A FRAMEWORK DESIGN FOR INTEGRATING KNOWLEDGE GRAPHS INTO RECOMMENDATION SYSTEMS

Yuhao Mao

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Science (Computer Science) at

Concordia University

Montréal, Québec, Canada

March 2021

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Yuhao Mao**

Entitled: **A Framework Design for Integrating Knowledge Graphs into Recommendation Systems**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Joey Paquet

_____ Examiner
Dr. Joey Paquet

_____ Examiner
Dr. Dhrubajyoti Gosmwami

_____ Supervisor
Dr. Sudhir Mudur

_____ Supervisor
Dr. Serguei A. Mokhov

Approved by _____
Dr. Hovhannes Harutunyan, Graduate Program Director

_____ 20 _____ _____
Dr. Mourad Debbabi, Interim Dean
Gina Cody School of Engineering and Computer Science

Abstract

A Framework Design for Integrating Knowledge Graphs into Recommendation Systems

Yuhao Mao

Online recommendation is a significant research domain in artificial intelligence. A recommendation system recommends different items to users, and has applications in varied domains, including news, music, movies, etc. Initially, recommendation systems were based on demographic, content-based filtering and collaborative filtering. But collaborative filtering often suffers from sparsity and cold start problems, therefore, side information is often used to address these issues and improve recommendation performance. Currently, incorporating knowledge into the recommendation algorithm has attracted increasing attention, as it can help improve recommendation system performance. Knowledge graph representation and construction, and recommendation system development are independent but related; the triples of knowledge graph form the input to the recommendation system. While, there are a number of independent solutions for each of these two tasks, currently, there is no existing solution that can combine the construction of knowledge graph and input it to the recommendation system to provide an integrated work pipeline.

Our major contribution is a modular, easy to use framework solution that fills this gap, essentially enabling integration of a structured knowledge graph and a recommendation system. Our framework provides multiple functionalities, including cross-language invocation and pipeline execution mechanism, and also knowledge graph query, modification and visualization. We instantiate our implementation of the proposed framework and evaluate its performance to show that we achieve higher accuracy in recommendations by using side information extracted from knowledge

graphs. Our framework addresses the complete pipeline from constructing structured data knowledge graph to training recommendation model to incorporating the recommendation system into application domains.

Acknowledgments

During my study, I was lucky enough to get help, suggestion and advice from many people. I would like to firstly express my sincere gratitude to my co-supervisors Dr. Sudhir Mudur and Dr. Serguei Mokhov for their valuable and constructive supervision and guidance during my research. I really appreciate their supports during the period of master study.

Secondly, I need to say thank you to those people who helped me with my study and research, including every members in my lab, thanks for their helpful suggestions and corporation. Besides that I would like to thank my parents for their support and encouragement.

Lastly, I need to thank my friends, Chen Ling, Zhanfan Zhou and Jianing Yao. I'm very lucky to have them during the difficult times. Thank you.

Contents

List of Figures

List of Tables

Acronyms

ACC Accuracy.

AUC Area under curve.

FR Functional Requirement.

JSON-LD JSON for Linking Data.

KG Knowledge Graph.

MKR Multi-Task Feature Learning for Knowledge Graph Enhanced Recommendation.

NFR Non-functional Requirement.

OWL Web Ontology Language.

RDF Resource Description Framework.

RDFa Resource Description Framework in Attributes.

RDFs Resource Description Framework Schema.

ROC receiver operating characteristic.

RS Recommendation System.

Chapter 1

Introduction

Our work is in the area of online recommendations based on user demographics and preferences derived from past history of the user, and information about items being recommended. Increasingly, recommendation systems have begun to use information, as much as available, about the user and about the items to improve the quality of recommendations. Specifically, in this dissertation, we present a new framework which enables easy integration of any such available information, called side information, by representing it as knowledge to be used by the recommender algorithm.

In this chapter, we will first briefly outline the research domain for the reader in Section 1.1. Next, we describe our motivation and goals in Section 1.2 followed by motivational scenarios in Section 1.3 which reflect possible usage scenarios for our framework. We also elicit specific functional and non-functional requirements from these motivational scenarios for our proposed recommendation systems framework. Essential background required is briefly touched upon in Section 1.4.

1.1 Research Domain

Recommendation System (RS) is a cross-research direction in multiple fields, and it involves machine learning and data mining.

The recommendation system is the product of the rapid development of the Internet. With the growth of user sales and the increasing variety of items provided

by the suppliers, the users are over loaded with information. In such situations, the recommendation system comes into play [6]. The recommendation system is essentially a technical means for users to narrow the information they are interested in from the massive amount of information available on the Internet, when the user desired product is not specific to a single item [88]. A recommendation system can be regarded as an information filtering system, which can learn the user's interests and preferences based on the user's files or historical behaviour, and predict the user's rating or preference for a given item, based on information about the item.

Applications of recommendation systems are very wide. According to reports, the recommendation system has brought 35% of sales revenue to Amazon [66] and up to 75% of consumption to Netflix [21], and 60% of the browsing on the Youtube homepage comes from recommendation services [67]. It is also widely used by various Internet companies. As long as the company has a large number of products to offer to the clients, the recommendation system will be useful [31, 19, 30]. The current application fields of the recommendation system include, but not limited to the following product categories:

- **E-commerce website**: Amazon, Bestbuy, etc.
- **Video**: Netflix, Youtube, etc.
- **Music**: Apple music, Spotify, Google play music, etc.
- **Information category**: Apple News, Google News, The New York Times, etc.
- **Dating**: Tinder, Bumble, etc.

While most recommendation systems make use of collaborative filtering-based or content-based algorithms [15], in recent years, given the success of deep learning technology in speech recognition, computer vision and natural language understanding, applying it to recommendation systems is also a research hotspot [93]. In our work too, the emphasis is on the latter.

In most recommendation scenarios, items may have rich associated knowledge in the form of interlinked information, and the network structure that depicts this

knowledge is called a knowledge graph. Information is encoded in a data structure called triples made of subject-predicate-object statements. A knowledge graph on the item side greatly increases information about the item, strengthens the connection between items, provides a rich reference value for the recommendation, and can bring additional diversity and interpretability to the recommendation result. In this context, we can state here that a more specific domain of this research work would be knowledge graph enhanced recommendation systems.

1.2 Motivation and Goal

A recommendation system is needed as long as there are users (clients) interested in items (products). The system is usually trained using ratings of items by users. But since users usually have ratings on few items, there are problems of data sparsity in this training process. It has been found that adding the knowledge graph to provide any side information about the user and the items can alleviate this problem. In existing solutions, such side information is stored in one or more separate text files. This has obvious disadvantages of inefficiency. The use of framework solutions can make framework developers spend the least time to complete their code, which also provides feasibility for algorithm comparison, and there is no existing framework that integrates the construction of knowledge graphs. Further, search and update of information, if not carried out across all the text files, would cause problems of incompleteness, and inconsistency. There is clearly a need for a general framework, which (i) integrates search and update of information, (ii) includes crawling of websites for additional information, (iii) supports storing of the information in the knowledge graph and (iv) enables easy retrieval of the triples in the knowledge graph as input for the training of a recommendation system. Adding a knowledge graph into the recommendation framework can help us better manage knowledge data, process data, and query the information we need faster.

Hence the main goal in this work is **to design a framework that integrates a knowledge graph and enables enhancement of the functionality of a**

recommendation system, for different classes of users as follows: (i) the end_user who receives recommendations from a trained recommendation system application, (ii) the application_developer who uses our framework's API to create end user applications, and (iii) the framework_developer who can update/modify the framework. While, for illustration and explanations, we will use the domain of movie recommendations, our framework should be generic to be applied to other domains.

1.3 Scenarios and System Requirements

We present a few specific scenarios, to help us derive the requirements for such a framework, and also to provide a general solution. We analyze these scenarios one by one and then extract functional requirements and non-functional requirements (FR and NFR). Meeting these requirements become the concrete realization goal of our work. For brevity, we will use the generic term "user" to refer to all of the above categories; the scenario will make clear the specific category.

1.3.1 Functional Requirements

1. **FR1: The solution shall provide an abstraction layer for the storage mode that enables the storage construct transparency property to the users.**

2. **FR2: The solution should ensure the adaptability of the abstraction layer required in FR1, which means that when a new storage method is needed, minimal or or no modifications are required, and it does not affect the existing system.**

   Imagine if the user needs to develop a recommendation system program, and needs to support different knowledge graph storage modes S_1 and S_2. Among them, S_1 is a storage mode based on graph storage, and S_2 is a storage mode based on RDF (Resource Description Framework) [45]. Therefore, according to the needs, we need to be able to switch between S_1 and S_2. From the

application's point of view, it should be able to use different storage modes with only some or no changes in the code. In addition, if there is a new storage mode S_3 in the future, the system should easily extend to use S_3.

3. **FR3: The system should provide a unified interface for modifying nodes.**

   Because the content in the knowledge graph may come from open-source data or it may be extracted by the user, there is a possibility of incorrect or inconsistent information. We need to modify the nodes with such data according to the new data. This includes modifying the name of the node, the label of the node, and the relationship between two or more nodes. An example is when the label of node1 is say, "male", and in the new schema structure, we would like to refer to "male" and "female" as "person". If the amount of data is large, it is near impossible to modify one by one manually, we need to provide an API to uniformly modify all the nodes in the knowledge graph.

4. **FR4: The KG solution should support all languages.**

   Imagine a scenario when some users want to use information in other languages to build a knowledge graph. Our framework should be able to support multiple languages in the knowledge graph, and ensure that no matter which language or languages the user prefers, they are displayed correctly.

5. **FR5: The Recommendation System solution should support all languages.**

   Imagine a scene where not all movies have English names, and some users want recommendations for non-English movies. We must make sure that the recommendation framework supports all languages, to ensure that the system can predict the ratings of movies in all languages.

6. **FR6: The KG storage module should support the fusion of different storage formats.**

Imagine a scenario where not all knowledge graphs use a unified storage format, and users want to integrate knowledge graphs of other storage formats. We must ensure that the recommendation framework supports the fusion of different knowledge graph formats.

1.3.2 Non-functional Requirements

With further analysis of the above scenarios, we find that our solution must meet the following non-functional requirements as well:

1. **NFR1: Real-time Response** *When back-end developer use this framework to instantiate a system, the predictor step should be able to predict the end-user's rating of the movie in real time.*

2. **NFR2: API Usability** *The Application Programming Interface provided by this solution should be easy to use and learn by users.*

   An application programming interface (API) is a computing interface that defines interactions between multiple software intermediaries. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc. It can also provide extension mechanisms so that users can extend existing functionality in various ways and to varying degrees. Compared to existing solutions, we need to provide a simpler method. Our users may come from completely different fields, but we can roughly divide them into programmers and non-programmers. Therefore, these factors should affect the way we design the API of our solution.

3. **NFR3: Extensibility** *Our framework should be able to acquire and integrate new components without impairing existing system functions.*

   Hardware and software technology will continue to develop and upgrade. We will need to use more advanced methods. Designing a solution with broad scalability is crucial. It enables us to add, modify, or integrate new storage methods or algorithms into our solutions with minimal effort.

4. **NFR4: Cross-Platform** *The ability of the system to run on multiple platforms.*

   The possibility of users using different platforms cannot be ruled out. So we need our solution to support Linux and Unix-based macOS operating systems, where the majority of our target applications run. Therefore, the solution we propose must be cross-platform.

1.4 Essential Background

Before delving into the details of the design of our framework, we will first introduce two independent bodies of work to readers, which will be mentioned many times later. Although they are explained in detail in Chapter 2, the purpose here is to provide readers with some essential background knowledge.

Firstly, knowledge graphs (KG) as a form of structured human knowledge have drawn great research attention from both the academia and the industry [26, 58, 81]. A knowledge graph is a structured representation of facts, consisting of entities, relationships, and semantic descriptions. The knowledge graph can be used wherever there is a relationship. It has successfully captured a large number of customers, including Walmart, Google, LinkedIn, Adidas, HP, FT Financial Times, etc. well-known companies and institutions. Applications are still growing.

Compared with traditional data storage and calculation methods, the advantages of knowledge graphs are the following:

- Strong ability to express relationships: Traditional databases are usually read through tables, fields, etc., and the relationship levels and expression methods are diverse. Based on graph theory and probability graph models, it can handle complex and diverse association analyses.

- Knowledge learning: Using interactive machine learning technology, it supports learning functions based on interactive actions such as reasoning, error correction, and annotation, and continuously accumulates knowledge logic and

models, improves system intelligence.

- High-speed feedback: Schematic data storage method, compared with traditional storage methods, the data retrieval speed is faster, can return the results in seconds.

Knowledge graphs usually have two main types of storage [94, 11]: one is RDF (Resource Description Framework) based storage, explained in detail later in (Chapter 2, Section 2.3.1), and the other is using graph database stores, explained further in (Chapter 2, Section 2.3.2).

Secondly, the recommendation system (RS) has become a relatively independent research direction and is generally considered to have started with the GroupLens system launched by the GroupLens research group of the University of Minnesota in 1994 [28].

In the early stages of the development of recommendation systems, a common recommendation method was to sort items based on sales of items, clicks on topics, or news reads, and then select the top N items to form a ranking list and recommend them to users. This method has been quite effective; until today we still often see similar functions on major websites. But on the other hand, this method also has a huge flaw, that is, only a small number of top-ranked items can be recommended, and more items are buried unknown. Therefore, how to make full use of the existing items and make the recommendation as comprehensive as possible has become the main goal of the research in the field of recommendation systems, particularly, since personalized recommendation systems came into being.

According to the different implementation methods, we can divide the existing personalized recommendation methods into the following categories: Content-based recommendations, collaborative filtering-based recommendations and recommendations based on deep learning [1, 86]. In the method of content-based recommendation, the system recommends items similar to their past interests to users [62]. We will explain in detail later in Section 2.5.1. Collaborative filtering is another mainstream research direction in recommendation systems [69]. Through continuous

interaction with users, this method enables users to filter out items that are not of interest to users in their recommendation lists, thereby meeting their needs better. In Section 2.5.2, we will explain this further in detail. With the rapid rise of deep learning technology in recent years, many studies have shown that applying deep learning to recommendation systems can also achieve good results. As a highly readable external knowledge carrier, knowledge graphs provide a great possibility to improve algorithm interpretation capabilities [29]. Therefore, combining knowledge graph with recommendation systems is one of the hottest topics in the current recommendation system research.

A recommendation systems performance is measured based on relevance and accuracy of recommended items. One measure is AUC-ROC. `Definition:` Area under curve (AUC) is the area under the receiver operating characteristic (ROC) curve, which can be used to indicate the accuracy of the recommendation system; this will be explained in detail in Section 6.3.2.

1.5 Framework Design Questions

We put forward the following design questions that must be answered in our work to achieve our main goal and meet system requirements. These research questions drive our evaluation plan in Chapter 6 to show that our proposed solution meets our requirements.

- When we use our framework to instantiate an application, how can we increase the accuracy of RS?
- Can we make the system independent of the knowledge storage format, say RDF, Neo4j, etc.?
- Can we design a general extensible solution that meets our requirements?
- Can we use our solution to compare different knowledge graph storage modes to make recommendations for which storage mode is suitable for which application

type? Specifically, what are the advantages and disadvantages of different storage modes?

- Is side information useful for improving accuracy?
- How can we get information about products from the website?

From the situation described above, the design tasks can be intuitively divided into two parts: (1) build a knowledge graph storage system (2) enable recommendation system to use the knowledge graph to yield higher accuracy.

Since Google published the knowledge graph on May 16, 2012 in order to improve the quality of answers returned by search engines and the efficiency of user queries, the knowledge graph has become a popular research topic. But it is a problem to choose which way to represent, store, update and make effective use of the knowledge graph.

1.6 Contributions

Our main contribution is the following:

- We present an overall architecture that allows users to build knowledge graphs, display knowledge graphs, and enable recommender algorithms to be trained with knowledge when predicting user ratings for items. We demonstrate our framework for the movie recommendation domain.

Other contributions include:

- Our framework design is driven by a comprehensive set of functional and non-functional requirements.
- We offer a way in general that can allow the user to access different methods within the same set of APIs with good extensibility.

- We provide a pipeline that allows users to extract data, build knowledge graphs in different formats to display knowledge graphs without knowing the underlying details.
- We offer a way for the recommendation systems researchers to enable recommendation system experiments on top of `TensorFlow` and `Keras`.

We have designed and implemented the following:

- A modular framework solution that consists of the core and specialized frameworks that enables: crawling of data, and building a knowledge graph, knowledge graph visualization and recommender system.
- A crawling module that can enable our solution to support crawling of product websites like IMDB for movies, without writing rules. In addition, to be able to write simple rules to crawl other websites.
- A storage module which can enable our solution to support different knowledge storage formats, such as RDF and Neo4j.
- A deep learning support module that enables the integration of Python-based machine learning recommendation system.
- A recommendation specialized framework, instantiated for recommendation system task with a machine learning-based algorithm named MKR.

1.7 Thesis Outline

In this chapter, we presented the essential background for our work, pointed out the limitations in existing solutions, and stated the research and design problems giving them clear definitions and restricted scope. The rest of the thesis is organized as follows:

- In Chapter 2, we will present the background, and review the existing literature related to our work, and the available software that may be useful for our

implementation. At the end of this chapter, according to our needs, we will select the corresponding software and libraries.

- In Chapter 3 we will describe the design of the solution framework in a top-down manner.
- In Chapter 4 we will describe how to instantiate our framework to meet requirements.
- In Chapter 5, we will describe the applications built on top of our proposed solution, which presents a proof of concept, that our solution can actually fulfill our listed requirements.
- In Chapter 6, we will first demonstrate how both the functional and non-functional requirements are fulfilled by our framework solution. Then we report our results showing the benchmarks for the main components in our solution with commonly acknowledged metrics.
- In Chapter 7, we sum up our work with advantages and limitations and point out some potential directions for future work.

Chapter 2

Background and Literature Review

It follows from the discussion in the previous chapter, that in order to create a recommendation systems framework we will need to address the following questions: (1) How to extract relevant information from the web. (2) How to represent and store knowledge graph. (3) How to visualize the knowledge graph. (4) How to implement a recommendation system that uses the knowledge graph. In this chapter, we will introduce the background needed before we can proceed towards the design of our framework.

2.1 Background

2.1.1 Knowledge Graphs

Knowledge graph is essentially a large-scale semantic network, which contains a variety of entities, concepts and semantic relationships [51].

There are two important aspects of seeking information for which knowledge graphs were introduced. The first aspect is the explosion of internet information and the disorder of information. A knowledge graph can enable people to retrieve desired information more quickly and effectively. The second aspect is that people expect machines to understand massive amounts of linked information like humans, and expect faster, accurate, and intelligent ways of retrieving the information one

needs. A knowledge graph representation of information can also meet this demand.

The knowledge graph in its present form was first officially proposed by Google in May 2012 [87], when they released their own knowledge graph. As an Internet giant, Google's core requirement at that time was that users searched with a keyword and could directly get an answer instead of giving a web page containing the keyword. The emergence of the knowledge graph has allowed search engines to understand search keywords and return accurate answers [7]. Its fundamental purpose is to help machines understand information [82, 76, 78].

Before the emergence of the knowledge graph, there have been a variety of knowledge representation schemes, such as ontology, semantic web, text, etc.[38, 16]. However in recent years, it is the knowledge graph form which is most used as a background knowledge base [10]. To make a machine capable of cognition, the background information database must meet several conditions. The first is that the scale must be large enough to cover enough entities and concepts. The second is that the knowledge base should cover common semantic relations. The third is that the knowledge base should be very friendly. Text is a huge carrier, but text is a form of unstructured data that is difficult for machines to process. Hence, knowledge graphs are often expressed as RDF (Resource Description Framework) structures, which is a structure-friendly expression that can be effectively processed by computers [20].

Briefly, the knowledge graph provides complex semantic associations between items. The knowledge graph has the following benefits:

- **Combine disparate data silos**: When looking at our datas: they are kept in silos, so it is a highly time-consuming task to identify the right dots and information pieces, to connect them, to make sense of them, and finally to communicate them in the right way. In many industries, you can see a shift to data-centric execution rather than document-based communication [39].

- **Bring together structured and unstructured data**: Accumulating data does not mean just assembling documents and excel sheets. Knowledge Graph technology means being able to connect different types of data in meaningful

ways and supporting richer data services than most knowledge management systems. Organizations will then use the technology with the help of AI and machine learning techniques to extract and discover deeper and more subtle patterns [8].

- **Make better decisions by finding things faster**: Using knowledge graph technology can provide users with richer and deeper search results, which can help users provide relevant facts and background-related answers to specific questions [4].

2.1.2 Recommendation Systems

A recommendation system is a type of application that judges the items/services that the user currently needs or is interested in, based on the user's historical behaviour, social relationships, points of interest, context and other information. With the development of information technology and the Internet, people have moved from an era of lack of information to an era of information overload.

For users, it is becoming more difficult to find the information they are interested in from a large amount of information. For information producers, it is also becoming more difficult for the information produced by themselves to stand out from the crowd. The recommendation system came into being just to address this problem.

Recommendation system applications can be seen on various websites on the Internet, although the technologies used may differ considerably. In general, almost all recommendation system applications are composed of the front-end display page, the back-end log system and a recommender algorithm.

There are three common approaches in recommender algorithms [5], which we will explain in detail in section 2.5:

- Content-based recommendation system.
- Collaborative filtering recommendation system.
- Machine learning enhanced recommendation system.

As we mentioned in Section 1.4, many studies have shown that applying deep learning to recommendation systems can achieve good results [86, 82, 79]. Combining the knowledge graph with recommendation systems is the hottest topic in the current recommendation system research. Yet, through our review of previous research, we found that most of the existing works that use knowledge graphs as side information do not integrate the construction or use of knowledge graphs. In fact, most of them use open source CSV files or txt files for input of knowledge to the recommender alorithm[79, 33, 84]. This has an obvious disadvantage. If we need to delete or modify pieces of information, we need to traverse each line of the file. Or when we have information in multiple files, we need to traverse each line of each file, which is very inefficient. But if our framework includes the construction of the knowledge graph, we can perform unified operations on the nodes in the knowledge graph by calling the framework's API. It is not only fast, but also makes it more convenient to query, and more convenient to manage.

2.2 Information Extraction

Since we will be illustrating our framework for movie recommendations, we consider that domain, Internet Movie Database (IMDB). An IMDB extractor is a program or script that automatically extracts information on the World Wide Web following certain rules [89]. IMDB extractor is roughly divided into general crawlers, focused crawlers, incremental crawlers, and deep web crawlers according to the system structure and implementation technology [12, 3]. However, in practice, it is generally achieved by combining several crawler technologies. This is a diagram of the internal workings of a typical web crawler Figure 1.

The extractor workflow generally follows the steps below [49]:

- First select some seed URLs.
- Put these URLs into the queue of URLs to be crawled.
- Take the URL from the URL queue to be crawled, parse the DNS, and adjust

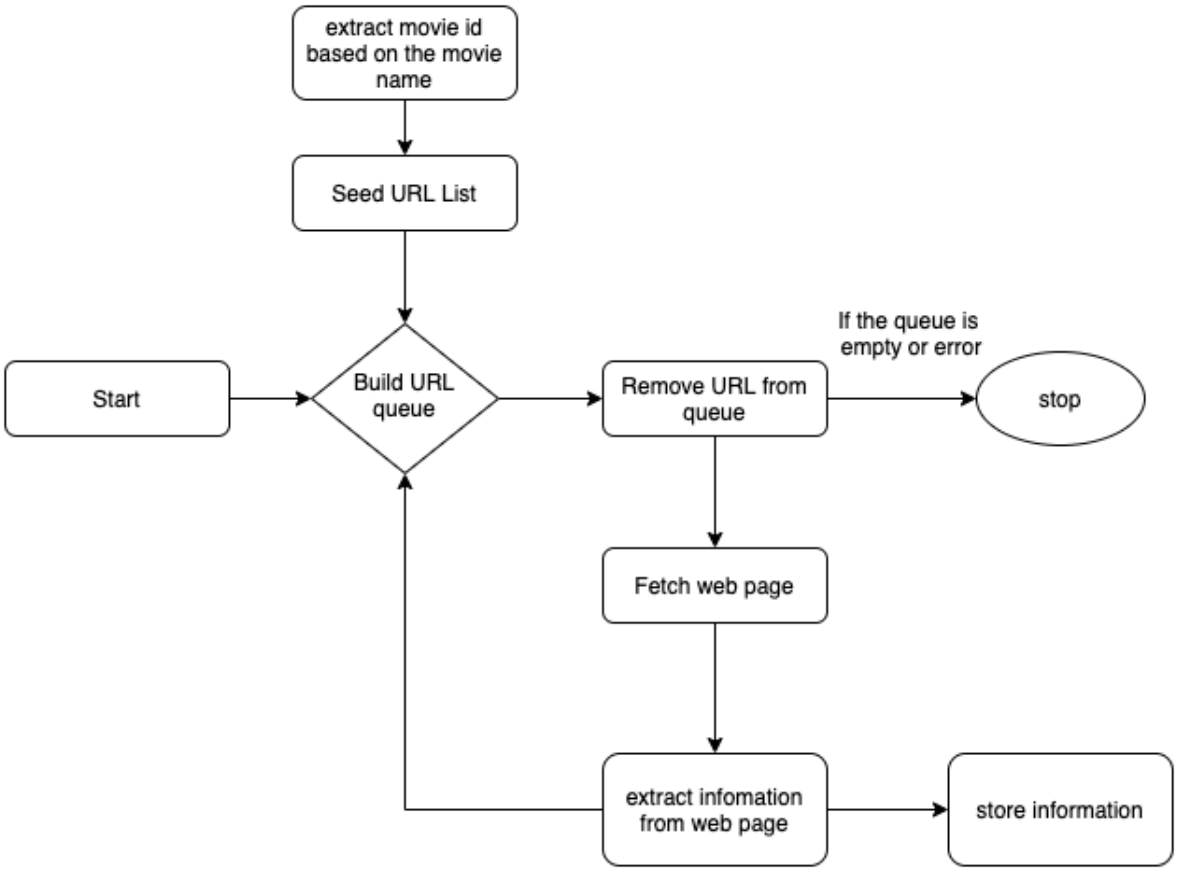

Figure 1: Basic crawler workflow

the regular expression according to the data that the user wants to intercept, download the data, and store it in the database. Besides, put these URLs into the crawled URL queue.

- Analyze the URL in the seed URL list, and put a part of the URL into the URL queue to be crawled, to enter the next cycle.
- If the queue is empty, then exit the program.

2.3 Knowledge Graph Storage

In this section, we present current forms used for knowledge graph storage. The value of the knowledge graph is reflected in its addition to downstream tasks such

☐ include secondary database models — 32 systems in ranking, June 2020

Rank Jun 2020	Rank May 2020	Rank Jun 2019	DBMS	Database Model	Score Jun 2020	Score May 2020	Score Jun 2019
1.	1.	1.	Neo4j	Graph	48.27	-1.49	-1.28
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	30.80	+0.13	+2.56
3.	3.	↑ 4.	ArangoDB	Multi-model	5.38	+0.70	+0.81
4.	4.	↓ 3.	OrientDB	Multi-model	4.82	+0.68	-0.77
5.	5.	5.	Virtuoso	Multi-model	2.28	-0.07	-0.83
6.	6.	↑ 7.	Amazon Neptune	Multi-model	2.17	+0.41	+0.93
7.	7.	↓ 6.	JanusGraph	Graph	2.01	+0.36	+0.46
8.	↑ 9.	↑ 11.	Dgraph	Graph	1.40	+0.31	+0.51
9.	↓ 8.	↓ 8.	GraphDB	Multi-model	1.25	+0.06	+0.16
10.	↑ 12.	↑ 18.	FaunaDB	Multi-model	1.19	+0.25	+0.84
11.	↓ 10.	↑ 13.	Stardog	Multi-model	1.15	+0.08	+0.44
12.	↓ 11.	↓ 9.	Giraph	Graph	0.97	+0.02	-0.11
13.	13.	↓ 12.	TigerGraph	Graph	0.90	+0.05	+0.18
14.	14.	↓ 10.	AllegroGraph	Multi-model	0.88	+0.04	-0.04
15.	15.	15.	Blazegraph	Multi-model	0.67	+0.03	+0.11
16.	↑ 17.	↑ 24.	Grakn	Multi-model	0.59	+0.13	+0.39
17.	↓ 16.	↓ 16.	Graph Engine	Multi-model	0.58	+0.03	+0.05
18.	18.	↓ 17.	InfiniteGraph	Graph	0.39	+0.00	+0.01
19.	↑ 20.	↑ 32.	Fluree	Graph	0.29	+0.05	+0.29
20.	↓ 19.	↓ 19.	FlockDB	Graph	0.27	+0.01	+0.00

Figure 2: Graph-based storage ranking [24]

as dialogue systems and information extraction. Therefore, the knowledge storage system should be able to support fast and frequent knowledge query operations [55].

Secondly, the knowledge in the graph is extracted according to the structure of the schema [32]. Therefore, the storage of knowledge should be able to reflect the hierarchy of knowledge and the relationships between knowledge items according to the structure of the schema.

Besides, knowledge storage should be as efficient as possible to avoid wasting too much storage space. From our exploration, we found that there are two main storage methods for knowledge graphs: one is RDF-based storage [11]; the other is graph-based storage [80]. DB-engines website announced the ranking based on these two storage methods Figure 2 and Figure 3.

2.3.1 RDF-based Storage

Resource Description Framework (RDF), its essence is a data model. It provides a unified standard for describing entities. Simply put, it is a method and means of

☐ include secondary database models — 19 systems in ranking, June 2020

Rank Jun 2020	Rank May 2020	Rank Jun 2019	DBMS	Database Model	Score Jun 2020	Score May 2020	Score Jun 2019
1.	1.	1.	MarkLogic	Multi-model	11.24	+0.28	-2.26
2.	2.	↑ 3.	Apache Jena - TDB	RDF	2.69	-0.10	+0.42
3.	3.	↓ 2.	Virtuoso	Multi-model	2.28	-0.07	-0.83
4.	4.	4.	Amazon Neptune	Multi-model	2.17	+0.41	+0.93
5.	5.	5.	GraphDB	Multi-model	1.25	+0.06	+0.16
6.	6.	↑ 7.	Stardog	Multi-model	1.15	+0.08	+0.44
7.	7.	↓ 6.	AllegroGraph	Multi-model	0.88	+0.04	-0.04
8.	8.	8.	Blazegraph	Multi-model	0.67	+0.03	+0.11
9.	9.	↑ 10.	RDF4J	RDF	0.48	+0.04	+0.11
10.	10.	↓ 9.	Redland	RDF	0.43	+0.02	-0.05
11.	11.	↑ 12.	4store	RDF	0.36	+0.01	+0.04
12.	12.	↑ 14.	CubicWeb	RDF	0.17	+0.01	-0.04
13.	13.	↑ 16.	RedStore	RDF	0.14	-0.01	+0.01
14.	↑ 17.	↑ 17.	Strabon	RDF	0.09	+0.01	-0.01
15.	↓ 14.	↓ 13.	AnzoGraph	Multi-model	0.09	-0.01	-0.12
16.	16.	↑ 19.	BrightstarDB	RDF	0.08	+0.00	+0.05
17.	↓ 15.	↓ 15.	Mulgara	RDF	0.07	-0.02	-0.07
18.	18.	18.	Dydra	RDF	0.07	+0.00	+0.03
19.	19.	↑ 20.	SparkleDB	RDF	0.00	±0.00	-0.02

Figure 3: RDF-based storage ranking [25]

representing things. RDF is formally expressed as SPO (Subject-Predicate-Object) triples [71]. In the knowledge graph, we also call it a piece of knowledge, as shown in Figure 4.

At present, there are mamy ways of RDF serialization: RDF/XML, N-Triples, Turtle, RDFa, JSON-LD and so on [75].

- RDF/XML: RDF data is expressed in XML format. The reason for this method is because the technology of XML is relatively mature, and there are many ready-made tools to store and parse XML. However, for RDF, the format of XML is too verbose and not easy to read. Hence, we do not use this method to process RDF data.

- N-Triples: Using multiple triples to represent the RDF data set is the most intuitive representation. In the file, each line represents a triplet, which is convenient for machine analysis and processing. The open domain knowledge graph DBpedia usually publishes data in this format.

- Turtle: It is the most used RDF serialization method. It is more compact than

RDF/XML and more readable than N-Triples.

- RDFa: "The Resource Description Framework in Attributes"(RDFa) is an extension of HTML5 that allows website builders to mark entities on the page like people, places, time, comments, etc. without changing any display effects. In other words, by embedding RDF data in web pages, search engines can better parse unstructured pages and obtain some usefully structured information. It intuitively displays the pages that ordinary users see.

- JSON-LD: "JSON for Linking Data"(JSON-LD) uses key-value pairs to store RDF data.

Below, we use Jon Favreau as an example to give a specific representation of its N-Triples and Turtle Figure 5. When expressed in Turtle, we will add a prefix to abbreviate the IRI (Internationalized Resource Identifier) of RDF Figure 6. The same entity has multiple data properties or relationships. We can use only one subject to make it more compact. We can change the Turtle above to Figure 7.

From the above examples, we can find that RDF has limited expressive power, cannot distinguish between classes and objects, and cannot define and describe the relationships/properties of classes. Because RDF lacks abstraction capabilities, it cannot define and describe things in the same class. Taking the knowledge graph of Jon Favreau as an example, RDF can express the properties of the two entities Jon Favreau and New York and the relationship between them. But if we want to define that Jon Favreau is a person, New York is a place, and what properties a person has, what properties a place has, and what relationship exists between a person and a place, for this RDF is inadequate. Whether in the concept of intelligence or in practical applications, this generalization and abstraction ability is very important; at the same time, this is also emphasized by the knowledge graph itself. Combining the two technologies RDFs and OWL Web Ontology Language, or schema/ontology language, helps solve this problem of RDF's limited expressive power.

Figure 4: SPO triples

Example1 N-Triples:

<http://www.kg.com/person/1> <http://www.kg.com/ontology/Name> "Jon Favreau"^^string.
<http://www.kg.com/person/1> <http://www.kg.com/ontology/career> "director"^^string.
<http://www.kg.com/person/1> <http://www.kg.com/ontology/birthDate> "1966-10-19"^^date.
<http://www.kg.com/person/1> <http://www.kg.com/ontology/height> "180"^^int.
<http://www.kg.com/person/1> <http://www.kg.com/ontology/weight> "98"^^int.
<http://www.kg.com/person/1> <http://www.kg.com/ontology/nationality> "America"^^string.
<http://www.kg.com/person/1><http://www.kg.com/ontology/hasBirthPlace><http://www.kg.com/place/10086>.
<http://www.kg.com/place/10086> <http://www.kg.com/ontology/address> "New York"^^string.
<http://www.kg.com/place/10086> <http://www.kg.com/ontology/coordinate> "-22.908333, -43.196389"^^string.

Figure 5: Jon Favreau N-triples

Example2 Turtle:

@prefix person: <http://www.kg.com/person/> .
@prefix place: <http://www.kg.com/place/> .
@prefix : <http://www.kg.com/ontology/> .

person:1 :Name "Jon Favreau"^^string.
person:1 :career "director"^^string.
person:1 :birthDate "1966-10-19"^^date.
person:1 :height "180"^^int.
person:1 :weight "98"^^int.
person:1 :nationality "America"^^string.
person:1 :hasBirthPlace place:10086.
place:10086 :address "New York"^^string.
place:10086 :coordinate "-22.908333, -43.196389"^^string.

Figure 6: Jon Favreau turtles

```
Example3 Turtle:

@prefix person: <http://www.kg.com/person/> .
@prefix place: <http://www.kg.com/place/> .
@prefix : <http://www.kg.com/ontology/> .

person:1 :Name "Jon Favreau"^^string;
        :career "director"^^string;
        :birthDate "1966-10-19"^^date;
        :height "180"^^int;
        :weight "98"^^int;
        :nationality "America"^^string;
        :hasBirthPlace place:10086.
place:10086 :address "New York"^^string;
          :coordinate "-22.908333, -43.196389"^^string.
```

Figure 7: Jon Favreau subject turtles

2.3.1.1 RDFs

Resource Description Framework Schema(RDFs) serialization is no different from RDF [22]. In fact, the form of expression in RDFs is RDF. In this example, we use the term “rdfs:Class" to define the two classes “person" and “place". In RFDs, there is no distinction between data properties and object properties. The term “rdf:Property" defines properties, which are the “edges" of RDF Figure 8.

The following are some of the more important and commonly used terms in RDFs:

- `rdfs:Class` Used to define classes.
- `rdfs:domain` Used to indicate which category this attribute belongs to.
- `rdfs:range` Used to describe the value type of this attribute.
- `rdfs:subClassOf` Used to describe the parent class of this class. For example, we can define an athlete class and declare that this class is a subclass of human.
- `rdfs:subProperty` Used to describe the parent property of this property. For example, we can define a name attribute and declare that the Chinese name

and full name are subclasses of name.

In order to more intuitively understand the layers represented by RDF and RDFs/OWL in the knowledge graph, we use the following diagram to represent the data layer and the pattern layer in the example Figure 9.

The Data layer is our specific description of Jon Favreau using RDF, the vocabulary layer is some vocabulary (category, attribute) we have defined, and the RDFs layer is a predefined vocabulary. In the figure, we use red rounded rectangles to represent classes, green fonts to represent “rdf:type", “rdfs:domain", and “rdfs:range" three predefined vocabularies, and dashed lines to represent “rdf:type". Besides, in order to reduce the intersection of the lines in the figure, we only kept the relationship of “rdf:type" of the attribute of career and omitted the relationship of other attributes.

2.3.1.2 OWL

As mentioned above, RDFs is essentially an extension of the RDF vocabulary. Later, it was discovered that the expression ability of RDFs was still quite limited, so OWL was proposed. We can also use OWL as an extension of RDFs, which adds additional predefined vocabulary [42, 72].

Web Ontology Language (OWL) is one of the core layers of the semantic web technology stack. OWL has two main functions:

- Provide fast and flexible data modeling capabilities.
- Efficient automatic reasoning.

The following is how one uses OWL for data modeling Figure 10. Say, we want to use OWL to describe Jon Favreau’s semantic layer:

- Here we use “`owl:Class`" to define the two classes “Person" and “Place".
- Unlike RDFs and RDF, OWL distinguishes between data properties and object properties (object properties represent the relationship between entities).

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://www.kg.com/ontology/> .

:Person rdf:type rdfs:Class.
:Place rdf:type rdfs:Class.

:Name rdf:type rdf:Property;
      rdfs:domain :Person;
      rdfs:range xsd:string .

:career rdf:type rdf:Property;
      rdfs:domain :Person;
      rdfs:range xsd:string .

:fullName rdf:type rdf:Property;
      rdfs:domain :Person;
      rdfs:range xsd:string .

:birthDate rdf:type rdf:Property;
      rdfs:domain :Person;
      rdfs:range xsd:date .
```

Figure 8: Jon Favreau RDFs example

"`owl:DatatypeProperty`" defines data properties, "`owl:ObjectProperty`" defines object properties.

After the description language of the schema layer is changed to OWL, the hierarchy diagram is expressed as: Data properties are represented in cyan, and object properties are represented in blue Figure 11. The following are some of the more important and commonly used vocabularies of OWL:

- `owl:TransitiveProperty` Indicates that the attribute has transitive properties. For example, we define "located" as a transitive property. If A is in B and B is in C, then A must be in C.
- `owl:SymmetricProperty` Indicates that the property has symmetry. For

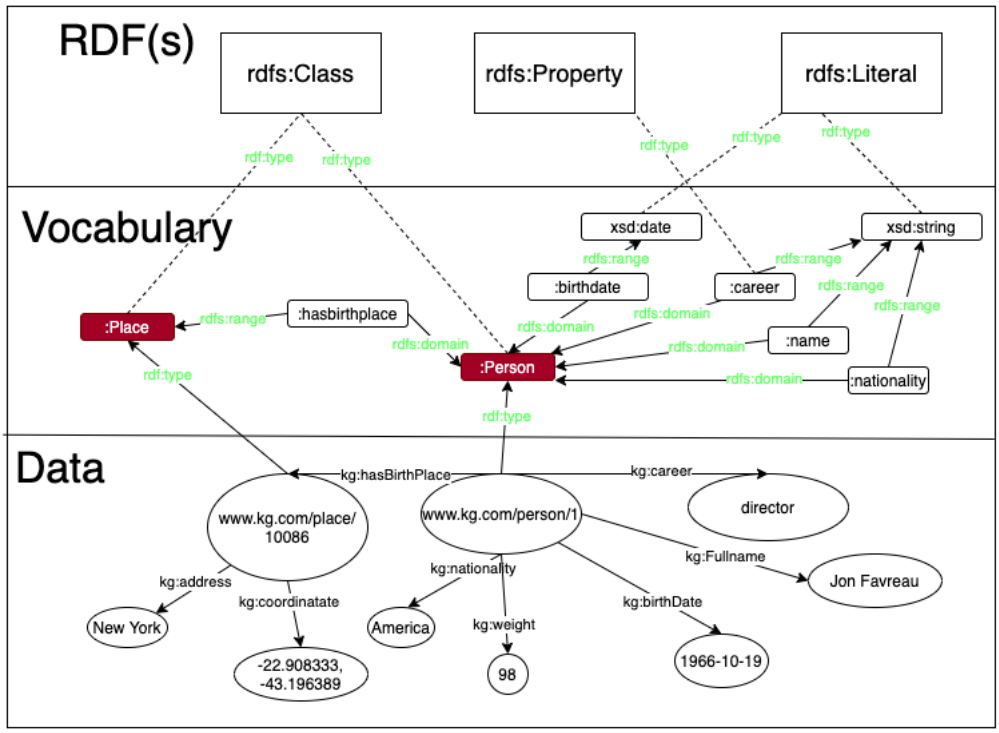

Figure 9: RDFs layers example

example, we define “awareness" as a symmetrical property. If A knows B, then B must know A.

- `owl:FunctionalProperty` Represents the uniqueness of the value of this property. For example, we define “mother" as a unique attribute. If A’s mother is B, and elsewhere we know that A’s mother is C, then B and C refer to the same person.

- `owl:inverseOf` Defines the inverse relationship of an attribute. For example, the reverse relationship that defines “parents" is “children". If A is B’s parents, then B must be A’s children.

- `owl:equivalentClass` Indicates that a certain class is the same as another class.

- `owl:equivalentProperty` Indicates that a certain property is the same as another property.

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://www.kg.com/ontology/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

:Person rdf:type owl:Class.
:Place rdf:type owl:Class.

:Name rdf:type owl:DatatypeProperty;
      rdfs:domain :Person;
      rdfs:range xsd:string .

:career rdf:type owl:DatatypeProperty;
      rdfs:domain :Person;
      rdfs:range xsd:string .

:birthDate rdf:type owl:DatatypeProperty;
      rdfs:domain :Person;
      rdfs:range xsd:date .
```

Figure 10: Jon Favreau OWL example

- `owl:sameAs` Indicates that the two entities are the same entity.

Ontology mapping is mainly used to merge multiple independent Ontology Schemas [56]. For example, PersonA constructs an ontology structure, which defines a class such as "Person" to represent people; PersonB defines "Human" class to represent people in the ontology that it builds. When we merge these two ontologies, we can use OWL's ontology mapping method. Without OWL, it is difficult to merge knowledge graphs.

As we mentioned before, one of the characteristics of OWL is reasoning [68]. The reasoning of knowledge graph is mainly divided into two categories: ontology-based reasoning and rule-based reasoning [85]. Here we are talking about reasoning based on ontology. It is not difficult to find that the property feature function introduced above creates the premise for reasoning about RDF data. At this point, we can add a reasoner that supports OWL reasoning to perform ontology-based reasoning. RDFs

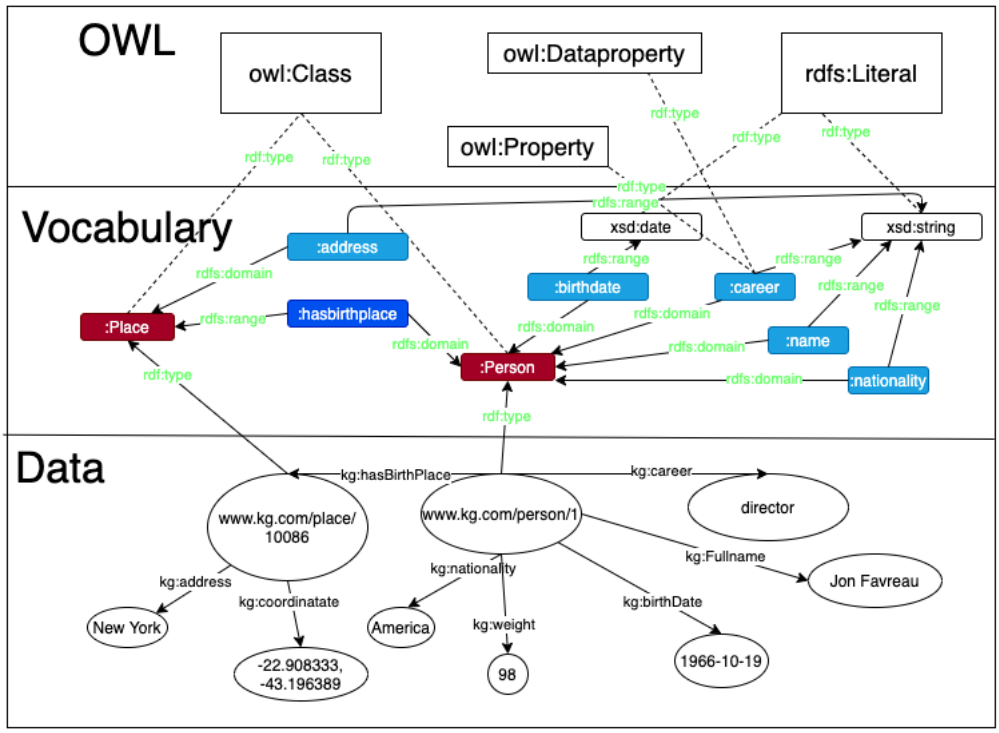

Figure 11: Jon Favreau OWL layers

also supports reasoning, but due to the lack of expressive ability, its reasoning ability is not strong. For example, we use RDFs to define two classes: human and animal. In addition, define human as a subclass of animal. At this time the inference engine can infer that if an entity is a human, then it is also an animal. OWL also supports this basic reasoning. In addition, with its strong expressive ability, we can make more complex reasoning. Imagine a scenario where we have a huge database that stores the kinship of characters. Many of the relationships in it are one-way. For example, Its only saved A's father (mother) is B, but there is no A in the child field of B, as shown in the following table Figure 12.

If there is only a single relationship and the amount of data is not large, we can manually complete this relationship. If there are hundreds of relationship types and hundreds of millions of characters, it is difficult to deal with when modifying, adding, or deleting relationships. If we use "inverseOf" to indicate that "hasParent" and "hasChild" are in an inverse relationship with each other, the above data can be

personNotation	hasParent	hasChild
A	B	
B		

Figure 12: OWL reasoning example

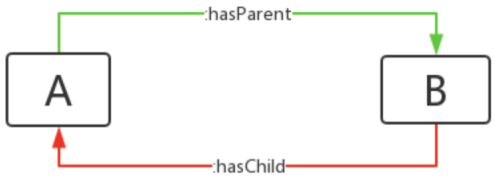

Figure 13: OWL inverse reasoning example

expressed as Figure 13. The relationship in green indicates that it actually exists in our RDF data, and the relationship in red is obtained by reasoning.

2.3.2 Graph-based Storage

Knowledge graph because its data contains entities, attributes, relationships, etc., and because common relational databases such as MySQL can not reflect these characteristics of the data, the storage of knowledge graph data is generally a graph database.

The advantage of the graph database is that it can naturally represent the structure of the knowledge graph, the nodes in the graph represent the objects of the knowledge graph, and the edges in the graph represent the object relationship of the knowledge graph; the advantage of this approach is that the database itself provides a complete graph query language, supporting various graph mining algorithms. Query speed is better than relational database, especially multi-hop query performance is better. However, its disadvantage is that the update of the graph database is more complicated, the distributed storage of the graph database is expensive, the data update speed is slow, and the processing overhead of large nodes is high.

2.3.2.1 Neo4j

The data storage form of Neo4j, a graph database, is mainly to organize data by nodes and edges. Node can represent the entity in the knowledge graph, edge can be used to represent the relationship between entities, the relationship can have a direction, the two ends correspond to the start node and end node. In addition, we can add one or more labels on the node to indicate the classification of the entity, and a set of key-value pairs to represent some additional attributes of the entity in addition to the related attributes.

As a graph database, Neo4j has the following advantages [14]:

- Faster database operation.
- The data is more intuitive and the SQL statements are better written.
- More flexible. No matter what new data needs to be stored, they are all nodes and edges, and only node attributes and edge attributes need to be considered.
- The speed of database operations does not decrease significantly with the increase of the database.

Neo4j uses Cypher as the query language. `Cypher` is a descriptive graph query language, it is not necessary to write the traversal code of the graph structure, which is convenient for efficient query of the graph storage [57]. Cypher is designed to be a human query language, and many of its keywords such as “like" and “order by" are inspired by SQL. The expression for pattern matching comes from SPARQL. The SPARQL query language contains the following main parts:

- START: At the starting point in the figure, it is obtained by looking up the ID or index of the element.
- MATCH: Matching pattern of graphics.
- WHERE: Filter condition.
- RETURN: Return what is needed.

2.3.2.2 ArangoDB

ArangoDB is a multi-model database, which has three data model storage formats with graph, document and key/value pair [91]. The reason for its quick and flexible query is that it has a unified kernel and unified database query language-AQL (ArangoDB Query Language), which applies to all three data models. It can cover all three data models, and also allows a mixture of three data models in a single query.

Therefore, users can mix and use multiple data models in a single query without switching between different data models or performing data transmission. And all three data models support horizontal expansion. Based on its local integrated multi-model feature, ArangoDB's native multi-model database is suitable for building high-performance applications. ArangoDB has the following characteristics:

- Multi-data model: Users can flexibly use document, graph, key-value or their combination as their data model.
- Convenient query: Support SQL-like query syntax AQL, or through REST and other queries.
- Ruby and JS extensions: No language range restrictions, user can use the same language from the front-end to the back-end.
- Simple and easy to use: It can be started and used in a few seconds, and the user can manage their ArangoDB through a graphical interface.
- Open source and free: ArangoDB complies with Apache protocol.

2.4 Knowledge Graph Visualization

Data visualization is not a necessary part, but visualization has some irreplaceable benefits. Graph data is easier to display visually, and graph visualization can help data analysts, business users and developers improve analysis efficiency. Anyone browsing the graph can view the connections, identify areas of interest or quickly assess the current state and organization of the data. As one can imagine, this can

provide insights that other types of data representations cannot provide, and bring tremendous value. Visualization helps make anomalies or related patterns stand out to help the human eye and brain detect them, and other types of data formats may not highlight hidden structures. In this section, we will review three of the existing methods for knowledge graph visualization:

- Neo4j Desktop
- Protege
- Networkx

2.4.1 Neo4j Desktop

The design of Neo4j is very intuitive. With nodes and relationships, users can easily model data into content that is easy to understand for developers, data analysts, and bosses. The visual structure also makes querying, building and maintaining easier to read and expand. In the absence of graphs, we can view the data in Neo4j in many ways. Neo4j can also return the results in JSON, XML, and Table formats. Neo4j has two main visualization tools that are specifically designed for use with data in the Neo4j graph database: Neo4j Browser and Neo4j Bloom. Neo4j browser is a tool for developers that allows them to perform Cypher queries and visualize the results. It is the default developer interface for the enterprise and community editions of the Neo4j database. Neo4j Bloom is a commercially licensed product that allows users to browse their graph data in natural language.

Neo4j embeds visual content into applications, allowing developers to create applications that use visual content as part of the user interface. It should be noted that these libraries usually do not support extremely complex or heavy workloads and do not have vendor support or SLAs (service level agreement) for feature requests. Because they are managed by the community, these tools rely on the community to provide support and feature improvements.

2.4.2 Protege

Protege is an open-source Java tool researched and designed by Stanford University. It provides an extensible framework for the development of application systems based on knowledge bases. It is currently a relatively complete ontology support tool and has many professional ontology fields built with Protege. Protege provides a friendly graphical user interface for editing classes, attributes, and instances involved in Ontology.

The various ontology visualization plug-ins supported by Protege can immediately display the edited ontology in a variety of visualization methods. There are 13 visualization plug-ins listed on Protege's official website. These visualization tools have their characteristics and are rich in functions.

2.4.3 Networkx

Networkx was produced in May 2002. It is a software package written in python, which is convenient for users to create, operate and learn complex networks. Users can use networkx to store networks in standardized and non-standardized data formats, generate a variety of random networks and classic networks, analyze network structures, establish network models, design new network algorithms, and perform network drawing, etc. Networkx supports the creation of four graph types:

- Graph undirected graph, allowing nodes to form a closed loop with itself.
- DiGraph directed graph.
- MultiGraph, multiple undirected graphs, a flexible graph type that allows multiple undirected edges between pairs of nodes. The extra flexibility leads to some performance degradation, but it is usually not significant.
- MultiDiGraph, multi-directed graph.

2.5 Recommendation System

In this section, we are going to review the existing approaches for the recommendation system task, most of them can be sorted into the following categories [5]:

- Content-based Filtering.
- Collaborative Filtering.
- Machine learning enhanced recommendation system.

2.5.1 Content-based Recommendations

Content-Based Recommendations build a recommender algorithm model based on item-related information, user-related information and user actions on items to provide users with recommendation services [53, 48]. The item-related information here may be metadata, tags, user comments, manually marked information, etc. that describe the item text. User-related information refers to demographic information (such as age, gender, preference, region, income, etc.). The user's operations on items can be comments, favourites, likes, viewing, browsing, clicking, adding a shopping cart, buying, etc. The content-based recommendation algorithm generally only relies on the user's behaviour to provide recommendations for the user and does not involve the behaviour of other users.

The disadvantage of content-based recommendation is that thorough content analysis is required, and there is a user cold-start problem, which cannot bring surprises to users (only recommend products with similar content).

2.5.2 Collaborative Filtering-based Recommendations

Collaborative filtering is different from traditional content-based filtering. It analyzes user interests, finds users with similar interests in specified users in the user group. Based on the evaluation of certain information by these similar users, the system predicts how much the specified user likes a specific item [70].

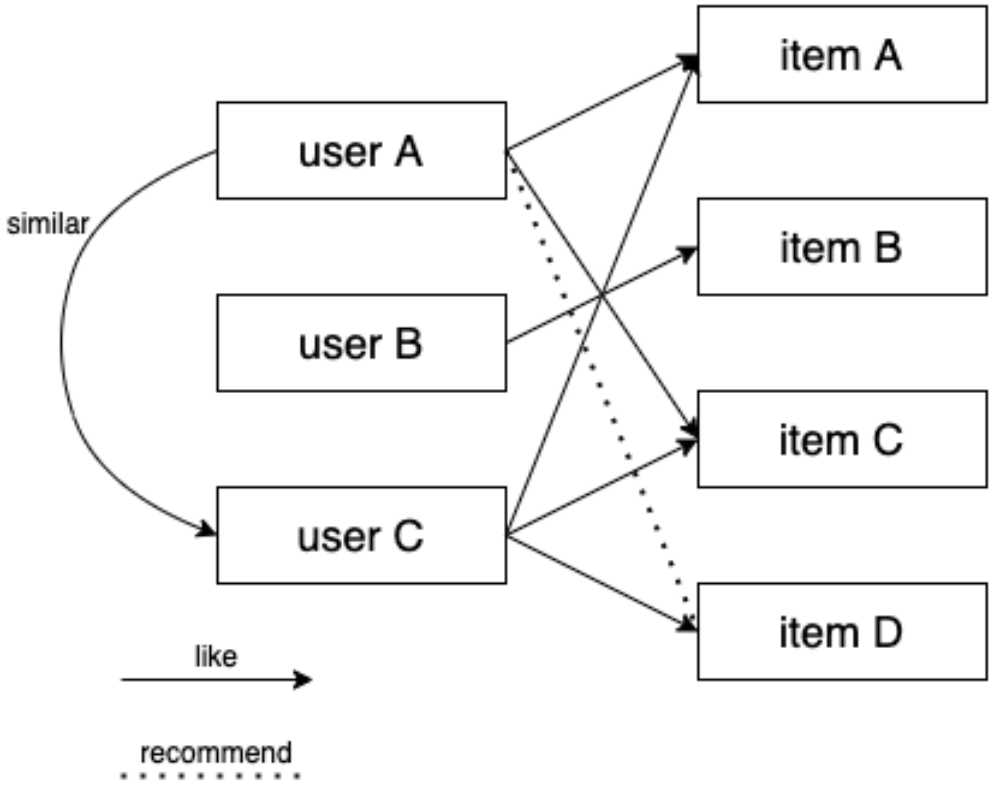

Figure 14: User-based collaborative filtering example

2.5.2.1 User-based Collaborative Filtering

The user-based collaborative filtering algorithm finds the user's liking for the product or content (such as product purchase, collection, content comment or sharing) through the user's historical behaviour data, normalize and score these preferences [18, 65]. The relationship between users is calculated based on the attitudes and preferences of different users towards the same product or content. Recommends products among users with similar preferences. The following are the specific steps.

- **Find users with similar preferences**: We simulated five users' ratings on two products to illustrate how to find similar users based on their attitudes and preferences for different products Figure 15. It is difficult to visually find the connection between the five users from the table. We will show the scores in a graph, and the relationship between the users is easy to find. In the figure, the Y-axis is the rating of item 1, and the X-axis is the rating of item 2. According

	A	B	C
1		item 1	item 2
2	user A	3.3	6.5
3	user B	5.8	2.6
4	user C	3.6	6.3
5	user D	3.4	5.8
6	user E	5.2	3.1
7			

Figure 15: User item rating example

to the distribution of users, it can be found that the three users A, C, and D are relatively close. User E and user B form another group Figure 16.

- **Calculate the distance between users**: Euclidean distance evaluation is a relatively simple user relationship evaluation method. The principle is to determine whether different users have the same preference by calculating the distance between two users in the mapping Figure 17.

- **Provide recommended items for similar users**: When we need to recommend products to users, we first check the previous similarity list and find that users A, C, and D have high similarities. These three users have the same preferences. Therefore, we can recommend the products purchased by A and D to user C.

2.5.2.2 Item-based Collaborative Filtering

The item-based collaborative filtering algorithm is very similar to the user-based collaborative filtering. Items and users are interchanged. The relationship between items is obtained by calculating the scores of different items by different users. Recommend similar items to users based on the relationship between items [9, 65]. The following are the specific steps.

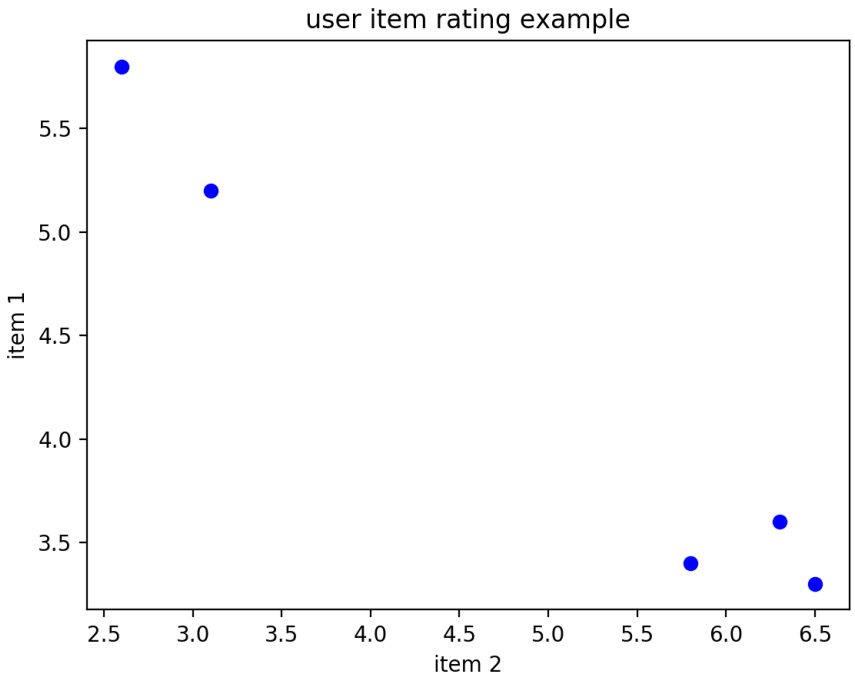

Figure 16: User item rating example

	A	B
1		distance
2	user A/B	4.63
3	user A/C	0.36
4	user A/D	0.71
5	user A/E	3.89
6	user B/C	4.3
7	user B/D	4
8	user B/E	0.78
9	user C/D	0.54
10	user C/E	3.58
11	user D/E	3.24

Figure 17: User distance example

	A	B	C
1		user A	user B
2	item 1	3.3	6.5
3	item 2	5.8	2.6
4	item 3	3.6	6.3
5	item 4	3.4	5.8
6	item 5	5.2	3.1

Figure 18: Item user rating

- **Find similar items**: The table shows the scores of 5 products by two users Figure 18. The the ratings of two users are used to obtain the similarity between the 5 products. From the table alone, we still find it difficult to find the connection, so we chose to show it through the graph Figure 19.

- **Calculate the distance between items**: From the Euclidean coefficient, we can see that items 1, 3, and 4 are closely related to each other Figure 20.

- **Provide users with recommendations based on similar items**: When we need to recommend users to products, we first check the previous similarity list and find that items 1, 3, and 4 have high similarities. These three items have the same preferences. Therefore, we can recommend the user who purchased item 1 and 4 to item 3.

Collaborative filtering does not require domain knowledge of items, so it is difficult to handle new items or new users, and it is difficult to take into consideration the side features of items.

2.5.3 Recommendation System Frameworks

While much of the research in recommendation systems has been focused on the underlying algorithms, there is some work on developing frameworks. Through our reading of paper [74], below we describe a number of frameworks found in the

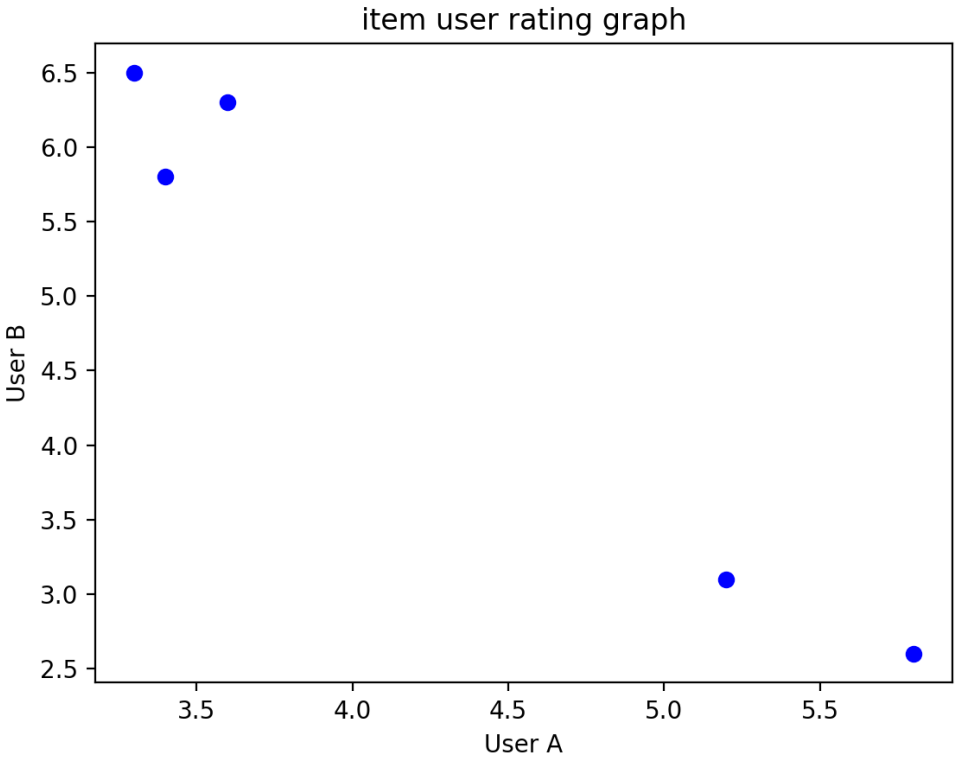

Figure 19: Item user rating graph

	A	B
1		distance
2	item 1/2	4.63
3	item 1/3	0.36
4	item 1/4	0.71
5	item 1/5	3.89
6	item 2/3	4.3
7	item 2/4	4
8	item 2/5	0.78
9	item 3/4	0.54
10	item 3/5	3.58
11	item 4/5	3.24

Figure 20: Item distance

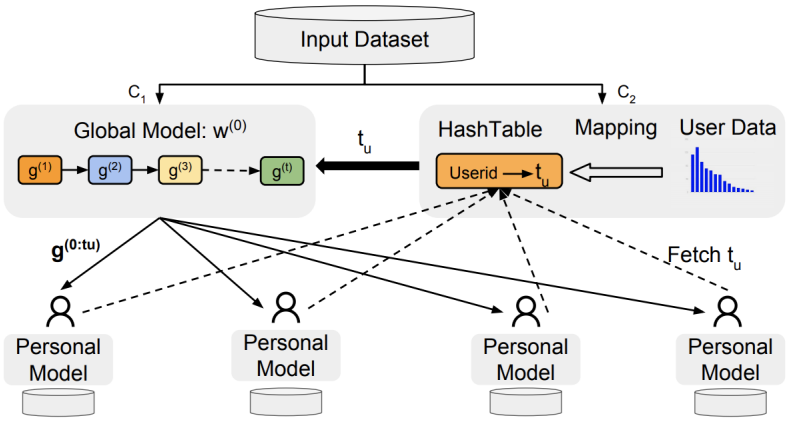

Figure 21: The structure of the gradient descent model.

literature.

2.5.3.1 A Gradient based Adaptive Learning Framework for Efficient Personal Recommendation

Yue et al. [59] use gradient descent to learn the user's model for the recommendation. Three machine learning algorithms (including logistic regression, gradient boosting decision tree and matrix decomposition) are used. Figure 21 describes the structure of the recommendation system.

Although gradient boosting decision tree can prevent overfitting and has strong interpretability, Boost is a serial process, which is not easy to parallelize and has high computational complexity. At the same time, it is not suitable for high-dimensional sparse features. If there are many features, each regression tree will consume a lot of time.

2.5.3.2 Raccoon Recommendation Engine

Raccoon [54] is built on top of Node.js and Redis. It is a recommendation system framework based on collaborative filtering. The system uses k-nearest-neighbours to classify data. The classification idea is similar to the collaborative filtering idea mentioned in Section 2.5.2.

Because Raccoon uses the collaborative filtering algorithm, it needs to calculate the similarity of users or items. Raccoon uses two different coefficients to calculate the similarity, namely Pearson and Jaccard. The original implementation of Raccoon uses Pearson which was good for measuring discrete values in a small range. But in order to make the calculation faster, you can also use Jaccard, which is a calculation method used to measure binary rating data (ie like/dislike). But the collaborative filtering algorithm doesn't care about the inner connection of characters or objects. The collaborative filtering algorithm only needs to enter the user ID and movie ID to make recommendations. For example, Robert Downey is an actor in Iron man, and he is also an actor in Iron man2. If another user related to this user has only seen ironman but not ironman2, the system will not recommend ironman2 to this user.

If the number of users is large, optimization needs to be performed, and it is inefficient to compare each user with other users.

2.5.3.3 Good Enough Recommendations (GER)

GER (Good Enough Recommendation) [37] is a scalable, easy-to-use and easy-to-integrate recommendation engine. GER is an open source NPM module. Its core is the same as the knowledge graph triplet (people, actions, things). GER recommends in two ways. One is comparing two people by looking at their history, another one is providing recommendations from a person's history.

GER is implemented in Coffee-Script on top of Node.js. Its core logic is implemented in an abstraction called the Event Storage Manager (ESM). Data can be stored in memory ESM or PostgreSQL ESM. It also provides corresponding interfaces to the framework developer, including the Initialization API for operating namespace, the Events API for operating on triples, the Thing Recommendations API for computing things, the Person Recommendations API for recommending users, and Compacting API for compressing items. If the framework developer completes these APIs according to the new storage model, it can also be integrated into the GER framework.

The process of GER's core recommendation module is as follows:

1. According to the entered user id, by checking their history, find similar people.
2. Calculate the similarity between the input user ID and a series of people.
3. Find a list of recent things done by the most similar person.
4. Use the similarity of people to calculate the weight of things.

2.5.3.4 Simple Python Recommendation System Engine

Simple Python Recommendation System Engine (Surprise) [35] is a recommendation system library in the scikit series. The main feature of Surprise is that it is easy to use and supports multiple recommendation algorithms.

- **Baseline algorithms**: Randomly give a predicted value according to the distribution characteristics of the training set.
- **Neighborhood methods**: K-nearest neighbour algorithm based on collaborative filtering.
- **Matrix factorization-based**: Algorithm based on Singular Value Decomposition.

The nearest neighbour-based method (collaborative filtering) supports different similarity metrics (cosine similarity, mean square difference similarity and Pearson correlation coefficient).

Surprise was designed to alleviate users' worries about data processing, so two processed data sets (Movielens, Jester) have been included in Surprise. Movielens is a public dataset of movie data maintained by members of GroupLens Research at the University of Minnesota. Jester is a data set maintained by AUTOLab, a member of UC Berkeley, which include 6.5 million anonymous user ratings for jokes. Users can also customize their own data sets. Users can also use various built-in prediction algorithms, such as neighbourhood methods, based on matrix factorization (SVD, PMF, SVD++, NMF). Various similarity measures (cosine, MSD, Pearson...) are also built-in.

2.5.3.5 LensKit

LensKit [44] is an open-source recommendation system based on java, produced by the GroupLens Research team of the University of Minnesota. But the java version of LensKit has been deprecated, and the latest version uses python. The python version of Lenskit is a set of tools for experimenting and researching recommendation systems. It provides support for training, running and evaluating recommendation systems.

The recommended algorithms in `LensKit` include SVD, Hierarchical Poisson Factorization, and KNN. `LensKit` can work with any data in `pandas.DataFrame` with the expected columns. `Lenskit` loads data through the `dataLoader` function. Each data set class or function takes a path parameter specifying the location of the data set. These data files have normalized column names to fit with LensKit's general conventions. They are UserID, Item ID, Rating and Timestamp.

2.5.3.6 A general graph-based framework for top-N recommendation using content, temporal and trust information

Armel et al. [60] proposed a model called GraFC2T2. It is a graph-based framework to combine and compare various kinds of information for recommendation. With GraFC2T2, users can calculate the results of using various information, and then find the appropriate combination for specific applications. The author also experiments on two data sets, Epinions and Ciao to illustrate the functions of GraFC2T2. GraFC2T2 uses content-based methods and adds the weight of user interests to reflect the changes in users' interests over time.

2.5.3.7 Deep Knowledge-Aware Network for News Recommendation (DKN)

DKN [77] proposes a model that integrates the embedded representation of knowledge graph entities with neural networks for news recommendation. News is characterized by highly condensed language and contains many knowledge entities. The previous

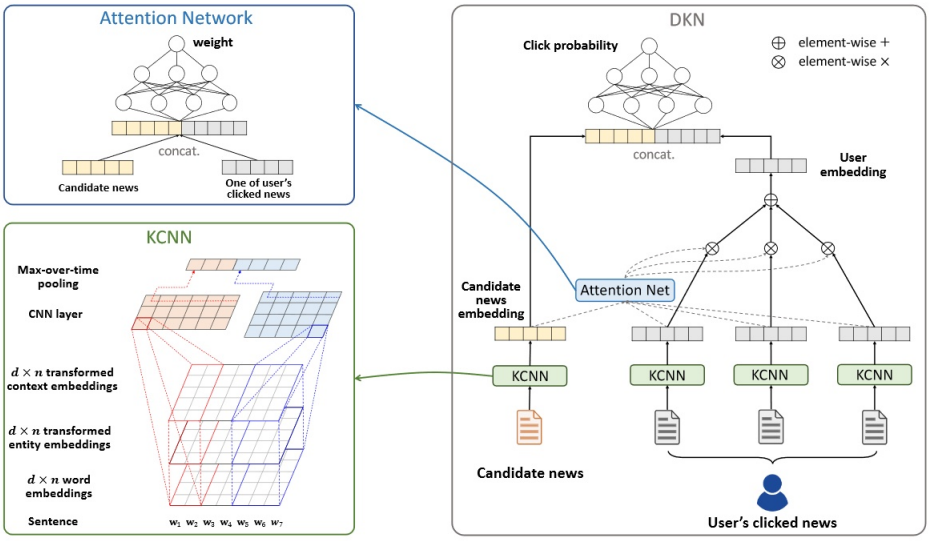

Figure 22: DKN architecture [77]

models did not consider the external knowledge contained in news, and only carried out representation learning from the semantic level, and did not fully explore the relationship of news at the knowledge level. News is timesensitive, and a good news recommendation algorithm should be able to make corresponding changes as users' interests change. To solve the above problems, the DKN model is proposed. First, a knowledge-aware convolutional neural network (KCNN) is used to integrate the semantic representation of news with the knowledge representation to form a new embedding, and then the attention from the user's news click history to the candidate news is established. The news with higher scores are recommended to users. The overall network architecture is shown in Figure 22.

2.5.3.8 Multi-task Feature Learning for Knowledge Graph enhanced Recommendation (MKR)

The traditional recommendation system only uses historical interaction information of users and items as input, which brings two problems: First, in actual scenarios, the interaction information of users and items are often very sparse. For example, a movie app may contain tens of thousands of movies, but a user-rated movie may only have dozens of movies on average. Using such a small amount of scoring data to predict

a large amount of unknown information will greatly increase the risk of overfitting of the algorithm; Second, for newly added users or items, because the system does not have its historical interaction information, it cannot do accurate modelling and recommendation, this situation is also called cold start problem [83, 64, 50, 92].

A common approach to solve the problem of sparsity and cold start is to introduce some side information as input in the recommendation algorithm. The auxiliary information can enrich the description of users and items and enhance the mining ability of the recommendation algorithm, thereby effectively making up for the sparseness or lack of interactive information. Commonly uxed auxiliary information includes:

- **Social networks**: A user is interested in an item, and his friends may also be interested in the item.
- **User/item attributes**: Users with the same attributes may be interested in the same type of items.
- **Multimedia information**: For example, images, video, audio, text, item pictures, movie trailers, music, news titles, etc.
- **Context**: The time, place, and current session information of the user-item interaction.

Among all kinds of side information, knowledge graph as a provider of side information has gradually attracted researchers' attention.

The triple (h, r, t) shown in the picture above Figure 23 expresses the fact that "John Lasseter wrote and directed toy story (1995)", where h=John Lasseter, t=toy story(1995), r=director, writer.

The knowledge graph has the potential to be applied in many recommendation scenarios, such as movies, news, restaurants, shopping, etc. Compared with other kinds of side information, the introduction of the knowledge graph can make the recommendation result have the following characteristics [61]:

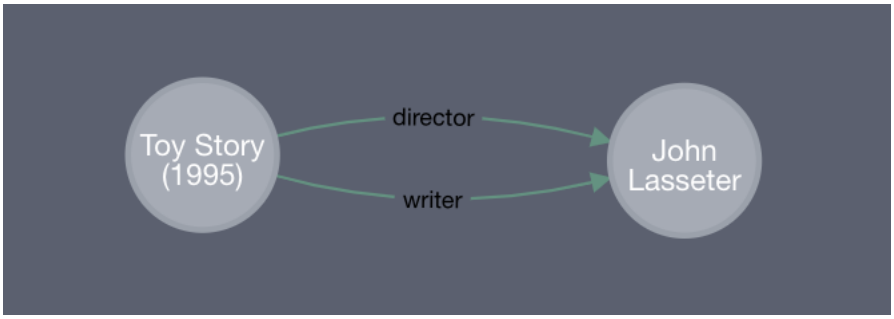

Figure 23: Toy story triples

- **Precision**: The knowledge graph introduces more semantic relationships for items, which can deeply discover the interests of users.
- **Diversity**: The knowledge graph provides different types of relationship connections, which is conducive to the divergence of recommendation results and avoids the limitation of recommendation results in a single category Figure 24.
- **Explainability**: The knowledge graph can connect the user's historical records and recommendation results, thereby improving the user's satisfaction with the recommendation results and enhancing the user's trust in the recommendation system Figure 24.
- **Context**: The time, place, and current session information of the user-item interaction.

A complete knowledge graph can provide deeper and longer-range associations between items, for example, "Iron man–Robert Downey Jr.–American–Will Smith–Men in Black 3." It is precisely because the knowledge graph has higher dimensions and richer semantic relationships, and its processing is therefore more complicated and difficult.

Knowledge Graph Embedding learns a low-dimensional vector for each entity and relationship in the knowledge graph while maintaining the original structure or semantic information in the graph. Because knowledge graphs contain unique semantic information, knowledge graph feature learning requires more careful and targeted model design than general network feature learning.

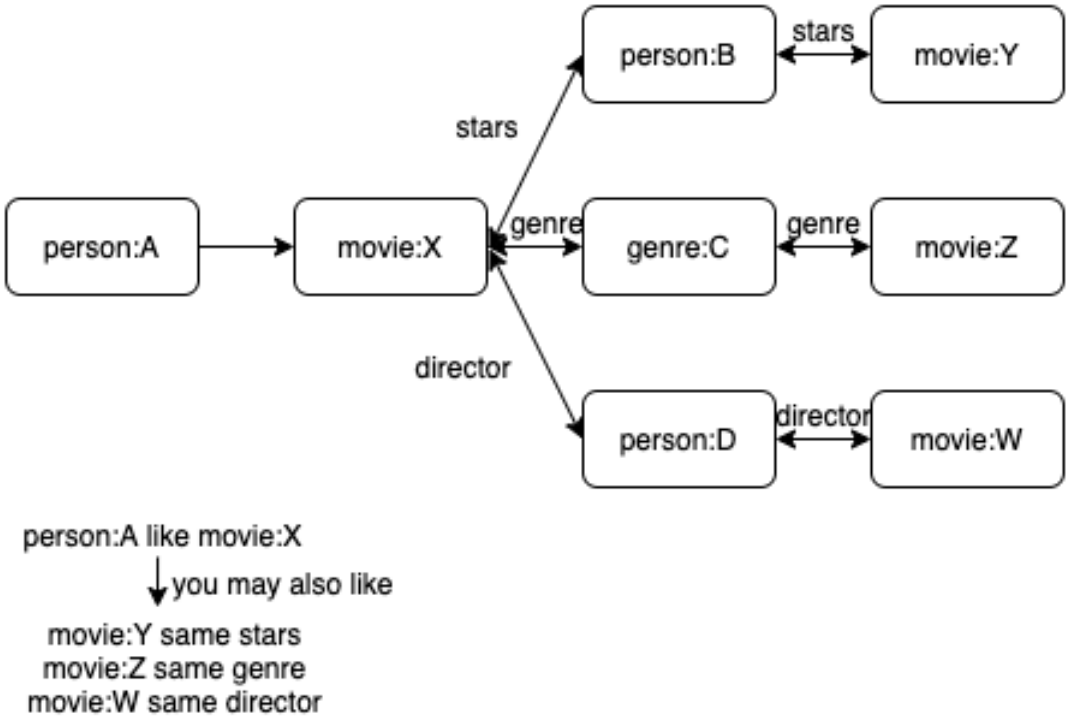

Figure 24: Recommended diversity

A good set of entity vectors can fully and completely represent the interrelationships. Most machine learning algorithms can easily handle low-dimensional vector inputs [34]. Therefore, use of knowledge graph feature learning facilitates introduction of the knowledge graph into various recommendation system algorithms. Knowledge graph feature learning:

- **Reduces the high dimensionality and heterogeneity of the knowledge graph.**
- **Enhances the flexibility of knowledge graph application.**
- **Reduces the workload of feature engineering.**
- **Reduces the extra computational burden caused by the introduction of a knowledge graph.**

Because the items in the recommendation system overlap with the entities in the

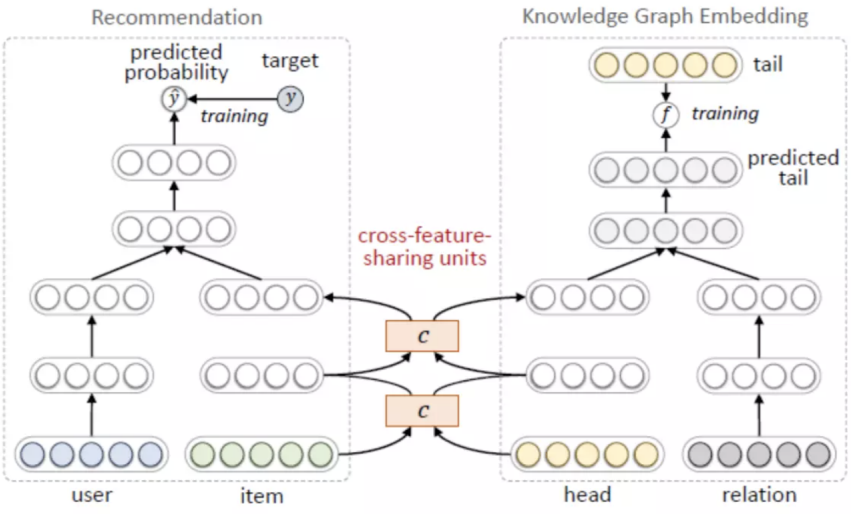

Figure 25: MKR structure [79]

knowledge graph, multi-task learning can be used to treat the recommendation system and knowledge graph feature learning as two separate but related tasks.

The structure of MKR is shown in the figure below Figure 25. The left is the recommendation system task, and the right is the knowledge graph feature learning task. The input to the recommendation part is the characteristic representation of the user and the item, and the output is the estimated value of the click rate. The knowledge graph feature learning part uses the head node and relationship of the triple as input, and the predicted tail node as output.

Further, cross-feature-sharing units can be designed as connecting links between two tasks of recommendation and knowledge graph feature learning.

- **Cross and Compress Unit:**

  The cross-feature sharing unit is a module that allows two tasks to exchange information. Since the item vector and the entity vector are two descriptions of the same object, the cross-sharing of information between them can allow both to obtain additional information from each other, thereby making up for the lack of information sparsity. Its structure is as follows Figure 26. Its structure is as follows:

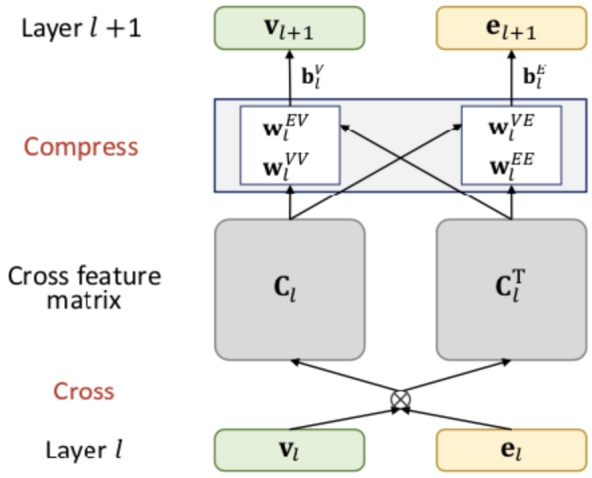

Figure 26: Cross feature sharing units structure [79]

– Cross operation: Construct a $d * d$ pairwise interaction of $v_l \in R^d$ and $e_l \in R^d$.

$$C_l = V_l e_l^T = \begin{bmatrix} v_l^1 e_l^1 & \dots & v_l^1 e_l^d \\ \dots & & \dots \\ v_l^d e_l^1 & \dots & v_l^d e_l^d \end{bmatrix} \tag{1}$$

– Compress operation: $w_l \in R^d$ and $b_l \in R^d$ are training weights and deviation vectors. The weight vectors compress the cross feature matrix from $R^d * d$ to R^d.

$$\begin{aligned} V_{l+1} &= C_l w_l^{VV} + C_l^T w_l^{EV} + b_l^V = v_l e_l^T w_l^{VV} + e_l v_l^T w_l^{EV} + b_l^V \\ e_{l+1} &= C_l w_l^{VE} + C_l^T w_l^{EE} + b_l^E = v_l e_l^T w_l^{VE} + e_l v_l^T w_l^{EE} + b_l^E \end{aligned} \tag{2}$$

– For simplicity, the cross compression unit is expressed as follows:

$$\left[v_{l+1}, e_{l+1} \right] = C_(v_l, e_l) \tag{3}$$

- **Recommendation Module:**

Vectors **u** and **v** represent user u and item v. **u** and **v** can be expressed based on application scenarios with one-hot, attributes, bag-of-words, or a combination

of them.

Given the original feature vector **u** of user u, use an L-layer multi-layer perceptron MLP to extract the user's potentially concentrated features.

$$u_L = M(M(M(...M(u)))) = M^L(u) \tag{4}$$

Where M is the fully connected neural network layer:

$$M(x) = \sigma(Wx + b) \tag{5}$$

The recommendation system module is a click-through rate estimation model: After obtaining the user feature vector and the item feature vector, the probability of user u participating in item v can be calculated by vector inner product:

$$\hat{y}_{uv} = \sigma(f_{RS}(u_L, v_L)) \tag{6}$$

- **KGE Module:**

  KGE maps entity and relationship to low-dimensional space while retaining their original spatial structure. For a given knowledge triple (h, r, t), cross-compression unit and multilayer perceptron are used to extract features from the original head h and relation r.

  The vectors corresponding to head and relationship are jointed together, and through a multi-layer neural network, a tail vector $\hat{t}$ is estimated.

  The tail vector predicted by the knowledge graph embedding module is expected to be similar to the real tail vector: The score of the last triple (h, r, t) is calculated by the similarity function f_{KG}. The f_{KG} function can be the inner product of t and $\hat{t}$ and then do the sigmoid transformation.

$$score(h, r, t) = f_{KG}(t, \hat{t}) \tag{7}$$

- **Learning Algorithm:**

  The complete loss function is as follows:

$$L = L_{RS} + L_{KG} + L_{REG} \tag{8}$$

  The first item is the cross-entropy loss of the recommender module. The second item is the loss of the KGE module, which aims to increase the score of correct triples and reduce the score of wrong triples. The third term is a regularization term to prevent overfitting.

  Although this framework uses knowledge graphs, their knowledge graphs are input in the form of txt documents. If the knowledge graph needs to be modified, a lot of manual work is required. If we integrate the building of the knowledge graph into the framework, it will save a lot of work in future management.

2.6 Software Available for Framework Development

To implement a general solution we need tools for (1) extracting information from the website (2) building knowledge graph (3) visualizing knowledge graph (4) building recommendation system. We survey below, currently available software that may be useful for our solution.

2.6.1 Beautiful Soup

Beautiful Soup is a Python library that can extract data from HTML or XML files. It can implement document navigation, search, and modify documents in the way you define. It automatically converts input documents to Unicode encoding and output documents to utf-8 encoding. Provides users with flexible analysis strategies and fast speed.

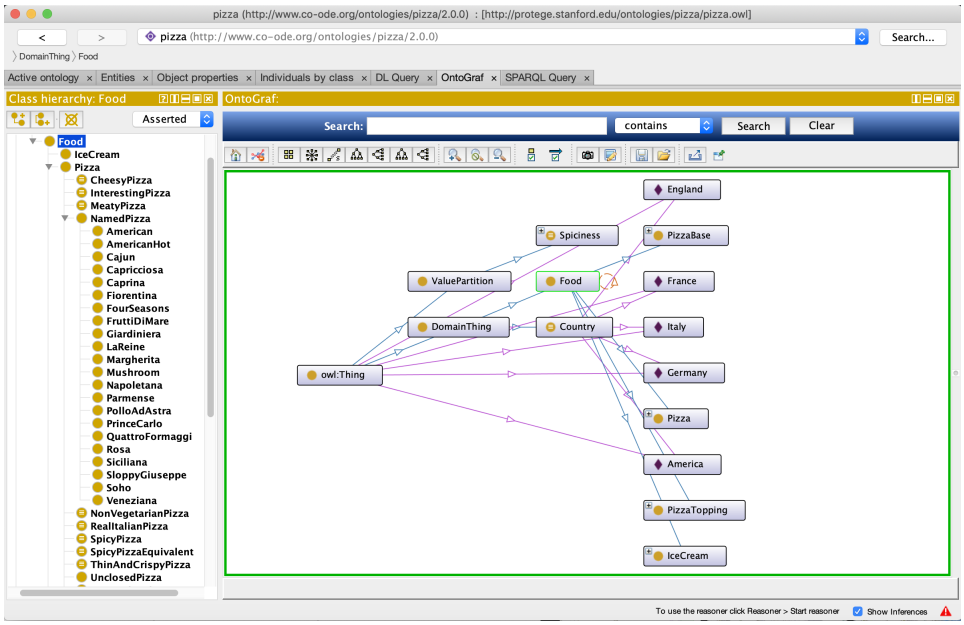

Figure 27: Protege example

2.6.2 Protege

As mentioned earlier, Protege is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies.

Its visual interface is shown below Figure 27.

Double-click “owl:Thing" and the ontology information will display in the right area. Among the graph displayed, double-click to expand, and you will see the relationship diagram of the entire ontology. Above the picture, the relationship diagram can be displayed in different ways, such as vertical direction display, horizontal direction display, etc. If you put the mouse on an information node, the detailed information of the information node will be displayed.

2.6.3 Owlready2

Generally speaking, the ontology requires us to build in an editor like Protege, but if it is manually added, the efficiency is very low. If you want to add it through the program, it needs some development interface. Jena is such a set of API developed

by HP. But Jena is based on java, so we explored and found `Owlready2`. `Owlready2` has been created at the LIMICS research lab, it is a module for ontology-oriented programming in Python. It can load OWL 2.0 ontologies as Python objects, modify them, save them, and perform reasoning via HermiT (included). `Owlready2` provides the following functionalities via its interface:

- Load the ontology from a local repository or from the Internet
- Create a new class in ontology
- Access ontology classes and create new individuals
- Export to RDF/XML file
- Perform reasoning and classify instances and classes

2.6.4 Neo4j

As described earlier, Neo4j is a high-performance NoSQL graph database.

The following is a Neo4j knowledge graph with "dangerous minds (1995)" as an example Figure 28.

2.6.5 Cypher

Neo4j uses `Cypher` to query graph data. `Cypher` is a descriptive graph query language with simple syntax and powerful functions. Neo4j is an absolute leader in the graph database family and has a large number of user bases, making `Cypher` very popular. `Cypher` is very similar to SQL. `Cypher` keywords are not case sensitive, but attribute values, labels, relationship types, and variables are case sensitive.

2.6.6 Py2neo

Py2neo is a client library and toolkit for working with Neo4j from within Python applications and from the command line. The library supports both Bolt and HTTP and provides a high level API, an OGM, admin tools, an interactive console. Unlike

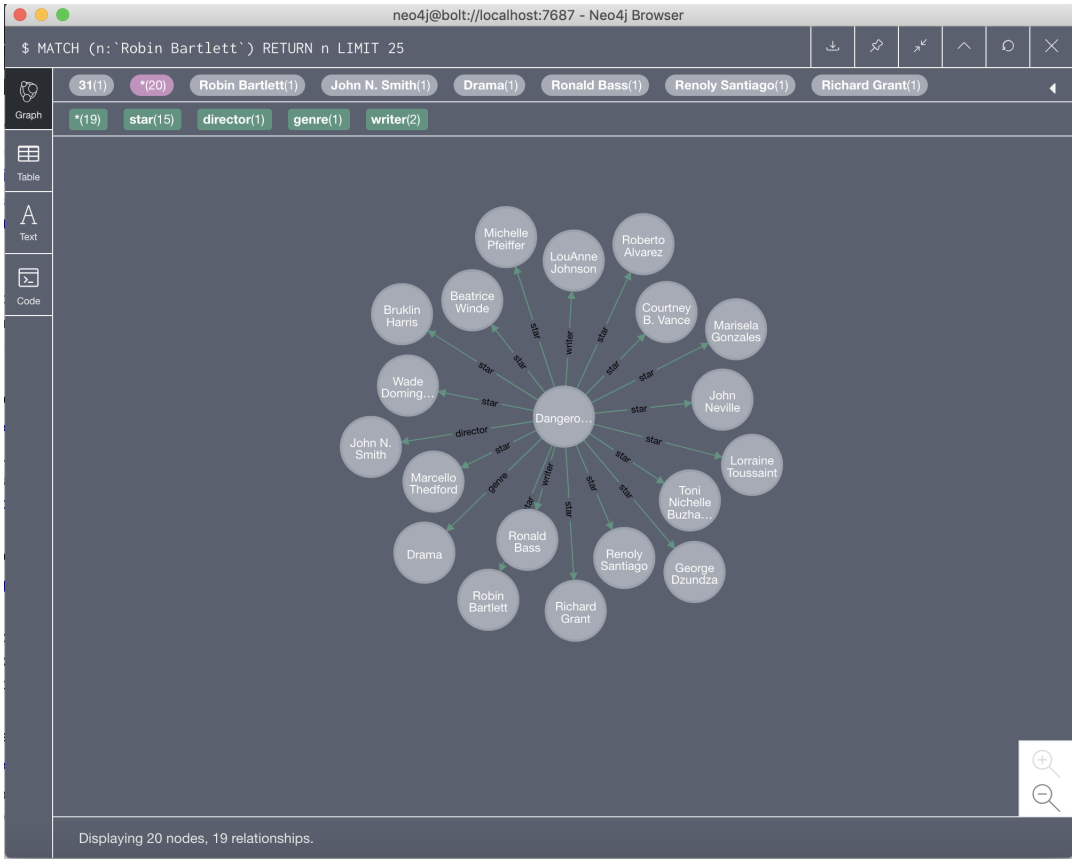

Figure 28: Neo4j example

previous releases, Py2neo v4 no longer requires an HTTP-enabled server and can work entirely through Bolt.

2.6.7 Tensorflow

`TensorFlow` is a popular open-source machine learning framework in the world [2]. It is fast, flexible, and suitable for large-scale product-level applications. It allows every developer and researcher to easily use artificial intelligence to solve diverse challenges. `TensorFlow` has rich applications in scenarios such as graphics classification, audio processing, recommendation systems, and natural language processing. `TensorFlow` supports various platforms such as Linux, Windows, Mac and even mobile devices. It also provides a very rich API for deep learning, including basic vector-matrix calculation, various optimization algorithms, the implementation of various basic

Name	Type	Storage types	Domain	Example uses	Language	ML Models	Maintained
Yue Ning et. al.	Decision Tree	-	content recommendation	content recommendation	-	Boosting	-
Racoon	Collaborative Filtering	Redis	cross-domain	$https://github.com/guymorita/benchmark_accoon_movielens$	Java	KNN	Jan 10, 2017
GER	Collaborative Filtering	PostgreSQL	movie	$https://github.com/grahamjenson/ger/tree/master/examples$	Java	-	Jul 9, 2015
Surprise	Collaborative Filtering	LocalFile	movie,joke	$https://github.com/NicolasHug/Surprise/tree/master/examples$	Python	matrix factorization, KNN	Aug 6, 2020
LensKit	Machine Learning	LocalFile	cross-domain	$https://github.com/lenskit/lkpy/tree/master/examples$	Python	SVD, hpf	Nov 10, 2020
GraFC2T2	Content-based	LocalFile	cross-domain	$https://github.com/nzekonarmel/GraFC2T2$	Python	-	Jan 18, 2019
DKN	knowledge graph enhanced	LocalFile	News	$https://github.com/hwwang55/DKN$	Python	tensorflow	Nov 22, 2019
MKR	knowledge graph enhanced	LocalFile	cross-domain	$https://github.com/hwwang55/MKR$	Python	tensorflow	Nov 22, 2019
PredictionIO	machine learning	Hadoop,HBase	-	$https://github.com/apache/predictionio$	Scala	Apache Spark MLlib	Mar 11, 2019

Table 1: Characteristics of Existing Frameworks

units of convolutional neural networks and recurrent neural networks, and visual aids, etc. `TensorFlow` has the following characteristics:

- **High flexibility**: Users can write their own superstructure and library in python based on `TensorFlow`. If `TensorFlow` does not provide the API we need, we can also write the underlying C++ code ourselves and add the newly written functions to `TensorFlow` through custom operations.
- **Portability**: `TensorFlow` can run on CPU and GPU, can run on a desktop, server, mobile devices.
- **Support multiple programming languages**: `TensorFlow` uses python to build and execute our logic, and it also supports C++, java, GO and other programming languages.
- **Complete documentation**: The official website of `TensorFlow` provides very detailed documentation, including the use of various APIs and examples of basic applications, as well as some basic theories of deep learning.

2.7 Summary of Recommendation System Literature Review

A comprehensive review of the literature in Section 2.5 shows that there are three types of recommender system models (collaborative filtering, content based, and machine learning based). Different types of models have different advantages. Table 1 is a tabulated summary of the frameworks mentioned above.

In the current literature review, we found that the model proposed by Yue et al. [59] mainly uses the boosting model. Boosting can prevent the decision tree from overfitting, and the trained model is also easy to explain. But the computational complexity of boosting is high, and cannot be parallelized. So the training will consume a lot of time.

Raccoon(Section 2.5.3.2) is a more comprehensive collaborative filtering based recommendation framework. Raccoon uses the K-nearest-neighbours model. It cannot handle new users or new items, well known as the cold start problem. Further, it also has data sparseness, and scalability issues.

The advantage of GER is that it contains no business rules, limited configuration, and almost no setup required. It does not require a complicated configuration to use and is easy to understand. But at the expense of the scalability of the engine. Besides, GER also has some other limitations. The first limitation is that it does not generate recommendations for a person with less history. For example, if a person has only liked one movie, their generated recommendations will probably be random. In this case GER return no recommendations and lets the client handle this situation [36]. Another is data set compression limit, i.e., certain items will never be used. For example, if items are older or belong to users with a long history, these items will not be used in any calculations, but will occupy space.

Surprise in Section 2.5.3.4 is a library in the scikit series. It supports a variety of recommendation algorithms and contains many similarity metrics and evaluation criteria. It is the preferred way to implement the recommendation algorithm using the python language. However, Surprise also has some drawbacks. It cannot recognize other languages, for example Chinese. When using Surprise to read local data, the local CSV file cannot have headers, and columns with Chinese in the headers and metadata need to be removed. Framework developer needs to provide a new Reader class.

The advantage of `LensKit` is that its framework contains many algorithms, such as funksvd and KNN, and users can call different algorithms according to their needs. In `LensKit` the data is called through the path parameter, and the data has a specific

format, This means that the LensKit framework can only process user, item and rating data. If the user has new data, such as user information or item classification, the LensKit framework cannot handle it.

The advantage of GraFC2T2 is that it refers to the weight of the user's interest so that when generating recommendations, it will focus more on the user's recent interests, but the disadvantage is that it uses text files as input, which costs more time when modifying or managing user's log file.

The advantage of DKN in Section 2.5.3.7 is that it uses knowledge graph embedding as an aid. However, because of the design of the model, DKN can only be used for news recommendations.

MKR Section 2.5.3.8 is a general recommendation framework that aims to use Knowledge Graph Embedding (KGE) to assist recommendation tasks. The knowledge graph embedding and recommendation system are independent of each other, but they are highly related because the item in RS and the entity in KG are related to each other. The entire framework can be trained by alternately optimizing the two tasks, giving the user a high degree of flexibility and adaptability in real recommendation scenarios. However, its main drawback is that it inputs text documents as knowledge graphs, if users want to change some of the nodes, they need to manually change the text files. If the amount of data is small, manual operations can be performed. If the amount of data is large, manual operations are inefficient.

2.8 Summary

In this chapter, we have conducted an extensive review of current methods in recommendation systems, the representation of knowledge graphs and some existing recommendation system frameworks. For the storage form of the knowledge graph, we discussed two forms, OWL and Neo4j. OWL is the RDF representation with the best descriptive ability, and Neo4j is the most popular graph storage. OWL can provide the most accurate description for triples, while Neo4j performs very well in terms of speed. For the recommendation system methods, we introduced various

models and explained their work flows, and pointed out the need and importance of using side information. Through our review of previous research, we found that most of the works that use knowledge for providing side information do not integrate the construction of knowledge graphs into the framework, and most of them use open source CSV files or txt files as input. This, as discussed earlier is a major drawback. In the last part of this chapter, we listed all available software that will be used by us in the development of our framework.

Chapter 3

Framework Design

In this chapter, we will first provide our motivation for why we decided to use a framework solution. Next, we will introduce the architecture of our framework. Lastly, we detail each component for the core and specialized frameworks.

3.1 Introduction to Software Frameworks

A framework is an abstraction in which the software provides generic functionality, but can be selectively changed by additional user-written code, thus providing application-specific software [90]. Compared with non-framework-based approaches, frameworks promise higher productivity and shorter time of application development. The software framework aims to facilitate software developments by allowing designers and programmers to devote their time to meeting software requirements rather than dealing with the more standard low-level details of providing a working system, thereby reducing overall development time [27]. In other words:

1. The framework itself is generally not complete enough to solve specific problems.
2. The framework is designed for specialization and expansion.
3. The framework provides many auxiliary, supportive and easy-to-use components for subsequent specialization and expansion.

Unlike in libraries or standard user applications, the overall flow of control in the program is not dictated by the caller, but by the framework.

It fulfills several purposes:

1. Change is required with time in every software application. So the framework helps out in that a changed scenario is implemented without affecting the other areas of the application.

2. Handles the routine tasks that every application requires to ease the life of the developers.

3. Handles all the validations which are required in the application so that the programmer can focus mainly on the application logic.

4. Reduces the development time and maintenance of the application.

So the framework is important for every application. There is certainly need for top-level design, and for the functional and non-functional requirements of the software framework to be allocated to the software components in a unified and reasonable manner.

While there is a lot of work on specific recommendation methods and systems in the literature, as we have seen in the last chapter, there is not much work on generic framework solutions for recommendation systems. In particular, with the maturing of the technology of knowledge graphs to represent knowledge, the increasing use of machine learning to enhance recommendation methods, and the importance of providing side information to these methods for improved performance, there is a need for a new framework that accommodates specialization and expansion with such new developments.

3.2 Why Framework Solution?

According to [63, 41], a software framework consists of two kinds of components:

- Frozen spots: Define the overall architecture of a software system, that is to say, the relationship between the basic components. These remain unchanged in any instantiation to derive a specialized framework.

- Hot spots: The part that requires programmers to add their code to meet specific functions for specialized frameworks.

The framework has three key differences from normal libraries:

Inversion of control: In our case, a major requirement is to enable the use of knowledge (side information) for enhancing the recommendation system. It can be divided into two sub-problems: the knowledge graph for representation of all information, and its availability to the recommendation system, i.e., the knowledge graph has to provide knowledge inputs to the recommendation system. From the end user's point of view, they do not need to care about processing in the middle. The flow of control from input to recommendations is completed by the framework. The user need only care about the input.

Extensibility: Developers can expand the framework with just a few simple operations according to user needs. As mentioned later in Section 1.3.2, extensibility is one of our non-functional requirements. Since we want to support various kinds of storage modes, (specified by FR1 and FR2) and as discussed in Chapter 2 the algorithms for both knowledge graph storage and recommendation systems are diverse. With an extensible design, we save on such integration work.

Non-modifiable framework code: When allowing users to implement extensions, users can extend the framework but cannot modify the framework. The hot spots of the framework allow developers to implement specialized functions. The frozen spot of the framework is used to control the flow of the program. This structure allows developers to add hot spots to the existing framework. It improves the reliability of our solutions and reduces programming and testing efforts.

3.3 Core Framework Design

Figure 29 shows the overall design of our framework solution. It is designed as a pipeline of tasks from end user input to final recommendations to the end user. Further, each stage in the pipeline is designed as a sub-framework, some stages are nested, enabling specialization and expansion at a more granular level. The data flow of the framework is shown in Figure 30. The four major stages in the pipeline have the following functionality:

- InfoExtractor framework: It provides the abstraction of a web extractor, it takes a URL address (such as `.html`, `.htm`, `.asp`, `.aspx`, `.php`, `.jsp`, `.jspx`) and extraction rules as input then formats the output that is extracted. This is further nested, to enable domain level (movies, news, etc.) specialization.
- StorageManager framework: It provides the abstraction of various kinds of storage. It takes string data stream as input then generates the output file according to the required format.
- Knowledge graph viewer framework: It provides the abstraction of the knowledge graph viewer. It takes triples stream as input then creates the visualization.
- Recommendation method: It provides the abstraction of knowledge input to the recommendation method. The recommendation method can take these knowledge graph triples as input and generate recommendations.

3.3.1 InfoExtractor

In Section 1.3, we explained the need for an information extractor. Therefore, it is necessary to abstract the common methods of information extractors. We will take the example of extracting movie information, we have designed a specialization module called `MovieExtractor` nested in `InfoExtractor`. `MovieExtractor` serves as a frozen spot to provide users with functions such as capturing movie information. To

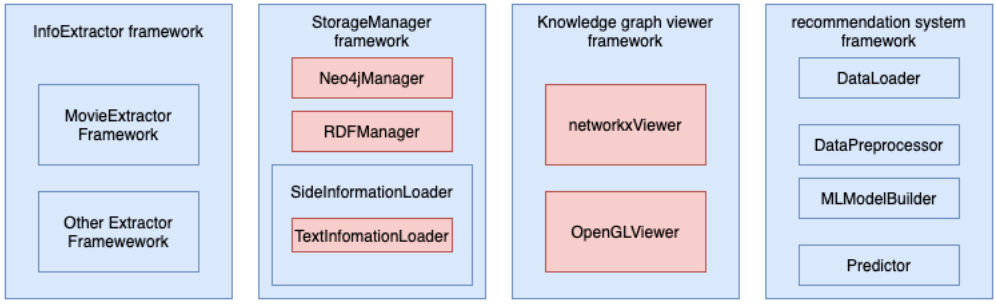

Figure 29: Core components of the framework, each box in blue means the framework's frozen spot and each box in red represents a set of the hot spots for each specialized framework.

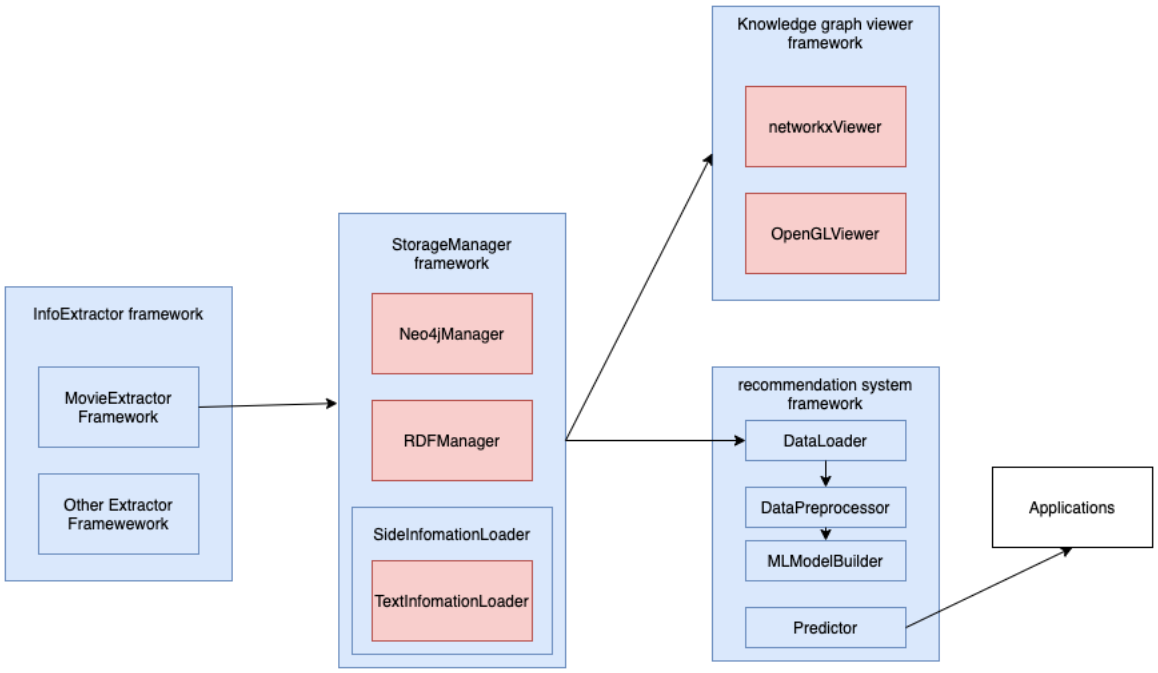

Figure 30: Data flow of the framework.

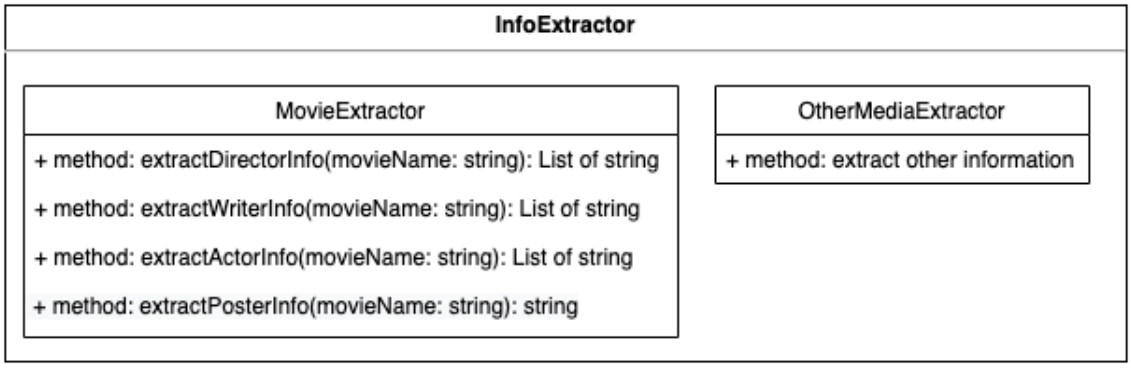

Figure 31: Design of the InfoExtractor Framework.

achieve this goal, we designed the MovieExtractor framework for movie information, as shown in Figure 31.

`MovieExtractor` is the abstract class of the extractor, it has three basic methods, `extractDirectorInfo`,`extractWriterInfo` and `extractActorInfo`. It is used to extract director information, author information and star information respectively. `extractPosterInfo` is the abstract class of the poster downloader. It is used to download the movie poster from website. Then converts the image into a string to facilitate the storage and coding of the knowledge graph and recommendation system. `extractDirectorInfo`,`extractWriterInfo` and `extractActorInfo` return a list, because there may be many directors, actors or authors of a movie.

Since it is a core module for movie recommendation, we are going to explain some of these important abstract methods defined in the `MovieExtractor` class:

1. `extractDirectorInfo:` This method is to extract the director information of the movie; the input to this method is movie name of type string and it returns a list.

2. `extractWriterInfo:` This method is to extract the writer information of the movie; the input to this method is movie name of type string and it returns a list.

3. `extractActorInfo:` This method is to extract the star information of the movie; the input to this method is movie name and it returns a list.

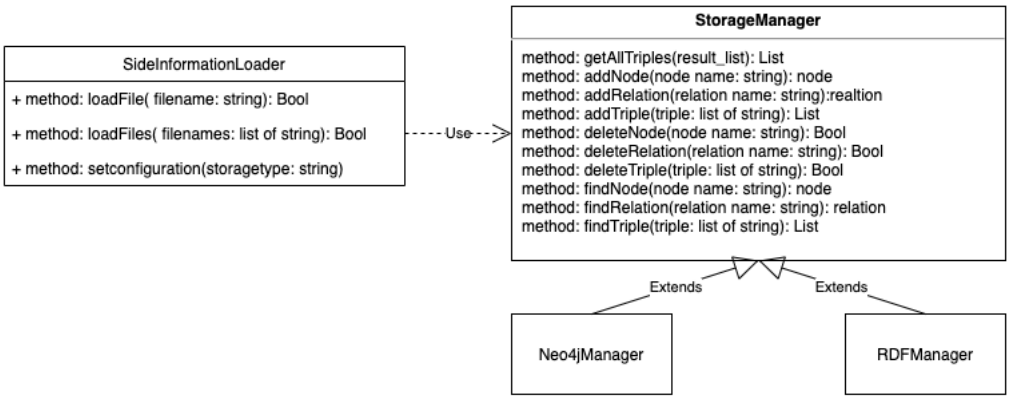

Figure 32: Design of the knowledge graph StorageManager framework.

4. `extractPosterInfo`: This method is to extract the poster image and then convert the image to string; the input to this method is movie name of type string and it returns a string.

`MovieExtractor` is used as the frozen spot in the framework. There are many websites on the Internet that store information about movies, such as IMDB, Wikipedia, Netflix and Douban. The extractors from these websites can be used as hotspots to access our `MovieExtractor` API.

In another extractor framework, we have a hot spot called `BookExtractor`, which is used to extract book information.

3.3.2 Knowledge Graph StorageManager Framework Design

As mentioned in Section 1.3, knowledge graphs enhance recommendation systems. So it is necessary to design a special framework for this. For flexibility and scalability, we need to extract the common parts of different types of knowledge graph storage, and then provide their abstractions.

Out of this consideration, `StorageManager` Framework serves as a frozen spot. We designed a dedicated frame as shown in Figure 32. Also, the design should allow us to add support to more kinds of storage modes easily without changing the frozen spots themselves. With these requirements in mind, our design makes it easier for framework developers to add their hot spot to our frozen spot. If users have a new

storage method to store their knowledge graph, they can provide a backend of their storage method, and in `StorageManager` they only need to change one line of code to call their storage module.

The following is a detailed explanation of some important methods.

1. `getAllTriples:` This method is to extract all the triples; the input of this method is an empty list and it returns all triples as a list of string.

2. `addNode:` This method is to check whether the node exists, and if it does not exist, generate a new node. Because all data in open source data exists in the form of triples, but if users use their data, there may be separate nodes. In this case, you need to use the addNode method. The input of this method is node name of type string, and it returns node.

3. `addRelation:` This method is to check whether the relation exists, and if it does not exist, generate a new relation. The input of this method is relation name of type string, and it returns relation.

4. `addTriple:` When we create the graph, or when we do knowledge graph fusion, we need to use this method. This method is to check whether this triple or node exists, and if it does not exist, generate a new triple. Its input is triple, a list composed of strings, and it returns this triple.

5. `deleteNode:` This method is used to delete node. When the user creates an error or needs to modify the graph, they need to call this method. Its input is the node name of the typed string and then it returns a true or false value.

6. `deleteRelation:` This method is used to delete relation. When the user creates an error or needs to modify the relationship of the triplet, they need to call this method. Its input is the name of the relationship and then it returns a true or false value.

7. `deleteTriple:` This method is used to delete triple. When the user creates a triplet by mistake, s/he needs to delete the triplet. They need to call this

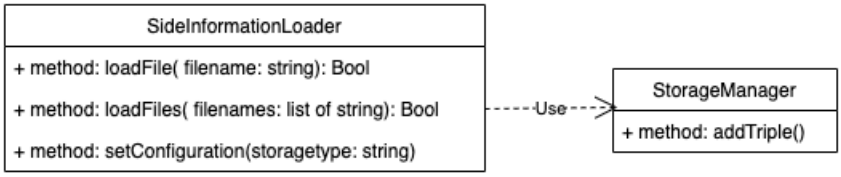

Figure 33: Design of the side information loader framework.

method. Its input is the triple of type string and then it returns a true or false value.

8. `findNode:` This method is used to check whether the node exists, and return to this node if it exists. When a user needs to create a node or modify a node, this method is needed. The node needs to be found first, and then the node is operated. The input of this method is the node name of string type, if there is this node, it will return the node. If not, it will return None.

9. `findRelation:` This method is used to check whether the relation exists. When users need to modify the relation, they need to use this method. We need to find the relation first and then operate on it. The input of this method is the relation name of the string type. If there is this relation, it returns the node. If not, it returns None.

10. `findTriple:` This method is used to check whether the triple exists. When the user needs to search for triples, we use this method. Its input is a list of triples composed of strings. The output is the triple or none.

For the `SideInformationLoader`, We design it as a frozen spot, and its responsibility is to add triples to the knowledge graph through the method in `StorageManager` to increase the richness of the knowledge graph. As shown in Figure 33, `SideInfomationLoader` is an abstract class. It contains three methods. The following is a detailed explanation of some important methods.

1. `loadFile:` This method is used to add a triplet in a single text file by reading the triplet of the year in the text file and then calling the API in the

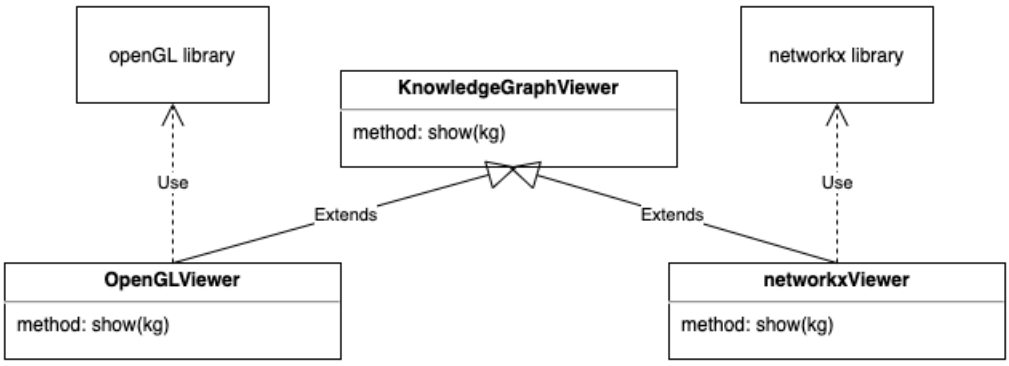

Figure 34: Design of the knowledge graph viewer framework.

`StorageManager` to add it to the knowledge graph. The input of this method is the file name of the string class, the output is True and False.

2. `loadFiles:` This method is used to add triples in multiple text files by reading each file name and then calling the `loadFile` mentioned above. The input of this method is a list of file names, and the output is also True or False.

3. `setConfiguration:` This method is used to set the configuration. The input is a string, which is used to input different storage modes, such as Neo4j or RDF.

3.3.3 Knowledge Graph Viewer Framework Design

We design the viewer module as a frozen spot, and its responsibility is to display triples for visualization. The design of the viewer module is simple. As shown in Figure 34, `KnowledgeGraphViewer` is an abstract class, which contains a show method for displaying a visual interface.

3.3.4 Recommendation Method Framework Design

As mentioned in Section 1.3, our focus in this work is on using a knowledge graph enhanced recommendation method. It can be divided into two parts, first part is the knowledge graph, already explained in the previous section. The other part is recommendation method. We need to extract the triples in the knowledge graph

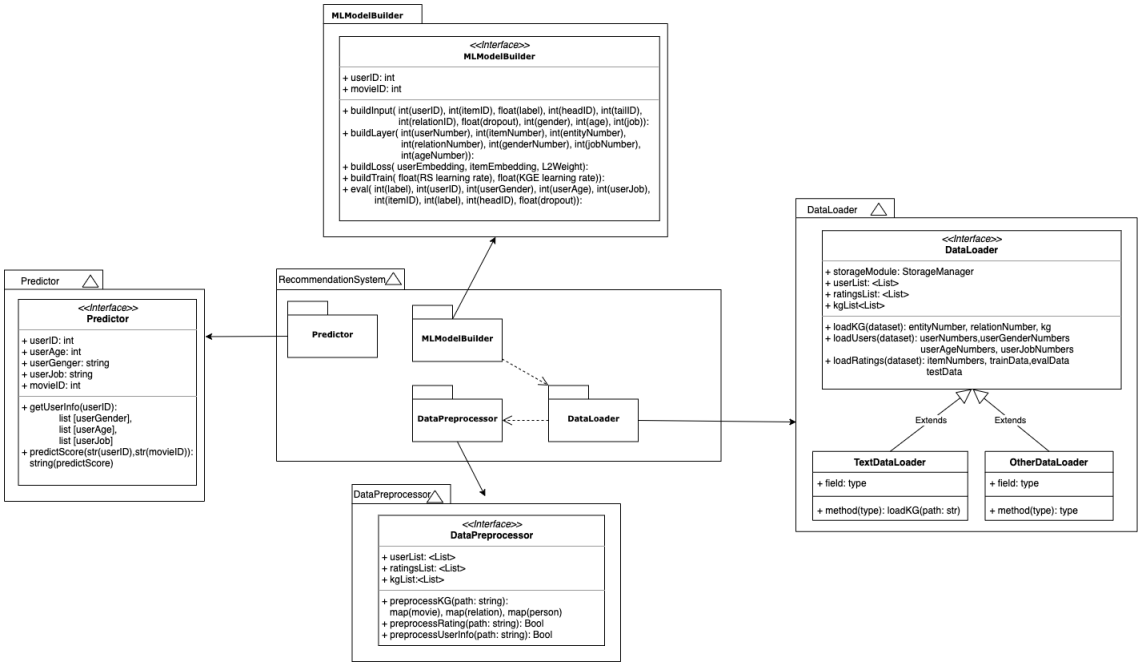

Figure 35: Design of the recommendation system framework.

first. Then, for the machine learning enhanced recommendation method, we treat the triples as training sets.

We designed a dedicated framework for the recommendation method as shown in Figure 35.

It consists of four frozen spots. Including `DataLoader` module, `DataPreprocessor` module, `MLModelBuilder` module and `Predictor` module. Since we need to preprocess the triples in the knowledge graph. The first step is `DataPreprocessor` module. In this step, the name of the string category in the triplet is represented by a unique int number, to facilitate the training of the recommendation system. The `DataPreprocessor` module includes three methods, which are `preprocessKG`, `preprocessUserInfo` and `preprocessRating`. They are used to process different data. The following is a detailed explanation of `DataPreprocessor` class.

1. `preprocessKG`: This method is to preprocess all the triples related to movie. It returns three dictionaries, which are used to store the id corresponding to the movie name, the id corresponding to the relation, and the id corresponding to

the person.

2. `preprocessUserInfo:` This method is to encode user information, including the user's gender, age, and occupation. It encodes the user information and stores it into a text file. Convenient for framework developers to debug.

3. `preprocessRating:` Re-encode the rating file through user and movie encoding. It uses two of the three dictionaries stored in the previous step to convert the movie name and user id in the rating information through the dictionary and then stores them in a text file.

The second step is the `DataLoader` module. The function of the DataLoader module is to read all triples from the file generated in the previous step, including the triples of the movie, the user's information, and the user's rating of the movie. And return three lists, which are used to store the above information. The following is a detailed explanation of `DataLoader` class.

1. `loadKG:` This method is used to load the KG file processed in the previous step, calculate the number of entities, the number of relations, and return these values. These parameters are used to create the matrix of entities and relations when building the neural network in the next step.

2. `loadUsers:` Load the user information file processed in the previous step, calculate the number of users, genders, ages and jobs. These parameters are used to create a user information matrix when building the neural network in the next step.

3. `loadRatings:` It is used to load the rating file, then calculate the number of items, and then divide the data into the training data, eval data and test data according to the ratio of 6:2:2. Finally, return the number of items and the divided data sets.

The last step is to build a machine learning model and get a pre-trained model through training using the data processed by `DataPreprocessor`. Among them

are `buildInput`, `buildLayer`, `buildLoss`, `buildTrain`, and `eval` methods. The following is a detailed explanation of `MLModelBuilder` class.

1. `buildInput:` This method is used to define the input data of the model. They are userID, itemID, label, headID, tailID, relationID, dropout, gender, age, job. Through the above various ids, we can quickly obtain the vector of the user or item, and quickly find the corresponding node from the matrix through the id of the head, relation and tail.

2. `buildLayer:` This method is used to define the dimensions of the model and the structure of the model. We use user number, item number, entity number, relation number, gender number, job number, age number to construct a matrix of the corresponding size. Each row represents one attribute. When we search for a vector by id, we can quickly find the corresponding person or item.

3. `buildLoss:` This method is used to define the loss function of the model. We calculate the loss of the recommendation system, the loss of the knowledge graph embedding and the L2 loss as the final loss.

4. `buildTrain:` This method is used to define the training steps.

5. `eval:` In this method, we calculate AUC and ACC through the predicted result and the actual label.

After the user trains the model, s/he will get a trained network. This network model will be used to predict the user's score in the later stage. `Predictor` framework is to facilitate user. It includes `getUserInfo` and `predictScore` two methods: The following is a detailed explanation of `Predictor` class.

1. `getUserInfo:` This method is used to extract the user's information. If it is an old user, the user only needs to provide the user ID to query the user's personal information, but if it is a new user, the user needs to enter information such as age, gender, and job type. Finally, three lists are returned to save the user's gender, age, and job information.

2. `predictScore:` Predict the user's rating of the movie based on the user's id and the movie's id. Finally, a float value is returned, which represents the user's rating of the movie.

3.4 Summary

In this chapter, we introduced the design of our framework. We start by explaining why we chose the framework design solution, and then we describe the structure of the framework. In our design, we divide the entire solution into four parts, each with a specific function. Then we explained the design and function of each module in each framework. In the next chapter, we will describe how to build concrete applications with the framework. In a later chapter, we will show how this design and its application fulfil the functional and non-functional requirements derived in Section 1.3.

Chapter 4

Framework Instantiation

In this chapter, we will show examples of how to create hot spots for those frozen spots introduced in Section 3.3. An instantiated architecture diagram is shown in Figure 36.

4.1 IMDBExtractor Instantiation

In Section 3.3.1, we described the design of the `InfoExtractor` module within the framework. We extract director, writer, stars information, movie genre and movie poster as side information. Figure 37 illustrates the workflow of the `IMDBExtractor` module. For each kind of information to be extracted, we create a list for them. Because there may be many directors, actors, and stars of the movie. The information for each category is returned as a list. If the relevant information cannot be found in IMDB, an empty list will be returned. Each triplet will be stored in the text file in

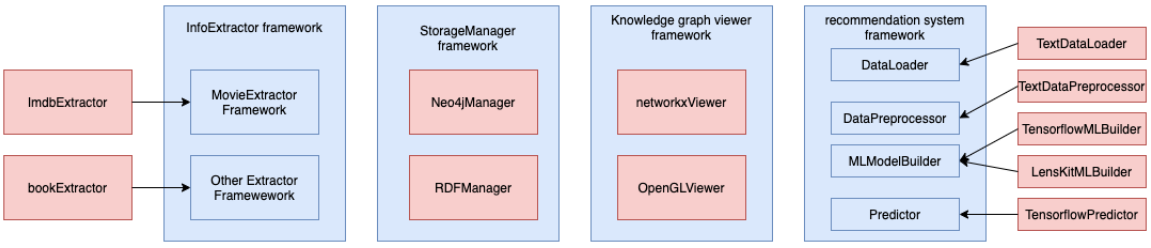

Figure 36: Implemented framework instance

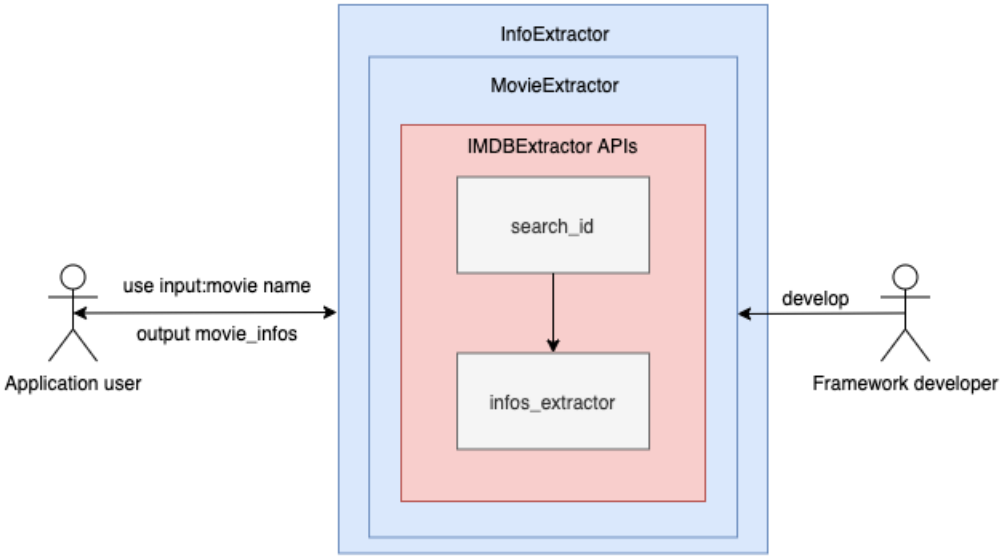

Figure 37: `IMDBExtractor` workflow.

```
Toy Story (1995)    directors   John Lasseter
Toy Story (1995)    writers John Lasseter
Toy Story (1995)    writers Pete Docter
Toy Story (1995)    stars   Tom Hanks
Toy Story (1995)    stars   Tim Allen
Toy Story (1995)    stars   Don Rickles
Jumanji (1995)  directors   Joe Johnston
Jumanji (1995)  writers Jonathan Hensleigh
Jumanji (1995)  writers Greg Taylor
Jumanji (1995)  stars   Robin Williams
```

Figure 38: An example of text format.

the form of head relation tail. Figure 38 gives an example. We can use this function by calling the `extractDirectorInfo`, `extractWriterInfo` and `extractActorInfo` methods. If framework developers or application developers want to add a new movie website extractor, they need to create a new class under `MovieExtractor`, and then add a line of declaration for the new extractor in `MovieExtractor`.

In the other extractor framework, we have a specialization `BookExtractor` as a hot spot. We have used this module only as a test to verify the scalability of the framework, but without actual implementation. At present, it just returns a triple of

open-source data.

4.2 Knowledge Graph StorageManager Instantiation

In Section 3.3.2, we proposed the design of frozen spot for a knowledge graph StorageManager framework in general. To achieve this requirement, we created two storage modules as hot spots for knowledge graphs. As we mentioned earlier in Section 2.3.2 and Section 2.3.1. They are Neo4jManager and RDFManager. Figure 39 shows the structure of this module. For framework developers, when a new storage module needs to be added, they do not need to know the working of the middle but need only to add a new storage mode to the module and reserve the corresponding interface.

In our implementation, since there are already some open-source data, which includes movie names and user ratings of movies, we did not start with extracting movie names. Instead based on the movie name we proceed to extract movie genres, directors, writers and stars information. We control different storage formats and data through command-line options. The following command line is an example of using Neo4j storage mode and using all data. `python3 StorageManager.py -mode neo4j -dataset all`. If the application developer wants to use their data, all files should be saved in the movie folder, otherwise the program will not recognize the file path.

The workflow of the storage module is shown in Figure 40. The complete process is described by the following steps:

1. Choose a storage mode through the command line option. There are two currently, one is Neo4j storage mode, the other is RDF storage mode.

2. Choose different side information through the command line option. There are three kinds of side information in our framework instance for movie recommendations. One is the file containing director, writer and star

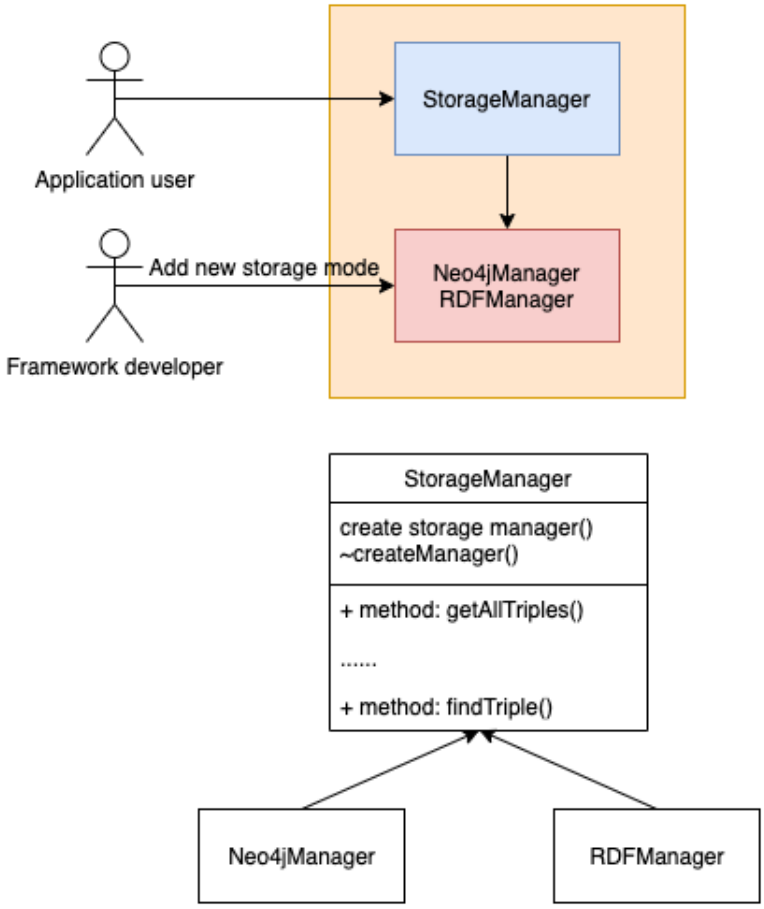

Figure 39: Instantiation of the StorageManager in framework.

information, and the other is user information (including age, gender and job category), movie genre, and the last one is the movie poster.

4.2.1 RDFManager

To facilitate users to use the RDFManager framework, we provide our framework's API for all operations of owl, including:

1. `RDF_Get_Ontology`: Load the ontology from a local repository or the internet.
2. `RDF_Add_Class`: Create a new class in ontology.
3. `RDF_Add_Individual`: Access ontology class, and create new instances or individuals.

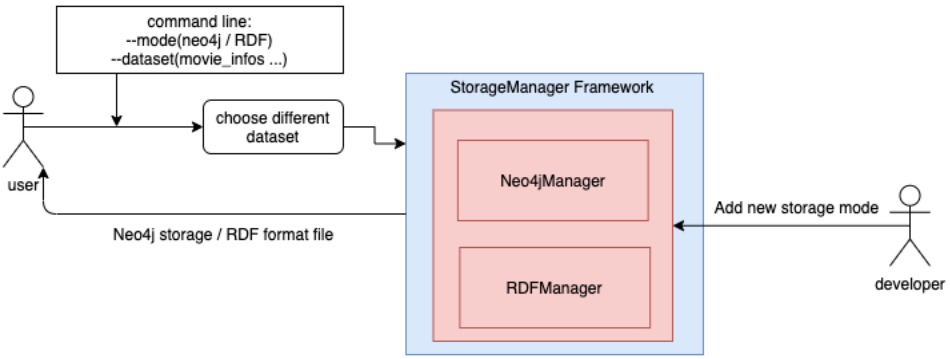

Figure 40: Workflow of storage framework

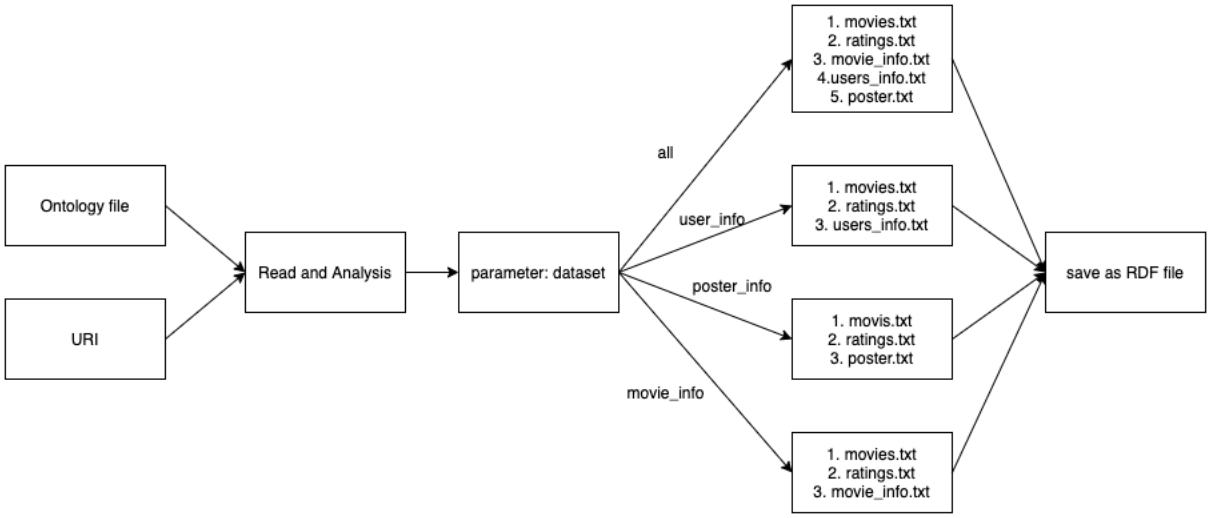

Figure 41: Structure of the RDF storage module in framework.

4. `RDF_Add_Dataproperty`: Access ontology, and create new dataproperty.

5. `RDF_Add_Dataproperty_Value`: Access ontology individual, and add new value for the dataproperty.

6. `RDF_Add_Objectproperty`: Access ontology classes, and add new objectproperty between two classes.

7. `RDF_Save`: Export to RDF or XML file.

Figure 41 illustrates the structure of RDF storage. When the user chooses to use RDF storage, it will call the RDFManager in the framework, use the API in our framework to operate on the triples, and then save it as an RDF file.

4.2.2 Neo4jManager

To facilitate users to use the Neo4jManager framework, we provide a framework API for Neo4j operations. Because most of the data is in txt format, we provide storage from txt format to Neo4j, also include API from a single triple to Neo4j. Figure 42 illustrates the structure of Neo4j storage.

Our process can be described by the following steps:

1. Start the Neo4j server, then enter the account and password.
2. Call the corresponding API, read the file according to the file address, and get the id, name, genre and other information of the movie.
3. Create a node based on the id of the movie.
4. The other information about the movie needs to judge first whether the node exists according to the name of the node. This operation is to avoid creating the same node repeatedly.
5. If the node exists, connect the two nodes directly.
6. If the node does not exist, we need to create a new node.
7. According to the content of the file, we define the relationship between the two nodes. If it is a director, we name the connection line as a director relationship. If the relationship is a user's rating of the movie, we define the relationship as a rating.

4.2.3 TextInfomationLoader

In Section 3.3.2, we described the design of the `SideInformationLoader`. To add side information to meet our requirements, based on the frozen spot of `SideInformationLoader`, we created a `TextInformationLoader` hot spot based on text documents. The overall idea is shown in Figure 43. The format of the file is the same as Figure 38.

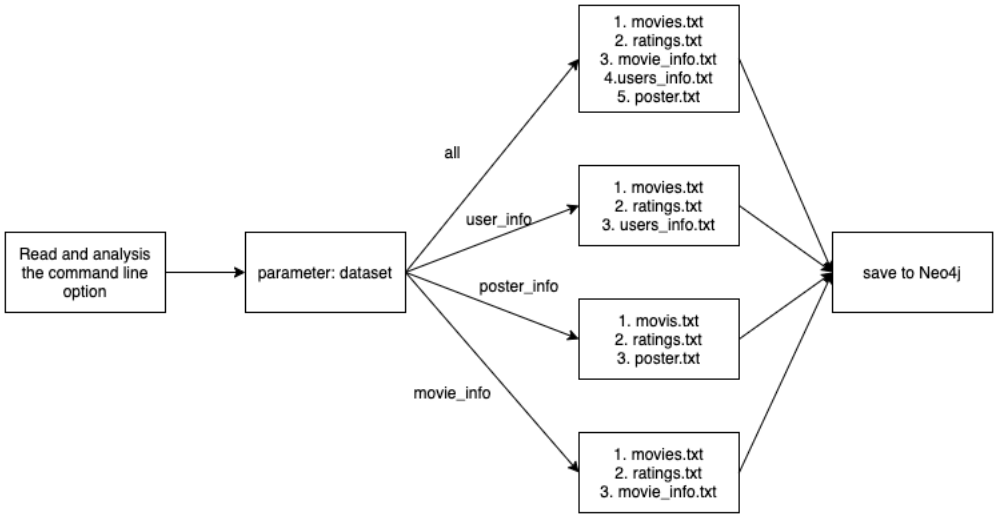

Figure 42: Structure of the Neo4j storage module in framework.

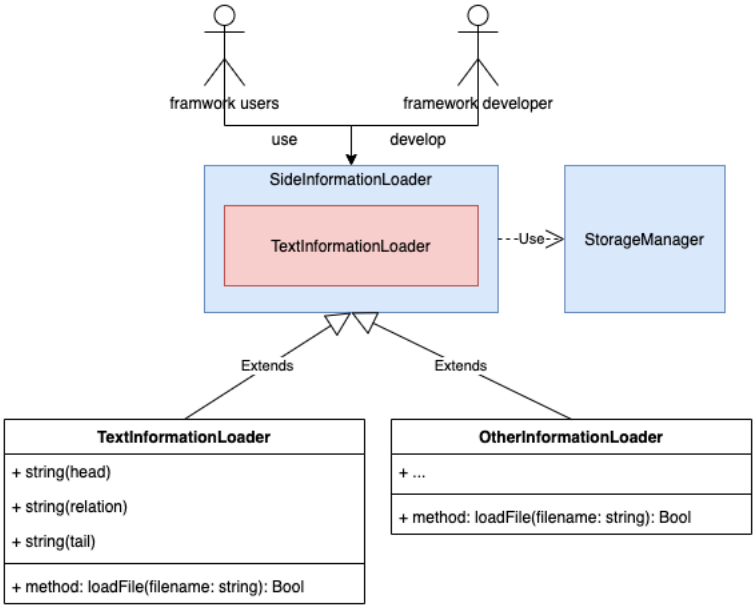

Figure 43: TextInformationLoader instantiation.

For framework developers, if they want to add a new information loader, they only need to provide requested methods to the `SideInformationLoader`. In our implementation, we need to read the file in the parameter. Parse the file according to the format, and extract the head, relation and tail of the triples. Then add new triples to the knowledge graph through `addTriple` in `StorageManager`. Since there is already a method for adding a single file, we only need to make some adjustments about `loadFiles`. When reading files, just call `loadFile` for each file.

4.3 networkxViewer Instantiation

In Section 3.3.3, we described the design of the viewer module. The workflow of the viewer module is shown in Figure 44. Figure 45 illustrates the architecture of the RDF knowledge graph viewer. Our process can be described by the following steps:

1. Call `query_individual` function, according to the RDF file address, to read all the individual names and store all the individuals in Viewer_individuals.
2. Take the individual list from the previous step, call the `individual_classes` function, and query the class to which each individual belongs.
3. For each individual, call the `query_individual_content` function, through this function, to get all the information connected to this individual.
4. If we get dataproperty, call `add_node` function directly to generate a new node.
5. If what we get is a subindividual connected to this individual, then recursively call `query_individual_content`.
6. If what we get is the current class name, judge whether there is a node of the current class. If not, call an `addNode` function to generate it directly.

For knowledge graph viewer in Neo4j format. Our process can be described by the following steps:

1. Call the `load_all_triples` function to extract all triples in Neo4j.

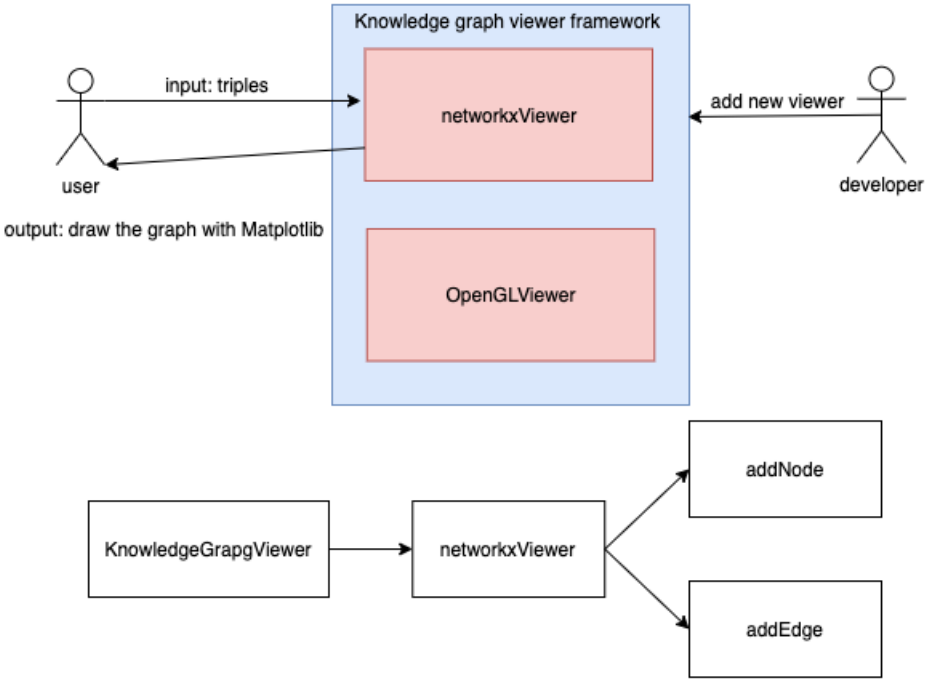

Figure 44: Instantiation of the networkxViewer framework.

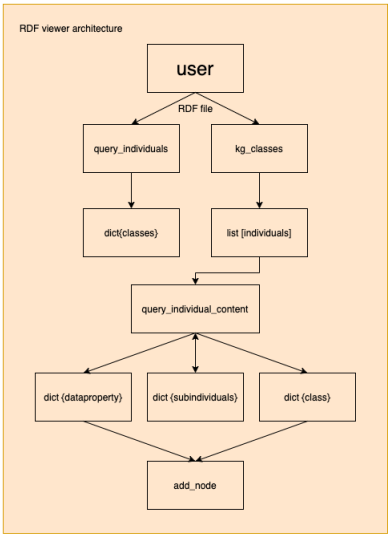

Figure 45: Implemented RDF viewer architecture.

2. According to the situation of each triplet, call `addNode` or `addEdge` function respectively.

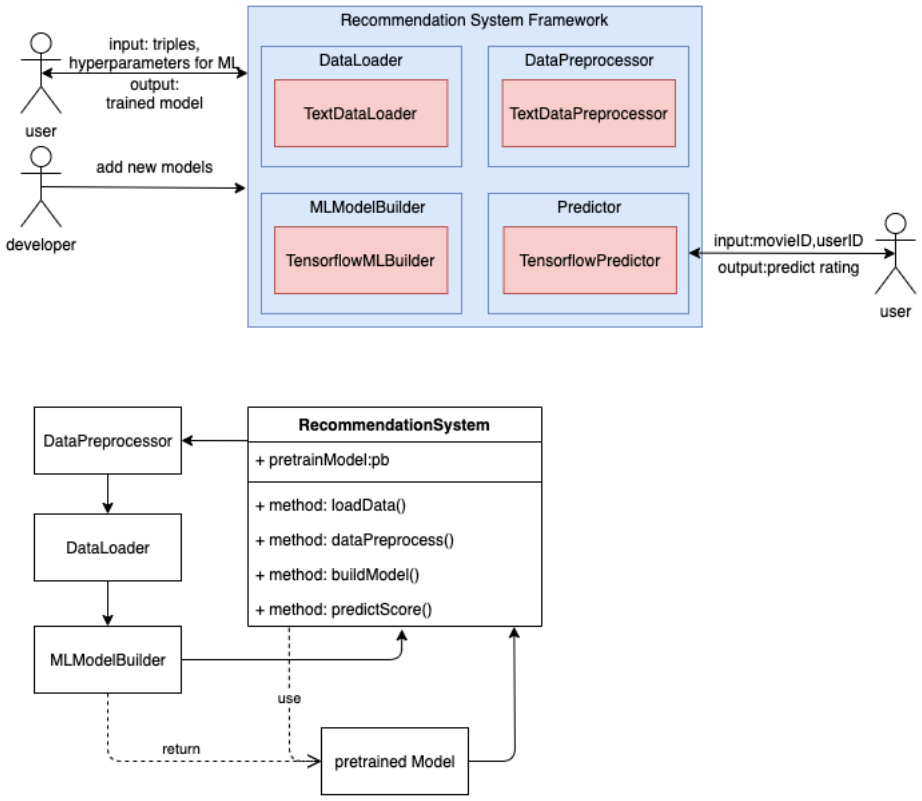

Figure 46: Workflow of recommendation system framework

4.4 Recommendation Method Instantiation

The recommendation method is the most important part of our framework. Without it, we would not be able to complete the final goal. In Section 3.3.4, we proposed the design of a generic recommendation method. Here we present an instance which adopts the deep learning method. We decided to implement the entire method ourselves, as we wanted to incorporate the benefits of side information obtained from using knowledge graphs.

The recommendation method is written in Python and built on top of TensorFlow. Figure 46 illustrates the workflow of the recommendation method module. In the following paragraphs, we will first introduce the network structure and then explain its training process, which includes data preprocessing, loss function, optimizer and some important hyperparameters.

4.4.1 Neural Network Architecture

We implemented the network architecture shown in Section 2.5.3.8, which is a combination of the knowledge graph embedding model, recommendation method model and cross&compress model.

For the recommendation method model, the structure is shown below in Figure 47. In our framework, we made some changes to the structure of the model. The original model does not contain user_gender, user_age and user_job embeddings at the bottom. It is based on the work of [23, 13]. User information is used as side information to improve the accuracy, so we decided to add the user's information in the recommendation method model. To unify the latitude, we also choose arg.dim as the dimension of user_age, user_job and user_gender.

Take our data as an example. There are a total of 6040 users 3883 items in the data, so the user matrix is a $6040 \times$ `arg.dim` matrix, user age matrix is a $7 \times$ `arg.dim` matrix, user gender matrix is a $2 \times$ `arg.dim` matrix, user job matrix is a $21 \times$ `arg.dim` matrix and the item is a $3883 \times$ `arg.dim` matrix. Each time we take out the vector corresponding to the user from the user embedding according to the user's index, extract the corresponding vector from the corresponding embedding file according to gender, age and job. Then go through a fully connected layer to obtain the final vector u_l representation of the user. The item is processed by the cross and compress unit to obtain a V_l vector. The processing of the cross and compress will be explained in detail later. Multiply U_l and V_l and with a sigmoid activation, the result is the predicted score.

For the knowledge graph embedding model, the structure is shown below in Figure 48. Take our data as an example. There are 3883 heads and four relations in the data. Therefore, the item matrix is the $3883 \times$ `arg.dim` matrix and the relation matrix is the $4 \times$ `arg.dim` matrix. Each time we take out the vector corresponding to the head from the head embedding according to the head index, and then process the crossover and compression unit to obtain an H_l vector. The R_l vector is obtained by looking up the vector of the relation index in the relation matrix and then passing

through a fully connected layer. Then merge the [batch_size, dim] dimension H_l and R_l into a [batch_size, $2 \times$ `arg.dim`] vector, and this vector will pass through a fully connected layer to get a [batch_size,arg.dim] vector. This vector is the predicted tail. The correct tail is the vector obtained from a fully connected layer.

For the cross and compress unit, the structure is shown in Figure 49 below. Item and head are a vector of dimension [batch_size,arg.dim]. To facilitate calculation, first expand them by one dimension so that they become [batch_size,arg.dim,1] and [batch_size,1,arg.dim] respectively, and then multiply them, to get the cross matrix c_matrix, a matrix of [batch_size,dim,dim], and a transpose matrix c_matrix_transpose, the two matrices are multiplied by different weights, then reshape, and finally V_l and H_l vectors are obtained.

4.4.2 Training

During the training, our model will be guided by three loss functions: L_{RS} loss and L_{KGE} loss and L_{REG} loss.

$$L = L_{RS} + L_{KG} + L_{REG} \tag{9}$$

The complete loss function is as follows:

- The first item is the loss in the recommendation module.
- The second item is the loss of the KGE module, which aims to increase the score of the correct triplet and reduce the score of the wrong triplet.
- The third item is L2 regularization to prevent overfitting.

Since the training set is too large to be stored in memory at one time, we take batch size data each time. And according to [40], It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize. So we did not read all the data at once.

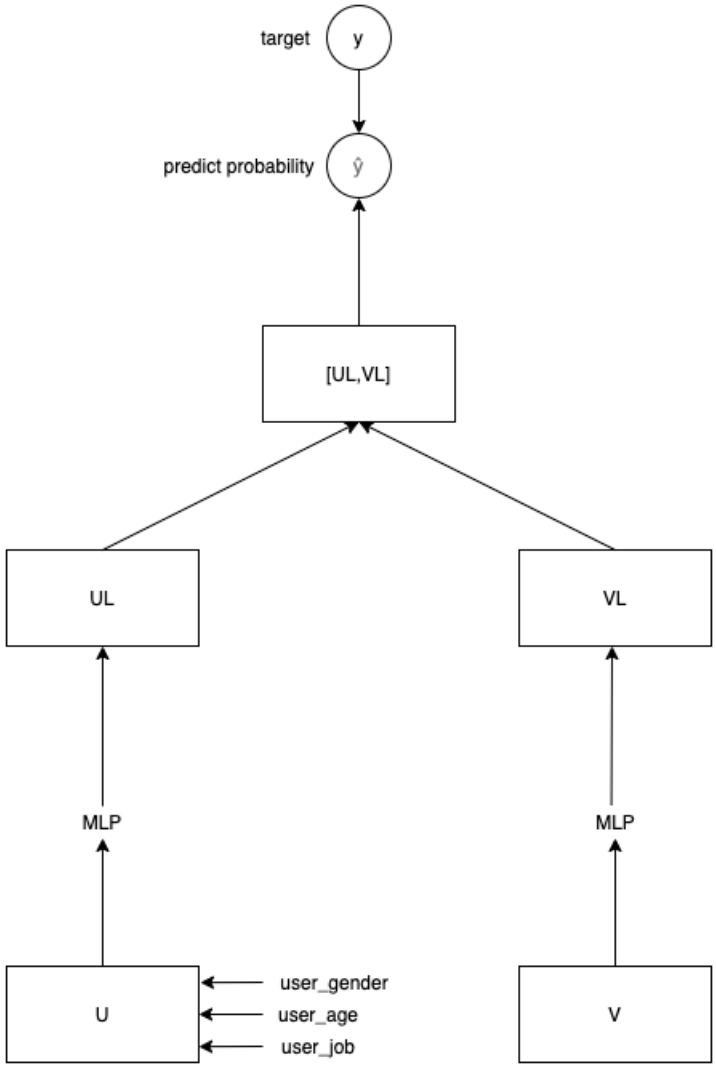

Figure 47: Structure of the recommendation module in RS framework.

Adam optimizer works well in practical applications, surpassing other optimizers. We used the built-in Adam optimizer algorithm provided by Tensorflow. It is an optimization algorithm that finds the best overall advantage and introduces quadratic gradient correction. Compared with the basic SGD (stochastic gradient descent) algorithm, it has the following advantages:

- Not easy to fall into local optima.
- Faster and more effective learning.
- Correct the problems existing in other optimization techniques, such as the

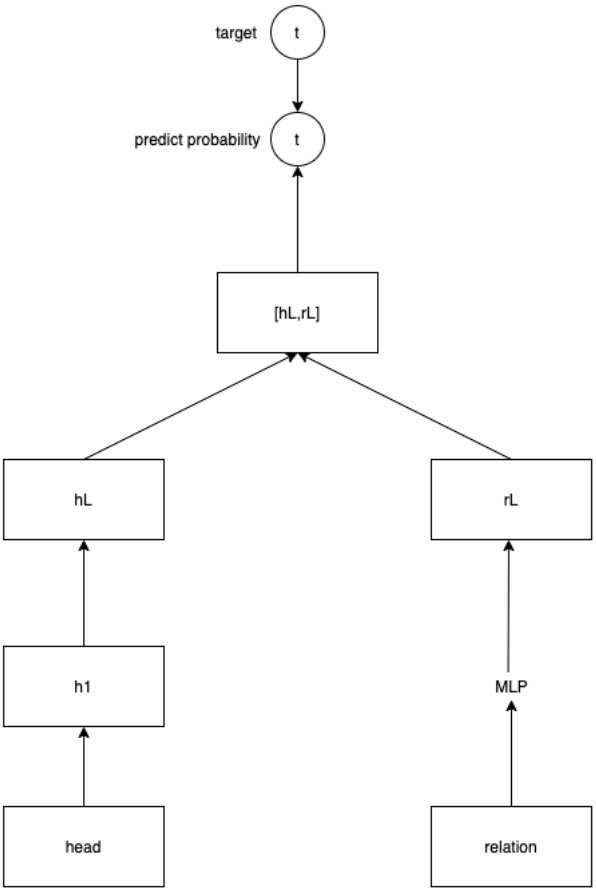

Figure 48: Structure of the knowledge graph embedding module in recommendation system framework.

disappearance of learning rate or high variance of parameter updates that cause large fluctuations in the loss function.

From the implementation point of view, the training program can be illustrated in Figure 50. First, we define a set of configuration variables and the structure of the model. Then we pass the configuration to the model and attach the defined loss function and optimizer. Finally, run the model mode during training. Before training, we first use a data loader to process the data, get the required data to create various weights. Then divide all the data into three, according to the ratio of 6:2:2 into the training set, validation set and test set. Each time, the data amount of batch size is taken for training. If the framework developer wants to make some adjustments to the model, they can modify the model structure in `model.py`. Adjust

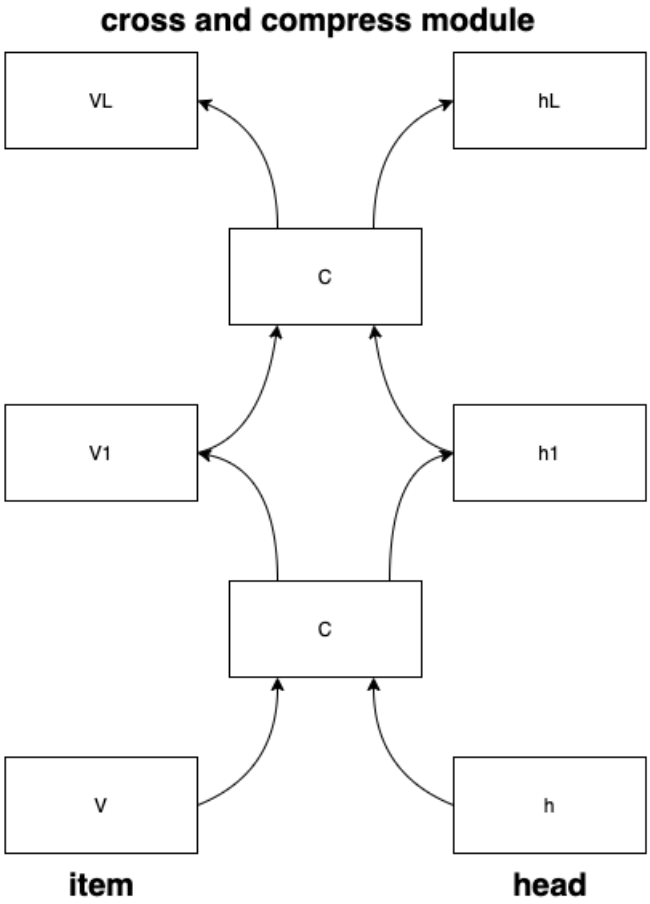

Figure 49: Structure of the cross and compress module in RS framework.

the way of reading data in `data_loader.py`. Use the following command line to train the model. `python3 main.py`. After training and obtaining the model, application users can use the trained model through `python3 predict.py -userid -movieid` to make predictions.

With these settings and after training, the result can be visualized in Figure 51.

4.4.3 Predictor

The above steps are for training the model. After the model is trained, we need to make predictions. At this time, we need to use our predictor module. There are two methods in this module, `getUserInfo` and `predictScore`. `getUserInfo` is a method used to extract user's information. The application user returns three lists by providing userID, saving the user's gender, age, and job category.

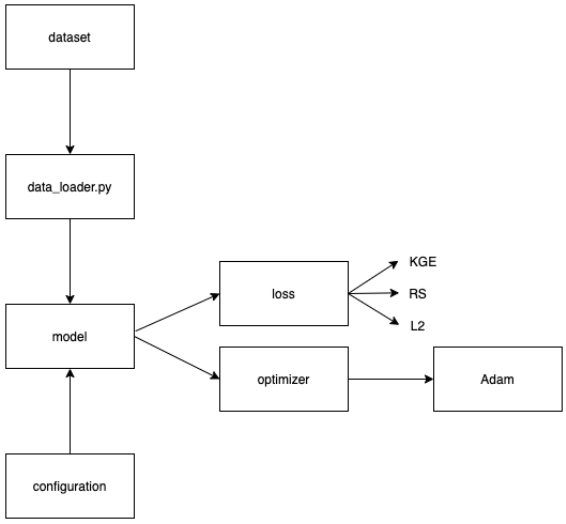

Figure 50: Code structure of the recommendation system training program.

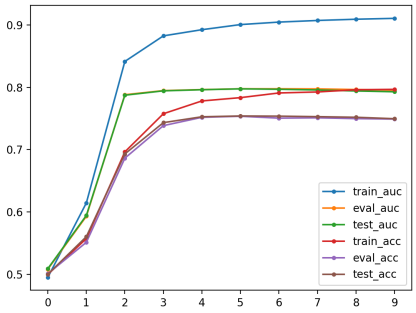

Figure 51: Training visualization diagram of ACC and AUC.

The model trained in the previous step is loaded into `predictScore`, and then different matrices are read by name. If the id entered by the user is greater than the dimension of the model. That means the id is a new user. Our recommendation will focus on the user's age and jobs. Finally, a predicted float value is returned.

4.5 Deployment

Figure 52 shows the diagram of all the required libraries. In `InfoExtractor`, we need `request`, `bs4`, `IMDB` and `base64` libraries. The `requests` library is used to issue standard HTTP requests in Python. It abstracts the complexity behind the request into an API so that users can focus on interacting with the service and using data in

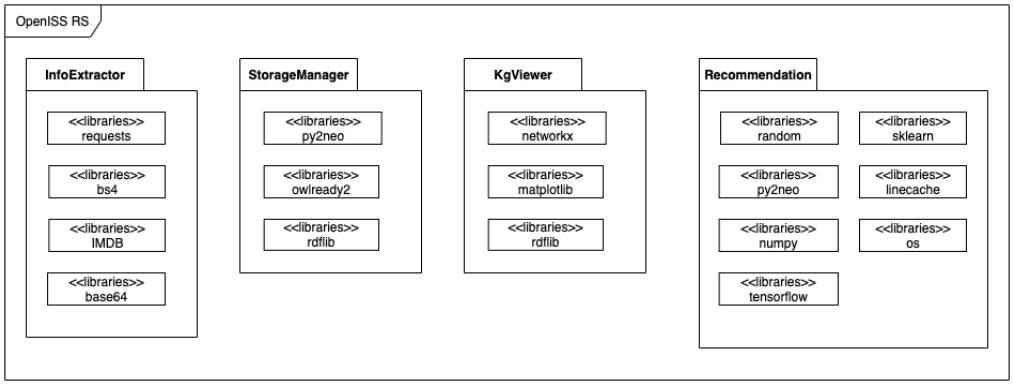

Figure 52: UML deployment diagram.

the application. HTML is composed of a "tag tree", and the `bs4` library is a functional library responsible for parsing, traversing, and maintaining the "tag tree". `IMDB` is an online database of movie information. `Base64` is a library that uses 64 characters to represent arbitrary binary data.

In `StorageManager`, we need `py2neo`, `owlready2` and `rdflib` libraries. `py2neo` can use Neo4j from within the Python application and from the command line. `owlready2` is a module for ontology-oriented programming in Python. `rdflib` is used to parse and serialize files in RDF, owl, JSON and other formats.

In `KGViewer`, we need `networkx` and `matplotlib` libraries. `networkx` is a graph theory and complex network modelling tool developed in Python language, which can facilitate complex network data analysis and simulation modelling. The `matplotlib` library is an essential data visualization tool.

In `RecommendationSystem`, we need `random`, `numpy`, `sklearn linecache` and `tensorflow` libraries. `random` library is used to generate random numbers. `NumPy` is a math library mainly used for array calculations. `sklearn` is an open-source Python machine learning library that provides a large number of tools for data mining and analysis. `linecache` is used to read arbitrary lines from a file. `TensorFlow` is a powerful open-source software library developed by the Google Brain team for deep neural networks.

4.6 Summary

In this chapter, we have illustrated instantiation and implementation of our framework. As can be seen it was easy to switch storage modes. As can be expected, much of the effort was on how to train and integrate a deep learning based recommender method into a framework. In the next chapter, we will take up a couple of typical recommendation system problems, and show how to apply this framework to provide suitable solutions.

Chapter 5

Framework Application Examples

In this chapter, we will explain how to build a recommendation system application using the framework we mentioned in Chapter 4 to achieve our goal. Besides the main recommendation system application, we will also introduce other support services: knowledge graph visualization, predicting users' ratings of movies. We build all these examples using our framework.

5.1 Integrated Lenskit Application

A complete integrated application is critical to this research work in different ways. As mentioned before in Chapter 3. it allows us to iterate on functions when there are new storage methods or new algorithms. This Lenskit application is a comparison of NDCG (Normalized Discounted Cumulative Gain) values for `Lenskit` recommendation algorithms.

Figure 53 shows the main components of the application. Figure 54 shows the result of the evaluation.

Now, we will elaborate on the application from the development perspective.

1. We first need to read the data, and divide the data into training set and test set according to a certain proportion.

2. Next, in the application, we initialize the model.

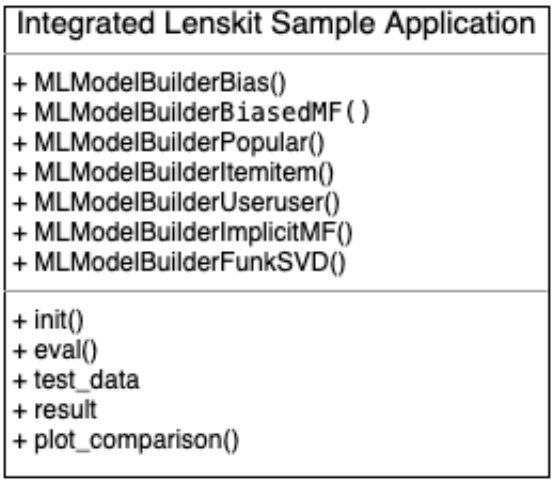

Figure 53: Integrated LensKit example UML diagram.

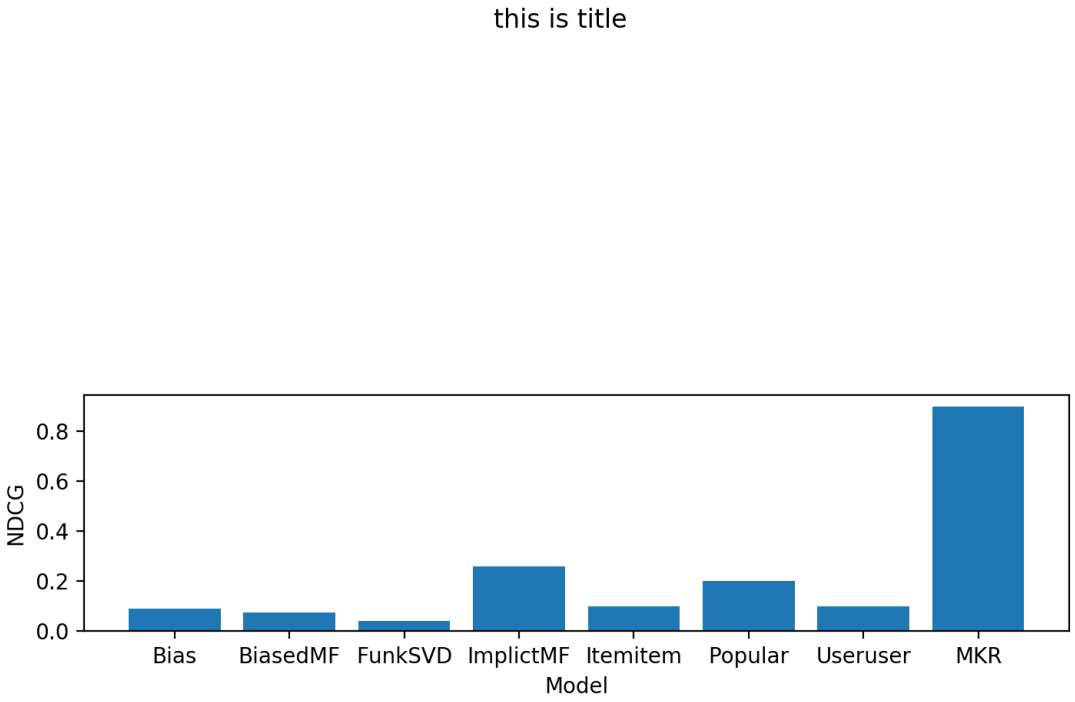

Figure 54: LensKit evaluation result.

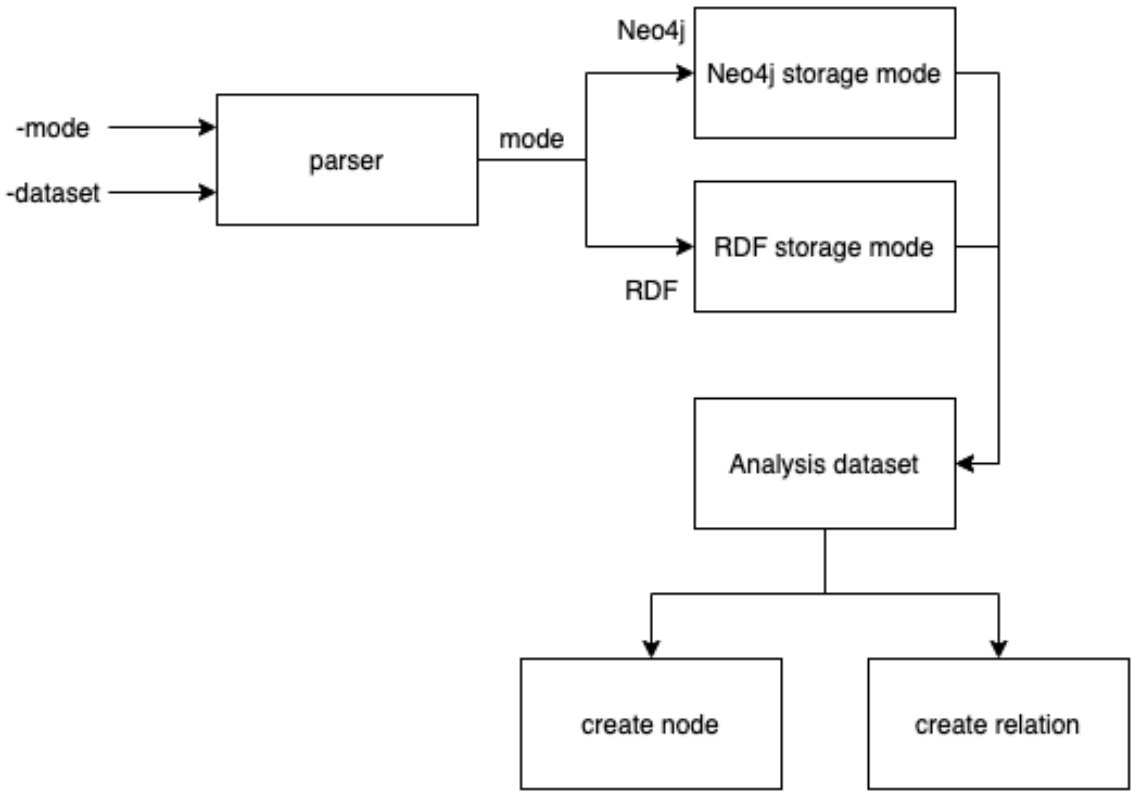

Figure 55: Storage pipeline.

3. The third step we loop over the data and the algorithms, and generate recommendations.

4. Then, we analyze our recommendation lists with a RecListAnalysis. To make sure that our test data and the recommendations line up properly.

5.2 Knowledge Graph Storage Application

As mentioned in item 1, we need to store the knowledge graph triples. For this, we have defined some modules in Section 4.2.

The workflow of the storage instance is depicted in Figure 55. First, application users need to determine whether to use Neo4j or RDF according to the mode in the parser. Second, determine what files are loaded according to the dataset in the parser. Finally, read the content of the file line by line, and generate different nodes and relationships according to the corresponding information. The procedure can be

described as Algorithm 1.

Algorithm 1: Storage application procedure

```
parser ← init a argument parser
mode ← init default storage mode
dataset ← init default dataset

if args.mode == "neo4j" then
    call neo4j storage mode
else
    call RDF storage mode

while args.dataset do
    storage_mode ← init a storage mode
    file_path ← init the file path
    while read file line do
        if node exist then
            find the corresponding node
            else
                create new node
        create relationship and connect two nodes
```

5.3 Prediction of User's Rating for a Movie

So far, what we have described in the previous chapters are the design, implementation and instantiation of our framework. Our prediction service in this application uses the framework to solve the problem of predicting users' ratings of movies.

As mentioned in Section 1.5, the prediction task can be divided into two parts: one is the knowledge graph construction, and the other is the recommendation system. We have shown how to create these two parts for movie recommendations. The workflow can be described as follows:

1. Read all the triple data through `dataLoader` to get the corresponding information.

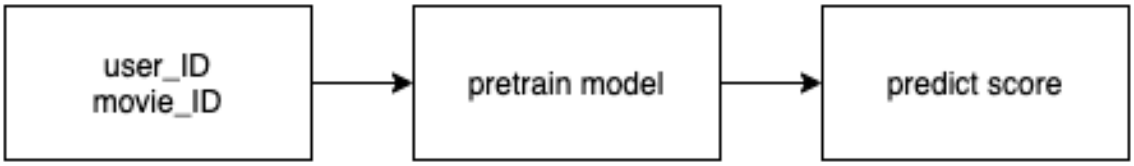

Figure 56: Prediction pipeline.

2. Use these triples and the user's rating of the movie as input, and train through the RS module to get the corresponding model.

3. By loading the trained model, we can predict the user's rating of the movie.

The above process explains some processes to users, but application developers need to know what the inputs and outputs of all these modules are, etc. In most cases, application developers do not care about how it is implemented internally. They only care about what functions this framework can achieve, what format should we provide for input and what format for output.

For the prediction task, we use the pipeline shown in Figure 56. Now, with this pipeline instance, application developers no longer need to worry about details. All the user needs to do is provide the right input. The steps we describe here can be represented by Algorithm 2.

Algorithm 2: prediction application procedure

```
parser ← init a argument parser
dataset ← init default dataset
userid ← init default userid
movieid ← init default movieid

if trained model path exist then
    load trained model
    predict user's rating to movie
else
    Error
```

As we see, the framework users only need to focus on their application development. And the framework can reuse the design, as it provides reusable

```
2686        book.author    14932
10491       book.author    14933
5227        book.author    14934
11584       book.author    14935
8073        book.author    14938
11680       book.date_of_first_publication        14939
14828       book.written_work.author      14940
2760        book.book.genre       14924
10349       book.written_work.author      14945
```

Figure 57: An example of the processed data.

abstractions. As long as it conforms to the interface definition, new components can be inserted into the framework.

5.4 Predict User's Rating for a Book

In the previous section, we introduced the use of our framework to predict users' ratings of movies. In this section, we use different data for training. The data we use is called `Book-Crossing` [17]. Book-Crossing dataset is Collected by Cai Nicolas Ziegler from the Book-Crossing community.

The Book-Crossing dataset comprises 3 tables. They are users, Books and Ratings. Users contain the user's id, the user's location, and the user's age. Books include book title, book author, year of publication, publisher and other information. Ratings include user reviews of the book. The processed book knowledge graph file is shown in Figure 57. Because the process used is the same as Section 5.3, we won't repeat it here.

5.5 Other Applications

In addition to the main knowledge graph storage and prediction applications, we have also implemented some other applications.

5.5.1 Knowledge Graph Visualization

As mentioned in Section 2.4, we need visualization of knowledge graph. For this, we have defined some modules in Section 4.3.

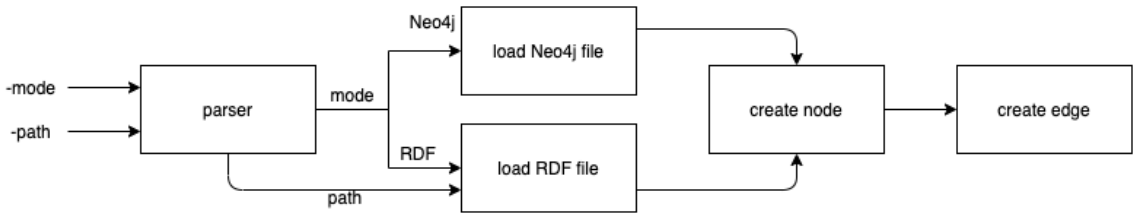

Figure 58: Knowledge graph viewer pipeline.

The workflow of the viewer instance is depicted in Figure 58. First, application users need to choose whether to use Neo4j or RDF. Second, read files according to different modes. Finally, read all the triples, and generate different nodes and relationships according to the corresponding information. This process can be described as Algorithm 3. The result is shown in Figure 59.

Algorithm 3: Knowledge graph viewer application procedure

```
parser ← init a argument parser
mode ← init default storage mode
path ← init default path

if args.mode == "neo4j" then
    load neo4j file
else
    load RDF file

triples_list ← init a list for all the triples
while read file do
    add triples to the triples_list
    while read triples_list do
        if node exist then
            find the corresponding node
            else
                create new node
    create nodes and edges
```

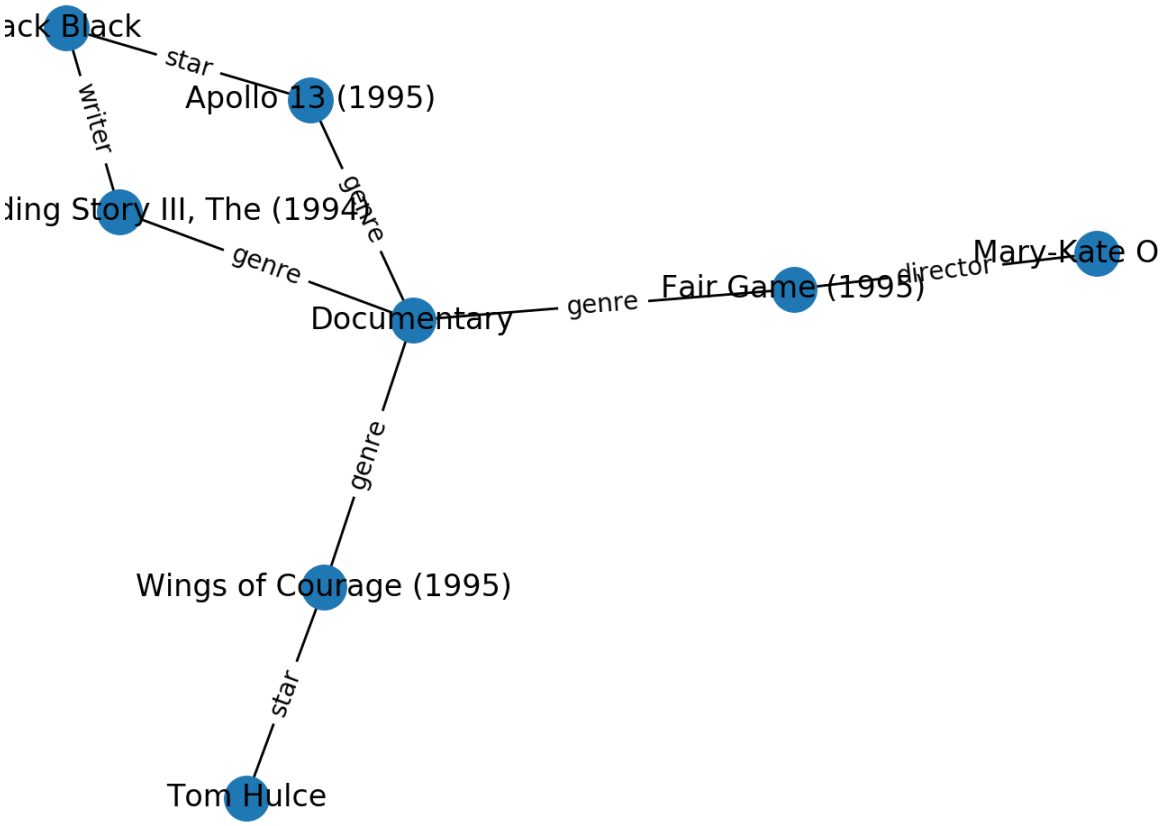

Figure 59: Knowledge graph viewer example.

5.5.2 Knowledge Graph Fusion Application

As mentioned in item 6, knowledge fusion is an effective way to solve the problem of node duplication. The original idea is that users have knowledge graph files in different formats. If they need to train using the knowledge graph, they need to read all the triples, but they may be duplicated. In our case, we use knowledge graph fusion to solve the problem of duplicate nodes.

In our implementation, we provide an API where users can convert Neo4j triples to RDF or convert them to Neo4j based on RDF triples according to their needs. The procedure can be described in Algorithm 4.

Algorithm 4: Knowledge graph fusion application procedure

```
parser ← init a argument parser
mode ← init default storage mode
path ← init default path

if args.mode == "neo4j2RDF" then
    call Neo4j to RDF API
else
    call RDF to Neo4j API

Neo4j_triples_list ← init a list for all the triples in Neo4j
RDF_triples_list ← init a list for all the triples in RDF
while read file do
    add triples to Neo4j_triples_list
    add triples to RDF_triples_list
    while read Neoj_triples_list or RDF_triples_list do
        if node exist in RDF_triples_list or Neo4j_triples_list then
            find the corresponding node
            else
                create new node
        connect nodes and relations
```

5.6 Summary

In this chapter, we described applications that utilize the framework we mentioned in Chapter 4. In the application of prediction and knowledge graph storage, we use the pipeline design architecture provided by the framework. Then we described another framework-based application that can help us modify nodes. In the next chapter, we will carry out evaluation of our framework solution, specifically the advantage of using knowledge graphs and the use of side information in making recommendations.

Chapter 6

Results and Evaluation

In this chapter, our purpose is to evaluate our framework to see if we have achieved the goals mentioned at the beginning (Section 1.2). In this section, we will go over the requirements and solutions, one-by-one, to illustrate how our solution solves them. While doing so, we will also show the advantages of the framework solution and explain the advantages of this method compared to the non-framework solution. We likewise demonstrate how our framework solution makes it easy for us to add various types of side information to increase accuracy of predictions.

6.1 Evaluation Testbed Specifications

Before starting the discussion about the evaluation, we first describe the environment, including the operating system used, processor power, memory, hardware, etc. The detailed specifications can be seen in Table 2. Table 3 lists the various libraries used in this research.

Due to different operating systems, the software is slightly different, so we also need to give the software details of the environment. For the two primary environments: one is a laptop and the other is a server, we call them *Setting 1* and *Setting 2*, respectively. There is a slight difference between the installed software versions. For these environments, we give more detailed information in Table 4.

We further perform both qualitative or quantitative analysis of the framework's

Setting	Name	Device
Laptop	Memory	8 GB
	Processor	2.3 GHz Intel Core i5
	Graphics card	Intel Iris Plus Graphics 640 1536 MB
	OS	Mac OS Mojave 10.14.6
Server (Google Colab)	Driver Version	418.67
	Graphics card	Tesla T4 (16 GB)
	CUDA Version	10.1
Desktop	Memory	12 GB
	Processor	Intel Core i7-920 CPU 2.67GHz
	Graphics card	GeForce GTX1080 Ti (12 GB)
	OS	Ubuntu 18.04.5 LTS 64-bit

Table 2: Environment hardware specifications.

Type	Name	Version
Libraries	`py2neo`	4.3.0
	`Tensorflow`	1.14.0
	`bs4`	4.8.0
	`csv`	1.0
	`networkx`	2.4
	`matplotlib`	3.1.2
	`rdflib`	4.2.2
	`numpy`	1.17.0
	`sklearn`	0.21.3
	`pandas`	0.25.0

Table 3: Python libraries used.

Software	Setting 1	Setting 2
PyCharm	2019.3.2	2019.3.5
Python	3.7.5	3.7.5
Neo4j Desktop	1.2.1	1.2.9
Protege	5.5.0	5.5.0

Table 4: Software packages and IDE tools used.

API, framework instance and its performance in recommendations of our use case with the side information.

6.2 Qualitative Evaluation

6.2.1 Switching of Knowledge Graph Storage Mode

In Section 4.2, we described the design of a storage module that provides an interface for various data storage formats. With the provided API we can easily switch between different storage modes and without changing the rest of the existing framework and its applications.

Storage switch: For Neo4j and RDF-based storage, we provide a short sample code for application developers to understand how to call our framework's components. The code can be found under the path `samples/kg_examples.py`. The storage mode selected by default is Neo4j as we found it to be more flexible. When the developer selects RDF, `StorageManager` calls `RDFManager` instead for applications that require the use of RDF format. This is shown in Figure 60. From this, we can say that our FR1 is fulfilled.

```
parser = argparse.ArgumentParser()
parser.add_argument('-mode', type=str, default='neo4j', help='default neo4j')
parser.add_argument('-dataset', type=str, default='all', help='default all')
args = parser.parse_args()

if __name__ == '__main__':
    storage_m = StorageManager()
    if args.mode == "neo4j":
        StorageManager.getManager("Neo4j")
        StorageManager.load(args.dataset)
    elif args.mode == "rdf":
        StorageManager.getManager("rdf")
        StorageManager.load(args.dataset)
    else:
        print("ERROR")
```

Figure 60: Switch example between Neo4j and RDF formats.

New storage format additions: because we have an abstraction layer on top of each specific storage mode implementation, additional formats can be implemented by framework developers.

Compared to solutions without abstract methods, when we try to add a new storage mode, it will be more work for the developers. However, if we consider the convenience (when developers switch between different storage modes, they only need to change a several lines), the abstraction layer will make these aspects easier. With the development of technology, there will be better storage methods to store triples in the future. Thus. if developers need different storage modes when developing various applications, or the current two storage methods may not suitable for their pipelines, they add a module that complies with the `StorageManager` API. Under our framework design, developers need to add a new storage mode according to the following steps:

1. Create a new class, and implement all operations of this storage type, such as `addNode()`, `addTriple()`, `addRelation()`, `findNode()`, `deleteNode()`, `deleteRelation()` and so on.

2. We also need to add `elif args.mode == "new storage mode"` in the application, then do a `StorageManger.getManager("new storage mode)"` call in the `StorageManager` file. Then, simply add a line of code to add the new storage mode to our framework.

Through the above steps, it is shown that our framework meets the necessary requirements of FR1 and FR2.

6.2.2 Modify the Nodes of the Knowledge Graph

As explained in Section 1.3.1, FR3, we want that the framework can also modify the node. Because we are currently using open-source structured data, all data are non-repeating, so there is no need to delete or modify nodes at this stage for validation purposes. However, it will be needed in the future and it may be necessary to merge

multiple knowledge graphs in the future. At that time, nodes or the relationship between nodes will need to be modified.

Take Figure 61 as an example: from the figure, we find that there are a total of six nodes, and the three relationships are *director*, *writer*, and *star*. A total of two labels distinguish the attributes of the node, one is *person* and the other is *movie*.

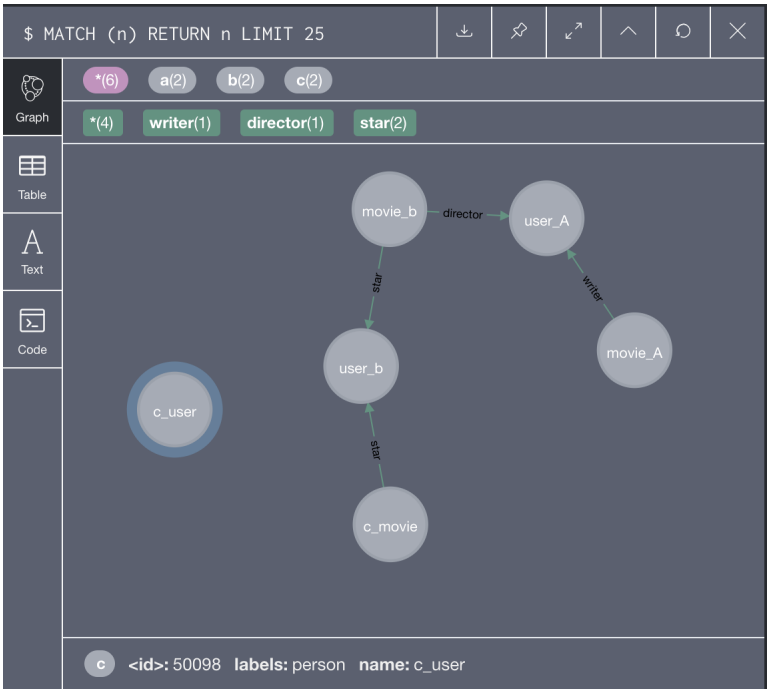

Figure 61: Original Neo4j nodes.

Suppose we created a wrong node when constructing the graph. In `Neo4j`, we need to write `cypher` regardless of any operation on the graph, so in this example, we integrate the `cypher` delete statement into the `deleteNode()` method shown in Figure 62.

For example, since the `c_user` node is not connected to other nodes, after we delete the `c_user` node, the remaining nodes are shown in Figure 63.

Sometimes, we also need to delete a wrong relationship between nodes. At this time, we need to call the `deleteRelation()` method (Figure 64). The result after relation deletion is shown in Figure 65. Figure 66 shows the result of deleting `user_b` and all relations related to it through the `deleteTriple()` method.

To simplify this step, we also provide the corresponding interface. We can easily

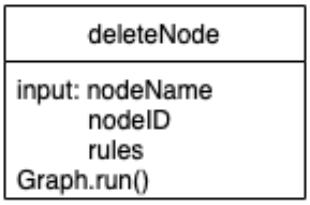

Figure 62: Neo4j node deletion.

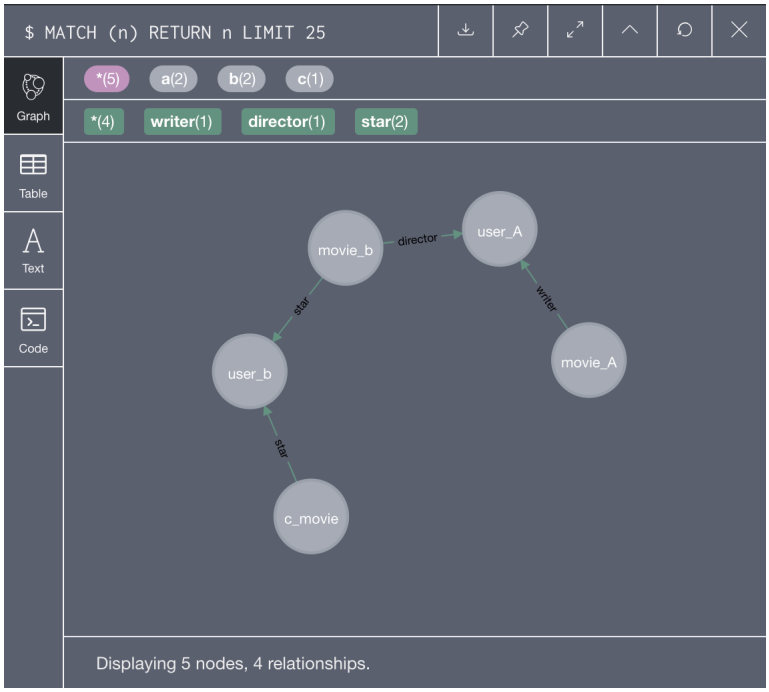

Figure 63: Neo4j node without `c_user`.

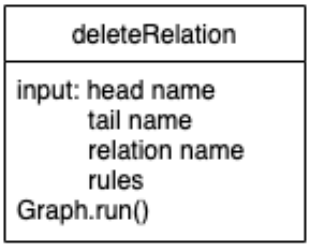

Figure 64: Neo4j delete relation.

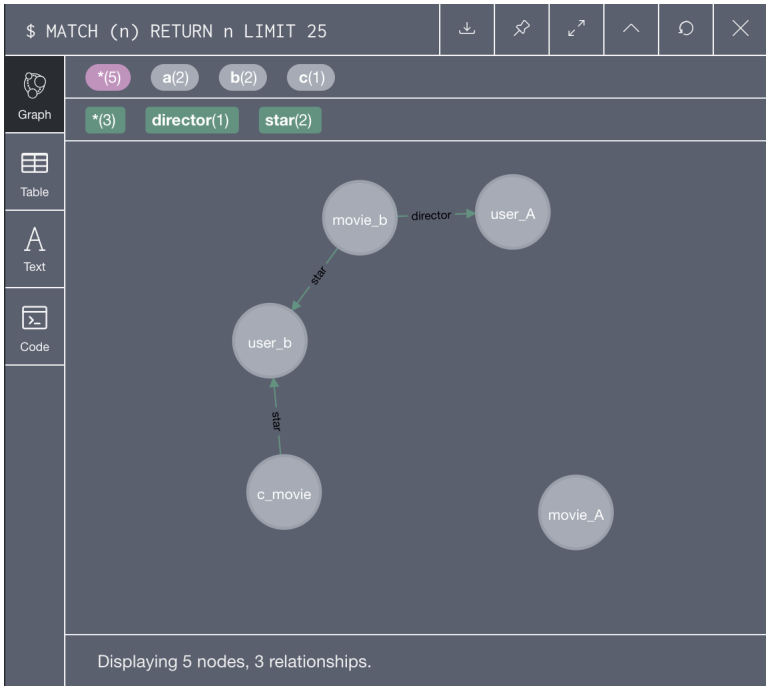

Figure 65: Neo4j node without relation between `user_A` and `movie_A`.

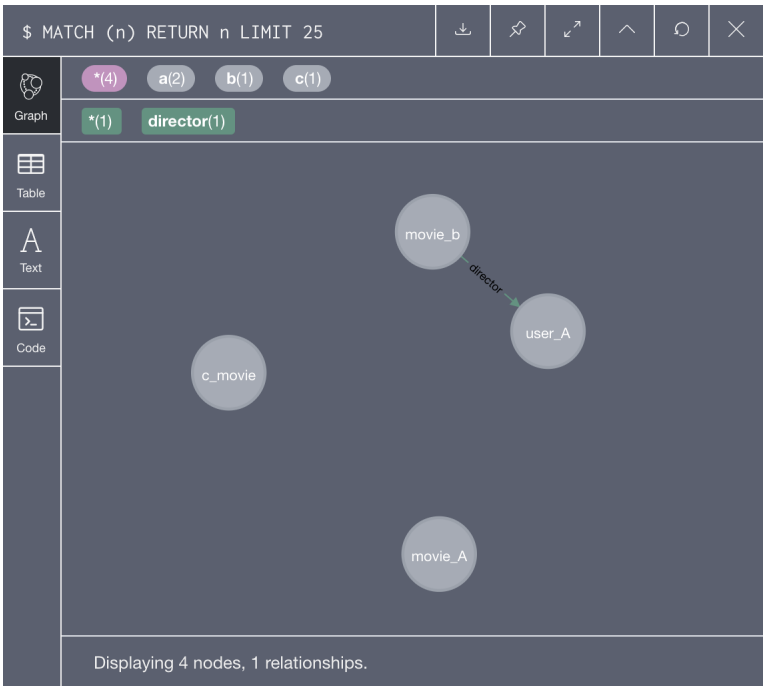

Figure 66: Neo4j node without `user_b` and all the relations connected with it.

call different method according to the definition of the interface. The supported functionality is to delete nodes, delete specific relationships between nodes, and delete all relationships connected to the node. If we need to implement any specific functionality, we can write rules based on the grammar of the cypher, and finally integrate it into the `modify_node.py` file. Through the above discussion, we have addressed the FR3 requirement.

6.2.3 Knowledge Graph and Recommendation System Support for Multiple Languages

As mentioned in Section 1.3.1, if a framework user develops using our framework requires using languages other than English or when it is necessary to integrate the knowledge graphs of multiple languages, we need our framework to support these at all levels – recommendations as well as display.

However, the data in our primary public dataset are in English, so for the knowledge graph construction stage, we need to make an additional small application to test whether the knowledge graph storage module supports multiple languages. We searched the Internet for various movie titles in various languages, namely in French, Chinese, Japanese, Spanish, Russian and German and recorded them in our multilingual example set. Figure 67 is an example of using Neo4j desktop to display a multilingual knowledge graph based on Neo4j storage. The code can be found in `sample/neo4j_multilingual.py`.

Figure 68 shows that the knowledge graph based on the RDF format can also store triples in multiple languages. In this picture, we used the same movie data as in the Neo4j example. Because of the RDF format, we need to add a schema layer. In the category of *person*, we are presently subdivided into *men* and *women* (for simplicity of illustration, and as of this writing children and other genders are not present in the source datasets, so we have not included them). When creating an individual, if we do not know the gender information, we can directly classify the individual to *person*. When we create a knowledge graph in the RDF format, we need to define

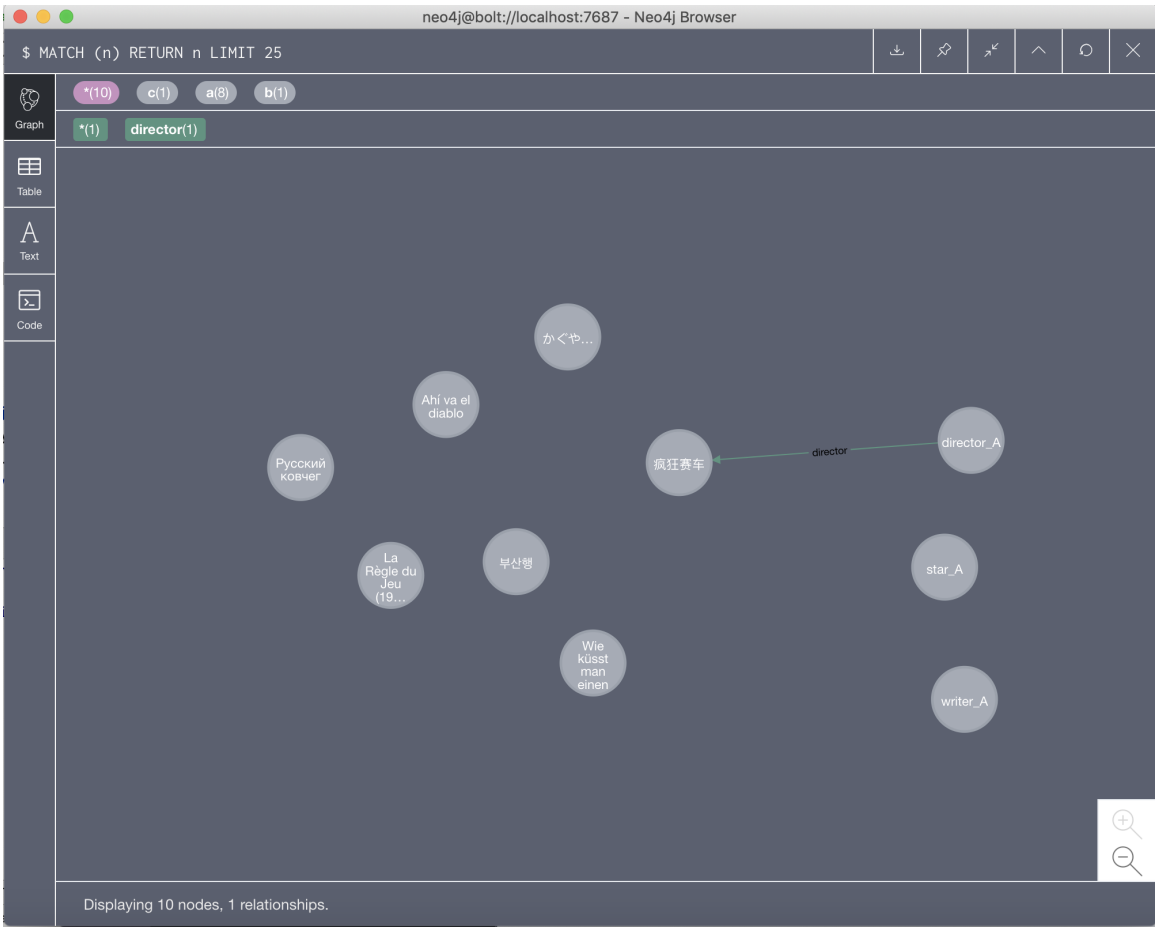

Figure 67: An example of using Neo4j desktop to display a multilingual knowledge graph based on Neo4j storage.

the URI or use a randomly generated URI. All resources in RDF are identified with a Uniform Resource Identifier. URI can be treated as a string that identifies network resources (as seen, e.g., in Figure 68 in the yellow box).

The yellow circle represents the schema layer. The purple diamond shape represents an individual. The line between `owl:thing` and `person` and the line between `person` and `man` represent the relationships between them – "has-a-subclass". The line connecting `writer_A` and `man` represents "`man` has an individual `writer_A`". The yellow dashed line between `person` and `movie` represents "A-is-a-director-of-B", which is an `objectproperty` defined by us. The `objectproperty` defines the relationship between two classes. The individual of the `person` class and the individual of the `movie` class are connected, which means that there is "an A who is

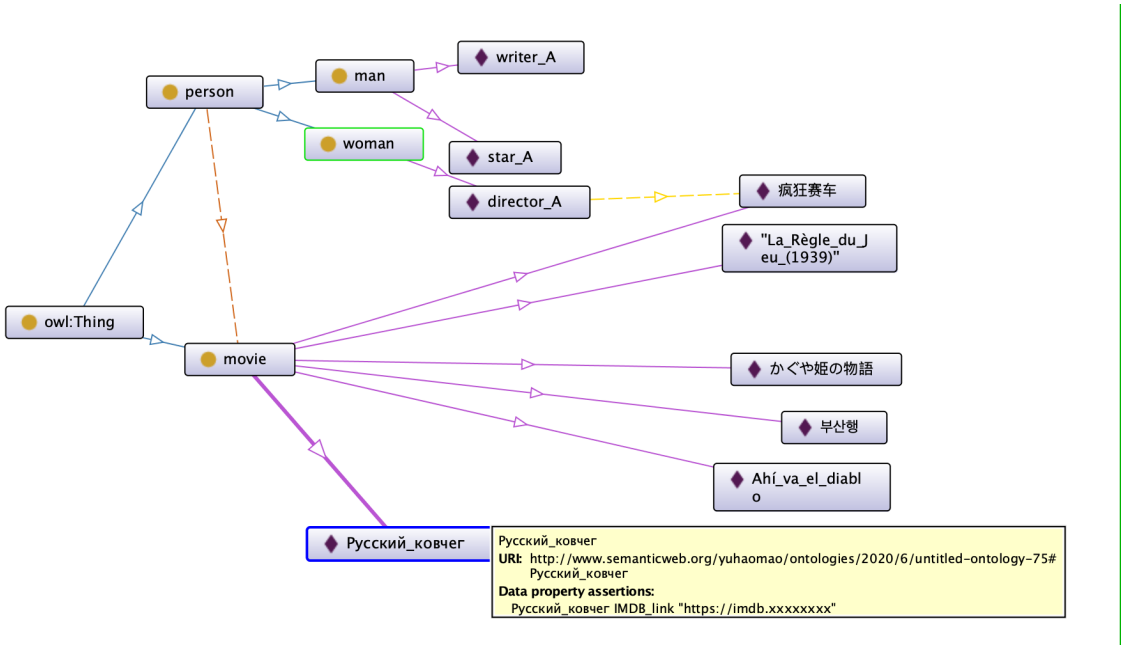

Figure 68: An example of using Protege to display a multilingual knowledge graph based on RDF storage.

director of B" relationship between them. Besides, we define a `dataproperty`, which is used to define individuals' attributes. The type of this `dataproperty` is an IMDB address. For example, we set up an IMDB URL for the movie "Русский ковчег" ("Russian Ark"), which can be seen in the picture. This illustrates that we meet the requirement FR4.

For the recommendation part in Section 1.3.1, the method we adopt is to preprocess the input text file `movie.txt` through the `DataPreprocessor` into the format illustrated in Figure 69. The first column is the encoding of the movie, and the first seven rows represent all the triple information of the first movie. The third column represents the *object* in the triple. The middle column represents the relationship between the *subject* and the *object*.

The rating file example processed by `data_process.py` is shown in Figure 70. The first column represents the user's id, the second column represents the movie's id, the third column represents the user's rating of the movie, and the last column represents the timestamp the user posted the rating comment. The specific processing

flow of the `DataPreprocessor` class is as follows:

1. Read the text file based on the file path.

2. Create three dictionary variables: `entity_id2index`, `relation_id2index` and `item_index_old2new`. They are used to store relevant information such that it is convenient to query whether relevant information has been stored in the previous processing run.

3. By reading each row of data, the relevant information is stored in different variables, and then they are output to the corresponding file.

As a result of these steps, we have solved the support for different languages in the recommendation system through unified coding, which includes both machine learning components, side information, as well as display. This clearly demonstrates that our solution fulfills FR5.

```
0   film.film.genre 3883
0   film.film.genre 3884
0   film.film.director  3885
0   film.film.writer    3885
0   film.film.writer    3886
0   film.film.star  3887
0   film.film.star  3888
1   film.film.genre 3883
1   film.film.genre 3889
1   film.film.director  3890
1   film.film.writer    3891
1   film.film.writer    3892
1   film.film.star  3893
1   film.film.star  3894
2   film.film.genre 3895
2   film.film.genre 3896
```

Figure 69: An example of preprocessed of the movie file.

6.2.4 Knowledge Graph Fusion Evaluation

As mentioned in Section 1.3.1, if an application developer uses our framework, and they want to integrate other people's knowledge graphs (possibly in different formats), then, we need our framework to support the fusion of knowledge graphs. However,

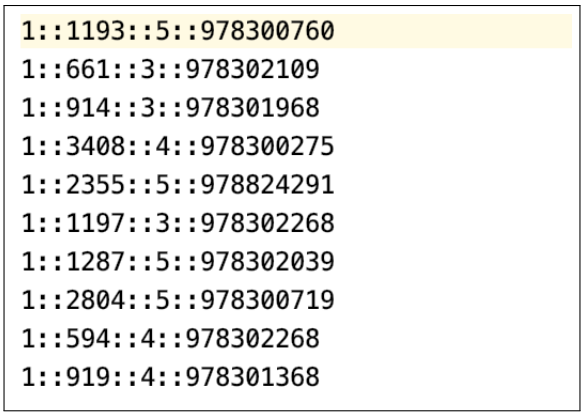

Figure 70: An example of preprocessed of the rating file.

the fusion of knowledge graphs is more than simply concatenating two knowledge graphs together to generate a larger graph. To merge two knowledge graphs, one needs to make sure the duplicate nodes from them are properly combined with their respective relationships.

Further, we use a short example to illustrate this function. Figure 71 and Figure 72 respectively show the original Neo4j and RDF format knowledge graph. Through different functions, we can get all the triples in the two storage modes respectively. Figure 73 shows all the triples extracted from the Neo4j sample. The first triple is to test whether the new node and existing relation can be added to the RDF knowledge graph. The second one is to test whether a new triple can be added to an existing node. The third and fourth triples are to test whether two existing triples can add new relations and whether the existing triple will be properly regenerated.

Figure 75 shows the result of the fusion of triples from Neo4j into the RDF. For the first triple mentioned above, we can see that two new nodes are generated as shown in the figure, and the green dashed line indicates the existing triple relationship. For the second one, we can see that the code does not repeatedly generate the same node, but only generates the new ones for the relations and tail that are not in the original graph. For the third triple, two existing nodes did not generate a new one, but we generate a new relationship that does not exist in RDF, and it is represented by a dark purple dashed line. For the fourth triple, we can see that nothing has been

changed on the existing triple.

Figure 74 shows all the triples extracted from the RDF sample. The first triple is used to test whether a new relationship can be directly created between two nodes when the head or tail exist respectively. The second triple is used to test whether a new head will be created if the tail of the new triple already exists. The third triple is used to test whether the existing triple will be regenerated. The fourth triple is used to test whether a new triple can be generated. Figure 76 shows the result of the fusion of triples from RDF into the Neo4j. For the first, second and third triples mentioned above, we can see that they all connect related nodes and generate corresponding relationships, without generating duplicate nodes. For the fourth triples, we can see that two new nodes are generated in the graph and the corresponding relationships.

To test at scale, we also merged two public datasets with a larger amount of data. These two data sets are ml-1m and ml-latest-small. The specific data size is shown in Table 5. We fused the two knowledge graphs through the framework's integrated API.

Dataset	**Relations**	**Movies**	**Movie triples**	**Nodes**
MovieLens 1M	1	3883	6408	3901
MovieLens latest small	1	9742	22085	9759
Fused MovieLens 1M small	1	11001	24328	11020

Table 5: MovieLens 1M dataset and MovieLens Small Latest dataset.

Then we read the triples in the two knowledge graphs through scripts and queried them in the new knowledge graph to verify the newly generated knowledge graph. We tested it on laptop and desktop, and the fusion of the two knowledge graphs requires 596ms and 472ms respectively. The fusion map has a total of 11020 nodes, one relation and 24328 triples. It is evidence which can clearly demonstrate that we fulfill FR6.

6.2.5 API Usability

From Section 1.3.2, API usability has been identified as one of the non-functional requirements of our solution. In this paragraph, we analyze usability in two aspects: one is that the system is usable for our goals and applications, another is that it

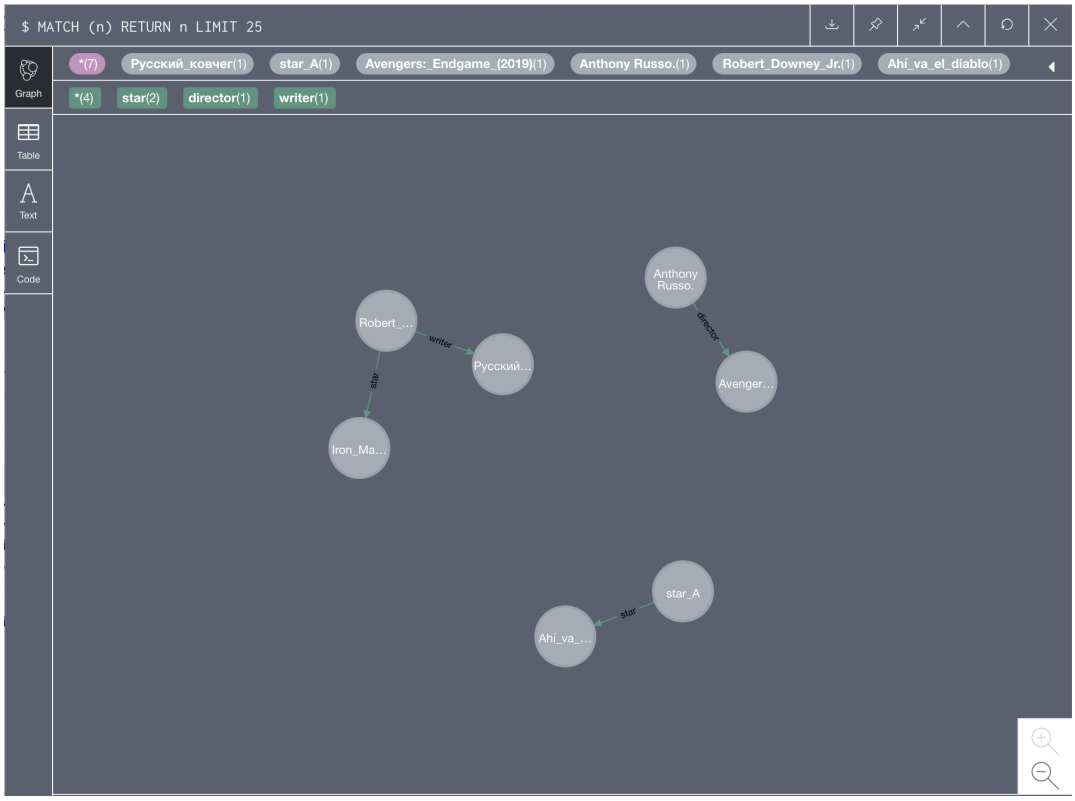

Figure 71: An example of Neo4j knowledge graph based on small data.

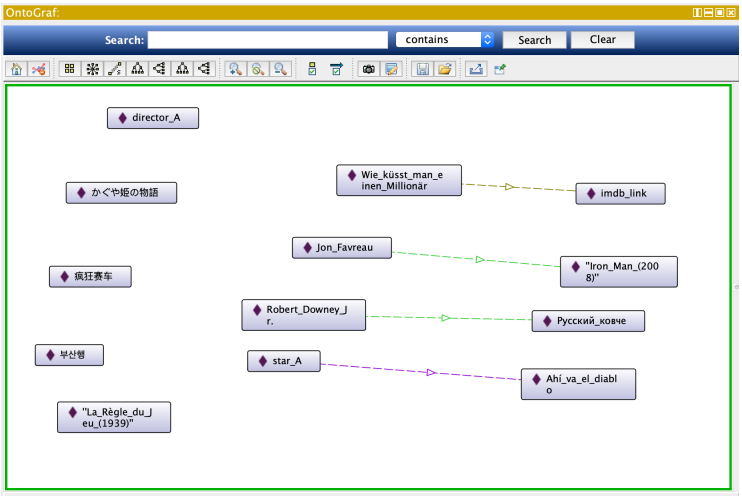

Figure 72: An example of RDF knowledge graph based on small data.

```
[' Anthony Russo.', 'director', 'Avengers:_Endgame_(2019)']
['star_A', 'star', 'Ahí_va_el_diablo']
['Robert_Downey_Jr.', 'writer', 'Русский_ковчег']
['Robert_Downey_Jr.', 'star', 'Iron_Man_(2008)']
```

Figure 73: The triples were extracted from Neo4j samples.

```
['Wie_küsst_man_einen_Millionär', 'poster_link', 'imdb_link']
['Robert_Downey_Jr.', 'director', 'Русский_ковче']
['Jon_Favreau', 'director', 'Iron_Man_(2008)']
['star_A', 'star', 'Ahí_va_el_diablo']
```

Figure 74: The triples were extracted from RDF samples.

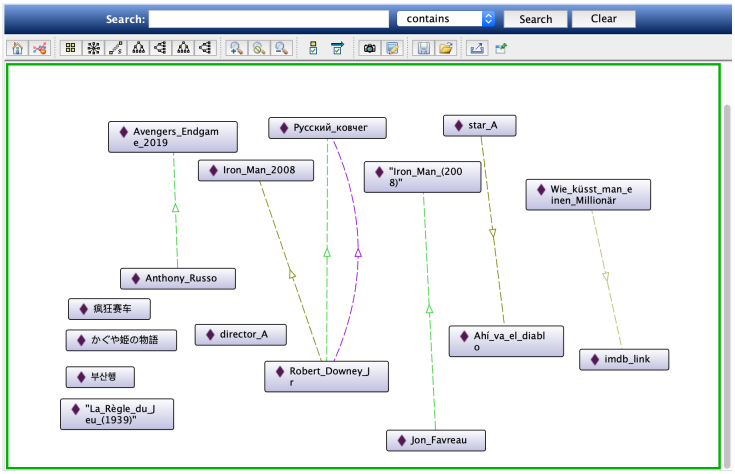

Figure 75: The triples were added from Neo4j samples to RDF format.

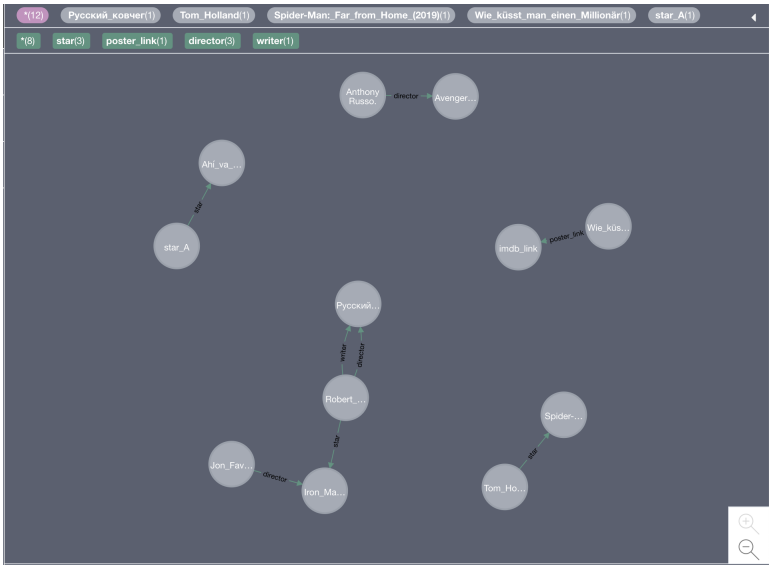

Figure 76: The triples were added from RDF samples to Neo4j format.

should be user-friendly. Please note however, in the scope of this work we do not conduct a formal usability study as is usually done for HCI applications as it goes beyond the thesis scope and this initial evaluation lays a ground work for the formal evaluation later on. As a result we limit the usability aspect to discussion of the API-based usability from the framework and application developer stakeholders point of view, similarly to as it was done by Singh in [73].

To illustrate the basic API usability factors of our framework, we listed all the sample applications provided with the framework. These applications show that our framework can be used to perform the following tasks:

- Use the movie name to search for information such as the director, author, and actors of the movie.
- Load the knowledge graph from Neo4j or RDF and modify the nodes on it.
- Query all triples in RDF or Neo4j knowledge graphs.
- Add new triples to the Neo4j knowledge graph or RDF knowledge graphs.
- Load the pre-trained model to predict the user's rating of a movie.
- Load the pre-trained model to predict the user's rating of a book. (Cross-domain application transfer).

All the samples mentioned above are implemented through the designed API calls. Each method only requires a few simple lines of code, from importing the corresponding library to calling the corresponding method. With provided illustrations throughout the thesis, we conclude we meet the NFR2 requirement.

6.2.6 Extensibility

In Section 1.3.2, we mentioned that our system needs to have extensibility as a nonfunctional requirement. The switching of knowledge graph storage mode and adding a new storage mode, which we just explained in Section 6.2.1 is a good first showcase for it. For the `MovieExtractor` framework, when we need to integrate a new

one, the work at the framework level is creating the corresponding class/methods, the process can be described as the following:

1. Create a subclass of the `MovieExtractor` frozen spot to extend it.
2. Write extraction rules according to the layout of the data source (e.g., a web page), and implement the API methods such as `extractDirector()`, `extractWriter()`, and `extractStar()`.
3. Add the new extractor to the `MovieExtractor`. The application needs not change except to select the newly defined extractor.

In terms of the extensibility of recommendation system components, as we explained in Chapter 5, when we need to develop and integrate a new recommendation method, we can follow the steps below:

1. Implement (import) new recommendation algorithms based on new requirements.
2. Add a new subclass in the recommendation system module.

We can use this, for example, to write wrappers around other recommenders if we need to include them and make available to our framework's applications. Therefore, we confirm that new implementations can be added to specific modules to extend the framework's instance's functionality. Based on the above description, we establish we meet the requirements of NFR3 without impairing the existing system.

6.2.7 Cross-Platform

Through our tests and reliance on Python, we have seen that our code runs on macOS and Linux systems. Windows has not been tested yet. The required tools and library version numbers are shown in Table 3 and Table 4. Through tests on different platforms, we have also sufficiently addressed NFR4 in Section 1.3.2.

6.3 Quantitative Evaluation

6.3.1 Real-time Response

Since we want our solution to address the problem of quickly predicting user ratings, real-time response is a non-functional requirement of our solution for possible deployment purposes. According to [52], we set the real-time response baseline to be 2000ms. To evaluate the whole system's processing ability, we performed the experiment described below:

1. Load the trained model to get the pre-trained graph structure and weights.
2. Prepare `feed_dict`, either with the new training data or test data. In this way, the same model can be used to train or test with different datasets.
3. Measure the difference between the timestamp t_s before the pipeline start and the timestamp t_d after system processing.
4. Repeat the previous operation 100 times, and then calculate the average processing time through the formula Equation 10.

$$Result = \frac{\sum_{i=1}^{100} t_e - t_s}{100}. \tag{10}$$

The result shows that the deployment runtime of our solution on a local laptop machine is `634.33ms` and the runtime on Google Colab is `68.42ms`. Its specification can be found in Table 2. It is faster than the real-time baseline of 2000ms. This fulfills NFR1 stated in Section 1.3.2.

6.3.2 Recommendation Performance Evaluation

As we have already evaluated the knowledge graph construction, we will introduce the evaluation methods of the recommendations obtained and report the results by applying these evaluation methods to our implementation using side information as a case study. This case study was the original motivation for this work that evolved

into the framework approach. As a result, we used our framework together with deeplearning-based `MLBuilder`-based-hotspot in this section. We begin with some definitions first.

6.3.2.1 ACC

To measure the quality of the recommendations we described in Section 4.4, we use the method called an "accuracy" (ACC) and "Area under curve (AUC)". ACC is called the "accuracy rate", which means the proportion of all samples that are accurately judged as positive and negative. The formula for ACC is as in Equation 11:

$$ACC = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{11}$$

The metrics TP (True Positive), TN (True Negative), FP (False Positive), FN (False Negative) mentioned in the above formula are used in the classification result, as shown in Figure 77:

		Actual	
		Positive	Negative
Predicted	Positive	**True Positive**	**False Positive**
	Negative	**False Negative**	**True Negative**

Figure 77: Possible result for classification problem.

- True positive, the model correctly predicts the positive class.
- True negative, the model correctly predicts the negative class.
- False positive, the model incorrectly predicts the positive class.
- False negative, the model incorrectly predicts the negative class.

ACC reflects the ratio of the classifier accurately identifying true positives and false negatives. It also means the proportion of the correct samples to the total samples.

The value range is $[0, 1]$. The larger the value, the better the predictive ability of the model. The ACC indicator treats each category equally, which means that the cost of correct (0) and wrong (1) for each sample is the same. However, ACC can easily cause the model to become "unpredictable" due to data skew. For example, we entered a set of data. Unfortunately, this set of data is very extreme. Among 10,000 data elements, 9900 are negative types, and only 100 are positive types. The model predicts the negative class very accurately, but the prediction of the positive class is very inaccurate. Since Equation 11, the values of TP and FP are small in this set of data. So the ACC formula is approximately equal to Equation 12 at this time.

$$ACC = \frac{(TN)}{(TN + FN)} \tag{12}$$

Because the prediction of the negative class is very accurate, the value of ACC is also good at this time. But for another set of data, such as 9000 positive classes and 1000 negative classes, the prediction of the positive class is bad at this time. There will be a large area of false positive prediction, so the value of ACC becomes very low. In summary, ACC cannot evaluate the model well enough, which is why we added another evaluation metric – AUC.

6.3.2.2 AUC

The AUC value is the area covered by the ROC curve. The ROC curve is a relationship curve between FPR (False Positive Rate) and TPR (True Positive Rate).

- The x-axis is FPR: in all negative samples, the proportion of the classifier's prediction errors (Equation 13).

$$FPR = \frac{(FP)}{(FP + TN)} \tag{13}$$

- The y-axis is TPR: in all positive samples, the proportion of correct predictions

by the classifier (equal to Recall), see Equation 14.

$$TPR = \frac{(TP)}{(TP + FN)} \tag{14}$$

Obviously, the larger the AUC, the better the classification effect of the classifier.

- $AUC = 1$, which means a perfect classifier.
- $0.5 < AUC < 1$, is better than random guessing and has a predictive value.
- $AUC = 0.5$, following the machine to guess the same. There is no predictive value.
- $AUC < 0.5$ is worse than a random guess, but as long as you always go against prediction, it is better than random guessing.

The examples illustrating the ROC curve and a AUC value are in Figure 78. It is also

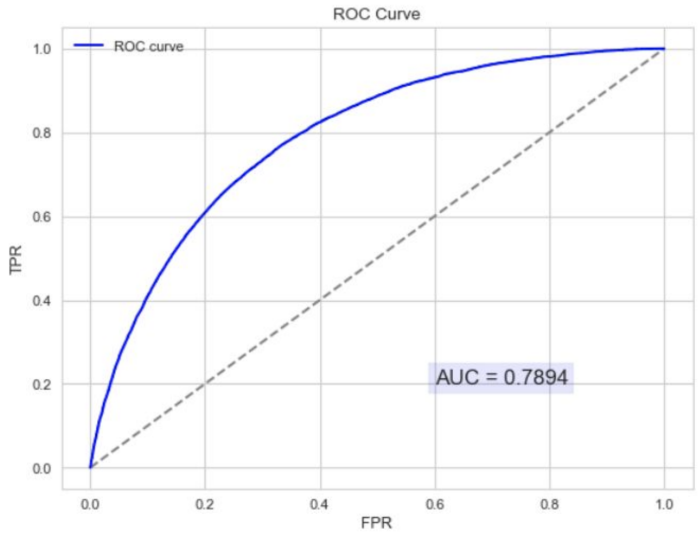

Figure 78: An example of a ROC curve and an AUC value.

worth noting that AUC is not sensitive to whether the sample category is balanced, which is one reason why unbalanced samples usually use AUC to evaluate classifier performance.

6.3.3 Dataset for Recommendation Performance Evaluation

Before proceeding with the experiments and results, it is necessary to review the data sets available in the field.

The MovieLens dataset is provided by the GroupLens research group at the University of Minnesota. MovieLens is a collection of movie ratings, available in various sizes. The data sets are named 1M, 10M and 20M with increasing number of ratings. As a base, we chose MovieLens 1M dataset, which has 1 million ratings from 6000 users on 4000 movies. Compared with other datasets, although this dataset does not have more movie sets and more user reviews than some data sets, it trains faster and comparable with other models. Richer data are better conducive to the training of the model. To remind the reader, to expand the dimension of information, we add the side information such as directors, writers, and stars. Therefore, we use an HTML extractor to search for corresponding information from the IMDB website. Therefore, the expanded MovieLens 1M dataset includes the following information in total Table 6.

Dataset	Users	Movies	Raitings	Movie triples
MovieLens 1M	6040	3883	1,150,560	23440

Table 6: MovieLens 1M dataset.

For completeness, we also compared the training speeds of different machines. It is the slowest on the laptop. If we take 20 epochs as an example, and it takes `140s` on the laptop. `113s` training time on desktop, and the shortest time on Google Colab is `42s`.

6.3.4 Experimental Classification Results

As described in Section 4.4, we trained the MKR model using the MovieLens-1m dataset, and then used the validation set to verify the model. We split all data according to the 6:2:2 ratio, i.e., 60% is the training set, 20% is the validation set, and 20% is the test set. The data of the validation set and test set will not be used for training.

As Section 4.2 shows, our side information has many types, including movie information, user information, and movie posters, we train with different side information types through our model and obtain the results through 20 epoch training, as shown in Table 7. Through our analysis of the table information, we found that by adding movie information and user information, the accuracy rate was increased by 0.5%. But after adding the poster information, the accuracy rate dropped. It shows that not all side information improves accuracy. After reading [43], we believe that the reason is that each poster information is unique, so this causes the problem of data sparseness. Therefore, the accuracy will be affected during training.

MovieLens 1M	train AUC	train ACC	evaluate AUC	evaluate ACC	test AUC	test ACC
baseline	0.9189	0.8392	0.9046	0.8277	0.9042	0.8261
baseline+movie	0.9227	0.8439	0.9081	0.8295	0.9061	0.8297
baseline+user	0.9238	0.8455	0.9096	0.8321	0.9091	0.8331
baseline+user+movie	0.9292*	0.8516*	0.9142*	0.8375*	0.9136*	0.8359*
baseline+poster	0.9173	0.8279	0.9041	0.8153	0.9029	0.8214
baseline+movie+user+poster	0.9273	0.8497	0.9113	0.8351	0.9111	0.8349

Table 7: Results of the same models on different side information datasets.

From the results, we can see that the accuracy of data with user information and movie information is the highest, which is 1% higher than the baseline. Because users may watch a movie because of a famous director or an actor, the other movie information can help improve part of the accuracy. The age, job and other information in the user information also help to improve the accuracy, because the user may choose some related movies to watch based on age and occupation. However, because the poster of each movie is different, in the knowledge graph, each poster is connected to only one movie node, so the poster data is sparse for the knowledge graph. Therefore, the poster information does not have a better accuracy for us at this time (however, if we could extract some useful information from the poster through technologies such as computer vision, it may be helpful in improving the accuracy of the recommendation in the future).

6.4 Summary

In this chapter, we evaluate and provide explanations of the qualitative and quantitative aspects of our recommendation system framework, which is based on machine learning and side information. We can conclude that all of our functional and non-functional requirements have been successfully met to a satisfactory degree and qualitatively or quantitatively evaluated according to our stated requirements, as shown in Table 8. In the next chapter, we will describe the current limitations of

Requirements	Qualitative	Quantitative	Addressed
FR1	X		Section 6.2.1
FR2	X		Section 6.2.1
FR3	X		Section 6.2.2
FR4	X		Section 6.2.3
FR5	X		Section 6.2.3
FR6	X	X	Section 6.2.4
NFR1		X	Section 6.3.1
NFR2	X		Section 6.2.5
NFR3	X		Section 6.2.6
NFR4	X		Section 6.2.7

Table 8: Results of the same models on different datasets.

our framework.

Chapter 7

Conclusion and Future Work

In this chapter, we will summarize the work and contributions we have made in this thesis, and we will also discuss the current limitations. In the end, we will point out and analyze other potential research and development paths for extending this work. A shorter summary of this work has been submitted to a special issue on graph manipulation of Science of Computer Programming [47].

7.1 Overview

In this dissertation, we have proposed a software framework solution that integrates knowledge representations for input to recommendation methods. To enable framework developers or researchers to build recommendation systems that meet their own special needs faster, and try not to change the code or change the code in a small amount. The framework can provide a variety of ways to store knowledge triples, especially side information, input the information into recommendation methods, and provide the function of predicting users' ratings of items, thus achieving the goal we set in Section 1.2.

By analyzing the requirements mentioned in Section 1.3, we designed and implemented our software framework, and have shown that in Section 3.2, that our software framework solution meets our requirements.

In the core of the framework, we provide extractor for information, knowledge

graph storage for extracted information, knowledge graph visualization and recommendation method. As mentioned in Section 3.3, the current `InfoExtractor` supports extraction of different information, say for example from the IMDB website, or it can extract different websites according to any new rules defined. The `StorageManager` framework allows programs to store knowledge graph triples in different storage modes, including RDF, RDFS, OWL and Neo4j. The knowledge graph viewer framework supports us to visualize the stored knowledge graph files. The recommendation system framework is trained based on the data we extracted with the `InfoExtractor` framework and the user's evaluation of the item, and finally carries out the function of predicting the user's rating for that item.

According to our specific needs, we instantiated the IMDBExtractor described in Section 4.1 and the Neo4jManager/RDFManager described in Section 4.2 to construct the knowledge graph, the knowledge graph viewer module described in Section 4.3, and the instantiated recommendation module described in Section 4.4 to train the model.

In order to show that our solution can meet the proposed scenarios listed in Section 1.3. We created a specific applications using the framework in Chapter 5 and evaluated and tested each integrated module one by one. The results sufficiently demonstratedshowed that all requirements were satisfactorily met (t. There are more details in Chapter 6). We additilsonally published the code on GitHhub [46], the address is as follows:
`https://github.com/OpenISS/kg-recommendation-framework` [46].

7.2 Concluding Remarks

Based on the evaluation results in the previous chapter, we summarize our research work in this section. First, we will begin by answering all the research questions raised in Section 1.5.

Question: *How to increase the accuracy of recommendation system?*

We increase the accuracy of recommendations by introducing knowledge graphs

which can provide side information for improving recommendation accuracy. Knowledge graphs contain rich semantic relationships between entities. Compared with other kinds of auxiliary information, the introduction of a knowledge graph makes it easier to systematically and uniformly provide information to the recommendation method and thus make the recommended results more precise, diverse and explainable.

Question: *Can we make the system independent of the knowledge graph storage format, say RDF, Neo4j, etc.?*

The core part of our system is the recommendation module. For the time being, the input of the recommendation module is a text file, so if the user provides the correct text file, the system can be independent of the knowledge graph storage module. But the knowledge graph storage module can help users manage and modify triples.

Question: *Can we use our solution to compare different knowledge graph storage modes to make recommendations for which storage mode is suitable for which application type?*

After our comparison, we found that Neo4j is a native graph database engine, and has a corresponding graph traversal algorithm, so its performance will not be affected as the data increases. Neo4j has a very high query performance. The structured data format of graph data structure allows Neo4j's database design to have great flexibility. So when we want to conduct quick response applications such as knowledge based question answering, we recommend using the Neo4j database. Although the query speed of RDF format storage is slower than Neo4j, because the RDF storage mode adds more description classes and property based on nodes, when we need to do some reasoning, RDF storage is more suitable for us.

Question: *Specifically, what are the advantages and disadvantages of different storage modes?*

The advantage of Neo4j is that it is faster to read and write. Unstructured data storage has great flexibility in database design. Easy to use and easy to model. The disadvantage is that the community version is free and open-source, but the cluster

cannot be used, and it supports up to 32 billion nodes. The Enterprise Edition can avoid these problems, but it is expensive.

The advantage of RDF storage is that RDF can keep the semantic information of the data to the maximum. We can learn from mature graph algorithms and graph databases to design RDF data storage schemes and query algorithms. The shortcomings of RDF are also obvious. The design is not flexible enough. When new attributes or data are added, the network needs to be reconstructed. Needs large storage space. Since there is no corresponding graph query engine, the query algorithm has high time complexity.

Question: *Is side information useful for improving accuracy?*

We used the same model to compare the results before and after adding side information in the movie recommendation domain. For side information here, we use user information, including the user's age, user's gender, and user's job category. Movie information, includes the director, actor and author of the movie.

The accuracy of the baseline without side information is AUC: 0.9046 ACC: 0.8277. After only adding movie information, the accuracy rate reached AUC: 0.9081 ACC: 0.8295. After adding only user information, the accuracy rate reached AUC: 0.9096 ACC: 0.8321. After using all the informations, the accuracy is AUC: 0.9142 ACC: 0.8375. We did not include accuracy in our non-functional requirement, because the framework is always accurate, but it allows us to test algorithm implementation for accuracy. It's a property of a user application rather than a property of a framework component.

Question: *How to get information about movies from the website?*

Our framework includes the function of extracting information from the web, and we wrap these codes into functions for users to call. This has been explained in detail through figures and text in Section 4.1.

7.3 Limitations

Even if the solution proposed in this article achieves the goals we set in Section 1.2 and meets the requirements we listed in Section 1.3, we must admit that there are still some limitations in our solution:

- Our recommendation framework includes `DataPreprocessor` and `DataLoader`, but the output of both parts are text files. Although it can be helpful for the framework developer to check or debug, there will be redundancy problems.
- Our framework reads all data into the memory when reading data, but if the amount of data exceeds the memory, an error will occur. We already have a solution, but have not yet updated it in our framework. We can use chunksize in pandas to read large files.
- Our framework uses the name of the item, say movie name, to search when creating a new node. If the name does not exist, then it generates a new node. If we use a public data set, there will be no problem. But if the user has his data set, the movie names in the data set may be different from those in the knowledge graph. If this is the case, different nodes of the same movie will occur.
- We haven't done many tests on the scalability of the framework for the time being. For example, when the triples we train increase, our training speed and prediction speed will increase in a linear or exponential way.
- Our current framework solution can only support storage when the input is triples or CSV/txt files. Other input formats such as JSON format or TTL format are currently not supported, which hinders our solution from achieving better scalability.
- The IMDBExtractor framework we currently implement is based on the IMDB website, so users need to define their own rules when extracting other website information, which is additional effort for users.

- Compared with the original MKR model, we added several layers of frozen points and hot spots, which increased the code running time, but we did not compare the running time difference between the original model and the model in our framework.

- We trained our current recommendation model on a single dataset. Although we have extracted more data through the model, it would be better if there are more movies and user information and user review information. If we train the model on multiple datasets, the model can learn more general features, making them more accurate in practice.

- Our framework requires a lot of dependencies, and users of the framework may need to install a lot of libraries. We may need to develop some scripts to help developers configure the environment.

- Our current viewer model is mainly based on the `networkx` library. Using this model, it can only expand all nodes at once. If there are many nodes, the processing speed will be slow. And it is not easy to find the corresponding node. However, the visualization interface of Neo4j desktop and protege can independently expand a node, and they have human-computer interaction operations. For example, double-click the node to expand the node and display all the nodes connected to the node.

7.4 Future Work

The ultimate goal of such work is to make it as a research platform for more developers in the recommendation systems field. Aside from addressing the limitations stated in the previous section, we list several future work items in more detail:

Java API wrapper: As discussed in Chapter 4, our framework was written in Python, but the movie recommendation system is mostly used on web pages, so to be better used by users, we must provide a Java wrapper for our API.

Support different machine learning backend: Currently, our recommended module only supports TensorFlow. But there are many different deep learning frameworks, such as PyTorch, Caffe, MindSpore, or Scikit-learn. Different frameworks have their advantages. We plan to add various machine learning frameworks to our framework in the future.

Support more storage methods and more input formats: Currently, we only support four storage formats, namely RDF, RDFS, OWL and Neo4j. The input format is triples or CSV/txt files. But with the advancement of technology, there will be a storage mode with faster storage or a storage mode with faster queries. For the input format, because we use CSV for storage, some users may choose JSON format or TTL format, so we also need to update the program to support these formats.

More effective loss function: In this thesis, only the cross entropy loss and the L2 loss are used for this recommendation system task. In the future, there may be a loss function that is more suitable.

Full-platform support: Until now, our solution only applies to Linux and macOS. In the future, we will provide support for more operating systems.

Auto installation: Currently, the installation process of our framework is manual. To make it more convenient to use the installation procedure needs to be automated. We plan to write install scripts to automate the installation of software and required libraries.

Bibliography

[1] Aamir, M., and Bhusry, M. Recommendation system: state of the art approach. *International Journal of Computer Applications 120*, 12 (2015).

[2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Abu Kausar, M., Dhaka, V., and Singh, S. Web crawler: A review. *International Journal of Computer Applications 63* (02 2013), 31–36.

[4] Adam Murray. How knowledge graphs boost the benefits of analytics. https://www.sisense.com/blog/how-knowledge-graphs-boost-the-benefits-of-analytics. Accessed: 2020-02-12.

[5] Adomavicius, G., and Tuzhilin, A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering 17*, 6 (2005), 734–749.

[6] Adomavicius, G., and Tuzhilin, A. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering 17*, 6 (June 2005), 734–749.

[7] Aijal, J. What is a knowledge graph and how does one work? https://thenextweb.com/podium/2019/06/11/what-is-a-knowledge-graph-and-how-does-one-work/. Accessed: 2019-12-09.

[8] Akash Tandon. Reconciling your data and the world with knowledge graphs. https://towardsdatascience.com/reconciling-your-data-and-the-world-with-knowledge-graphs-bce66b377b14. Accessed: 2020-07-27.

[9] Almazro, D., Shahatah, G., Albdulkarim, L., Kherees, M., Martinez, R., and Nzoukou, W. A survey paper on recommender systems, 2010.

[10] Arora, S. A survey on graph neural networks for knowledge graph completion, 2020.

[11] Auer, S., Kovtun, V., Prinz, M., Kasprzik, A., Stocker, M., and Vidal, M. E. Towards a knowledge graph for science. In *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics* (2018), pp. 1–6.

[12] Bassil, Y. A survey on information retrieval, text categorization, and web crawling, 2012.

[13] Berg, R. v. d., Kipf, T. N., and Welling, M. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[14] Besta, M., Peter, E., Gerstenberger, R., Fischer, M., Podstawski, M., Barthels, C., Alonso, G., and Hoefler, T. Demystifying graph

databases: Analysis and taxonomy of data organization, system designs, and graph queries, 2019.

[15] Bobadilla, J., Ortega, F., Hernando, A., and Rrez, G. Recommender systems survey. *Know.-Based Syst.* (July 2013).

[16] Brachman, R. J., and Levesque, H. J. *Readings in Knowledge Representation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1985.

[17] Cai-Nicolas Ziegler, D. F. Book-crossing dataset. [online], 2004. http://www2.informatik.uni-freiburg.de/~cziegler/BX/.

[18] Caruso, F., Giuffrida, G., and Zarba, C. Subjective collaborative filtering, 2011.

[19] Chah, N. Ok google, what is your ontology? or: Exploring freebase classification to understand google's knowledge graph, 2018.

[20] Cheng, L., Zhang, Y., Wu, D., Jie, Z., Bing, L., Lu, W., and Si, L. Knowledge graph empowered entity description generation, 2020.

[21] Chong, D. Deep dive into netflix recommender system. https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48. Accessed: 2016-04-30.

[22] Christophides, V. *Resource Description Framework (RDF) Schema (RDFS)*. Springer US, Boston, MA, 2009, pp. 2425–2428.

[23] Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (New York, NY, USA, 2016).

[24] DB-ENGINES. Db-engines ranking of graph dbms. [Online; accessed 25-May-2020], 2020. https://db-engines.com/en/ranking/graph+dbms.

[25] DB-ENGINES. Db-engines ranking of rdf stores. [Online; accessed 25-May-2020], 2020. https://db-engines.com/en/ranking/rdf+store.

[26] Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S., and Zhang, W. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), pp. 601–610.

[27] Edwin, N. M. Software frameworks, architectural and design patterns. *Journal of Software Engineering and Applications 2014* (2014).

[28] Ekstrand, M. D., Riedl, J. T., and Konstan, J. A. *Collaborative filtering recommender systems*. Now Publishers Inc, 2011.

[29] Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H., and He, Q. A survey on knowledge graph-based recommender systems, 2020.

[30] Hanika, T., Marx, M., and Stumme, G. Discovering implicational knowledge in wikidata, 2019.

[31] Heist, N., Hertling, S., Ringler, D., and Paulheim, H. Knowledge graphs on the web – an overview, 2020.

[32] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., Gayo, J. E. L., Kirrane, S., Neumaier, S., Polleres, A., et al. Knowledge graphs. *arXiv preprint arXiv:2003.02320* (2020).

[33] Huang, X., Fang, Q., Qian, S., Sang, J., Li, Y., and Xu, C. Explainable interaction-driven user modeling over knowledge graph for sequential recommendation. In *Proceedings of the 27th ACM International Conference on Multimedia* (2019), pp. 548–556.

[34] Huang, X., Zhang, J., Li, D., and Li, P. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (2019), pp. 105–113.

[35] Hug, N. SurPRISE scikit for recommender systems. [online], 2016–2020. http://surpriselib.com/.

[36] Jenson, G. GER's anatomy: How to generate good enough recommendations. [online], Dec. 2014. https://maori.geek.nz/gers-anatomy-how-to-generate-good-enough-recommendations-2ad21ed58f6f.

[37] Jenson, G. Good enough recommendations (GER) recommendation engine. [online], 2014–2015. https://github.com/grahamjenson/ger.

[38] Ji, S., Pan, S., Cambria, E., Marttinen, P., and Yu, P. S. A survey on knowledge graphs: Representation, acquisition and applications, 2020.

[39] Kendall Clark. Turning data into knowledge graphs. https://www.forbes.com/sites/forbestechcouncil/2020/04/28/if-you-want-to-deal-with-data-silos-embrace-them/#4cd75bf7443d. Accessed: 2018-03-09.

[40] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR abs/1609.04836* (2016).

[41] Lai, H. An openiss framework specialization for deep learning-based person re-identification. Master's thesis, Concordia University, 2019.

[42] Lawan, A., and Rakib, A. The semantic web rule language expressiveness extensions-a survey, 2019.

[43] Lee, N., Torr, P. H., and Jaggi, M. Data parallelism in training sparse neural networks. *arXiv preprint arXiv:2003.11316* (2020).

[44] LensKit Project. LensKit recommendation framework. [online], 2018–2021. https://github.com/lenskit/lkpy.

[45] Lopes, N., Polleres, A., Passant, A., Decker, S., Bischof, S., Berrueta, D., Campos, A., Corlosquet, S., Euzenat, J., Erling, O.,

ET AL. Rdf and xml: Towards a unified query layer. In *Proc. W3C workshop on RDF next steps* (2010), pp. No–pagination.

[46] MAO, Y. Knowledge graph-based deep learning rrecommendation system framework. [online], Jan. 2021. https://github.com/OpenISS/kg-recommendation-framework.

[47] MAO, Y., MOKHOV, S. A., AND MUDUR, S. P. Application of knowledge graphs to provide side information for improved recommendation accuracy, 2021. https://arxiv.org/abs/2101.03054, submitted for review to Science of Computer Programming.

[48] MCFEE, B., BARRINGTON, L., AND LANCKRIET, G. Learning content similarity for music recommendation, 2011.

[49] MEDIUM. Introduction to Web Crawling and Scraping. [Online; accessed 13-April-2018], 2018. https://medium.com/@allisonmorgan/short-essay-on-web-crawling-scraping-8abf1b232b65.

[50] MENG, Y., CHEN, G., LIAO, B., GUO, J., AND LIU, W. Wasserstein collaborative filtering for item cold-start recommendation, 2019.

[51] MERZI, O. Introduction to knowledge graphs. https://medium.com/analytics-vidhya/introduction-to-knowledge. Accessed: 2019-12-09.

[52] MIZGAJSKI, J., AND MORZY, M. Affective recommender systems in online news industry: How emotions influence reading choices. *User Modeling and User-Adapted Interaction* (Apr. 2019).

[53] MOONEY, R. J., AND ROY, L. Content-based book recommending using learning for text categorization, 1999.

[54] MORITA, G. recommendationRaccoon (raccoon). [online], 2016–2018. https://github.com/guymorita/recommendationRaccoon#recommendationraccoon-raccoon.

[55] Namaki, M. H., Chowdhury, F. A. R. R., Islam, M. R., Doppa, J. R., and Wu, Y. Learning to speed up query planning in graph databases, 2018.

[56] Narula, G. S. Ontology mapping and merging aspects in semantic web. *International Robotics and Automation Journal 4* (01 2018).

[57] Neo4j. Cypher query language. https://neo4j.com/developer/cypher-query-language. Accessed: 2020-04-01.

[58] Nickel, M., Murphy, K., Tresp, V., and Gabrilovich, E. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE 104*, 1 (2015), 11–33.

[59] Ning, Y., Shi, Y., Hong, L., Rangwala, H., and Ramakrishnan, N. A gradient-based adaptive learning framework for efficient personal recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems* (2017), pp. 23–31.

[60] Nzeko'o, A. J. N., Tchuente, M., and Latapy, M. A general graph-based framework for top-n recommendation using content, temporal and trust information. *arXiv preprint arXiv:1905.02681* (2019).

[61] Paulheim, H. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web 8*, 3 (2017), 489–508.

[62] Pazzani, M. J., and Billsus, D. Content-based recommendation systems. In *The adaptive web*. Springer, 2007, pp. 325–341.

[63] Pree, W. Meta patterns - a means for capturing the essentials of reusable object-oriented design. In *ECOOP* (1994).

[64] Qu, Y., Bai, T., Zhang, W., Nie, J., and Tang, J. An end-to-end neighborhood-based interaction model for knowledge-enhanced recommendation, 2019.

[65] Rastin, N., and Jahromi, M. Z. Using content features to enhance performance of user-based collaborative filtering performance of user-based collaborative filtering, 2014.

[66] REJOINER. The amazon recommendations secret to selling more online. http://rejoiner.com/resources/amazon-recommendations-secret-selling-online. Accessed: 2018-03-27.

[67] Rodriguez, A. Youtube recommendations drive 70 percent of what we watch. https://qz.com/1178125/youtubes-recommendations-drive-70-of-what-we-watch/. Accessed: 2018-01-13.

[68] Rodriguez Garcia, M. A., and Hoehndorf, R. Inferring ontology graph structures using owl reasoning. *BMC bioinformatics 19*, 1 (2018), 7.

[69] Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. Collaborative filtering recommender systems. In *The adaptive web*. Springer, 2007, pp. 291–324.

[70] Shang, M.-S., Zhang, Z.-K., Zhou, T., and Zhang, Y.-C. Collaborative filtering with diffusion-based similarity on tripartite graphs, 2009.

[71] Shinavier, J. Optimizing real-time rdf data streams, 2010.

[72] Siddiqui, F., and Alam, M. A. Web ontology language design and related tools: a survey. *Journal of Emerging Technologies in Web Intelligence 3*, 1 (2011), 47–59.

[73] Singh, J. Universal gesture tracking framework in OpenISS and ROS and its applications. Master's thesis, Concordia University, Jan. 2020.

[74] Sun, Z., Guo, Q., Yang, J., Fang, H., Guo, G., Zhang, J., and Burke, R. Research commentary on recommendations with side information: A survey

and research directions. *Electronic Commerce Research and Applications 37* (2019), 100879.

[75] Tomaszuk, D., and Hyland-Wood, D. Rdf 1.1: Knowledge representation and data integration language for the web, 2020.

[76] Wang, H., Zhang, F., Wang, J., Zhao, M., Li, W., Xie, X., and Guo, M. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems, 2018.

[77] Wang, H., Zhang, F., Xie, X., and Guo, M. Dkn: Deep knowledge-aware network for news recommendation, 2018.

[78] Wang, H., Zhang, F., Zhang, M., Leskovec, J., Zhao, M., Li, W., and Wang, Z. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems, 2019.

[79] Wang, H., Zhang, F., Zhao, M., Li, W., Xie, X., and Guo, M. Multi-task feature learning for knowledge graph enhanced recommendation, 2019.

[80] Wang, P., Jiang, H., Xu, J., and Zhang, Q. Knowledge graph construction and applications for web search and beyond. *Data Intelligence 1*, 4 (2019), 333–349.

[81] Wang, Q., Mao, Z., Wang, B., and Guo, L. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering 29*, 12 (2017), 2724–2743.

[82] Wang, X., He, X., Cao, Y., Liu, M., and Chua, T.-S. Kgat: Knowledge graph attention network for recommendation, 2019.

[83] Wang, X., Peng, Z., Wang, S., Yu, P. S., Fu, W., and Hong, X. Cross-domain recommendation for cold-start users via neighborhood based feature mapping, 2018.

[84] Wang, X., Wang, D., Xu, C., He, X., Cao, Y., and Chua, T.-S. Explainable reasoning over knowledge graphs for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 5329–5336.

[85] Wang, X. H., Zhang, D. Q., Gu, T., and Pung, H. K. Ontology based context modeling and reasoning using owl. In *IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second* (2004), pp. 18–22.

[86] Wei, J., He, J., Chen, K., Zhou, Y., and Tang, Z. Collaborative filtering and deep learning based recommendation system for cold start items. *Expert Systems with Applications 69* (2017), 29–39.

[87] Wikipedia. Knowledge graph — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Knowledge_Graph. Accessed: 2012-05-01.

[88] Wikipedia. Recommender system — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Recommender_system. Accessed: 2020-07-27.

[89] Wikipedia. Web crawler — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Web_crawler. Accessed: 2020-07-27.

[90] Wikipedia. Software framework. [online], wikipedia, 2006–2011. https://en.wikipedia.org/wiki/Software_framework.

[91] Wikipedia contributors. Arangodb — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=ArangoDB&oldid=947546389, 2020. [Online; accessed 3-August-2020].

[92] Wu, Q., Zhang, H., and Zha, H. Inductive relational matrix completion, 2020.

[93] Zhang, S., Yao, L., Sun, A., and Tay, Y. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR) 52*, 1 (2019), 1–38.

[94] Zhao, Z., Han, S.-K., and So, I.-M. Architecture of knowledge graph construction techniques. *International Journal of Pure and Applied Mathematics 118*, 19 (2018), 1869–1883.

Appendix A

Towards a Modular System for Gu Zheng Instrument Using OpenISS Core

This application is used in conjunction with issv2. ISSv2 uses OpenISS to extract the position of the hand in the camera. The pitch value is obtained by compressing the coordinates of the hand. The system obtains a series of pitch values through the userś wave of hands and finishes generating pitch values through a push operation. The pitch value is passed from Processing to Python via OSC. Python calls the model trained in machine learning to generate the pitch value and then uses the music player library to play the generated pitch value through the music font of Guzheng. When ISSv2 detects a push operation, it will pass instructions through OSC to make python stop playing. Figure 79 is a screenshot of the demo.

Appendix B

Visualizing and Interacting with Forensic Evidence Using OpenISS

Open Illimitable Space System, is primarily designed to serve as the open-source core for this real product experience. In the process of HCI, OpenISS provides the ability to interact with different 3D visual devices or take several data inputs to provide and calculate real-world information for a software system to analyze. The OpenISS core can capture images, analyze motions, map coordinates and more functions to help developers and users to convert real-world information to structured data. Forensic Lucid programming language is used for analyzing cyber forensics. FLucid provides a set of formulated definitions, and by treat the data of an object as observations, it is easy to analyze the relationships between objects by linking the observations as a sequence. By triggering the compilation of FLucid, it is easy to analyze and see the relationships between virtual items. it can easily know what happens when the user interacts with the objects and generates new images or other prompts then sends back to the user. Figure 69 is a screenshot of the demo. Figure 80 is a screenshot of the demo.

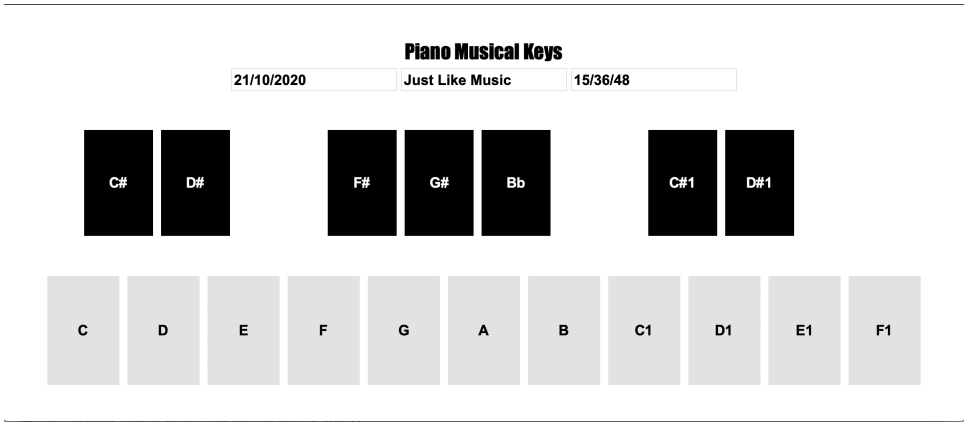

Figure 79: Screenshot of the interface of the demo.

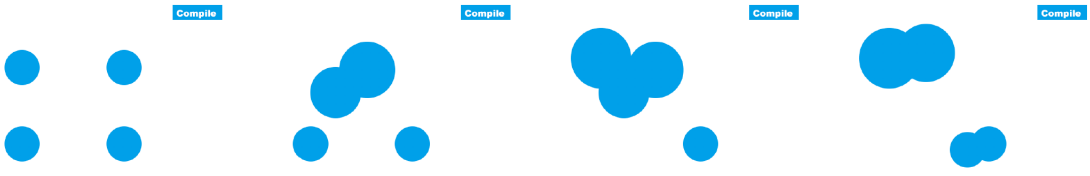

Figure 80: Screenshot of OpenISSFLucid Demo.