big data and cognitive computing

MDPI

Article

Context-Aware Explainable Recommendation Based on Domain Knowledge Graph

Muzamil Hussain Syed [1,*], Tran Quoc Bao Huy [2] and Sun-Tae Chung [2,3]

[1] Department of Information and Telecommunication, Graduate School, Soongsil University, Seoul 06978, Korea
[2] Department of Intelligent Systems, Graduate School, Soongsil University, Seoul 06978, Korea; huy.tsusak@gmail.com (T.Q.B.H.); cst@ssu.ac.kr (S.-T.C.)
[3] School of Artificial Intelligence Convergence, Soongsil University, Seoul 06978, Korea
* Correspondence: engr.muzamilshah@gmail.com

Abstract: With the rapid growth of internet data, knowledge graphs (KGs) are considered as efficient form of knowledge representation that captures the semantics of web objects. In recent years, reasoning over KG for various artificial intelligence tasks have received a great deal of research interest. Providing recommendations based on users' natural language queries is an equally difficult undertaking. In this paper, we propose a novel, context-aware recommender system, based on domain KG, to respond to user-defined natural queries. The proposed recommender system consists of three stages. First, we generate incomplete triples from user queries, which are then segmented using logical conjunction ($\wedge$) and disjunction ($\vee$) operations. Then, we generate candidates by utilizing a KGE-based framework (Query2Box) for reasoning over segmented logical triples, with $\wedge$, $\vee$, and $\exists$ operators; finally, the generated candidates are re-ranked using neural collaborative filtering (NCF) model by exploiting contextual (auxiliary) information from GraphSAGE embedding. Our approach demonstrates to be simple, yet efficient, at providing explainable recommendations on user's queries, while leveraging user-item contextual information. Furthermore, our framework has shown to be capable of handling logical complex queries by transforming them into a disjunctive normal form (DNF) of simple queries. In this work, we focus on the restaurant domain as an application domain and use the Yelp dataset to evaluate the system. Experiments demonstrate that the proposed recommender system generalizes well on candidate generation from logical queries and effectively re-ranks those candidates, compared to the matrix factorization model.

Keywords: domain knowledge graph; natural language query; recommendation system

Citation: Syed, M.H.; Huy, T.Q.B.; Chung, S.-T. Context-Aware Explainable Recommendation Based on Domain Knowledge Graph. *Big Data Cogn. Comput.* **2022**, *6*, 11. https://doi.org/10.3390/bdcc6010011

Academic Editors: Konstantinos Kotis, Dimitris Spiliotopoulos and Min Chen

Received: 14 November 2021
Accepted: 13 January 2022
Published: 20 January 2022

1. Introduction

Recommender systems provide personalized recommendations for a set of products or items that may be of interest to a particular user [1]. In today's digital world, recommender systems have shown to be extremely valuable and essential tools. Numerous applications, ranging from e-commerce to social networks, are developed with enhanced algorithms to make intelligent recommendations [2–4]. Although numerous efforts have been made toward more personalized recommendations, these recommender systems remain unable to address context and other challenges, such as data sparsity and cold start problems. Recently, recommendations based on the knowledge graph (KG) have attracted considerable interest as a source of context information. This approach not only alleviates the problems mentioned above for a more accurate recommendation but also provides explanations for recommended items [5–7]. The KG is a graph representation of real-world knowledge, whose nodes represent entities and edges illustrate the semantic relation between them [8]. KGs explain the semantic and attribute relationships between concepts, in order to facilitate reasoning about them. Moreover, node and edge vectors derived from KG embedding (KGE), where semantics of KG are preserved, are used for training and inferencing machine

learning (ML) models, which is useful for analytics, deeper queries, and more accurate recommendation. However, processing and making an efficient context-aware recommender system based on user-defined complex queries, such as "*Recommend best Chinese restaurants in Toronto which serve sweet Noodles or Spicy Chicken Biryani*", on large scale incomplete KGs still remains a challenging task. Finding candidate entities that satisfy queries can be approached by reasoning on KG.

Reasoning is the process of consciously applying logic to draw new conclusions from new or existing information. Logical reasoning mainly relies on first-order predicate logic (FOL), which uses propositions as the basic unit for reasoning. Simple propositions in FOL are declarative sentences that do not contain a connection and are represented in a simple form of triple (h, r, t). A simple query is a one-hop triple query, in the form of $(v_?, r, t)$, where $v_?$ denotes the target entity answering the query (e.g., *which restaurant has noodles on the menu?*) or $(h, r, v_?)$ (e.g., *which menu does the Alps restaurant serve?*). Complex queries in FOL can be decomposed into combinations of simple queries connected by logical operations (*AND* ($\wedge$), *OR* ($\vee$)) with an existential quantifier (*there exists;* $\exists$) and allows *negation* ($\neg$). Link prediction on the KG embedding space can be used to locate target entities that satisfy simple queries. Thus, complex queries consisting of any combination of simple queries with the same target entity can be answered by set operations (intersection, union) on sets of target entities that answer each simple query. However, complex queries that involve intermediate unknown entities to answer target entities, such as "*Which famous dishes Chinese restaurants serves in Toronto?*", illustrated in Figure 1, cannot be handled by link prediction.

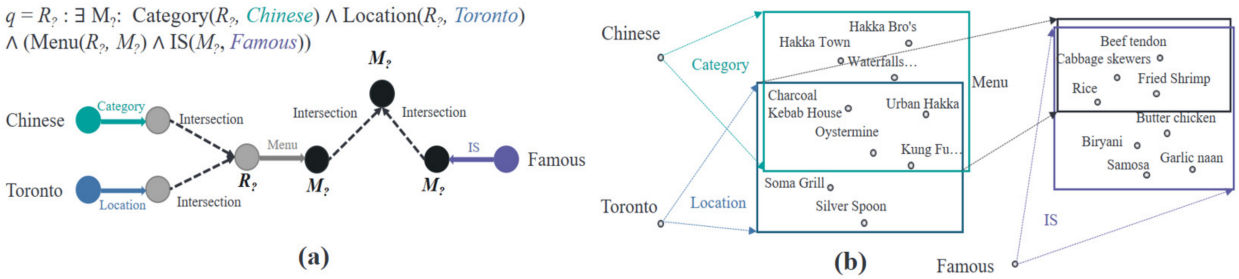

Figure 1. FOL query and its (**a**) computation graph and (**b**) vector space representation for a domain specific natural query "*Which famous dishes Chinese restaurants serves in Toronto?*".

In this work, we propose a novel, context-aware recommender system based on user-defined complex queries. Context-aware recommendation is accomplished by translating natural language queries into complex logical queries, reasoning over KGE space to identify candidate entities satisfying complex logical queries, and then re-ranking those candidates by incorporating contextual information. As for a concrete setting of the proposed system, we construct a KG for the restaurant domain and utilize the Yelp [9] review dataset to perform experiments. The proposed recommender system consists of three modules: (i) the triple generator module: generates logical triple segments from natural query; (ii) the candidate generator module: generates candidate item set with a *relevance score* (z), using the Query2Box (Q2B) model [10], which embeds logical queries, as well as KG entities, into a low-dimensional vector space, such that entities that satisfy the query are embedded close to the query in a hyper-rectangle boxes—candidate finder locates entities that satisfy or closely approximate existential positive first order (EPFO) logical queries, by reasoning over the KGE space and logical operations ($\wedge$, $\vee$, and $\exists$), which are processed by set operations (intersection, union and projection); and (iii) the ML-based re-ranker module: to incorporate interaction and content features for personalized recommendation—similar to the factorization machine [11], we train a neural collaborative filtering (NCF) model [12] and feed the contextual embeddings obtained from the GraphSAGE [13], which incorporates user and item node and attribute information to generate contextual embedding. The final candidate items, with relevance scores z, obtained in the second step, are further

passed through the trained NCF model to predict a *ranking score* (r) between a pair of user and candidate items and re-rank the candidates by calculating a weighted average between ranking (r) and relevance scores (z). Since our approach considers the content and collaborative features for recommendation, and KG provides a natural explanation of the recommended candidates, the proposed approach is more user-friendly and produces accurate explainable recommendations based on defined queries.

For reasoning over EPFO logical queries, we utilize the Q2B model [10]. We prepare a domain-specific, ontology-based triple dataset, from the Neo4j graph database, for training and reasoning over the Q2B model. However, the Q2B model does not support the conversion of complex natural queries into the EPFO logical query. In this work, we provide the generation of EPFO logical queries from user-defined queries.

To the best of our knowledge, there are only a few research works that address context-aware recommendation by incorporating side features based on query expansion. However, prior works lack in handling and providing recommendation based on user-defined natural language queries.

The following is the demonstration of the major contributions of this work:

- We propose a novel framework for providing context-aware explainable recommendations based on domain KG by utilizing the existing model and embedding framework.
- We carefully design the domain ontology to capture the semantic meaning of entities and relations to make relevant recommendations, in response to user queries.
- We develop a template-based framework to transform users' natural queries into logical triple segments and extract entity concepts and their relationships using the knowledge base.

The remainder of the paper is structured as follows. In Section 2, we review the related work. In Section 3, we present the implementation and techniques of the proposed system and its components. In Section 4, we describe the data, system evaluation, and experiment results. Finally, in Section 5, we present the conclusion of the paper.

2. Related Work

Substantial work has been conducted on contextual recommendation based on KGs. Our system is built on a foundation of prior research and query-based logical reasoning.

2.1. Knowledge Graph (KG) and Knowledge Graph Embedding (KGE)

The knowledge graph (KG) is a graph representation of real-world knowledge, whose nodes represent entities and edges that illustrate the semantic relation between them [8]. A KG is based on the facts that are typically represented as "SPO" triples (subject entity, predicate (relation), and object entity) and are denoted as (h, r, t) or r (h, t). A domain KG defines the entities and relationships of a specific domain. Constructing a domain KG entails creating ontology, designing rules, extracting relations, and storing semantic data. The KG has revolutionized how information is retrieved, in the traditional sense [14]. Additionally, KGs explain the semantic and attribute relationships between concepts, in order to facilitate reasoning about them. One of the key benefits of a KG is that it can easily integrate the newly discovered facts from the information retrieval process, which captures more detailed facts and attributes on an ongoing basis and significantly resolves the KGs incompleteness problem. Moreover, structured data, in the form of KG triples, reveals numerous interesting patterns that are useful for analytics, deeper queries, and more accurate recommendation. KG is effectively used for a variety of crucial AI tasks, including semantic search [15], intelligent question answering [16], and recommendation systems [5,6].

Formally, a KG is defined as follows: $G = \{(sub, pred, obj)\} \subseteq E \times R \times E$ is a set of $(sub, pred, obj)$ triples, each including a subject $sub \in E$, predicate $pred \in R$, and object $obj \in E$. E and R are the sets of all entities and relation types of G [17]. KGE models embed entities E and relations R into a low-dimensional real or complex vector space, while preserving the structure of the KG and its underlying semantic information [18].

Such KGE models have proven to be extremely useful for a range of prediction and graph analysis tasks [11]. KGE models are classified into three categories: translational distance-, semantic matching-, and neural network-based models. Translational distance models or additive models TransE [19], TransH [20], and TransR [21], use distance-based scoring functions to calculate the similarity between the different entities and relations, thus build the embedding accordingly [22]. On the other hand, semantic matching models, or multiplicative models RESCAL [23], DistMult [24], and ComplEx [25], employ a similarity-based scoring function. In translational-based models, given a triple (h, r, t), the relation r translates the head entity h to the tail entity t. These models define a scoring function to measure the correctness of the triple in the embedding space. However, these models are reported to have low expressive power and do not capture semantic information [18]. The multiplicative models outperform the additive models by capturing more semantic information [18]. The third category includes models built on graph neural networks (GNNs), such as ConvE [26], ConvKB [27], and GraphSAGE [13]. These models consider the type of entity or relation, temporal information, path information, and substructure information [18]. Among these models, GraphSAGE is a framework for learning inductive representations on large graphs [13]. Furthermore, it achieves low-dimensional vector representations of nodes, while utilizing their attribute information, which makes GraphSAGE more appropriate for generating contextual embedding for recommendation and prediction tasks than any other KGE models. Thus, we adopt this model for obtaining user and item latent vector for training and inferencing our NCF model.

2.2. Recommendation

Two main techniques, collaborative filtering (CF) and content-based filtering (CBF) are traditionally used for recommendation [28]. The CF approach recommends items for a user by predicting the ranking score $\hat{r}(u, i)$ of item (i), based on users' (u) profiles, common preferences, and historical interactions. A number of research efforts have been devoted to the CF approach for implementing efficient recommender systems that utilize user and item side information, by transforming them into feature vectors to predict the rating score. Matrix factorization (MF) [29] is a well-known collaborative filtering technique that projects users and items into a shared latent space using a latent feature vector. Thereafter, the interaction between a user and item is modelled as the inner product of their latent vectors. However, CF techniques usually suffer from data-sparse and cold-start problems. Numerous studies have been conducted to improve MF [12,29–33] and addressing the cold-start problem by incorporating content (side) features to represent the user and item in latent space. Therefore, exploiting context information has great importance in resolving such problems and enhancing recommender systems. The conventional recommendation methods (CRMs) recommend items for a given user and utilize context information, such as *When?* (day, night, weekday, weekend, etc.), *What conditions?* (whether, climate, region, etc.), and with *Whom?* (family, lover, or friend). Given this information, the CRMs learn more contextual prediction rating for recommendation between user and item. However, these methods are incapable of adequately capturing the context information and are lacking in handling and providing context-aware explicable recommendations based on user-defined natural queries. Our system is capable of capturing context information from users' queries and recommending by reasoning over KG, as well as employing contextual information, similar to that found in CRMs, to add more context and re-rank the recommended candidates.

Recent studies have started to consider KGs as a source of contextual information, since they can provide valuable context information for recommendation. For example, entities, and their associated attributes, can be incorporated and mapped into the KG, in order to comprehend their mutual relationships [34]. Additionally, the heterogeneous relations in a KG help to improve the accuracy of recommender systems and increase the diversity of recommended items. Moreover, KGs facilitate the interpretation of recommender systems. In general, the majority of the known techniques for developing KG-based recommender

systems can be classified into two categories: path- and knowledge graph embedding (KGE)-based approaches. Path-based methods build a user-item graph and utilize it to discover path-level similarity for items, either by predefining meta-paths or mining connective patterns automatically. However, the disadvantage of these methods is that they require domain knowledge to specify the types and number of meta-paths [35]. On the other hand, KGE-based methods expand the representation of entities and relations by embedding them in a continuous vector space. Hence, the representation of entities, and their associated relations, can be obtained from KGE space. These methods typically require a two-module approach for recommendation systems (RS) [18,22,35–37]: a KGE module for obtaining the user and item latent vectors and recommendation module for inferring recommendation (usually by computing candidate ranking score from user and item latent vectors). The advantages of these methods lie in the simplicity and scalability; however, since the two modules are loosely coupled, the approach might not be suitable for recommendation tasks. There are a few research works that deals with recommendations based on query context. Sitar-Tăut et al. [38] suggested a knowledge-driven product recommendation, using a digital nudging mechanism, to tackle the cold-start problem. They utilized a KG to integrate managerial preferences, along with product attributes that enable semantic reasoning using SPARQL queries. Zhu et al. [39] addressed the context of interactive recommendation in e-commerce by utilizing user interaction history with the e-commerce site. Bhattacharya et al. [40] utilized queries as context and found candidate items by collecting similar items while the user was searching for a target item. For candidate items, [40] calculated recommendation rank scores via a gradient boosted decision tree (GBDT). Both [39] and [40] dealt with query context by user interaction and query expansion, which is different from the context defined by a user's queries in natural language, as proposed in this paper.

2.3. Semantic Search and Reasoning over KG

FOL queries can be expressed as directed acyclic graphs (DAGs) and reasoned over KG to obtain a set of candidates. A conjunctive query performs a conjunction operation ($\wedge$) between two one-hop queries, such as $(v_?, r_1, t) \wedge (h, r_2, v_?)$. The path query, on the other hand, needs to traverse the path in a KG, such as "$\exists v\colon (v_?, r_1, v) \wedge (v, r_2, t)$" or "$(\exists v\colon (v, r_1, v_?) \wedge (v, r_2, t)$", where $v_?$ denotes the target entity, v denotes an intermediate unknown entity with a known type, and $\exists$ denotes the existential quantifier. One concrete example of a path query is, "*Where did Turing Award winners graduate?*" ($\exists v\colon$ (*v, Graduate*, $v_?$) $\wedge$ (*v, Win, Turing Award*). EPFL query allows disjunction ($\vee$) of queries, in addition to conjunction ($\wedge$) with existential quantifier $\exists$.

With the evolution of KGs, query-based methods for recommendations [10,16,19,41] have become a more intriguing topic of research. These approaches decouple entities and concepts from the query and create recommended candidates from KGs. We build our recommender framework on the concept of semantic search and reasoning over KG, since this enables the generation of explainable and context-aware query-based recommended candidates. Pirro [42] developed RECAP, based on a similar concept; the tool utilizes RDF and SPARQL for determining explainable knowledge for a given pair of entities in KG. Zhu et al. [15], Feddoul et al. [43], and Yan et al. [44] proposed searching techniques using query processing and expansion over KG. For an incomplete knowledge base (KB), some real-world queries fail to answer. The purpose of query embedding (QE) on KGs is to represent KB queries in an embedding space. A promising approach to this problem is to embed KG entities, as well as the query, into a vector space, such that entities that answer the query are embedded close to the query [34]. GQE (graph query embedding) [45] embeds graph nodes in a low-dimensional space and represents logical operators ($\wedge$ with$\exists$), modeled as learned geometric operations (e.g., translation and rotation). Ren et al. [10] further extended the concept and proposed Q2B to represent logical queries as hyper-rectangles, instead of points in a KGE vector space. Geometrical operations on those rectangles allowed for answering queries with disjunction ($\vee$), re-written in the disjunctive

normal form (DNF). This technique of modeling the answering entities of logical queries resolves the candidate generation problem for each incomplete triple and finds candidate entities of a given query, without traversing the KG. Additionally, the Q2B model [10] is capable of handling EPFO logical queries. Changing the query representation form to beta distributions enabled Beta-E [41], which can further tackle queries with negation (¬) and, hence, is able to handle FOL queries. However, since our recommendation framework is constrained to a certain template for parsing natural language queries into logical triple segments, and since the domain ontology graph lacks deeper linkages, it is not possible to train and answer very complex and arbitrary queries. Therefore, we use the Q2B model [10] to train the KG triples with limited query structures, as described in the original paper.

2.4. Translating Natural Language Query to Triple

In order to process users' queries expressed in natural language, we need to convert them into triples that reflect the semantic structures of domain KGs, based on defined ontology. PAROT [46] provides a dependency-based framework for converting user's queries into user and ontology triples to construct SPARQL queries. The framework processes compound sentences by employing negation, scalar adjectives, and numbered lists.

3. Methods

3.1. Ontology Design

Ontologies represent the backbone of the formal semantics of a knowledge graph. They can be seen as the schema of the KG. A well-defined ontology ensures a shared understanding of KG entities, attributes, and their relationships. Moreover, it enables deeper queries and reasoning over KG for efficient recommendations. Figure 2 shows the ontology of our restaurant domain KG. We apply natural language processing (NLP) techniques to extract information and produce KG triples. We utilized the Neo4j database for KG storage and visualization. The description of entities, relations, and their attributes is given in Table 1. The 'Time', 'Weather', and 'Date' nodes in the ontology do not exist in domain KGs. However, by extending multi-source data acquisition and implementing more robust rules, the KG can be extended to obtain additional knowledge.

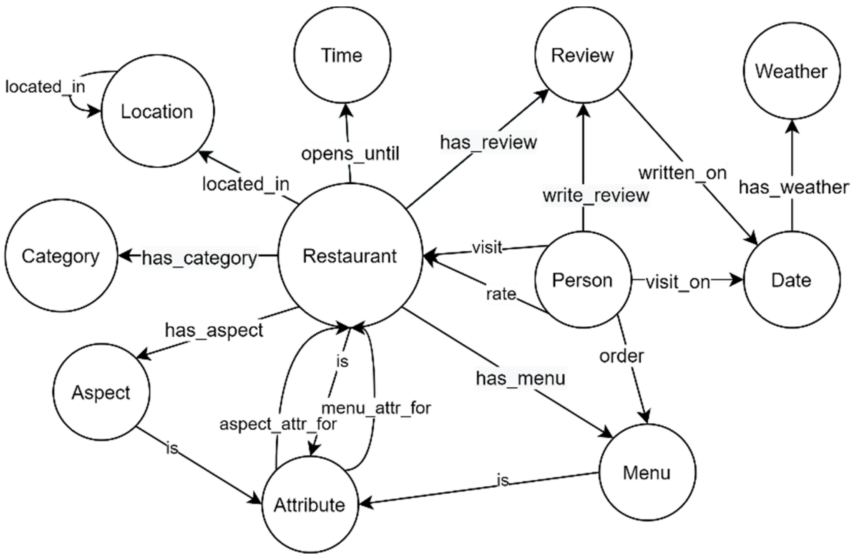

Figure 2. Conceptual graph schema (ontology) for restaurant domain knowledge graph.

Table 1. Description of domain KG entities and relations.

Entity	No. of Nodes	Relationship Types		Attributes
		Incoming	Outgoing	
Aspect	15	[HAS_ASPECT]	[IS]	aspect_id, name
Attr	241	[IS]	[ASPECT_ATTR_FOR, MENU_ATTR_FOR]	attr_id, name
Category	56	[HAS_CATEGORY]	[]	category_id, name
City	31	[LOCATED_IN]	[LOCATED_IN]	city_id, name
Country	2	[LOCATED_IN]	[]	country_id, name
Menu	624	[HAS_MENU]	[IS]	menu_id, name
Restaurant	102	[ASPECT_ATTR_FOR, MENU_ATTR_FOR, VISIT, RATE]	[LOCATED_IN, HAS_CATEGORY, IS, HAS_ASPECT, HAS_MENU, HAS_REVIEW]	rest_id, name, address, postal_code, rating
Review	3452	[HAS_REVIEW, WRITE_REVIEW]	[]	review_id, text
User	3227	[HAS_FRIEND]	[VISIT, ORDER, RATE, WRITE_REVIEW]	user_id, name, gender, age, review_count, avg_star, fans

3.2. Proposed KG-Based Context-Aware Recommendation

Our proposed recommender system, illustrated in Figure 3, is composed of the following three components: (i) triple generator module: generates template-based logical triple segments from a natural query; (ii) candidate generator module: comprises of the Q2B model; and (iii) machine learning-based re-ranker module: comprises of the NCF model. The NCF model is trained to learn user-item interactions by embedding latent vectors obtained from GraphSAGE framework. As a result, the model is capable of re-ranking the candidates generated in the second step of the candidate generation process by incorporating contextual features.

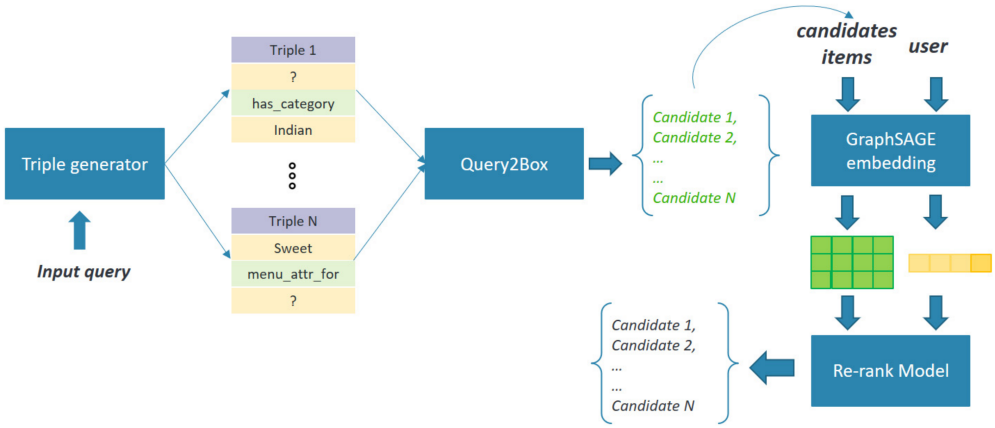

Figure 3. Proposed recommendation system.

3.2.1. Triple Generation from Natural Language Query

The triple generator module, illustrated in Figure 4, is a template-based framework that employs a simple, lexical-based technique to convert natural language query into logical triple segments. It first identifies the target word from the given user query. For our restaurant domain, we target restaurant and menus for recommendations (i.e., *recommend best* ***restaurant*** *in . . . , most* ***popular dishes*** *of Silver Spoon restaurant*, etc.). It then recognizes

the entity concepts, i.e., '*Category*', '*City*', and '*Menu*', etc., from user query, maps them to their associated relations, using the knowledge base (KB), and generates a set of triples. The goal is to convert natural language queries to logical triple segments that can be mapped to Q2B query structures and are created within the scope of the defined rules and KB entity concepts and relationships. Therefore, the framework is restricted to a specific template and can only process any arbitrary FOL queries that satisfy the template-specific rules.

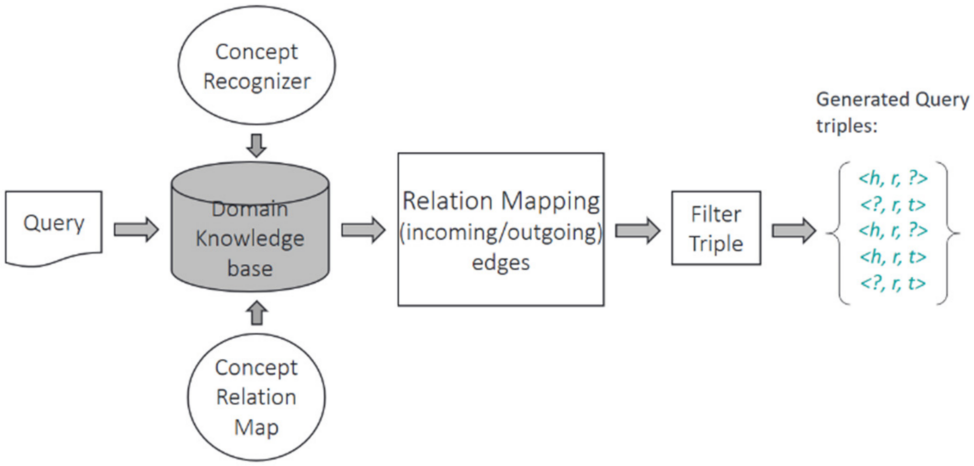

Figure 4. Triple generator module.

Table 2 illustrates the results of triple extraction, filtration, and logical query conversion for a given user query: *"Recommend best Chinese restaurant in Toronto which serves sweet Noodles or spicy Chicken Biryani"*. The extracted entity concept is mapped to its associated incoming and outgoing relations, in order to generate triples with logical conjunction ($\wedge$) and disjunction ($\vee$) (AND, OR) segments. Additionally, the module eliminates extraneous triples (in red) and filters the created triples to establish logical groupings of triple segments for reasoning and candidate retrieval. The strategy for filtration is described below:

- We eliminate all triples that do not have a direct relation to the target word or triples that do not describes the attribute of the entity concepts given in the user queries. The triple, for instance, ('Toronto', 'LOCATED_IN', ?), which represents the LOCATED_IN ('City', 'Country') triple, neither has a direct relationship to the target word nor defines the attribute of any entity concepts given in the user query. Therefore, this triple gets eliminated. On the other hand, the triple ('?' LOCATED_IN, 'Toronto') represents the LOCATED_IN ('Restaurant', 'City') triple, which has a direct relation with the restaurant concept. Similarly, the triple (?, 'ORDER', 'Noodles'), which represents ORDER ('User', 'Menu'), gets eliminated.
- The triples that do not belong to the similar concept are eliminated. For example, the 'Attribute' entity has two outgoing relationships, i.e., 'MENU_ATTR_FOR' and 'ASPECT_ATTR_FOR'. Therefore, the attribute 'Spicy' generates two triples: ('Spicy', 'MENU_ATTR_FOR', '?') and ('Spicy', 'ASPECT_ATTR_FOR', '?'). However, the former does not fall under the 'Menu' concept and, hence, the triple is eliminated.
- The two triples that result in a complete triple fact are considered a true fact and removed from the incomplete triple segments, i.e., [('Noodles', 'IS', '?'), ('?', 'IS', 'Sweet')] or [('Chicken Biryani', 'IS', '?'), ('?', 'IS', 'Spicy')] generates ('Noodles', 'IS', 'Sweet') or ('Chicken Biryani', 'IS', 'Spicy').

Furthermore, we transform query segments to a logical query format for querying the Q2B model for candidate retrieval.

- The logical operations ($\wedge$, $\vee$) are defined between triple segments.

Table 2. Results of user query "*Recommend best Chinese restaurant in Toronto which serves sweet Noodles or spicy Chicken Biryani*" conversion to logical query.

Extracted Triples	[{'triple_segment': [('?', 'HAS_CATEGORY', 'Chinese')], 'concept': 'Category', 'op': None}, {'triple_segment ': [('?', 'LOCATED_IN', 'Toronto'), ('Toronto', 'LOCATED_IN', '?')], 'concept': 'City', 'op': None}, {'triple_segment ': [('?', 'ORDER', 'Noodles'), ('?', 'HAS_MENU', 'Noodles'), ('Noodles', 'IS', '?'), ('?', 'IS', 'Sweet'), ('Sweet', 'MENU_ATTR_FOR', '?'), ('Sweet', 'ASPECT_ATTR_FOR', '?')], 'concept': 'Menu', 'op': None}, {'triple_segment ': [('?', 'ORDER', "Chicken Biryani '), ('?', 'HAS_MENU', 'Chicken Biryani'), ('Chicken Biryani', 'IS', '?'), ('?', 'IS', 'Spicy'), ('Spicy', 'MENU_ATTR_FOR', '?'), ('Spicy', 'ASPECT_ATTR_FOR', '?')], 'concept': 'Menu', 'op': 'AND'}]
Filtered Triples	[{'triple_segment ': [('?', 'HAS_CATEGORY', 'Chinese')], 'concept': 'Category', 'op': None}, {'triple_segment ': [('?', 'LOCATED_IN', 'Toronto')], 'concept': 'City', 'op': None}, {'triple_segment ': [('?', 'HAS_MENU', 'Noodles'), ('Sweet', 'MENU_ATTR_FOR', '?')], 'concept': 'Menu', 'op': None}, {'triple_segment ': [('?', 'HAS_MENU', 'Chicken Biryani'), ('Spicy', 'MENU_ATTR_FOR', '?')], 'concept': 'Menu', 'op': 'AND'}]
Logical Query	$q = R_?$: $\exists$ R: *Category*($R_?$, *Chinese*) $\wedge$ *Location*($R_?$, *Toronto*) $\wedge$ (((*Menu*($R_?$, *Noodles*) $\wedge$ *MenuAttrFor*(*Sweet*, $R_?$)) $\vee$ (*Menu*($R_?$, *Butter Chicken*) $\wedge$ *MenuAttrFor*(*Spicy*, $R_?$)))

The resultant logical query shows the converted output of the framework from the user query. To simplify logical query triples, the relationship aliases are converted to their simple form (i.e., HAS_CATEGORY to Category, LOCATE_IN to Location, HAS_MENU to Menu, MENU_ATTR_FOR to MenuAttrFor, etc.).

3.2.2. Query2Box (Q2B) Model

In KG, the entity that answers a query (e.g., $\{v_?; (v_?, \ r_1, \ t) = True\}$) is typically a set, not a point. If we take a query to be equivalent to a set of answer entities, then logical operations on queries are equivalent to operations on set of answer entities. A set of answer entities is embedded in a hyper-rectangle (box), formed by vectors corresponding to the entities in the KGE space. The Q2B [10] model is an embedding-based framework for reasoning over KGs that is capable of handling arbitrary existential positive first-order (EPFO) logical queries (i.e., queries having any set of $\wedge$, $\vee$, and $\exists$) in a scalable manner. It embeds queries as hyper-rectangles (boxes) in a KGE space and logical operations, such as existential quantifier ($\exists$), conjunctive ($\wedge$), and disjunctive operations ($\vee$), as box operations over queries (*projection*, *intersection*, and *union*).

The Q2B operations are illustrated in Figure 5, which is accomplished by defining a box with its center and offset $b = (Cen(b), Off(b)) \in R^{2d}$ (d is the dimension of KGE space) as:

$$Box_p \equiv \left\{ v \in R^d \ : Cen(p) - Off(p) \leq v \leq Cen(p) + Off(p) \right\}$$

where $\leq$ is an element-wise inequality, $Cen(p) \in R^d$ is the center of the box, and $Off(p) \in R^d_{\geq 0}$ is the positive offset of the box, representing the size of the box. Then, it describes the box projection and intersection operations, as well as the entity-to-box distance $dist_{box}$ (box distance).

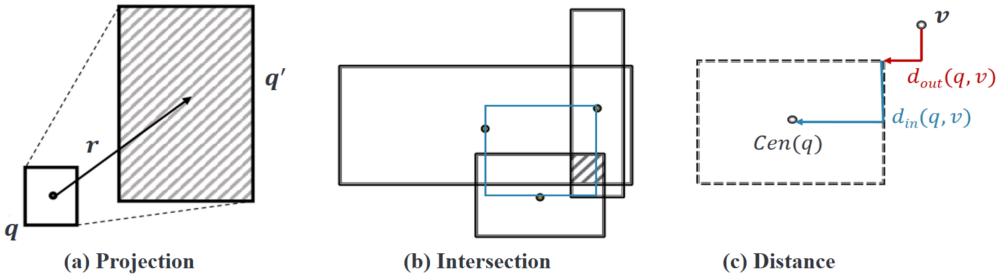

Figure 5. The geometric intuition of the projection and intersection operations in Q2B.

Q2B represents each entity $v \in V$ as a single point (zero-volume box): $v = (Cen(v), 0)$ and relation (r) as an embedding in R^{2d} and performs relation projection and box intersection, i.e., $\mathcal{P}$: Box × Relation → Box and $\mathcal{J}$: Box × . . . × Box → Box in the embedding space, respectively.

Each relation (r) takes a box and produces a new box. The projection operation ($\mathcal{P}$) (Figure 5a) takes the current box as input and uses the relation embedding to project and expand the box. The geometric intersection operation ($\mathcal{J}$) (Figure 5b) takes the multiple boxes $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ as input $\mathbf{p}_{inter} = (Cen(\mathbf{p}_{inter}), Off(\mathbf{p}_{inter}))$ and produces the intersection box. The center of the new box should be close to the centers of the input boxes, accomplished by shrinking the offset of the input boxes. The center of the new box (in blue Figure 5b) is a weighted sum of the input box centers, given as follows:

$$Cen(\mathbf{p}_{inter}) = \sum_i w_i \odot Cen(\mathbf{p_i}), \; \therefore w_i = \frac{\exp(MLP(\mathbf{p_i}))}{\Sigma_j \exp\left(MLP\left(\mathbf{p_j}\right)\right)}$$

$$Off(\mathbf{p}_{inter}) = Min(\{Off(\mathbf{p}_1), \ldots, Off(\mathbf{p_n})\}) \odot \sigma(DeepSets(\{\mathbf{p}_1, \ldots, \mathbf{p_n}\}))$$

where $\odot$ is the dimension-wise product, $w_i \in R^d$ is calculated by a multi-layer perceptron (with trainable weights), and w_i represents a self-attention score for the center of each input $Cen(\mathbf{p_i})$. It takes the minimum of the offset of the input box and makes the model more expressive by introducing a $DeepSets(.)$ function to extract the representation of the input boxes with a sigmoid function to guarantee shrinking.

The entity-to-distance (Figure 5c) function defines the scoring function, in terms of distance. Given a query box (q) and entity embedding (box) ($\mathbf{v}$), the distance is defined as:

$$dist_{box}(\mathbf{v}, q) \equiv dist_{out}(\mathbf{v}, q) + \alpha \cdot dist_{in}(\mathbf{v}, q) \;\; \therefore 0 < \alpha < 1$$

If the point is enclosed in the box, the distance should be downweighted. Q2B handles conjunctive queries in a natural way by taking projection and intersection. However, allowing disjunction (union) over arbitrary queries requires high-dimensional embeddings, as it cannot be embedded in a low-dimensional vector space. To handle EPFL queries that contain disjunction, it makes use of a well-known fact—any FOL query may be translated into its corresponding disjunctive normal form (DNF), which has a form of disjunction of conjunctive queries. Thus, in order to answer to an EPFO queries, Q2B first applies box projection and intersection to calculate the intersected box for each conjunctive and compute the final answer entities by utilizing the aggregated distance function to a set of intersected boxes. The aggregated distance function between an EPFO query (q) and an entity ($\mathbf{v}$) utilizing the box distance $dist_{box}$ and the fact that a complex query (q) in FOL is logically equivalent to $q^{(1)} \vee \cdots \vee q^{(N)}$, as follows:

$$dist_{agg}(\mathbf{v}, q) \equiv Min\left(\left\{dist_{box}\left(\mathbf{v}, q^{(1)}\right), \cdots, dist_{box}\left(v, q^{(N)}\right)\right\}\right)$$

As long as **v** is the answer to one conjunctive query (q^i), it should also be the answer to q, and as long as **v** is the close to one conjunctive query (q^i), it should also be close to q in the embedding space.

For the purpose of learning entity embeddings, as well as geometric projection and intersection operators, Q2B optimizes a negative sampling loss, in order to successfully optimize the distance-based model. During optimization process, Q2B randomly samples a query (q) from the training graph (G_{train}), answer ($v \in [|q|]_{train}$), and a negative sample ($v' \notin [|q|]_{train}$). The negative sample entity (v') is the entity of same type as v but not the answer entity. After sampling, the query is embedded into the vector space and calculates the score $dist_{box}(v;q)$ and $dist_{box}(v'_i;q)$ and optimizes the loss (L) to maximize $dist_{box}(v;q)$, while minimizing the $dist_{box}(v'_i;q)$;

$$L = -log\sigma(\gamma - dist_{box}(v;q)) - \sum_{i=1}^{k} \frac{1}{k} log\sigma(dist_{box}(v'_i;q) - \gamma)$$

where σ is the sigmoid function, γ represents a fixed scalar margin, v is a positive entity (answer to the query q), v'_i is the i-th negative entity (non-answer to the query q), and k is the number of negative entities.

3.2.3. ML-Based NCF Re-Rank Model

Our ML-based collaborative re-ranking model, illustrated in Figure 6, provides a ranking score function $\hat{r}\ (u,\ i)$ for recommendation between the user (u) and candidate items (i) obtained from the Q2B model. The ranking score (r) is used to re-rank the obtained candidate items, based on user and item interaction and side (auxiliary) information.

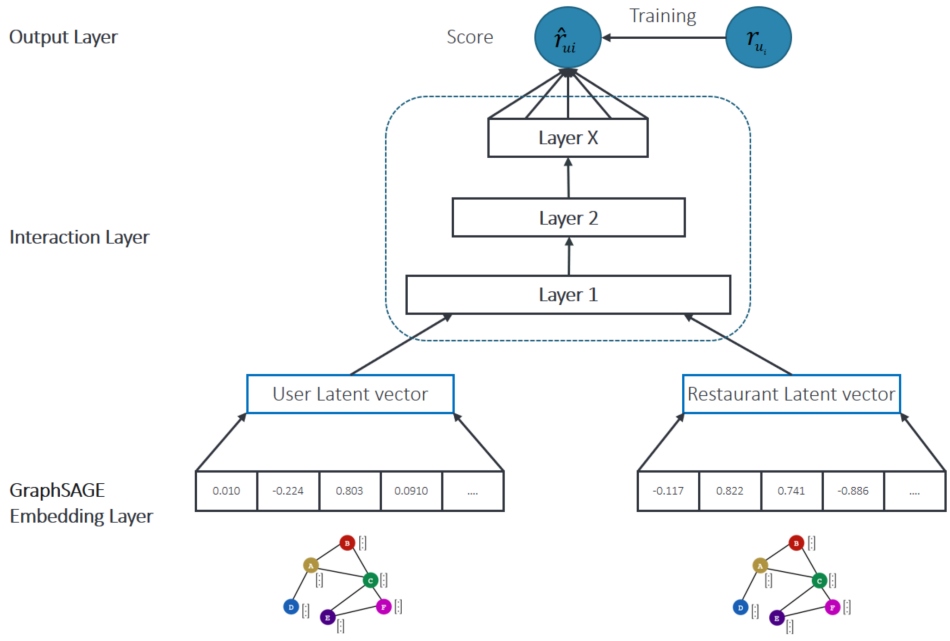

Figure 6. NCF-based re-rank model.

The NCF model leverages both the bipartite interaction and side information of users and items, as shown in Figure 7. We utilized the GraphSAGE framework to obtain the contextualized latent vectors from the domain KG. GraphSAGE generates node embeddings, while incorporating side features. The auxiliary features of user and restaurant nodes are given in the 'Attributes' column in Table 1. By combining content information and bipartite interactions into the NCF model, on top of the Q2B model, we can not only recommend candidates based on user-defined queries but also re-rank the recommended candidates to provide more contextual recommendations and address the cold-start problem.

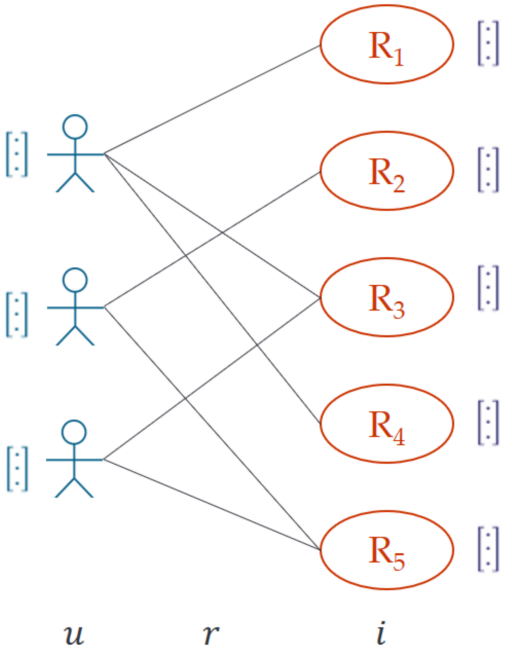

Figure 7. Example of bipartite interaction between user and item. The brackets represent the side features of users and items.

We leverage the Graph Data Science (GDS) library, provided by Neo4j, to generate GraphSAGE embeddings for nodes with context features from the domain KG. GraphSAGE is a framework for learning inductive representations on large graphs [13]. It generates low-dimensional vector representations of nodes, while utilizing their attribute information, making GraphSAGE more appropriate for training and learning KG representation for recommendation and other tasks than any other KGE model. Furthermore, it preserves the latent structural information of the network. Previous matrix factorization-based embedding frameworks, which are transductive in nature and computationally expensive, due to the fact that they can generate embeddings only for a single fixed graph and do not generalize well on unseen nodes. GraphSAGE performs sampling on the neighboring nodes and aggregates their feature representations to generate node embedding. GraphSAGE generates node embeddings using the pool aggregator function, which performs element-wise max-pooling operations to aggregate information across neighbor nodes [13].

$$AGGREGATE_k^{pool} = max\left(\left\{\sigma\left(W_{pool}h_{u_i}^k + b\right), \forall_{u_i} \in N(v)\right\}\right)$$

The learned contextual embeddings are used to train the NCF model for re-ranking the candidates. We process the final candidates, returned by the Q2B model, by utilizing user and restaurant node and attribute information to predict a rating score (r). The final recommendation score (s) for each candidate restaurant is calculated by the weighted average between relevance score (z) of the Q2B model and predicted rating score (r) of the NCF model to re-rank the final restaurant candidates.

$$s = (\alpha z + (1 - \alpha)r) \ \therefore 0 < \alpha < 1$$

The α factor is the weight that determines the importance of model results to be considered for final ranking of the candidates. The relevance score (z) of the generated candidate items is obtained from the Q2B model, based on the given user query. On the other hand, the NCF model predicts the rating score (r) between the user and generated candidate items by incorporating the interaction and side information. Therefore, assigning a higher weight (α) to the rating score (r) leads in re-ranking candidates, based on more accurate contextual information.

4. Evaluation

In this part, we describe the dataset and experiments. We conducted experiments to evaluate the candidate generation and re-ranking, using the proposed recommendation framework.

4.1. Dataset

We utilized the YELP dataset [9] and stored it in the MongoDB database. We filtered out only restaurant categories to make a restaurant domain dataset. We considered 100 restaurants, with at least 20 reviews and extract entities (User, Category, Reviews, City, etc.), associated with those restaurants. For instance, we stored only those users who had left a review for those restaurants. We extracted entities and their relations that constituted a KG triple. First, we processed the structured data to extract the factual information (entity and relation), based on the defined ontology, shown in Figure 2. For example, the restaurant data contains information about *location*, *category*, etc., while the review data contains information about *reviewer* (i.e., ID, name), *review_for*, *review_text*, and *date*. Then, we manipulated this structured data, based on domain ontology, and stored it in our Neo4j graph database. Then, we applied different NLP techniques to process unstructured text, in the form of review text. Table 3 shows the data description of the restaurant domain KG.

Table 3. No. of nodes and relation in the restaurant domain KG.

Total No. of Nodes	Total No. of Relations
7750	39,158

4.2. Evaluation on Natural Query Conversion

The triple generator module was evaluated on domain-specific user queries. For evaluation settings, we defined the natural query, corresponding to the query structures used to evaluate the Q2B model on the domain KG. We also evaluated the conversion of user-defined natural queries, corresponding to the arbitrary FOL query (having any set of $\wedge$, $\vee$, and $\exists$) that satisfied the template-specific rules. The results depict that the framework generalizes well on the queries that adhere to the template rules.

Failure case analysis: The framework generated an incorrect query that lacked a valid target word (restaurant or menu). The natural query with the 'pi' structure (Table 4 (6)) does not contain a valid target word. As a result, the framework produced an incorrect query. Additionally, the query (Table 4 (10)) also failed to convert to the valid logical query segments. The 'Aspect' node in the domain KG contains 'Delivery' and 'Service' as two distinct aspects of a restaurant. Therefore, the framework extracted two aspects and produced the wrong result, containing an extra triple: 'Aspect ($R_?$, Service)', which is invalid.

4.3. Evaluation on Query2Box (Q2B) Query and Candidate Generation

To train our system on the restaurant domain KG build from the Yelp dataset, we simulated [10] the construct of a set of queries and their answers during training time and then learned the entity embeddings and geometric operators, in order to enable accurate query response. The model examines nine distinct types of query structures, as seen in Figure 8. The first five query structures were used to train the model and were evaluated on all nine query structures. The technique demonstrates a high degree of generalization, when applied to queries and logical structures not seen during training.

Table 4. Results of natural query conversion to logical triple segments. $R_?$ denotes the restaurant target entity, and $M_?$ denotes the menu target entity.

#	User Query	Query Structure	Logical Query Segments	Result
1	Recommend best Chinese restaurants	1p	Category ($R_?$, Chinese)	Correct
2	What special menus Chinese restaurants serve?	2p	Category ($R_?$, Chinese) $\wedge$ Menu ($R_?$, $M_?$)	Correct
3	Recommend best Indian restaurant in Toronto	2i	Category ($R_?$, Indian) $\wedge$ Location ($R_?$, Toronto)	Correct
4	Recommend best Indian restaurant in Toronto which serves Butter Chicken.	3i	Category ($R_?$, Indian) $\wedge$ Location($R_?$, Toronto) $\wedge$ Menu ($R_?$, Butter Chicken)	Correct
5	What special menus Indian restaurants serve in Toronto?	ip	Category ($R_?$, Indian) $\wedge$ Location($R_?$, Toronto) $\wedge$ Menu ($R_?$, $M_?$)	Correct
6	Who visited Indian restaurant and ordered Butter Chicken?	pi	Category ($R_?$, Indian) $\wedge$Menu ($R_?$, Butter Chicken)	Wrong
7	Recommend restaurants which serve Butter Chicken or Chicken Biryani	2u	Menu ($R_?$, Butter Chicken) $\vee$ Menu ($R_?$, Chicken Biryani)	Correct
8	Which restaurants in Toronto serves Butter Chicken or Chicken Biryani	up	(Menu ($R_?$, Butter Chicken) $\vee$ Menu ($R_?$, Chicken Biryani)) $\wedge$ Location ($R_?$, Toronto)	Correct
9	Recommend best Chinese restaurant in Toronto which serves sweet Noodles or spicy Chicken Biryani	arbitrary	Category ($R_?$, Chinese) $\wedge$ Location ($R_?$, Toronto) $\wedge$ (((Menu ($R_?$, Noodles) $\wedge$ MenuAttrFor (Sweet, $R_?$)) $\vee$ (Menu ($R_?$, Butter Chicken) $\wedge$ MenuAttrFor (Spicy, $R_?$)))	Correct
10	Which Chinese restaurants in Toronto have delivery service?	3i	Category ($R_?$, Chinese) $\wedge$ Location ($R_?$, Toronto) $\wedge$ Aspect ($R_?$, Delivery) $\wedge$ Aspect ($R_?$, Service)	Wrong
11	Which Chinese restaurants in Toronto have delivery?	3i	Category ($R_?$, Chinese) $\wedge$ Location ($R_?$, Toronto) $\wedge$ Aspect ($R_?$, Delivery)	Correct

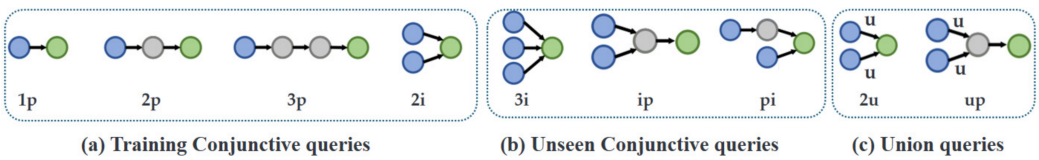

Figure 8. Query structures for experiments, where 'p', 'i', and 'u' stand for 'projection', 'intersection', and 'union', respectively.

We follow a similar evaluation protocol to that presented in [10], which is briefly stated here; the data preparation required splitting the KG edges into training, test, and evaluation sets and began by augmenting the KG to include inverse relations, thus dou-

bling the amount of edges in the graph. Following that, we constructed three graphs: $G_{train} \subseteq G_{valid} \subseteq G_{test}$. While G_{train}, contains only training edges and is used to train node embeddings, as well as box operators, G_{valid} contains G_{train}, as well as the validation edges, and G_{test} includes G_{valid}, as well as the test edges. Considering the nine query structures presented in [10] for training the graphs, 80% of generated queries were utilized for training, 10% were used for validation, and the remaining 10% were used for testing. For a given query (q), the denotation sets (answer entities sets) $[|q|]_{train}$, $[|q|]_{valid}$, and $[|q|]_{test}$ were obtained by running subgraph matching of q on G_{train}, G_{valid}, and G_{test}, respectively. $[|q|]_{train}$ were utilized as positive samples for the query during training, while negative samples were generated from other random entities. However, for testing and validation, the method was validated against only those answers that have missing relations, instead of validating against whole validation $[|q|]_{valid}$ or test $[|q|]_{test}$ sets of answers. Table 5 summarizes the results of the Q2B model simulation on a restaurant domain dataset for different query structures. As stated in [10], the complex logical queries (particularly 2p, 3p, ip, pi, and up) require modeling a significantly greater number of answer entities (often more than 10 times) than the simple 1p queries do. Therefore, it is expected that the box embeddings will perform well when handling complex queries with a large number of answer entities; this was also observed in our experiments, particularly for queries with 2i, 3i, ip, and pi. According to our restaurant domain ontology triple dataset, queries with 2i 3i, ip, and pi structures are frequently observed with a greater number of entities than other query structures and, therefore, produced better results on the Yelp (domain-KG) dataset than other datasets. The model performs comparably to FB15k-237 and NELL995 but is inferior to FB15k on our generated restaurant domain KG.

Table 5. Results of Q2B on restaurant domain dataset for different query structures. The other dataset results are illustrated from the original paper for comparison.

Query	Avg	1p	2p	3p	2i	3i	ip	pi	2u	up
Yelp (domain-KG) dataset										
MRR	0.286	0.309	0.164	0.144	0.402	0.604	0.234	0.38	0.171	0.168
Hits@1	0.188	0.19	0.062	0.05	0.334	0.545	0.143	0.26	0.012	0.099
Hits@3	0.347	0.392	0.234	0.207	0.429	0.627	0.287	0.46	0.289	0.201
Hit@10	0.44	0.46	0.362	0.303	0.498	0.699	0.399	0.535	0.393	0.308
FB15k										
MRR	0.41	0.654	0.373	0.274	0.488	0.602	0.194	0.339	0.468	0.301
Hits@3	0.484	0.786	0.413	0.303	0.593	0.712	0.211	0.397	0.608	0.33
FB15k-237										
MRR	0.235	0.4	0.225	0.173	0.275	0.378	0.105	0.18	0.198	0.178
Hits@3	0.268	0.467	0.24	0.186	0.324	0.453	0.108	0.205	0.239	0.193
NELL995										
MRR	0.254	0.413	0.227	0.208	0.288	0.414	0.125	0.193	0.266	0.155
Hits@3	0.306	0.555	0.266	0.233	0.343	0.48	0.132	0.212	0.369	0.163

Our recommendation system enables generating results for the user-specific queries in an explainable way, based on user and query context.

"Recommend best Indian restaurant in Toronto which serves sweet Butter Chicken" Query (1)

Q2B produces set of candidate restaurant for a given logical query. An equivalent illustration of the Q2B result for the Query (1) is given in Figure 9, using cypher in Neo4j.

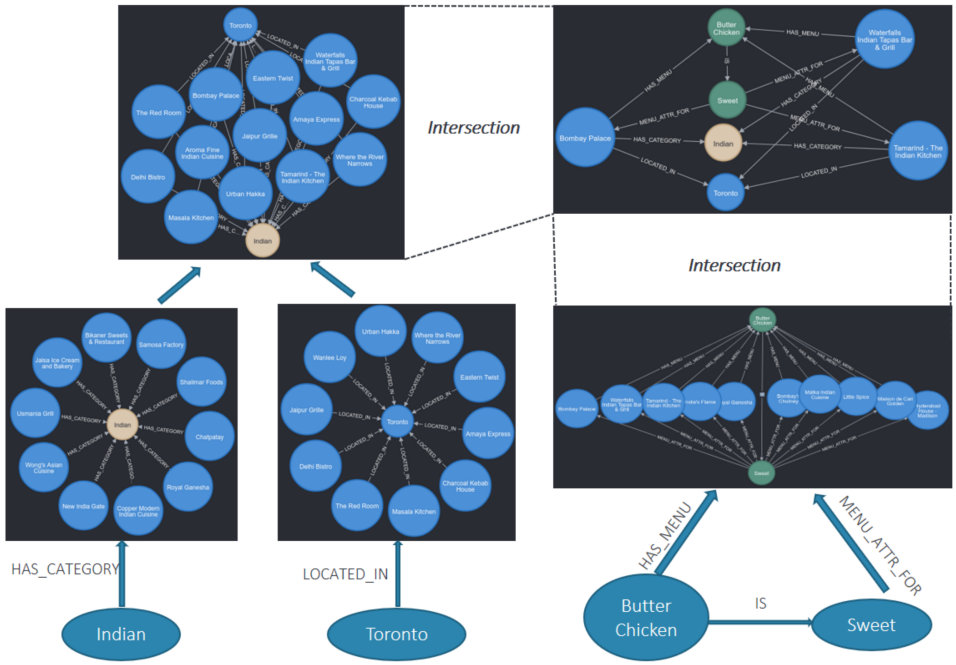

Figure 9. Visual explainable equivalent cypher query result from user Query (1).

The cypher query approach produces actual candidates from the domain KG. It cannot recommend candidates if no match is found. However, Q2B aims to recommend the item, not to perform a search from the database. Therefore, it produces *top-K*, the semantically similar and closest candidates for each query triple. Hence, it can recommend items, even if there is no match found in the KG. Table 6 depicts the Q2B results for the input Query (1). The resultant restaurant candidates in bold (Rank 1, 2, and 3) are the actual restaurants that satisfy the result for input Query (1) from the stored KG, as shown in Figure 10. The candidate restaurants with Ranks 4 and 5, on the other hand, missed some entities or relationships to match exact candidates based on the given query. However, because the recommender system's task is to recommend the *top-K* candidates, we consider the top five results, based on the semantic and contextual similarity of the query and KG. Additionally, this approach of producing candidates enables the generation of explainable findings. We define explainability, in terms of KG paths, for recommended candidates.

Table 6. Q2B results for input Query (1).

Rank	RestID	Q2B (*z*)
1	**4873**	**7.478498**
2	**4641**	**7.431061**
3	**3744**	**3.110984**
4	3926	1.378697
5	5324	0.376228

The result of the Q2B model can be defined for the explainable recommendation. We demonstrate the visual explainability of each generated restaurant candidate for input query (1) in Figure 11. The projection of the resulting restaurant candidates with connected paths determines the applicability of the recommendation. From the explicable approach, we show that candidate 4 does not match the attribute '*Sweet*', while candidate 5 does not match the menu, as well as attribute entity (*Butter Chicken*, *Sweet*). However, the other facts of these candidates are matched with higher semantic and structural similarity and obtained a relevance score lower than the exact match candidates of the query from the KG.

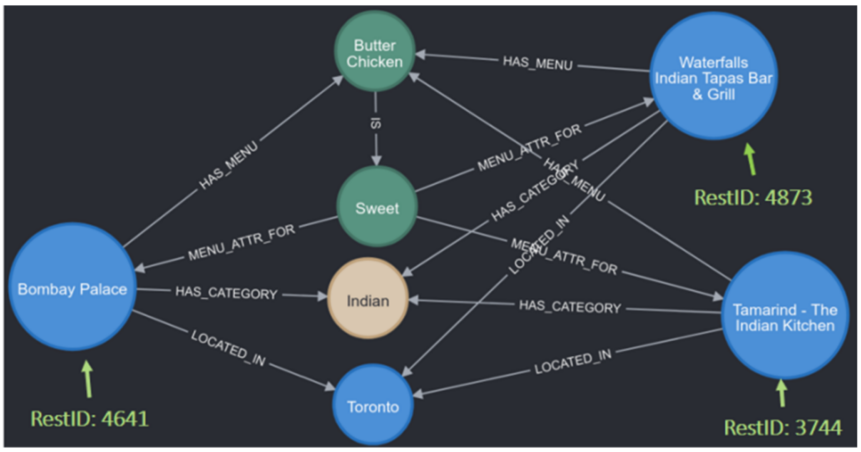

Figure 10. Comparison of Q2B results with native cypher query results for user Query (1).

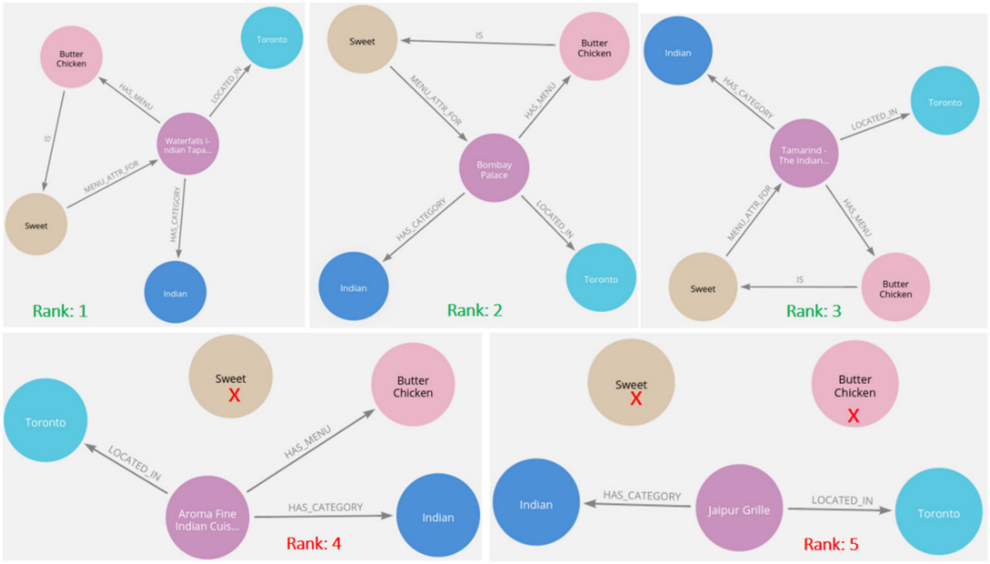

Figure 11. Visualization of explainable results from the domain KG for candidates, obtained from the Q2B model.

4.4. Evaluation of Neural Collaborative Filtering (NCF)-based Re-Rank Module

The scope of our recommender system is not only limited to domain KG-based explainable recommendation. It is a two-stage procedure, in which, similar to other factorization approaches, we employ a ML-based NCF model to re-rank the generated candidates from the previous stage, in order to incorporate user and candidate restaurant context information. We train a NCF model by leveraging contextual GraphSAGE embedding. We generate contextual GraphSAGE embedding by employing the entities and attributes from the domain KG, stored in the Neo4j database. The dataset for training the model is shown in Table 7.

Table 7. Dataset for model training.

Train	Test
30,000	4524

The proposed approach of utilizing GraphSAGE, embedding with the NCF network model, outperforms the classic matrix factorization method on the domain KG.

Table 8 shows the ablation study for model training, while in Table 9, we compare our proposed approach with the matrix factorization model.

Table 8. Ablation results of the NCF model training with GraphSAGE features.

Epochs	MAE	RMSE
5000	1.0690	1.5949
10,000	1.0674	1.5925
20,000	**1.0673**	**1.5917**

Table 9. NCF model comparison with matrix factorization technique.

Method	MAE	RMSE
MF	1.169	1.994
NCF Model	**1.0673**	**1.5917**

As discussed earlier, the NCF model processes candidates from the Q2B results and predicts a rating score (r) for the user and candidate restaurant pair. The final score (s) is then taken as a weighted average between the relevance score (z) of the Q2B model and predicted rating score (r) of the NCF model. The demonstration of the model results, between generated restaurant candidates in Table 6 and two users (*UserID:3178*, and *UserID:17*), pair is shown in Table 10. To calculate final score (s), we consider $\alpha = 0.3$ to assign more attention to r. The final candidate items (restaurants) are re-ranked, based on the final score.

Table 10. Re-rank model results.

Rank	RestID	Q2B (z)	UserID: 3178			UserID: 17		
			NCF (r)	Score (s)	Re-Ranking	NCF (r)	Score (s)	Re-Ranking
1	4873	7.478498	4.3	5.25	2	4.2	5.18	2
2	4641	7.431061	4.7	5.52	1	4.5	5.38	1
3	3744	3.110984	4.8	4.29	3	3.9	3.66	3
4	3926	1.378697	4.2	3.35	4	4.0	3.21	4
5	5324	0.376228	4.1	2.98	5	3.5	2.56	5

We further validate the NCF model results by applying cosine similarity on the GraphSAGE embeddings, learned using the Neo4j GDS framework. The cosine similarity computes the similarity between two vectors, which could be then utilized in a recommendation query. For instance, to obtain restaurant recommendations based on the preferences of users who have previously given similar ratings to other restaurants visited by the user. We measure the similarity between user (*UserID: 3178*) and resultant restaurant candidate pairs from the Q2B. The results in Table 11 show that the restaurant with (*RestID: 4641*) and (*RestID: 3744*) had the first and second highest similarity, respectively, which the NCF model also predicted as higher.

Table 11. NCF model results comparison, with the cosine similarity algorithm applied on GraphSAGE embedding, using the Neo4j GDS framework.

RestID	RestName	Similarity	NCF (r)
4641	Bombay Palace	0.99996	4.7
3744	Tamarind—The Indian Kitchen	0.99993	4.8
3926	OM Restaurant	0.99988	4.2
4873	Waterfalls Indian Tapas Bar & Grill	0.99985	4.3
5324	Pakwan Indian Bistro	0.99903	4.1

5. Conclusions

We proposed a novel, domain-specific, context-aware, explainable recommendation framework, based on a domain knowledge graph (KG) and machine learning model. Our proposed recommender system deals with user-defined natural language query. It consists of three modular stages: (1) decompose a complex natural language query into segments of logical triples that reflect the semantic and structural mapping over the domain knowledge graph; (2) find the final set of candidate restaurants with relevance score (z) by performing logical conjunction and or disjunction operations, using the Q2B model; and (3) predicts the rating score (r) for the final candidate items, through the neural collaborative filtering model, by incorporating user and candidate items latent vectors. Through the evaluation of its application to the restaurant domain, with Yelp data, it is shown that the proposed approach works effectively for a domain-specific system. We have shown that combining KG techniques with the traditional collaborative filtering approach can solve data sparsity cold-start problems and provide a content-aware explainable recommendation. Careful design of domain ontology is important for the decomposition of natural queries into triples for a domain KG. However, extending domain ontology with additional parameters and developing more robust rules for natural query processing could result in complex query answering and parsing. Moreover, replacing the candidate generation and re-rank processes with a single GraphSAGE framework could enable inductive learning and does not require the use of two distinct models for reasoning over KG and learning its representation (nodes, relation, and attributes) for creating latent vector separately.

Author Contributions: Conceptualization, M.H.S.; methodology, M.H.S.; software, M.H.S. and T.Q.B.H.; validation, M.H.S. and T.Q.B.H.; formal analysis, M.H.S.; data curation, M.H.S.; writing—original draft preparation, M.H.S.; writing—review and editing, S.-T.C.; supervision, S.-T.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the AI-based High Added Value New Product Technology Development Program (Project No. S2953591) and SME R&D project for the Start-up & Grow stage company (Project No. S3044682) of the Ministry of SMEs and Startups (MSS, Korea).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wang, H.; Zhang, F.; Wang, J.; Zhao, M.; Li, W.; Xie, X.; Guo, M. RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, Torino, Italy, 22–26 October 2018; pp. 417–426.
2. Covington, P.; Adams, J.; Sargin, E. Deep Neural Networks for YouTube Recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, 15–19 September 2016; ACM: New York, NY, USA, 2016; pp. 191–198.
3. Wang, H.; Zhang, F.; Xie, X.; Guo, M. DKN: Deep Knowledge-Aware Network for News Recommendation. In Proceedings of the 2018 World Wide Web Conference (WWW), Lyon, France, 23–27 April 2018; pp. 1835–1844.
4. Yu, X.; Ren, X.; Sun, Y.; Gu, Q.; Sturt, B.; Khandelwal, U.; Norick, B.; Han, J. Personalized entity recommendation: A heterogeneous information network approach. In Proceedings of the 7th International Conference Web Search and Data Mining (WSDM), New York, NY, USA, 24–28 February 2014; ACM: New York, NY, USA, 2014; pp. 283–292.
5. Wang, H.; Zhang, F.; Zhao, M.; Li, W.; Xie, X.; Guo, M. Multi-Task Feature Learning for Knowledge Graph Enhanced Recommendation. In Proceedings of the 2019 World Wide Web Conference (WWW), San Francisco, CA, USA, 13–17 May 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 2000–2010.
6. Wang, X.; He, X.; Cao, Y.; Liu, M.; Chua, T.S. KGAT: Knowledge Graph Attention Network for Recommendation. In Proceedings of the 25th ACM SIGKDD International Conference of Knowledge Discovery & Data Mining (KDD), Anchorage, AK, USA, 4–8 August 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 950–958.
7. Gao, Y.; Li, Y.F.; Lin, Y.; Gao, H.; Khan, L. Deep learning on knowledge graph for recommender system: A survey. *arXiv* **2020**, arXiv:2004.00387.

8. Hogan, A.; Blomqvist, E.; Cochez, M.; d'Amato, C.; Melo, G.; Gutierrez, C.; Gayo, J.E.L.; Kirrane, S.; Neumaier, S.; Polleres, A.; et al. Knowledge graphs. *ACM Comput. Surv.* **2021**, *54*, 1–37. [CrossRef]
9. Yelp.com. Available online: https://www.yelp.com/ (accessed on 23 October 2021).
10. Ren, H.; Hu, W.; Leskovec, J. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In Proceedings of the International Conference Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
11. Rendle, S.; Gantner, Z.; Freudenthaler, C.; Thieme, L.S. Fast context-aware recommendations with factorization machines. In Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, Beijing, China, 24–28 July 2011; pp. 635–644.
12. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T.-S. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web (WWW 2017), Perth, Australia, 3–7 April 2017; pp. 173–182. Available online: https://dblp.org/rec/conf/www/HeLZNHC17 (accessed on 13 November 2021).
13. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. In Proceedings of the NeuralPS, Long Beach, CA, USA, 4–9 December 2017; Curran Associates Inc.: Red Hook, NY, USA, 2017; pp. 1024–1034.
14. Hao, X.; Ji, Z.; Li, X.; Yin, L.; Liu, L.; Sun, M.; Liu, Q.; Yang, R. Construction and Application of a Knowledge Graph. *Remote Sens.* **2021**, *13*, 2511. [CrossRef]
15. Zhu, G.; Iglesias, C.A. Sematch: Semantic entity search from knowledge graph. In Proceedings of the SumPre 2015—1st International Workshop Extended Semantic Web Conference (ESWC), Portoroz, Slovenia, 1 June 2015.
16. Yasunaga, M.; Ren, H.; Bosselut, A.; Liang, P.; Leskovec, J. QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Online, 6—11 June 2021; pp. 535–546. Available online: https://aclanthology.org/2021.naacl-main.45/ (accessed on 13 November 2021).
17. Papadopoulos, D.; Papadakis, N.; Litke, A. A Methodology for Open Information Extraction and Representation from Large Scientific Corpora: The CORD-19 Data Exploration Use Case. *Appl. Sci.* **2020**, *10*, 5630. [CrossRef]
18. Wang, M.; Qiu, L.; Wang, X. A Survey on Knowledge Graph Embeddings for Link Prediction. *Symmetry* **2021**, *13*, 485. [CrossRef]
19. Bastos, A.; Nadgeri, A.; Singh, K.; Mulang, I.O.; Shekarpour, S.; Hoffart, J.; Kaul, M. RECON: Relation Extraction using Knowledge Graph Context in a Graph Neural Network. In Proceedings of the World Wide Web, Ljubljana, Slovenia, 19–23 April 2021; pp. 1673–1685.
20. Arakelyan, E.; Daza, D.; Minervini, P.; Cochez, M. Complex Query Answering with Neural Link Predictors. *arXiv* **2011**, arXiv:2011.03459.
21. Bordes, A.; Usunier, N.; García-Durán, A.; Weston, J.; Yakhnenko, O. Translating embeddings for modeling multi-relational data. In Proceedings of the Advances in Neural Information Processing Systems 26 (NIPS 2013), Stateline, NV, USA, 5—10 December 2013; pp. 2787–2795. Available online: https://papers.nips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html (accessed on 13 November 2021).
22. Choudhary, S.; Luthra, T.; Mittal, A.; Singh, R. A survey of knowledge graph embedding and their applications. *arXiv* **2021**, arXiv:2107.07842.
23. Wang, Z.; Zhang, J.; Feng, J.; Chen, Z. Knowledge graph embedding by translating on hyperplanes. In Proceedings of the 28th AAAI Conference of Artificial Intelligence, Quebec City, QC, Canada, 27–31 July 2014; pp. 1112–1119.
24. Lin, Y.; Liu, Z.; Sun, M.; Liu, Y.; Zhu, X. Learning entity and relation embeddings for knowledge graph completion. In Proceedings of the 29th AAAI, Austin, TX, USA, 25–30 January 2015; pp. 2181–2187.
25. Nickel, M.; Tresp, V.; Kriegel, H.P. A three-way model for collective learning on multi-relational data. In Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA, 28 June –2 July 2011; pp. 809–816.
26. Yang, B.; Yih, W.T.; He, X.; Gao, J.; Deng, L. Embedding entities and relations for learning and inference in knowledge bases. In Proceedings of the International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015; pp. 1–12.
27. Dettmers, T.; Minervini, P.; Stenetorp, P.; Riedel, S. Convolutional 2D knowledge graph embeddings. In Proceedings of the AAAI, Quebec City, QC, Canada, 27–31 July 2014; pp. 1811–1818.
28. Shah, L.; Gaudani, H.; Balani, P. Survey on Recommendation System. *Int. J. Comp. Appl.* **2016**, *137*, 43–49. [CrossRef]
29. Koren, Y.; Bell, R.; Volinsky, C. Matrix factorization techniques for recommender systems. *Computer* **2009**, *42*, 30–37. [CrossRef]
30. Zhang, H.; Ganchev, I.; Nikolov, N.S.; Ji, Z.; O'Droma, M. FeatureMF: An Item Feature Enriched Matrix Factorization Model for Item Recommendation. *IEEE Access* **2021**, *9*, 65266–65276. [CrossRef]
31. He, X.; Chua, T.S. Neural factorization machines for sparse predictive analytics. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Tokyo, Japan, 7–11 August 2017; pp. 355–364.
32. Cheng, H.T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H.; Anderson, G.; Corrado, G.; Chai, W.; Ispir, M. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop Deep Learning for Recommender Systems, Boston, MA, USA, 15 September 2016; pp. 7–10.
33. Lian, J.; Zhou, X.; Zhang, F.; Chen, Z.; Xie, X.; Sun, G. XDeepFM: Combining explicit and implicit feature interactions for recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference of Knowledge Discovery and Data Mining, London, UK, 19–23 August 2018; pp. 1754–1763.

34. Zhang, F.; Yuan, N.J.; Lian, D.; Xie, X.; Ma, W.Y. Collaborative knowledge base embedding for recommender systems. In Proceedings of the 22nd. International Conference of ACM SIGKDD on KDD, San Francisco, CA, USA, 13–17 August 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 353–362.
35. Guo, Q.; Zhang, F.; Qin, C.; Zhu, H.; Xie, X.; Xiong, H.; He, Q. A survey on knowledge graph-based recommender systems. *arXiv* **2020**, arXiv:2003.00911. [CrossRef]
36. Liu, C.; Li, L.; Yao, X.; Tang, L. A Survey of Recommendation Algorithms Based on Knowledge Graph Embedding. In Proceedings of the IEEE International Conference on Computer Science and Education informalization (CSEI), Xinxiang, China, 16–19 August 2019; pp. 168–171. [CrossRef]
37. Zhang, S.; Yao, L.; Sun, A.; Tay, Y. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.* **2019**, *52*, 1–38. [CrossRef]
38. Sitar-Tăut, D.-A.; Mican, D.; Buchmann, R.A. A knowledge-driven digital nudging approach to recommender systems built on a modified Onicescu method. *Expert Syst. Appl.* **2021**, *181*, 115170. [CrossRef]
39. Zhu, Y.; Gong, Y.; Liu, Q.; Ma, Y.; Ou, W.; Zhu, J.; Wang, B.; Guan, Z.; Cai, D. Query-based interactive recommendation by meta-path and adapted attention-gru. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, Beijing, China, 3–7 November 2019; pp. 2585–2593.
40. Bhattacharya, M.; Barapatre, A. Query as Context for Item-to-Item Recommendation. *Comput. J.* **2021**, *64*, 1016–1027.
41. Ren, H.; Leskovec, J. Beta embeddings for multi-hop logical reasoning in knowledge graphs. In Proceedings of the Advances in Neural Information Processing System, Vancouver, Canada, 6–12 December 2020. Available online: https://papers.nips.cc/paper/2020/hash/e43739bba7cdb577e9e3e4e42447f5a5-Abstract.html (accessed on 13 November 2021).
42. Pirro, G. Explaining and suggesting relatedness in knowledge graphs. In *ISWC*; Springer: Cham, Switzerland, 2015; pp. 622–639.
43. Feddoul, L. Semantics-driven Keyword Search over Knowledge Graphs. In Proceedings of the DC@ISWC, Vienna, Austria, 3 November 2020; pp. 17–24.
44. Yan, H.; Wang, Y.; Xu, X. SimG: A Semantic Based Graph Similarity Search Engine. In Proceedings of the 27th International Conference of Advanced Cloud and Big Data, Suzhou, China, 21–22 September 2019; pp. 114–120. [CrossRef]
45. Hamilton, W.; Bajaj, P.; Zitnik, M.; Jurafsky, D.; Leskovec, J. Embedding logical queries on knowledge graphs. In *NeurIPS*; Curran Associates Inc.: Red Hook, NY, USA, 2018; pp. 2030–2041.
46. Ochieng, P. PAROT: Translating natural language to SPARQL. *Expert Syst. Appl. X* **2020**, *5*, 100024.