-F_{MiniLM6+}$
B_{TFIDF}	paraphrase-multilingual-MiniLM-L12-v2	dot	Multivalue	fine-tuned	$B_{TFIDF}-F_{MiniLM7*}$
B_{TFIDF}	paraphrase-multilingual-MiniLM-L12-v2	dot	Mean	fine-tuned	$B_{TFIDF}-F_{MiniLM7+}$
B_{TFIDF}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{MiniLM8*}$
B_{TFIDF}	paraphrase-multilingual-MiniLM-L12-v2	cos	Mean	fine-tuned	$B_{TFIDF}-F_{MiniLM8+}$
B_{TFIDF}	all-mpnet-base-v2	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{MPNET1*}$
B_{TFIDF}	all-mpnet-base-v2	cos	Mean	fine-tuned	$B_{TFIDF}-F_{MPNET1+}$
B_{TFIDF}	all-mpnet-base-v2	dot	Multivalue	fine-tuned	$B_{TFIDF}-F_{MPNET2*}$

Table G.1 continued from previous page

QM Model	CJN Model	Similarity	Agg. Approach	CJN Type	Abbreviation
B_{TFIDF}	all-mpnet-base-v2	dot	Mean	fine-tuned	$B_{TFIDF}-F_{MPNET2+}$
B_{TFIDF}	multi-qa-mpnet-base-dot-v1	dot	Multivalue	fine-tuned	$B_{TFIDF}-F_{MPNET3*}$
B_{TFIDF}	multi-qa-mpnet-base-dot-v1	dot	Mean	fine-tuned	$B_{TFIDF}-F_{MPNET3+}$
B_{TFIDF}	paraphrase-multilingual-mpnet-base-v2	cos	Multivalue	fine-tuned	$B_{TFIDF}-F_{MPNET4*}$
B_{TFIDF}	paraphrase-multilingual-mpnet-base-v2	cos	Mean	fine-tuned	$B_{TFIDF}-F_{MPNET4+}$
B_{TFIDF}	paraphrase-albert-small-v2	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{Albert*}$
B_{TFIDF}	paraphrase-albert-small-v2	cos	Mean	pre-trained	$B_{TFIDF}-P_{Albert+}$
B_{TFIDF}	all-distilroberta-v1	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{Distil1*}$
B_{TFIDF}	all-distilroberta-v1	cos	Mean	pre-trained	$B_{TFIDF}-P_{Distil1+}$
B_{TFIDF}	distiluse-base-multilingual-cased-v1	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{Distil2*}$
B_{TFIDF}	distiluse-base-multilingual-cased-v1	cos	Mean	pre-trained	$B_{TFIDF}-P_{Distil2+}$
B_{TFIDF}	distiluse-base-multilingual-cased-v1	dot	Multivalue	pre-trained	$B_{TFIDF}-P_{Distil3*}$
B_{TFIDF}	distiluse-base-multilingual-cased-v1	dot	Mean	pre-trained	$B_{TFIDF}-P_{Distil3+}$
B_{TFIDF}	distiluse-base-multilingual-cased-v2	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{Distil4*}$
B_{TFIDF}	distiluse-base-multilingual-cased-v2	cos	Mean	pre-trained	$B_{TFIDF}-P_{Distil4+}$
B_{TFIDF}	distiluse-base-multilingual-cased-v2	dot	Multivalue	pre-trained	$B_{TFIDF}-P_{Distil5*}$
B_{TFIDF}	distiluse-base-multilingual-cased-v2	dot	Mean	pre-trained	$B_{TFIDF}-P_{Distil5+}$
B_{TFIDF}	multi-qa-distilbert-cos-v1	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{Distil6*}$
B_{TFIDF}	multi-qa-distilbert-cos-v1	cos	Mean	pre-trained	$B_{TFIDF}-P_{Distil6+}$
B_{TFIDF}	all-MiniLM-L12-v2	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{MiniLM1*}$
B_{TFIDF}	all-MiniLM-L12-v2	cos	Mean	pre-trained	$B_{TFIDF}-P_{MiniLM1+}$
B_{TFIDF}	all-MiniLM-L6-v2	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{MiniLM2*}$
B_{TFIDF}	all-MiniLM-L6-v2	cos	Mean	pre-trained	$B_{TFIDF}-P_{MiniLM2+}$
B_{TFIDF}	all-MiniLM-L6-v2	dot	Multivalue	pre-trained	$B_{TFIDF}-P_{MiniLM3*}$
B_{TFIDF}	all-MiniLM-L6-v2	dot	Mean	pre-trained	$B_{TFIDF}-P_{MiniLM3+}$
B_{TFIDF}	multi-qa-MiniLM-L6-cos-v1	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{MiniLM4*}$
B_{TFIDF}	multi-qa-MiniLM-L6-cos-v1	cos	Mean	pre-trained	$B_{TFIDF}-P_{MiniLM4+}$
B_{TFIDF}	multi-qa-MiniLM-L6-cos-v1	dot	Multivalue	pre-trained	$B_{TFIDF}-P_{MiniLM5*}$
B_{TFIDF}	multi-qa-MiniLM-L6-cos-v1	dot	Mean	pre-trained	$B_{TFIDF}-P_{MiniLM5+}$
B_{TFIDF}	paraphrase-MiniLM-L3-v2	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{MiniLM6*}$
B_{TFIDF}	paraphrase-MiniLM-L3-v2	cos	Mean	pre-trained	$B_{TFIDF}-P_{MiniLM6+}$
B_{TFIDF}	paraphrase-multilingual-MiniLM-L12-v2	dot	Multivalue	pre-trained	$B_{TFIDF}-P_{MiniLM7*}$
B_{TFIDF}	paraphrase-multilingual-MiniLM-L12-v2	dot	Mean	pre-trained	$B_{TFIDF}-P_{MiniLM7+}$
B_{TFIDF}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{MiniLM8*}$
B_{TFIDF}	paraphrase-multilingual-MiniLM-L12-v2	cos	Mean	pre-trained	$B_{TFIDF}-P_{MiniLM8+}$
B_{TFIDF}	all-mpnet-base-v2	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{MPNET1*}$
B_{TFIDF}	all-mpnet-base-v2	cos	Mean	pre-trained	$B_{TFIDF}-P_{MPNET1+}$
B_{TFIDF}	all-mpnet-base-v2	dot	Multivalue	pre-trained	$B_{TFIDF}-P_{MPNET2*}$
B_{TFIDF}	all-mpnet-base-v2	dot	Mean	pre-trained	$B_{TFIDF}-P_{MPNET2+}$
B_{TFIDF}	multi-qa-mpnet-base-dot-v1	dot	Multivalue	pre-trained	$B_{TFIDF}-P_{MPNET3*}$
B_{TFIDF}	multi-qa-mpnet-base-dot-v1	dot	Mean	pre-trained	$B_{TFIDF}-P_{MPNET3+}$
B_{TFIDF}	paraphrase-multilingual-mpnet-base-v2	cos	Multivalue	pre-trained	$B_{TFIDF}-P_{MPNET4*}$
B_{TFIDF}	paraphrase-multilingual-mpnet-base-v2	cos	Mean	pre-trained	$B_{TFIDF}-P_{MPNET4+}$
$F_{MiniLM1}$	TF-IDF	cos	TF-IDF	bayesian	$F_{MiniLM1}-B_{Simple}$
$F_{MiniLM1}$	paraphrase-albert-small-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Albert*}$
$F_{MiniLM1}$	paraphrase-albert-small-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{Albert+}$
$F_{MiniLM1}$	all-distilroberta-v1	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Distil1*}$
$F_{MiniLM1}$	all-distilroberta-v1	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{Distil1+}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Distil2*}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{Distil2+}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Distil3*}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	dot	Mean	fine-tuned	$F_{MiniLM1}-F_{Distil3+}$

Table G.1 continued from previous page

QM Model	CJN Model	Similarity	Agg. Approach	CJN Type	Abbreviation
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Distil4*}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{Distil4+}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v2	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Distil5*}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v2	dot	Mean	fine-tuned	$F_{MiniLM1}-F_{Distil5+}$
$F_{MiniLM1}$	multi-qa-distilbert-cos-v1	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{Distil6*}$
$F_{MiniLM1}$	multi-qa-distilbert-cos-v1	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{Distil6+}$
$F_{MiniLM1}$	all-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM1*}$
$F_{MiniLM1}$	all-MiniLM-L12-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM1+}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM2*}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM2+}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM3*}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	dot	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM3+}$
$F_{MiniLM1}$	multi-qa-MiniLM-L6-cos-v1	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM4*}$
$F_{MiniLM1}$	multi-qa-MiniLM-L6-cos-v1	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM4+}$
$F_{MiniLM1}$	multi-qa-MiniLM-L6-cos-v1	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM5*}$
$F_{MiniLM1}$	multi-qa-MiniLM-L6-cos-v1	dot	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM5+}$
$F_{MiniLM1}$	paraphrase-MiniLM-L3-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM6*}$
$F_{MiniLM1}$	paraphrase-MiniLM-L3-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM6+}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM7*}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	dot	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM7+}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MiniLM8*}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{MiniLM8+}$
$F_{MiniLM1}$	all-mpnet-base-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MPNET1*}$
$F_{MiniLM1}$	all-mpnet-base-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{MPNET1+}$
$F_{MiniLM1}$	all-mpnet-base-v2	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MPNET2*}$
$F_{MiniLM1}$	all-mpnet-base-v2	dot	Mean	fine-tuned	$F_{MiniLM1}-F_{MPNET2+}$
$F_{MiniLM1}$	multi-qa-mpnet-base-dot-v1	dot	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MPNET3*}$
$F_{MiniLM1}$	multi-qa-mpnet-base-dot-v1	dot	Mean	fine-tuned	$F_{MiniLM1}-F_{MPNET3+}$
$F_{MiniLM1}$	paraphrase-multilingual-mpnet-base-v2	cos	Multivalue	fine-tuned	$F_{MiniLM1}-F_{MPNET4*}$
$F_{MiniLM1}$	paraphrase-multilingual-mpnet-base-v2	cos	Mean	fine-tuned	$F_{MiniLM1}-F_{MPNET4+}$
$F_{MiniLM1}$	paraphrase-albert-small-v2	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{Albert*}$
$F_{MiniLM1}$	paraphrase-albert-small-v2	cos	Mean	pre-trained	$F_{MiniLM1}-P_{Albert+}$
$F_{MiniLM1}$	all-distilroberta-v1	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{Distil1*}$
$F_{MiniLM1}$	all-distilroberta-v1	cos	Mean	pre-trained	$F_{MiniLM1}-P_{Distil1+}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{Distil2*}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	cos	Mean	pre-trained	$F_{MiniLM1}-P_{Distil2+}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	dot	Multivalue	pre-trained	$F_{MiniLM1}-P_{Distil3*}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v1	dot	Mean	pre-trained	$F_{MiniLM1}-P_{Distil3+}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v2	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{Distil4*}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v2	cos	Mean	pre-trained	$F_{MiniLM1}-P_{Distil4+}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v2	dot	Multivalue	pre-trained	$F_{MiniLM1}-P_{Distil5*}$
$F_{MiniLM1}$	distiluse-base-multilingual-cased-v2	dot	Mean	pre-trained	$F_{MiniLM1}-P_{Distil5+}$
$F_{MiniLM1}$	multi-qa-distilbert-cos-v1	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{Distil6*}$
$F_{MiniLM1}$	multi-qa-distilbert-cos-v1	cos	Mean	pre-trained	$F_{MiniLM1}-P_{Distil6+}$
$F_{MiniLM1}$	all-MiniLM-L12-v2	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{MiniLM1*}$
$F_{MiniLM1}$	all-MiniLM-L12-v2	cos	Mean	pre-trained	$F_{MiniLM1}-P_{MiniLM1+}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{MiniLM2*}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	cos	Mean	pre-trained	$F_{MiniLM1}-P_{MiniLM2+}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	dot	Multivalue	pre-trained	$F_{MiniLM1}-P_{MiniLM3*}$
$F_{MiniLM1}$	all-MiniLM-L6-v2	dot	Mean	pre-trained	$F_{MiniLM1}-P_{MiniLM3+}$
$F_{MiniLM1}$	multi-qa-MiniLM-L6-cos-v1	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{MiniLM4*}$
$F_{MiniLM1}$	multi-qa-MiniLM-L6-cos-v1	cos	Mean	pre-trained	$F_{MiniLM1}-P_{MiniLM4+}$

Table G.1 continued from previous page

QM Model	CJN Model	Similarity	Agg. Approach	CJN Type	Abbreviation
$F_{MiniLM1}$	multi-qa-MiniLM-L6-cos-v1	dot	Multivalue	pre-trained	$F_{MiniLM1}-P_{MiniLM5*}$
$F_{MiniLM1}$	multi-qa-MiniLM-L6-cos-v1	dot	Mean	pre-trained	$F_{MiniLM1}-P_{MiniLM5+}$
$F_{MiniLM1}$	paraphrase-MiniLM-L3-v2	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{MiniLM6*}$
$F_{MiniLM1}$	paraphrase-MiniLM-L3-v2	cos	Mean	pre-trained	$F_{MiniLM1}-P_{MiniLM6+}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	dot	Multivalue	pre-trained	$F_{MiniLM1}-P_{MiniLM7*}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	dot	Mean	pre-trained	$F_{MiniLM1}-P_{MiniLM7+}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{MiniLM8*}$
$F_{MiniLM1}$	paraphrase-multilingual-MiniLM-L12-v2	cos	Mean	pre-trained	$F_{MiniLM1}-P_{MiniLM8+}$
$F_{MiniLM1}$	all-mpnet-base-v2	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{MPNET1*}$
$F_{MiniLM1}$	all-mpnet-base-v2	cos	Mean	pre-trained	$F_{MiniLM1}-P_{MPNET1+}$
$F_{MiniLM1}$	all-mpnet-base-v2	dot	Multivalue	pre-trained	$F_{MiniLM1}-P_{MPNET2*}$
$F_{MiniLM1}$	all-mpnet-base-v2	dot	Mean	pre-trained	$F_{MiniLM1}-P_{MPNET2+}$
$F_{MiniLM1}$	multi-qa-mpnet-base-dot-v1	dot	Multivalue	pre-trained	$F_{MiniLM1}-P_{MPNET3*}$
$F_{MiniLM1}$	multi-qa-mpnet-base-dot-v1	dot	Mean	pre-trained	$F_{MiniLM1}-P_{MPNET3+}$
$F_{MiniLM1}$	paraphrase-multilingual-mpnet-base-v2	cos	Multivalue	pre-trained	$F_{MiniLM1}-P_{MPNET4*}$
$F_{MiniLM1}$	paraphrase-multilingual-mpnet-base-v2	cos	Mean	pre-trained	$F_{MiniLM1}-P_{MPNET4+}$
F_{MPNET3}	TF-IDF	cos	TF-IDF	bayesian	$F_{MPNET3}-B_{Simple}$
F_{MPNET3}	paraphrase-albert-small-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{Albert*}$
F_{MPNET3}	paraphrase-albert-small-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{Albert+}$
F_{MPNET3}	all-distilroberta-v1	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{Distil1*}$
F_{MPNET3}	all-distilroberta-v1	cos	Mean	fine-tuned	$F_{MPNET3}-F_{Distil1+}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{Distil2*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	cos	Mean	fine-tuned	$F_{MPNET3}-F_{Distil2+}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{Distil3*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	dot	Mean	fine-tuned	$F_{MPNET3}-F_{Distil3+}$
F_{MPNET3}	distiluse-base-multilingual-cased-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{Distil4*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{Distil4+}$
F_{MPNET3}	distiluse-base-multilingual-cased-v2	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{Distil5*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v2	dot	Mean	fine-tuned	$F_{MPNET3}-F_{Distil5+}$
F_{MPNET3}	multi-qa-distilbert-cos-v1	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{Distil6*}$
F_{MPNET3}	multi-qa-distilbert-cos-v1	cos	Mean	fine-tuned	$F_{MPNET3}-F_{Distil6+}$
F_{MPNET3}	all-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM1*}$
F_{MPNET3}	all-MiniLM-L12-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM1+}$
F_{MPNET3}	all-MiniLM-L6-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM2*}$
F_{MPNET3}	all-MiniLM-L6-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM2+}$
F_{MPNET3}	all-MiniLM-L6-v2	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM3*}$
F_{MPNET3}	all-MiniLM-L6-v2	dot	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM3+}$
F_{MPNET3}	multi-qa-MiniLM-L6-cos-v1	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM4*}$
F_{MPNET3}	multi-qa-MiniLM-L6-cos-v1	cos	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM4+}$
F_{MPNET3}	multi-qa-MiniLM-L6-cos-v1	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM5*}$
F_{MPNET3}	multi-qa-MiniLM-L6-cos-v1	dot	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM5+}$
F_{MPNET3}	paraphrase-MiniLM-L3-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM6*}$
F_{MPNET3}	paraphrase-MiniLM-L3-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM6+}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM7*}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	dot	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM7+}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{MiniLM8*}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{MiniLM8+}$
F_{MPNET3}	all-mpnet-base-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{MPNET1*}$
F_{MPNET3}	all-mpnet-base-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{MPNET1+}$
F_{MPNET3}	all-mpnet-base-v2	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{MPNET2*}$
F_{MPNET3}	all-mpnet-base-v2	dot	Mean	fine-tuned	$F_{MPNET3}-F_{MPNET2+}$
F_{MPNET3}	multi-qa-mpnet-base-dot-v1	dot	Multivalue	fine-tuned	$F_{MPNET3}-F_{MPNET3*}$

Table G.1 continued from previous page

QM Model	CJN Model	Similarity	Agg. Approach	CJN Type	Abbreviation
F_{MPNET3}	multi-qa-mpnet-base-dot-v1	dot	Mean	fine-tuned	$F_{MPNET3}-F_{MPNET3+}$
F_{MPNET3}	paraphrase-multilingual-mpnet-base-v2	cos	Multivalue	fine-tuned	$F_{MPNET3}-F_{MPNET4*}$
F_{MPNET3}	paraphrase-multilingual-mpnet-base-v2	cos	Mean	fine-tuned	$F_{MPNET3}-F_{MPNET4+}$
F_{MPNET3}	paraphrase-albert-small-v2	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{Albert*}$
F_{MPNET3}	paraphrase-albert-small-v2	cos	Mean	pre-trained	$F_{MPNET3}-P_{Albert+}$
F_{MPNET3}	all-distilroberta-v1	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{Distil1*}$
F_{MPNET3}	all-distilroberta-v1	cos	Mean	pre-trained	$F_{MPNET3}-P_{Distil1+}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{Distil2*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	cos	Mean	pre-trained	$F_{MPNET3}-P_{Distil2+}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	dot	Multivalue	pre-trained	$F_{MPNET3}-P_{Distil3*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v1	dot	Mean	pre-trained	$F_{MPNET3}-P_{Distil3+}$
F_{MPNET3}	distiluse-base-multilingual-cased-v2	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{Distil4*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v2	cos	Mean	pre-trained	$F_{MPNET3}-P_{Distil4+}$
F_{MPNET3}	distiluse-base-multilingual-cased-v2	dot	Multivalue	pre-trained	$F_{MPNET3}-P_{Distil5*}$
F_{MPNET3}	distiluse-base-multilingual-cased-v2	dot	Mean	pre-trained	$F_{MPNET3}-P_{Distil5+}$
F_{MPNET3}	multi-qa-distilbert-cos-v1	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{Distil6*}$
F_{MPNET3}	multi-qa-distilbert-cos-v1	cos	Mean	pre-trained	$F_{MPNET3}-P_{Distil6+}$
F_{MPNET3}	all-MiniLM-L12-v2	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{MiniLM1*}$
F_{MPNET3}	all-MiniLM-L12-v2	cos	Mean	pre-trained	$F_{MPNET3}-P_{MiniLM1+}$
F_{MPNET3}	all-MiniLM-L6-v2	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{MiniLM2*}$
F_{MPNET3}	all-MiniLM-L6-v2	cos	Mean	pre-trained	$F_{MPNET3}-P_{MiniLM2+}$
F_{MPNET3}	all-MiniLM-L6-v2	dot	Multivalue	pre-trained	$F_{MPNET3}-P_{MiniLM3*}$
F_{MPNET3}	all-MiniLM-L6-v2	dot	Mean	pre-trained	$F_{MPNET3}-P_{MiniLM3+}$
F_{MPNET3}	multi-qa-MiniLM-L6-cos-v1	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{MiniLM4*}$
F_{MPNET3}	multi-qa-MiniLM-L6-cos-v1	cos	Mean	pre-trained	$F_{MPNET3}-P_{MiniLM4+}$
F_{MPNET3}	multi-qa-MiniLM-L6-cos-v1	dot	Multivalue	pre-trained	$F_{MPNET3}-P_{MiniLM5*}$
F_{MPNET3}	multi-qa-MiniLM-L6-cos-v1	dot	Mean	pre-trained	$F_{MPNET3}-P_{MiniLM5+}$
F_{MPNET3}	paraphrase-MiniLM-L3-v2	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{MiniLM6*}$
F_{MPNET3}	paraphrase-MiniLM-L3-v2	cos	Mean	pre-trained	$F_{MPNET3}-P_{MiniLM6+}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	dot	Multivalue	pre-trained	$F_{MPNET3}-P_{MiniLM7*}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	dot	Mean	pre-trained	$F_{MPNET3}-P_{MiniLM7+}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{MiniLM8*}$
F_{MPNET3}	paraphrase-multilingual-MiniLM-L12-v2	cos	Mean	pre-trained	$F_{MPNET3}-P_{MiniLM8+}$
F_{MPNET3}	all-mpnet-base-v2	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{MPNET1*}$
F_{MPNET3}	all-mpnet-base-v2	cos	Mean	pre-trained	$F_{MPNET3}-P_{MPNET1+}$
F_{MPNET3}	all-mpnet-base-v2	dot	Multivalue	pre-trained	$F_{MPNET3}-P_{MPNET2*}$
F_{MPNET3}	all-mpnet-base-v2	dot	Mean	pre-trained	$F_{MPNET3}-P_{MPNET2+}$
F_{MPNET3}	multi-qa-mpnet-base-dot-v1	dot	Multivalue	pre-trained	$F_{MPNET3}-P_{MPNET3*}$
F_{MPNET3}	multi-qa-mpnet-base-dot-v1	dot	Mean	pre-trained	$F_{MPNET3}-P_{MPNET3+}$
F_{MPNET3}	paraphrase-multilingual-mpnet-base-v2	cos	Multivalue	pre-trained	$F_{MPNET3}-P_{MPNET4*}$
F_{MPNET3}	paraphrase-multilingual-mpnet-base-v2	cos	Mean	pre-trained	$F_{MPNET3}-P_{MPNET4+}$
F_{MPNET4}	TF-IDF	cos	TF-IDF	bayesian	$F_{MPNET4}-B_{Simple}$
F_{MPNET4}	paraphrase-albert-small-v2	cos	Multivalue	fine-tuned	$F_{MPNET4}-F_{Albert*}$
F_{MPNET4}	paraphrase-albert-small-v2	cos	Mean	fine-tuned	$F_{MPNET4}-F_{Albert+}$
F_{MPNET4}	all-distilroberta-v1	cos	Multivalue	fine-tuned	$F_{MPNET4}-F_{Distil1*}$
F_{MPNET4}	all-distilroberta-v1	cos	Mean	fine-tuned	$F_{MPNET4}-F_{Distil1+}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	cos	Multivalue	fine-tuned	$F_{MPNET4}-F_{Distil2*}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	cos	Mean	fine-tuned	$F_{MPNET4}-F_{Distil2+}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	dot	Multivalue	fine-tuned	$F_{MPNET4}-F_{Distil3*}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	dot	Mean	fine-tuned	$F_{MPNET4}-F_{Distil3+}$
F_{MPNET4}	distiluse-base-multilingual-cased-v2	cos	Multivalue	fine-tuned	$F_{MPNET4}-F_{Distil4*}$
F_{MPNET4}	distiluse-base-multilingual-cased-v2	cos	Mean	fine-tuned	$F_{MPNET4}-F_{Distil4+}$

Table G.1 continued from previous page

QM Model	CJN Model	Similarity	Agg. Approach	CJN Type	Abbreviation
F_{MPNET4}	distiluse-base-multilingual-cased-v2	dot	Multivalue	fine-tuned	F_{MPNET4}−$F_{Distil5*}$
F_{MPNET4}	distiluse-base-multilingual-cased-v2	dot	Mean	fine-tuned	F_{MPNET4}−$F_{Distil5+}$
F_{MPNET4}	multi-qa-distilbert-cos-v1	cos	Multivalue	fine-tuned	F_{MPNET4}−$F_{Distil6*}$
F_{MPNET4}	multi-qa-distilbert-cos-v1	cos	Mean	fine-tuned	F_{MPNET4}−$F_{Distil6+}$
F_{MPNET4}	all-MiniLM-L12-v2	cos	Multivalue	fine-tuned	F_{MPNET4}−$F_{MiniLM1*}$
F_{MPNET4}	all-MiniLM-L12-v2	cos	Mean	fine-tuned	F_{MPNET4}−$F_{MiniLM1+}$
F_{MPNET4}	all-MiniLM-L6-v2	cos	Multivalue	fine-tuned	F_{MPNET4}−$F_{MiniLM2*}$
F_{MPNET4}	all-MiniLM-L6-v2	cos	Mean	fine-tuned	F_{MPNET4}−$F_{MiniLM2+}$
F_{MPNET4}	all-MiniLM-L6-v2	dot	Multivalue	fine-tuned	F_{MPNET4}−$F_{MiniLM3*}$
F_{MPNET4}	all-MiniLM-L6-v2	dot	Mean	fine-tuned	F_{MPNET4}−$F_{MiniLM3+}$
F_{MPNET4}	multi-qa-MiniLM-L6-cos-v1	cos	Multivalue	fine-tuned	F_{MPNET4}−$F_{MiniLM4*}$
F_{MPNET4}	multi-qa-MiniLM-L6-cos-v1	cos	Mean	fine-tuned	F_{MPNET4}−$F_{MiniLM4+}$
F_{MPNET4}	multi-qa-MiniLM-L6-cos-v1	dot	Multivalue	fine-tuned	F_{MPNET4}−$F_{MiniLM5*}$
F_{MPNET4}	multi-qa-MiniLM-L6-cos-v1	dot	Mean	fine-tuned	F_{MPNET4}−$F_{MiniLM5+}$
F_{MPNET4}	paraphrase-MiniLM-L3-v2	cos	Multivalue	fine-tuned	F_{MPNET4}−$F_{MiniLM6*}$
F_{MPNET4}	paraphrase-MiniLM-L3-v2	cos	Mean	fine-tuned	F_{MPNET4}−$F_{MiniLM6+}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	dot	Multivalue	fine-tuned	F_{MPNET4}−$F_{MiniLM7*}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	dot	Mean	fine-tuned	F_{MPNET4}−$F_{MiniLM7+}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	fine-tuned	F_{MPNET4}−$F_{MiniLM8*}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	cos	Mean	fine-tuned	F_{MPNET4}−$F_{MiniLM8+}$
F_{MPNET4}	all-mpnet-base-v2	cos	Multivalue	fine-tuned	F_{MPNET4}−$F_{MPNET1*}$
F_{MPNET4}	all-mpnet-base-v2	cos	Mean	fine-tuned	F_{MPNET4}−$F_{MPNET1+}$
F_{MPNET4}	all-mpnet-base-v2	dot	Multivalue	fine-tuned	F_{MPNET4}−$F_{MPNET2*}$
F_{MPNET4}	all-mpnet-base-v2	dot	Mean	fine-tuned	F_{MPNET4}−$F_{MPNET2+}$
F_{MPNET4}	multi-qa-mpnet-base-dot-v1	dot	Multivalue	fine-tuned	F_{MPNET4}−$F_{MPNET3*}$
F_{MPNET4}	multi-qa-mpnet-base-dot-v1	dot	Mean	fine-tuned	F_{MPNET4}−$F_{MPNET3+}$
F_{MPNET4}	paraphrase-multilingual-mpnet-base-v2	cos	Multivalue	fine-tuned	F_{MPNET4}−$F_{MPNET4*}$
F_{MPNET4}	paraphrase-multilingual-mpnet-base-v2	cos	Mean	fine-tuned	F_{MPNET4}−$F_{MPNET4+}$
F_{MPNET4}	paraphrase-albert-small-v2	cos	Multivalue	pre-trained	F_{MPNET4}−$P_{Albert*}$
F_{MPNET4}	paraphrase-albert-small-v2	cos	Mean	pre-trained	F_{MPNET4}−$P_{Albert+}$
F_{MPNET4}	all-distilroberta-v1	cos	Multivalue	pre-trained	F_{MPNET4}−$P_{Distil1*}$
F_{MPNET4}	all-distilroberta-v1	cos	Mean	pre-trained	F_{MPNET4}−$P_{Distil1+}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	cos	Multivalue	pre-trained	F_{MPNET4}−$P_{Distil2*}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	cos	Mean	pre-trained	F_{MPNET4}−$P_{Distil2+}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	dot	Multivalue	pre-trained	F_{MPNET4}−$P_{Distil3*}$
F_{MPNET4}	distiluse-base-multilingual-cased-v1	dot	Mean	pre-trained	F_{MPNET4}−$P_{Distil3+}$
F_{MPNET4}	distiluse-base-multilingual-cased-v2	cos	Multivalue	pre-trained	F_{MPNET4}−$P_{Distil4*}$
F_{MPNET4}	distiluse-base-multilingual-cased-v2	cos	Mean	pre-trained	F_{MPNET4}−$P_{Distil4+}$
F_{MPNET4}	distiluse-base-multilingual-cased-v2	dot	Multivalue	pre-trained	F_{MPNET4}−$P_{Distil5*}$
F_{MPNET4}	distiluse-base-multilingual-cased-v2	dot	Mean	pre-trained	F_{MPNET4}−$P_{Distil5+}$
F_{MPNET4}	multi-qa-distilbert-cos-v1	cos	Multivalue	pre-trained	F_{MPNET4}−$P_{Distil6*}$
F_{MPNET4}	multi-qa-distilbert-cos-v1	cos	Mean	pre-trained	F_{MPNET4}−$P_{Distil6+}$
F_{MPNET4}	all-MiniLM-L12-v2	cos	Multivalue	pre-trained	F_{MPNET4}−$P_{MiniLM1*}$
F_{MPNET4}	all-MiniLM-L12-v2	cos	Mean	pre-trained	F_{MPNET4}−$P_{MiniLM1+}$
F_{MPNET4}	all-MiniLM-L6-v2	cos	Multivalue	pre-trained	F_{MPNET4}−$P_{MiniLM2*}$
F_{MPNET4}	all-MiniLM-L6-v2	cos	Mean	pre-trained	F_{MPNET4}−$P_{MiniLM2+}$
F_{MPNET4}	all-MiniLM-L6-v2	dot	Multivalue	pre-trained	F_{MPNET4}−$P_{MiniLM3*}$
F_{MPNET4}	all-MiniLM-L6-v2	dot	Mean	pre-trained	F_{MPNET4}−$P_{MiniLM3+}$
F_{MPNET4}	multi-qa-MiniLM-L6-cos-v1	cos	Multivalue	pre-trained	F_{MPNET4}−$P_{MiniLM4*}$
F_{MPNET4}	multi-qa-MiniLM-L6-cos-v1	cos	Mean	pre-trained	F_{MPNET4}−$P_{MiniLM4+}$
F_{MPNET4}	multi-qa-MiniLM-L6-cos-v1	dot	Multivalue	pre-trained	F_{MPNET4}−$P_{MiniLM5*}$
F_{MPNET4}	multi-qa-MiniLM-L6-cos-v1	dot	Mean	pre-trained	F_{MPNET4}−$P_{MiniLM5+}$

Table G.1 continued from previous page

QM Model	CJN Model	Similarity	Agg. Approach	CJN Type	Abbreviation
F_{MPNET4}	paraphrase-MiniLM-L3-v2	cos	Multivalue	pre-trained	$F_{MPNET4}-P_{MiniLM6*}$
F_{MPNET4}	paraphrase-MiniLM-L3-v2	cos	Mean	pre-trained	$F_{MPNET4}-P_{MiniLM6+}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	dot	Multivalue	pre-trained	$F_{MPNET4}-P_{MiniLM7*}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	dot	Mean	pre-trained	$F_{MPNET4}-P_{MiniLM7+}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	cos	Multivalue	pre-trained	$F_{MPNET4}-P_{MiniLM8*}$
F_{MPNET4}	paraphrase-multilingual-MiniLM-L12-v2	cos	Mean	pre-trained	$F_{MPNET4}-P_{MiniLM8+}$
F_{MPNET4}	all-mpnet-base-v2	cos	Multivalue	pre-trained	$F_{MPNET4}-P_{MPNET1*}$
F_{MPNET4}	all-mpnet-base-v2	cos	Mean	pre-trained	$F_{MPNET4}-P_{MPNET1+}$
F_{MPNET4}	all-mpnet-base-v2	dot	Multivalue	pre-trained	$F_{MPNET4}-P_{MPNET2*}$
F_{MPNET4}	all-mpnet-base-v2	dot	Mean	pre-trained	$F_{MPNET4}-P_{MPNET2+}$
F_{MPNET4}	multi-qa-mpnet-base-dot-v1	dot	Multivalue	pre-trained	$F_{MPNET4}-P_{MPNET3*}$
F_{MPNET4}	multi-qa-mpnet-base-dot-v1	dot	Mean	pre-trained	$F_{MPNET4}-P_{MPNET3+}$
F_{MPNET4}	paraphrase-multilingual-mpnet-base-v2	cos	Multivalue	pre-trained	$F_{MPNET4}-P_{MPNET4*}$
F_{MPNET4}	paraphrase-multilingual-mpnet-base-v2	cos	Mean	pre-trained	$F_{MPNET4}-P_{MPNET4+}$