

Article

Cluster Relation-Augmented Hierarchical Reinforcement Learning for Sparse Knowledge Graph Reasoning

Shun Mao ¹, Yinghan Hong ^{1*}, Guizhen Mai ¹, Jiawei Li ¹, Zhao Chen ^{1,2}, Yike Li ³ and Dan Xiang ¹

¹ School of Artificial Intelligence, Guangzhou Maritime University, Guangzhou 510725, China; maoshun@gzmtu.edu.cn (S.M.); maiguizhen@gzmtu.edu.cn (G.M.); lijiawei@gzmtu.edu.cn (J.L.); danxiang@gzmtu.edu.cn (D.X.)

² School of Remote Sensing and Information Engineering, Wuhan University, Wuhan 430072, China; oliver@whu.edu.cn

³ School of Computer Science and Technology, Beijing Jiaotong University, Beijing 100044, China

* Correspondence: hongyinghan@gzmtu.edu.cn

Abstract

Knowledge graphs (KGs) organize relational facts in a form that supports machine reasoning across many domains. In decentralized Internet of Things (IoT) environments, such graphs can describe interactions among distributed devices and services. However, privacy constraints, intermittent communication, and incomplete observations can leave the resulting graphs sparse and incomplete. This sparsity weakens path availability and makes missing-fact inference difficult. Multi-hop reasoning can recover unobserved links through explicit reasoning paths, but conventional hierarchical reinforcement learning (HRL) approaches may explore inefficiently when only a few valid paths are available. This paper presents *attentiON exploration of cluster relation* (ONION), a framework for sparse KG reasoning. ONION introduces a cluster relation augmentation (CRA) module to add cluster relation cues to the action space, together with a relation-level attention (RLA) mechanism to emphasize relations that are relevant to the query. These components guide hierarchical exploration toward more informative paths. ONION is evaluated on four public sparse-KG benchmarks, and the results show competitive performance with the clearest gains on highly sparse datasets.

Keywords: knowledge graph reasoning; cluster relation; multi-hop reasoning; reinforcement learning



Academic Editors: Arkaitz Zubiaga, Chengzu Dong, Shao Di, Aiting Yao, Xiangyu Song and Juan Zhao

Received: 14 June 2026

Revised: 15 July 2026

Accepted: 16 July 2026

Published: 20 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The rapid deployment of *Internet of Things* (IoT) systems in smart cities and healthcare automation has made privacy protection an essential requirement. At the edge layer, techniques such as data anonymization, hidden-policy access control, and network coding are commonly adopted to reduce unauthorized access. Such mechanisms can restrict which facts are shared or observed, which motivates the sparse and incomplete KG setting studied in this paper. When only partial facts are available, continuous logical inference becomes more difficult. *Knowledge graphs* (KGs) provide a structured way to represent entities and semantic relations in such environments. Related applications include *question answering* [1,2], *recommendation systems* [3], and *information retrieval* [4]. In this paper, the IoT/privacy setting serves as the application motivation rather than a directly simulated experimental environment. We focus on the downstream structural reasoning problem that can arise when only privacy-permitted or intermittently observed facts are available.

The practical value of KGs is often limited by missing facts, especially in domain-specific or privacy-constrained scenarios. Therefore, we focus on the structural reasoning problem induced by such constraints, namely how to recover plausible links when the available KG contains only sparse and incomplete connectivity. Reinforcement learning (RL)-based multi-hop reasoning is appealing because each prediction can be associated with an explicit path. Unlike embedding models [5–8], which mainly use vector representations and one-step scoring, RL-based methods learn graph traversal policies and generate explicit reasoning paths. Hierarchical reinforcement learning (HRL) methods [9–12] decompose the decision process into relation selection and entity selection to reduce the complexity of large action spaces. However, sparse graphs still introduce several challenges:

- Hierarchical decomposition may weaken the semantic connection between high-level planning and low-level execution. When intermediate node information is limited, the state representation may provide insufficient global structural guidance [13,14].
- Many existing frameworks assume relatively dense relation connectivity, whereas practical IoT-oriented KGs may contain very few explicit paths. When a node has only a small number of outgoing relations, the available action space becomes too narrow for effective exploration [15,16].
- Relation-level goals may be poorly aligned with entity-level actions. Without additional semantic guidance or link compensation, repeated or suboptimal paths are easily selected, especially when the target entity is several hops away [13,14].

To address these problems, this paper proposes ONION (*attentiON exploratIon of cluster relatiON*), an HRL framework designed for incomplete knowledge graphs. The framework relies on two mechanisms:

- The CRA mechanism groups entities into semantic clusters and derives cluster relations. These relations expand the action space and provide additional choices in sparse regions.
- The RLA mechanism introduces attention into relation selection, allowing the policy network to emphasize query-relevant relations and avoid unnecessary exploration.

ONION does not treat clustering, attention, or HRL as isolated novelties. Instead, it converts embedding-induced entity clusters into explicit cluster relation actions and lets the high-level relation policy select these actions together with original graph relations. The primary contributions of this paper are summarized below:

- ONION is introduced as an HRL architecture for multi-hop reasoning in sparse KGs, with particular attention to sparse and incomplete graph observations motivated by privacy-aware IoT settings where only part of the triples may be accessible.
- The CRA and RLA mechanisms are designed to expand the reachable reasoning space and improve relation selection through query-aware semantic weighting.
- Results on four sparse-KG benchmark datasets show that ONION achieves competitive performance among path-based multi-hop reasoning baselines, with clearer gains on sparse datasets.

2. Related Work

2.1. Knowledge Graph Embedding

The objective of *Knowledge Graph Embedding* (KGE) is to encode entities and relations as low-dimensional vectors while retaining semantic regularities in the graph. In most KGE methods, a scoring function $f(e_s, r_q, e_o)$ estimates the plausibility of a triple (e_s, r_q, e_o) , and the score is used to rank candidate links. Existing KGE models can be broadly grouped into three categories: (1) translation-based models [5], in which relations are modeled as transformations between head and tail entities; (2) tensor decomposition-based models [6,7],

which learn latent factors from observed triples; and (3) neural network-based models [8,17], which combine embedding features through neural architectures. These methods are effective for link prediction, but they do not directly provide explicit multi-hop paths for individual predictions.

2.2. Multi-Hop Reasoning

Multi-hop reasoning with *Reinforcement Learning* (RL) casts KG inference as a sequential decision process, where relational paths are searched to infer entity links. Earlier approaches, such as PRA [18], relied on depth-first search or random walks. NTP [19] learned logical rules, while NeuralLP [20] combined rule learning with confidence estimation. Although these methods produced explicit reasoning paths, predictive accuracy remained limited in many settings. Later RL-based models strengthened path search in different ways. DeepPath [21] used Monte Carlo policy gradients, MINERVA [22] adopted RNNs with REINFORCE [23], and MultiHopKG [24] combined LSTM [25] with action dropout and reward shaping. DackGR [15] dynamically expanded actions using KG embeddings, PS [26] considered fake path detection, and GaussianPath [27] introduced Bayesian RL to represent uncertainty. To address sparse rewards and large action spaces, HRL models such as HRL [9], RLH [12], HiKGR [14], CURL [13], SbsKGR [10], and HSMH [11] decompose reasoning tasks and enrich state information. Despite these advances, decomposition-related information loss and insufficient paths in sparse graphs remain difficult to resolve.

Different from embedding-guided action expansion methods such as DackGR [15], which use KGE scores to supplement local actions, ONION constructs cluster relation actions from entity groups and injects them into the high-level relation policy. Compared with HRL-based methods such as HiKGR [14], CURL [13], and HSMH [11], which mainly refine hierarchical decomposition or state representation, ONION focuses on path scarcity by expanding the relation layer with data-driven cluster relations. In contrast to concept-level reasoning settings that depend on predefined concepts or external taxonomies, ONION induces clusters from pre-trained KG embeddings and maps them back to intra-cluster and inter-cluster relations in the original graph. Relation-level attention is then applied over the augmented action space to select query-relevant original and cluster relations jointly. The next section describes ONION in detail.

3. Methodology

This section presents the *attentiON exploration of cluster relatioN* (ONION) model for KG reasoning. The model improves the relation layer of the HRL framework through action-space augmentation and policy-network refinement. Graph clustering and cluster relation construction are introduced to provide additional semantic information. The overall architecture is shown in Figure 1. The key distinction of ONION is that cluster information is not used only as an auxiliary embedding feature, but is converted into explicit relation-level actions and optimized inside the HRL decision process.

As illustrated from left to right, ONION first performs entity clustering on the KG and then builds relations among the obtained clusters. The resulting cluster relations are added to the environment used by the relation layer. At each hop, a relation is selected from the augmented space and transferred to the original KG environment, where the entity layer selects the next entity. A complete relation–entity action pair is therefore formed. The clustering module generates entity clusters from pre-trained KG embeddings, and the clusters are mapped back to the original graph for cluster relation construction. An attention mechanism is further used in the relation layer to strengthen the policy network. The following subsections describe these modules in detail. To prevent information

leakage, ConvE is trained only on the training triples, denoted by $\mathcal{T}_{\text{train}}$. The entity embeddings used for clustering, the entity clusters, the cluster relation action space, the bucket structure, the training-time adjacency information, and the cluster-supported candidate transitions are all constructed exclusively from $\mathcal{T}_{\text{train}}$. Validation triples are used only for hyperparameter selection, while test triples are reserved for the final evaluation.

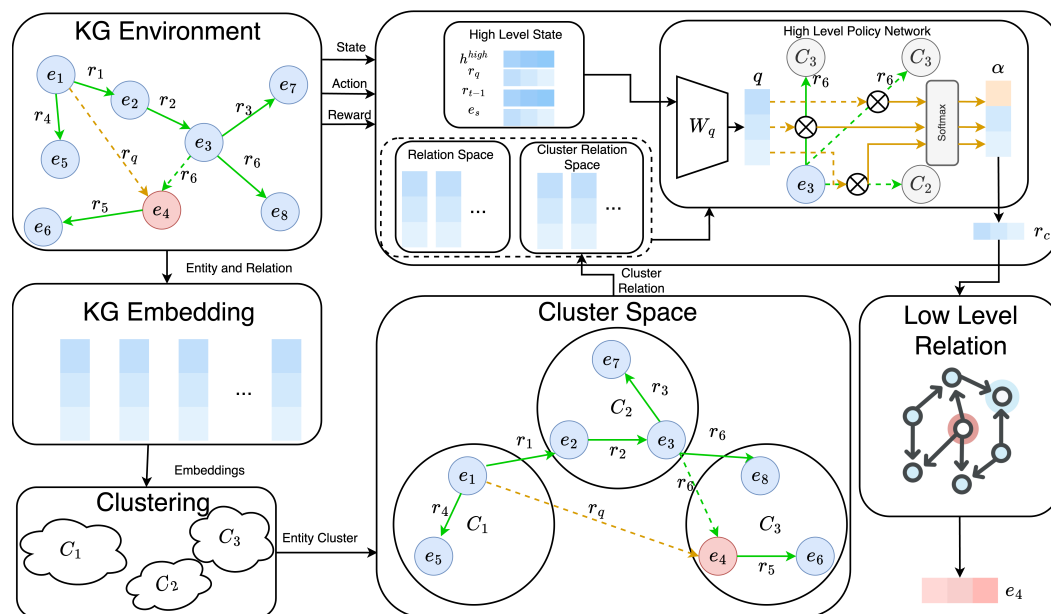


Figure 1. Overview of the ONION framework with the CRA and RLA mechanisms. CRA clusters pre-trained KGE entity embeddings and builds cluster relations to supplement the relation action space. RLA assigns attention to candidate relations, including cluster relations, to improve the relation-layer policy network.

3.1. Problem Definition

Generally, a knowledge graph (KG) is written as $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$, where $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{T}$ denote the entity set, relation set, and triple set, respectively. Each triple $(e_s, r_q, e_o) \in \mathcal{T}$ connects a source entity e_s to a target entity e_o through the directed query relation r_q . KG reasoning predicts the missing target entity for the query $(e_s, r_q, ?)$. Unlike KGE-based scoring, multi-hop reasoning requires explicit paths to support the prediction.

3.2. Entity Clusters and Cluster Relation Construction

Large-scale KGs usually contain many entities and relations, which makes direct graph processing expensive. For example, adjacency-list storage can become costly when frequent access to large graph neighborhoods is required. Relation types, however, are usually far fewer than entities. In FB15K-237-10% and FB15K-237-50%, which are sampled from Freebase [28], increasing the retained triples from 10% to 50% introduces many entities while leaving the relation set unchanged. This pattern indicates that relation information is comparatively concentrated. Entities can also be organized into categories, such as athletes or teams, and entities in the same category often share similar relation patterns. Although such category structure may be learned gradually by an RL agent, the learning process is slow and incomplete under sparse paths. Therefore, entity clusters are obtained through sampling and clustering, and cluster relations are then constructed from these clusters.

Graph clustering provides a way to extract category-level information and inject it into the reasoning model. Structural features such as adjacency lists can be used for clustering, but this strategy is inefficient for large graphs. In this work, ConvE entity embeddings

trained only on $\mathcal{T}_{\text{train}}$ are used instead. These embeddings retain useful information from the training graph while avoiding repeated access to the full graph structure. K-means is then applied to the embeddings to obtain entity clusters. The entity embedding matrix is defined as follows:

$$\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_N]^\top \in \mathbb{R}^{N \times d} \quad (1)$$

where N denotes the number of entities in $\mathcal{E}_{\text{train}}$, d is the embedding dimension, and $\mathbf{e}_i \in \mathbb{R}^d$ is the embedding of the i -th entity. Here, $\mathcal{E}_{\text{train}}$ is the entity set observed in $\mathcal{T}_{\text{train}}$. Because the appropriate cluster number varies across KGs, candidate values of k are evaluated using SSE analysis and validation performance. The clustering error is measured by the sum of squared errors (SSE), computed from squared distances between entities and corresponding cluster centroids:

$$\text{SSE} = \sum_{j=1}^k \sum_{\mathbf{e}_i \in C_j} \|\mathbf{e}_i - \boldsymbol{\mu}_j\|_2^2 \quad (2)$$

where C_j denotes the j -th cluster, $\boldsymbol{\mu}_j$ denotes the centroid of C_j , and $\|\mathbf{e}_i - \boldsymbol{\mu}_j\|_2^2$ is the squared Euclidean distance between entity embedding $\mathbf{e}_i$ and centroid $\boldsymbol{\mu}_j$. SSE values are computed for different values of k to identify the elbow trend, and validation performance is used to select the final setting. After k is determined, the entity embeddings are partitioned by k-means:

$$\{C_1, C_2, \dots, C_k\} = \text{K-Means}(\mathbf{E}, k) \quad (3)$$

In the following definitions, $\mathcal{T}$ denotes $\mathcal{T}_{\text{train}}$ and $\mathcal{E}$ denotes $\mathcal{E}_{\text{train}}$. Thus, the cluster-conditioned records, relation action spaces, bucket assignments, and candidate entities in this construction are obtained exclusively from the training triples.

After entity clusters are obtained, a cluster relation action space is built to supplement the original relation action space. In ONION, a cluster relation is not a newly defined KG predicate and is not a permanent edge between two clusters. It is an original relation label that is supported by a source entity cluster. Each entity is first assigned to a cluster identifier, denoted as $e_i \rightarrow C_j$. Each triple (e_s, r, e_o) is then mapped to the cluster-conditioned record $(C(e_s), r, C(e_o))$, where $C(e)$ denotes the cluster containing entity e . This mapping records whether an original relation label is observed inside one cluster or across different clusters. The intra-cluster and inter-cluster relation sets are defined as $\mathcal{R}_{\text{intra-C}} = \{r \mid \exists C_i \in \{C_1, \dots, C_k\}, \exists e_s, e_o \in C_i, (e_s, r, e_o) \in \mathcal{T}\}$ and $\mathcal{R}_{\text{inter-C}} = \{r \mid \exists C_i, C_j \in \{C_1, \dots, C_k\}, i \neq j, \exists e_s \in C_i, \exists e_o \in C_j, (e_s, r, e_o) \in \mathcal{T}\}$, respectively. Their union forms the global cluster relation action space $\mathcal{A}_C = \mathcal{R}_{\text{intra-C}} \cup \mathcal{R}_{\text{inter-C}}$. For a current entity $e_t \in C_i$, the cluster-specific relation action subset is $\mathcal{A}_{C_i} = \{r \in \mathcal{R} \mid \exists \tilde{e} \in C_i, \exists e' \in \mathcal{E}, (\tilde{e}, r, e') \in \mathcal{T}\}$. Thus, $\mathcal{A}_{C_i}$ contains original relation labels that have outgoing support from the current source cluster, and the cluster identifier is used to retrieve candidates rather than to define new relation types.

3.3. Cluster Relation Augmentation Mechanism

In large KGs, the cluster relation action space may still contain many candidates. A bucketing algorithm is therefore used to organize access to cluster relation candidates and avoid directly scanning the full cluster relation action space at each reasoning step. Clusters with similar relation counts are assigned to the same bucket, and the bucket index is used for accessing the cluster relation action space:

$$\text{Bucket}(C_i) = \max\{l \mid a_l < R(C_i) \leq b_l\} \quad (4)$$

where Bucket denotes the bucketing operation, $R(C_i)$ denotes the number of cluster relation actions associated with cluster C_i , and a_l and b_l are the lower and upper bounds of the

l -th bucket, respectively. In our implementation, these bounds are fixed deterministically rather than tuned as additional hyperparameters, and buckets are assigned only to clusters with $R(C_i) > 0$. Let $R_{\max} = \max_i R(C_i)$. When $R_{\max} > 0$, we set $L = \lceil \log_2(R_{\max} + 1) \rceil$. For $l = 1, \dots, L$, the bucket boundaries are set as $a_l = 2^{l-1} - 1$ and $b_l = 2^l - 1$. If $R(C_i) = 0$, no cluster relation action is added for C_i and the agent uses only the original local relation action space.

When the agent enters the relation layer, the full cluster relation set may not be appropriate for the current entity state. To improve generalization and avoid dependence on a fixed relation subset, action dropout is applied to the cluster-specific relation action space. For a current entity $e_t \in C_i$, relation actions are randomly retained according to a Bernoulli distribution:

$$\mathcal{A}_{C_i}^{\text{drop}} = \left\{ r \mid r \in \mathcal{A}_{C_i}, z_r = 1, z_r \sim \text{Bernoulli}(1 - p_{\text{drop}}^C) \right\} \quad (5)$$

where z_r is the retention indicator for relation action r , and p_{drop}^C is the cluster relation action dropout probability.

After dropout, the remaining cluster relations are merged with the local relation-layer action space $\bar{\mathcal{A}}_t^{\text{high}}$. The augmented high-level action space is written as

$$\mathcal{A}_t^{\text{high}} = \bar{\mathcal{A}}_t^{\text{high}} \cup \mathcal{A}_{C_i}^{\text{drop}}, \quad e_t \in C_i \quad (6)$$

The CRA mechanism introduces hidden category information into the relation-layer action space. Once a cluster-supported relation is selected by the high-level policy, the same original relation label is passed to the low-level entity layer. The model then retrieves candidate entities from relation instances supported by entities in the same source cluster. This operation creates a temporary cluster-supported transition during reasoning, but it does not permanently add new triples to the original KG. Consequently, the agent can explore beyond the immediate outgoing edges of the current entity when local paths are scarce. The goal of this mechanism is to improve sparse KG reasoning by enriching the candidate action space, rather than to explicitly recover or validate missing triples.

3.4. Relation-Level Attention Mechanism

After cluster relations are added to the action space, attention is introduced to improve relation selection in ONION. This subsection first describes the high-level relation layer and then specifies the low-level entity layer for reproducibility, since both policies are optimized during training. In the relation layer, the next-hop relation is selected from the augmented action space and then passed to the entity layer. The high-level layer is defined by four elements: state, action, policy, and reward, denoted as $(\mathcal{S}_{\text{high}}, \mathcal{A}_{\text{high}}, \mathcal{P}_{\text{high}}, \mathcal{R}_{\text{high}})$. These elements are specified as follows:

- **High-Level State $\mathcal{S}_{\text{high}}$:** The high-level state $\mathcal{S}_{\text{high}}$ is formed by the query relation r_q , the previous relation r_{t-1} , and the current entity e_t . At the start of reasoning, the initial state is $\mathcal{S}_0^{\text{high}} = (r_q, r_q^{\text{inv}}, e_s)$, where e_s is the source entity. For the first LSTM update, we set $r_0 = r_q^{\text{inv}}$ and $e_0 = e_s$. After several hops, the final state becomes $\mathcal{S}_T^{\text{high}} = (r_q, r_T, e_T)$.
- **High-Level Action $\mathcal{A}_{\text{high}}$:** The high-level action space includes relations connected to the current entity e_t and the retained cluster relations. A relation action a_t^{high} is sampled from this space before control moves to the low-level entity layer. The local relation action space is $\bar{\mathcal{A}}_t^{\text{high}} = \{r' \mid (e_t, r', e') \in \mathcal{T}_{\text{train}}\}$, where e' denotes a reachable tail entity. The augmented action space $\mathcal{A}_t^{\text{high}}$ is then obtained by Equation (6).

- **High-Level Policy** $\mathcal{P}_{\text{high}}$: The high-level policy network estimates the probability assigned to each candidate relation under the current state. Scaled dot-product attention is adopted for this calculation. Entities and relations are represented by embedding vectors $\mathbf{e} \in \mathbb{R}^d$ and $\mathbf{r} \in \mathbb{R}^d$, respectively. At step t , the high-level state embedding is $\mathbf{s}_t^{\text{high}} = [\mathbf{r}_q, \mathbf{r}_{t-1}, \mathbf{e}_t]$. Let $n_t = |\mathcal{A}_t^{\text{high}}|$ denote the number of candidate relation actions, and let the candidate relation–action embedding matrix be $\mathbf{X}_t^{\text{high}} = [\mathbf{a}_{t,1}^{\text{high}}; \dots; \mathbf{a}_{t,n_t}^{\text{high}}] \in \mathbb{R}^{n_t \times d}$. A high-level LSTM encodes path history from the previous history embedding $\mathbf{h}_{t-1}^{\text{high}}$, previous relation embedding $\mathbf{r}_{t-1}$, and previous entity embedding $\mathbf{e}_{t-1}$, producing the current history embedding $\mathbf{h}_t^{\text{high}}$:

$$\mathbf{h}_t^{\text{high}} = \text{LSTM}_{\text{high}}(\mathbf{h}_{t-1}^{\text{high}}, (\mathbf{r}_{t-1}, \mathbf{e}_{t-1})) \quad (7)$$

The state embedding $\mathbf{s}_t^{\text{high}}$ and history embedding $\mathbf{h}_t^{\text{high}}$ are concatenated and projected into the query vector $\mathbf{q}_t = [\mathbf{s}_t^{\text{high}}; \mathbf{h}_t^{\text{high}}] \mathbf{W}_Q \in \mathbb{R}^{1 \times d_k}$. The candidate relation–action embeddings are projected into the key and value matrices $\mathbf{K}_t = \mathbf{X}_t^{\text{high}} \mathbf{W}_K \in \mathbb{R}^{n_t \times d_k}$ and $\mathbf{V}_t = \mathbf{X}_t^{\text{high}} \mathbf{W}_V \in \mathbb{R}^{n_t \times d_v}$, respectively. The attention output is computed using scaled dot-product attention:

$$\mathbf{o}_t = \text{Attention}(\mathbf{q}_t, \mathbf{K}_t, \mathbf{V}_t) = \text{softmax}\left(\frac{\mathbf{q}_t \mathbf{K}_t^T}{\sqrt{d_k}}\right) \mathbf{V}_t \quad (8)$$

Equivalently, if $\mathbf{k}_{t,i}$ and $\mathbf{v}_{t,i}$ denote the i -th rows of $\mathbf{K}_t$ and $\mathbf{V}_t$, respectively, then the attention weight for the i -th candidate relation is

$$\alpha_{t,i} = \frac{\exp(\mathbf{q}_t \mathbf{k}_{t,i}^T / \sqrt{d_k})}{\sum_{j=1}^{n_t} \exp(\mathbf{q}_t \mathbf{k}_{t,j}^T / \sqrt{d_k})}, \quad \mathbf{o}_t = \sum_{i=1}^{n_t} \alpha_{t,i} \mathbf{v}_{t,i}. \quad (9)$$

Here, the softmax normalization is performed over all n_t candidate relation actions, and $\sqrt{d_k}$ scales the dot product to improve training stability. The attention output $\mathbf{o}_t$ is then used to calculate action logits for all candidate relations. The high-level policy network $\mathcal{P}_{\text{high}}$ therefore produces the distribution $\pi_{\theta_{\text{high}}}(\cdot | \mathbf{s}_t^{\text{high}})$ over candidate relations in $\mathcal{A}_t^{\text{high}}$:

$$\pi_{\theta_{\text{high}}}(\cdot | \mathbf{s}_t^{\text{high}}) = \text{softmax}(\mathbf{X}_t^{\text{high}} \mathbf{W}_o \mathbf{o}_t^T) \quad (10)$$

where $\mathbf{W}_o \in \mathbb{R}^{d \times d_v}$ is a trainable weight matrix. The i -th entry of this distribution gives the probability of selecting action $a_{t,i}^{\text{high}}$. Multinomial sampling is then used to choose the next-hop relation.

- **High-Level Reward** $\mathcal{R}_{\text{high}}$: The high-level reward R_T^{high} is determined by whether the final entity e_T matches the target entity e_o . If the target is reached, the reward is set to 1. Otherwise, a scoring function $f(\mathbf{r}_{\text{sum}}, \mathbf{r}_q)$ derived from pre-trained KG embeddings is used:

$$R_T^{\text{high}} = \begin{cases} 1, & e_T = e_o, \\ f(\mathbf{r}_{\text{sum}}, \mathbf{r}_q), & e_T \neq e_o. \end{cases} \quad (11)$$

where $\mathbf{r}_{\text{sum}} = \sum_{t=1}^T \mathbf{r}_t$ denotes the relation–summary embedding obtained by summing the embeddings of the selected high-level relations along the sampled path, and $\mathbf{r}_q$ denotes the query–relation embedding.

After the high-level policy selects a relation, the decision process is transferred to the low-level entity layer, which chooses the next entity conditioned on the selected relation.

- **Low-Level State $\mathcal{S}_{\text{low}}$:** The low-level entity layer is specified in the same state–action–policy–reward form. After the high-level policy samples relation $r_t = a_t^{\text{high}}$, the selected high-level action is passed to the low-level layer as the original relation label r_t . The low-level state is defined as $s_t^{\text{low}} = (r_q, r_t, e_t)$, where r_q is the query relation, r_t is the selected relation, and e_t is the current entity.
- **Low-Level Action $\mathcal{A}_{\text{low}}$:** The low-level action space is built from local one-hop neighbors and cluster-supported candidates. The local candidate set contains entities directly reachable from the current entity under relation r_t :

$$\mathcal{N}_{\text{local}}(e_t, r_t) = \{e' \mid (e_t, r_t, e') \in \mathcal{T}_{\text{train}}\}. \quad (12)$$

When the selected relation belongs to the retained cluster relation subset $\mathcal{A}_{C_i}^{\text{drop}}$, the low-level layer additionally retrieves entities connected by the same relation from other entities in the current source cluster C_i :

$$\mathcal{N}_{\text{cluster}}(e_t, r_t) = \{e' \mid e_t \in C_i, \exists \tilde{e} \in C_i, (\tilde{e}, r_t, e') \in \mathcal{T}_{\text{train}}\} \setminus \mathcal{N}_{\text{local}}(e_t, r_t). \quad (13)$$

The final low-level action space is defined as

$$\mathcal{A}_t^{\text{low}} = \begin{cases} \mathcal{N}_{\text{local}}(e_t, r_t) \cup \mathcal{N}_{\text{cluster}}(e_t, r_t), & r_t \in \mathcal{A}_{C_i}^{\text{drop}}, \\ \mathcal{N}_{\text{local}}(e_t, r_t), & \text{otherwise.} \end{cases}$$

The low-level action space contains observed KG edges and cluster-supported virtual transitions. Local candidates correspond to observed edges. Cluster-supported candidates are temporary transitions obtained from relation instances of other entities in the same cluster. They are not observed triples or permanent KG edges. The set difference in Equation (13) separates these two cases. This mechanism expands candidate entities only during the current reasoning step and does not modify $\mathcal{T}$.

For example, let e_a and e_b belong to the same cluster. Suppose that $(e_b, r, e_c) \in \mathcal{T}_{\text{train}}$ but $(e_a, r, e_c) \notin \mathcal{T}_{\text{train}}$. When e_a selects r , e_c can be retrieved through e_b as a cluster-supported candidate. This is a virtual transition. By contrast, $(e_a, r, e_d) \in \mathcal{T}_{\text{train}}$ is an observed KG transition. Accordingly, an explicit reasoning trajectory in ONION records the selected relation and entity sequence, but its transitions may be either observed KG edges or cluster-supported virtual transitions.

- **Low-Level Policy $\mathcal{P}_{\text{low}}$:** The low-level policy network estimates the probability of each candidate entity in $\mathcal{A}_t^{\text{low}}$. Let $\mathbf{X}_t^{\text{low}} \in \mathbb{R}^{|\mathcal{A}_t^{\text{low}}| \times d}$ be the embedding matrix of candidate entities. The entity-layer history is updated by a separate low-level LSTM:

$$\mathbf{h}_t^{\text{low}} = \text{LSTM}_{\text{low}}(\mathbf{h}_{t-1}^{\text{low}}, (\mathbf{r}_t, \mathbf{e}_t)). \quad (14)$$

The low-level policy is then computed by

$$\pi_{\theta_{\text{low}}}(\cdot \mid s_t^{\text{low}}) = \text{softmax}(\mathbf{X}_t^{\text{low}} \mathbf{W}_{\text{low}} [\mathbf{r}_q; \mathbf{r}_t; \mathbf{e}_t; \mathbf{h}_t^{\text{low}}]^T) \quad (15)$$

where $\mathbf{W}_{\text{low}}$ is a trainable projection matrix. The sampled entity action a_t^{low} is used as the next entity in the reasoning path. After sampling, the agent sets $e_{t+1} = a_t^{\text{low}}$. If the sampled entity comes from $\mathcal{N}_{\text{cluster}}(e_t, r_t)$, the transition is treated as a cluster-supported reasoning step induced by entities in the same cluster.

- **Low-Level Reward $\mathcal{R}_{\text{low}}$:** The low-level reward follows the terminal-success criterion: $R_T^{\text{low}} = 1$ if $e_T = e_o$ and $R_T^{\text{low}} = 0$ otherwise.

The improved relation layer strengthens relation–semantic learning and assigns greater attention to suitable actions in the augmented cluster relation action space. This design helps the model identify more useful paths and improves reasoning performance.

3.5. Policy Optimization

The training objective is to maximize the expected cumulative rewards of the high-level policy network θ_{high} and the low-level policy network θ_{low} . For each query triple in the training set, the parameters are updated by stochastic gradient ascent. The gradients for the two policies are given as follows:

$$\nabla_{\theta_{\text{high}}} J(\theta_{\text{high}}) \approx \sum_{t=1}^T R_T^{\text{high}} \nabla_{\theta_{\text{high}}} \log \pi_{\theta_{\text{high}}} (a_t^{\text{high}} | s_t^{\text{high}}) \quad (16)$$

$$\nabla_{\theta_{\text{low}}} J(\theta_{\text{low}}) \approx \sum_{t=1}^T R_T^{\text{low}} \nabla_{\theta_{\text{low}}} \log \pi_{\theta_{\text{low}}} (a_t^{\text{low}} | s_t^{\text{low}}) \quad (17)$$

Algorithm 1 summarizes the training process of ONION.

Algorithm 1 Training procedure of the ONION model

Input: Training graph $\mathcal{G}_{\text{train}} = (\mathcal{E}_{\text{train}}, \mathcal{R}_{\text{train}}, \mathcal{T}_{\text{train}})$, training set D_{train} , pre-trained ConvE embeddings Θ_{ConvE}

Output: Optimizable policy network parameters θ_{high} and θ_{low}

- 1: Initialize the entity embeddings using the pre-trained ConvE parameters Θ_{ConvE} , and randomly initialize the policy network parameters θ_{high} and θ_{low}
 - 2: **for** each candidate cluster number k **do**
 - 3: Compute the Sum of Squared Errors (SSE) as defined in Equation (2)
 - 4: **end for**
 - 5: Use SSE analysis and validation performance to select the cluster number k , then apply K-Means clustering on the entity embeddings in Equation (1) to obtain entity clusters in Equation (3)
 - 6: Construct the cluster relation action space $\mathcal{A}_C$ from $\mathcal{T}_{\text{train}}$, and organize it using the bucketing algorithm Bucket as defined in Equation (4)
 - 7: **for** $m \leftarrow 1$ to maximum number of training epochs **do**
 - 8: Randomly sample a query triple (e_s, r_q, e_o) from D_{train}
 - 9: Set the current entity to e_s and initialize the step counter
 - 10: Initialize historical path information $\mathbf{h}_0^{\text{high}}$ and $\mathbf{h}_0^{\text{low}}$ using the high-level and low-level LSTM states
 - 11: **for** $t \leftarrow 1$ to maximum path length T **do**
 - 12: Update the hidden state $\mathbf{h}_t^{\text{high}}$ using the high-level LSTM as defined in Equation (7), then compute high-level state s_t^{high}
 - 13: Compute current cluster relation action space $\mathcal{A}_{C_i}$ for $e_t \in C_i$ and action space $\mathcal{A}_t^{\text{high}}$; apply the action dropout method defined in Equation (5) to $\mathcal{A}_{C_i}$, and augment it into the relation action space as defined in Equation (6)
 - 14: Sample a relation action a_t^{high} using the high-level policy defined in Equation (10), and proceed to the low-level entity layer
 - 15: Update the hidden state $\mathbf{h}_t^{\text{low}}$ using the low-level LSTM as defined in Equation (14)
 - 16: Construct the low-level entity action space $\mathcal{A}_t^{\text{low}}$ by combining local candidates from Equation (12) and, when $a_t^{\text{high}} \in \mathcal{A}_{C_i}^{\text{drop}}$, cluster-supported candidates from Equation (13)
 - 17: Sample an entity action a_t^{low} using the low-level policy defined in Equation (15), and move the agent to the selected entity
 - 18: **end for**
 - 19: Calculate the high-level reward R_T^{high} using Equation (11) and the low-level reward R_T^{low} using the terminal-success criterion
 - 20: Optimize θ_{high} and θ_{low} using the gradient updates defined in Equation (16) and Equation (17), respectively
 - 21: **end for**
-

4. Experimental Results

4.1. Datasets

The proposed method is tested on FB15K-237-10%, FB15K-237-50%, NELL23K, and WD-singer. These benchmarks originate from Freebase [28], NELL [29], and Wikidata [30]. NELL23K is adopted from HSMH [11] as a sparse benchmark constructed from the NELL995 benchmark introduced in DeepPath [21], WD-singer is collected from Wikidata, and the two FB15K-237 subsets are sampled from FB15K-237 [31] by retaining 10% and 50% of triples, respectively. The two FB15K-237 subsets provide controllable triple-retention ratios, whereas NELL23K and WD-singer contain weakly connected regions.

These public benchmarks are not intended to model a specific IoT privacy mechanism. Instead, they provide public sparse-KG settings for evaluating reasoning under limited graph connectivity. Table 1 summarizes the dataset statistics.

Table 1. Statistics of the four experimental datasets.

Dataset	#Ent	#Rel	#Fact	#Degree	
				Mean	Median
FB15K-237-10%	11,512	237	60,985	5.8	4
FB15K-237-50%	14,149	237	173,830	13.0	13
NELL23K	22,925	200	35,358	2.21	1
WD-singer	10,282	135	20,508	2.35	2

4.2. Experimental Setup

4.2.1. Baseline Models

ONION is compared with four KGE models: TransE [5], DistMult [6], ComplEx [7], and ConvE [8]. Three rule- or path-based models are included, namely NeuralLP [20], NTP [19], and AnyBURL [32]. Seven RL-based multi-hop reasoning baselines are also considered: MINERVA [22], MultiHopKG [24], RuleGuider [33], DacKGR [15], SbsKGR [10], SparKGR [16], and HSMH [11].

4.2.2. Evaluation Protocol

For evaluation, each test triple (e_s, r_q, e_o) is converted into the query $(e_s, r_q, ?)$, after which candidate target entities are ranked. We use the filtered ranking protocol. For each query, other known correct entities are removed from the candidate set, while the target entity is retained. Candidate entities are ranked in descending order according to their final prediction scores. During inference, ONION uses deterministic beam search with a beam size of 128 and a maximum reasoning path length of three hops. No additional trajectories are sampled during inference. When multiple beam paths reach the same candidate entity, the maximum cumulative log path probability is retained as its score. The 20 rollouts used during training are only used for policy-gradient optimization and are not used for test-time ranking. Following prior studies, Mean Reciprocal Rank (MRR) and Hits@K are used as metrics. MRR averages the reciprocal ranks of correct target entities, whereas Hits@K measures the proportion of correct entities ranked within the top K. For readability, all MRR and Hits@K values reported in the experimental tables are multiplied by 100. Higher metric values indicate stronger reasoning performance.

4.2.3. Implementation Details

For fair comparison, part of the experimental configuration follows MultiHopKG [24]. Entity and relation embeddings are set to 200 dimensions for all datasets. History information is encoded by three-layer and four-layer LSTMs [25] with a hidden size of 200. Embedding and network parameters are initialized with Xavier initialization [34]. Adam [35] is adopted for optimization. ConvE [8] serves as the embedding model on all datasets. ConvE is trained only on the training triples, and all graph-derived structures used by ONION are constructed from these triples. Validation triples are used only for hyperparameter selection, while test triples are reserved for final evaluation. The final hyperparameter choices for each dataset are reported in Table 2. For each training triple (e_s, r_q, e_o) , a reverse triple (e_o, r_q^{-1}, e_s) is added. ONION introduces two additional hyperparameters: the cluster number k and the cluster relation action dropout probability p_{drop}^C . The parameter p_{drop}^C controls the proportion of cluster relation actions removed during training to improve generalization. Additionally, some baseline results are taken from the

corresponding papers, while others are reproduced using released code when available. A dash indicates that the result was not reported or could not be reproduced under the same dataset setting. For ONION in the main results and for the ablation and clustering analyses, we report mean $\pm$ standard deviation over three random seeds.

Table 2. Final dataset-specific hyperparameter settings used for the ONION experiments.

Dataset	Learning Rate	Entropy Coefficient β	Batch Size	Epochs	k	p_{drop}^C
FB15K-237-10%	0.001	0.02	128	40	15	0.5
FB15K-237-50%	0.001	0.02	128	40	35	0.5
NELL23K	0.003	0.05	128	30	30	0.1
WD-singer	0.001	0.03	128	30	30	0.3

4.3. Link Prediction Results

Table 3 reports the link prediction performance of ONION and the baselines. The ONION results in Tables 3–5 are reported as the mean $\pm$ standard deviation obtained from the same three random seeds, whereas many baseline results are single estimates taken from the original papers or reproduced under the available settings. Therefore, comparisons with these baselines are descriptive rather than formal significance tests. The results evaluate sparse-KG reasoning under public benchmark settings, providing evidence for reasoning under structural sparsity conditions motivated by privacy-aware IoT applications. Among path-based multi-hop reasoning models, ONION achieves competitive performance on most metrics across the four datasets, with clearer gains on sparser datasets. This trend is mainly due to attention-guided exploration over the cluster relation action space, which provides informative relation choices when explicit graph paths are limited.

The gains are more pronounced on NELL23K and WD-singer than on FB15K-237-10% and FB15K-237-50%. The two FB15K-237 subsets have denser relation coverage, so the CRA mechanism contributes less visibly. In sparser KGs, the added cluster-level guidance leads to clearer improvements. Among path-based baselines, ONION shows margins larger than its own three-seed standard deviation on most metrics, but it does not outperform every baseline on every metric. For example, HSMH remains stronger on H@10 for FB15K-237-10%, and ConvE achieves higher MRR and H@10 than ONION on WD-singer.

Knowledge graph embedding models still show strong raw prediction accuracy. Nevertheless, ONION narrows the gap with embedding-based methods when compared with other multi-hop reasoning models, while retaining explicit path-level reasoning. Since ConvE is used to initialize the embeddings in ONION, ConvE is treated as an embedding-only reference rather than a directly comparable path-based reasoning baseline. Embedding models can produce accurate rankings, but the relation paths behind a prediction are not directly revealed. In contrast, ONION records explicit relation–entity traversal paths for its predictions.

Table 3. Prediction results on four benchmarks. @3/@10 denote Hits@3/Hits@10.

Model	FB15K-237-10%			FB15K-237-50%			NELL23K			WD-Singer		
	MRR	@3	@10	MRR	@3	@10	MRR	@3	@10	MRR	@3	@10
TransE *	10.4	16.0	27.8	17.6	23.5	40.3	8.5	10.8	24.8	20.9	32.2	44.5
DistMult *	7.5	7.4	17.0	17.9	20.3	38.0	11.5	12.0	23.1	24.5	26.9	39.9
ComplEx *	22.3	24.0	34.8	30.2	32.6	48.1	24.1	26.4	40.8	39.0	43.3	51.3
ConvE *	<u>24.6</u>	<u>26.1</u>	<u>39.2</u>	<u>31.2</u>	<u>34.3</u>	<u>50.0</u>	<u>27.7</u>	<u>30.0</u>	<u>46.5</u>	<u>44.7</u>	<u>47.9</u>	<u>56.8</u>
NeuralLP *	8.0	7.1	13.9	18.1	19.3	24.5	12.3	13.0	26.4	31.8	33.5	48.1
NTP *	8.2	11.5	16.8	22.3	23.0	30.8	13.1	15.0	24.0	29.3	31.0	44.3
AnyBURL *	15.0	15.4	26.8	19.7	21.4	35.0	17.7	18.4	25.3	39.1	34.2	48.5
MINERVA *	7.7	7.9	12.1	23.1	23.9	31.2	14.9	15.3	25.3	33.6	37.3	45.0
MultiHopKG *	13.7	14.5	21.7	29.1	31.8	44.8	17.9	18.7	29.8	35.5	41.2	47.4
RuleGuider *	17.7	15.5	21.8	28.3	29.6	44.6	18.2	20.0	29.8	39.4	43.2	50.6
DacKGR *	21.9	23.8	33.8	29.4	31.9	45.8	20.0	21.7	33.1	38.2	42.2	50.5
SbsKGR [†]	21.5	24.6	34.8	27.9	32.1	42.9	16.5	18.1	28.2	-	-	-
SparKGR [†]	22.8	24.5	35.0	29.2	32.0	46.2	20.3	22.2	33.9	39.3	43.7	50.7
HSMH [†]	23.0	25.1	37.2	29.6	32.6	46.7	20.2	22.6	34.1	-	-	-
ONION	23.7 _{±0.2}	25.7 _{±0.2}	35.8 _{±0.4}	30.4 _{±0.6}	33.5 _{±0.4}	47.5 _{±0.8}	24.4 _{±0.1}	26.1 _{±0.1}	40.0 _{±0.3}	43.6 _{±0.2}	48.3 _{±0.3}	55.2 _{±0.3}

* denotes a result reproduced in our experiments, whereas [†] denotes a result reported in the corresponding publication. Boldface and underlining indicate the best scores among path-based and embedding-based methods, respectively.

Table 4. Ablation study results on WD-singer. CRA and RLA denote the cluster relation augmentation mechanism and relation-level attention mechanism, respectively.

Model	MRR	H@3	H@10
MultiHopKG	35.5 _{±0.6}	41.2 _{±0.3}	47.4 _{±0.5}
ONION (w/o CRA+RLA)	41.7 _{±0.7}	45.8 _{±0.4}	52.2 _{±0.4}
ONION (w/o CRA)	41.8 _{±0.2}	45.9 _{±0.3}	52.4 _{±0.2}
ONION (w/o RLA)	42.8 _{±0.3}	47.4 _{±0.5}	54.5 _{±0.6}
ONION	43.6 _{±0.2}	48.3 _{±0.3}	55.2 _{±0.3}

Boldface indicates the best scores.

Table 5. Comparison of clustering methods for ONION on WD-singer.

Model	MRR	H@3	H@10
ONION (k-means)	43.6 _{±0.2}	48.3 _{±0.3}	55.2 _{±0.3}
ONION (BIRCH)	42.9 _{±0.4}	47.8 _{±0.1}	54.7 _{±0.3}
ONION (hierarchical clustering)	43.1 _{±0.6}	44.9 _{±0.2}	55.0 _{±0.2}

Boldface indicates the best scores.

4.4. Ablation Study

An ablation study is conducted to examine the two main components of ONION, namely cluster relation augmentation (CRA) and relation-level attention (RLA). Each component is removed separately to measure its influence on reasoning performance. We conduct the ablation study on WD-singer because the CRA mechanism is expected to have a clear effect in sparse KGs. The results are listed in Table 4, where CRA refers to the cluster relation augmentation mechanism and RLA refers to the relation-level attention mechanism.

The ablation experiments are carried out on the WD-singer dataset, with MultiHopKG [24] used as a reference model. When CRA and RLA are both removed, model performance decreases clearly. Removing either component also reduces the scores, and the decrease is larger after CRA removal. The difference between ONION (w/o CRA + RLA) and ONION (w/o CRA) is small and within the observed variation, suggesting that RLA

has a limited effect when the cluster-augmented relation space is removed. Its effect becomes clearer when applied together with CRA, where it helps weight relations in the augmented action space. This indicates that both modules are useful, while cluster relation information is particularly important for sparse graph reasoning. With CRA and RLA jointly enabled, semantic information among entities and relations can be captured more effectively, thereby improving reasoning ability.

4.5. Analysis

This section further analyzes ONION from two aspects: entity-cluster visualization and the effect of cluster number. Several common clustering methods are also tested under the same experimental setting to observe how clustering choice affects model performance.

The left part of Figure 2 shows a two-dimensional UMAP projection of the 200-dimensional ConvE entity embeddings on WD-singer under the selected cluster number $k = 30$. The projection uses $n_{\text{components}} = 2$, $n_{\text{neighbors}} = 30$, and $\text{min_dist} = 0.2$. Each point represents an entity, while its color and label C_j indicate the k-means cluster assigned to that entity. These labels are cluster identifiers rather than predefined entity categories or KG relation types. The visualization suggests that pre-trained KG embeddings provide useful features for grouping entities in this dataset. Based on these clusters, a cluster relation action space carrying additional KG information is built, which supports multi-hop path discovery.

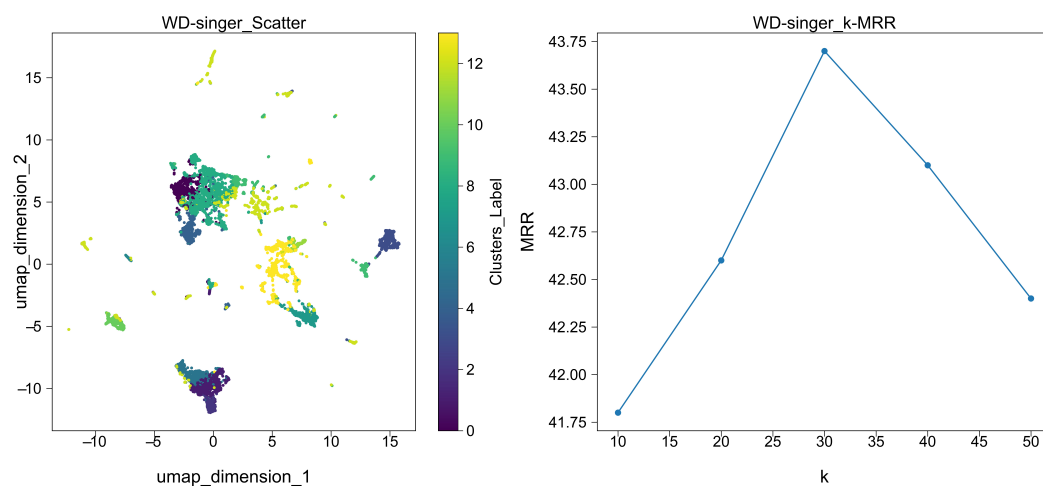


Figure 2. Clustering visualization analysis. The left panel shows a two-dimensional UMAP projection of the 200-dimensional ConvE entity embeddings on WD-singer, using $n_{\text{components}} = 2$, $n_{\text{neighbors}} = 30$, and $\text{min_dist} = 0.2$. Each point represents an entity, and its color and label C_j denote the k-means cluster assigned to that entity. The right panel reports the reasoning performance obtained with different values of k and identifies $k = 30$ as the selected setting.

The right part of Figure 2 reports the results obtained with different values of k on WD-singer. The best performance is observed at $k = 30$. When k is too small or too large, the generated clusters become less suitable, introducing noisy cluster relations and weakening reasoning performance.

Table 5 compares k-means with BIRCH and hierarchical clustering for constructing cluster relations on WD-singer. The results show that clustering choice affects the quality of generated cluster relations, but all three methods can support reasoning under appropriate settings. Hierarchical clustering obtains comparable MRR and H@10 but a lower H@3 than k-means and BIRCH, and this gap is larger than the reported standard deviations across three runs. This suggests that its cluster structure may place some correct entities outside the top-three positions. This finding indicates that clustering can build a useful structured

relation space. In ONION, clustering mainly serves as an auxiliary module for exploring higher-level semantic structures, similar to the role of pre-trained embeddings in supplying additional information. A detailed comparison of clustering algorithms is beyond the main scope of this work.

5. Conclusions

This paper has presented ONION, a KG reasoning method that combines cluster relation augmentation with relation-level attention. The cluster relation module adds category-level relation information to the action space, thereby mitigating path scarcity in sparse KGs. The attention module guides the relation-layer policy toward query-relevant relations and improves exploration efficiency. Experiments on four benchmark datasets show that ONION achieves competitive multi-hop reasoning performance, and the ablation results support the utility of the proposed components. These findings support the sparse-KG reasoning component motivated by privacy-aware IoT data collection, while the present evaluation is based on public sparse-KG benchmarks rather than IoT-specific privacy datasets. Overall, ONION achieves competitive results among existing multi-hop reasoning baselines and narrows the performance gap with embedding-based models. Its recorded reasoning trajectories distinguish observed KG edges from cluster-supported virtual transitions.

Author Contributions: Conceptualization, S.M. and Y.H.; methodology, S.M.; software, S.M.; validation, S.M., G.M., J.L., Z.C. and Y.L.; formal analysis, S.M.; investigation, S.M., G.M., Y.L.; data curation, S.M. and G.M.; writing—original draft preparation, S.M.; writing—review and editing, S.M., Y.H. and G.M.; visualization, S.M.; supervision, Y.H. and D.X.; project administration, Y.H. and D.X.; funding acquisition, Y.H. and D.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Technology Planning Project of Guangdong Province, China (grant numbers KTP20240831 and KTP20240254); the Key Scientific Research Projects in General Colleges and Universities of Guangdong Province (grant number 2025ZDZX3020); and the Guangdong Key Discipline Research Capacity Building Project (grant numbers 2024ZDJS055 and 2024ZDJS056).

Data Availability Statement: The source code and experiment configurations of ONION are available at <https://github.com/syunnm/ONION> (accessed on 18 July 2026).

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

References

1. Zhan, X.; Huang, Y.; Dong, X.; Cao, Q.; Liang, X. PathReasoner: Explainable Reasoning Paths for Commonsense Question Answering. *Knowl.-Based Syst.* **2022**, *235*, 107612.
2. Li, J.; Song, S.; Yan, S.; Hu, G.; Lai, C.; Zhou, Y. DANTE: Dialog Graph Enhanced Prompt Learning for Conversational Question Answering over KGs. *Knowl.-Based Syst.* **2024**, *301*, 112294.
3. Guo, Q.; Zhuang, F.; Qin, C.; Zhu, H.; Xie, X.; Xiong, H.; He, Q. A Survey on Knowledge Graph-Based Recommender Systems. *IEEE Trans. Knowl. Data Eng.* **2022**, *34*, 3549–3568.
4. Liu, Z.; Xiong, C.; Sun, M.; Liu, Z. Entity-Duet Neural Ranking: Understanding the Role of Knowledge Graph Semantics in Neural Information Retrieval. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Melbourne, Australia, 15–20 July 2018; pp. 2395–2405.
5. Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; Yakhnenko, O. Translating Embeddings for Modeling Multi-Relational Data. In Proceedings of the 27th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; pp. 2787–2795.
6. Yang, B.; Yih, S.W.t.; He, X.; Gao, J.; Deng, L. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
7. Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; Bouchard, G. Complex Embeddings for Simple Link Prediction. In Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 2071–2080.

8. Dettmers, T.; Minervini, P.; Stenetorp, P.; Riedel, S. Convolutional 2d Knowledge Graph Embeddings. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 1811–1818.
9. Zheng, M.; Zhou, Y.; Cui, Q. Hierarchical Policy Network with Multi-Agent for Knowledge Graph Reasoning Based on Reinforcement Learning. In Proceedings of the 14th International Conference on Knowledge Science, Engineering and Management, Tokyo, Japan, 14–16 August 2021; pp. 445–457.
10. Zhu, A.; Ouyang, D.; Liang, S.; Shao, J. Step by Step: A Hierarchical Framework for Multi-Hop Knowledge Graph Reasoning with Reinforcement Learning. *Knowl.-Based Syst.* **2022**, *248*, 108843.
11. Wang, D.; Li, B.; Song, B.; Chen, C.; Yu, F.R. HSMH: A Hierarchical Sequence Multi-Hop Reasoning Model with Reinforcement Learning. *IEEE Trans. Knowl. Data Eng.* **2023**, *36*, 1638–1649.
12. Wan, G.; Pan, S.; Gong, C.; Zhou, C.; Haffari, G. Reasoning Like Human: Hierarchical Reinforcement Learning for Knowledge Graph Reasoning. In Proceedings of the 29th International Joint Conference on Artificial Intelligence, Yokohama, Japan, 7–15 January 2021; pp. 1926–1932.
13. Zhang, D.; Yuan, Z.; Liu, H.; Xiong, H.; et al. Learning to Walk with Dual Agents for Knowledge Graph Reasoning. In Proceedings of the 36th AAAI Conference on Artificial Intelligence, Virtual, 22 February–1 March 2022; pp. 5932–5941.
14. Yan, C.; Zhao, F.; Zhang, Y. From Concept to Instance: Hierarchical Reinforced Knowledge Graph Reasoning. *IEEE Trans. Emerg. Top. Comput. Intell.* **2024**, *8*, 3665–3677.
15. Lv, X.; Han, X.; Hou, L.; Li, J.; Liu, Z.; Zhang, W.; Zhang, Y.; Kong, H.; Wu, S. Dynamic Anticipation and Completion for Multi-Hop Reasoning over Sparse Knowledge Graph. In Proceedings of the 25th Conference on Empirical Methods in Natural Language Processing, Suzhou, China, 4–9 November 2020; pp. 5694–5703.
16. Xia, Y.; Lan, M.; Luo, J.; Chen, X.; Zhou, G. Iterative Rule-Guided Reasoning over Sparse Knowledge Graphs with Deep Reinforcement Learning. *Inf. Process. Manag.* **2022**, *59*, 103040.
17. Shang, C.; Tang, Y.; Huang, J.; Bi, J.; He, X.; Zhou, B. End-to-End Structure-Aware Convolutional Networks for Knowledge Base Completion. In Proceedings of the 33rd AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; pp. 3060–3067.
18. Lao, N.; Mitchell, T.; Cohen, W. Random Walk Inference and Learning in a Large Scale Knowledge Base. In Proceedings of the 16th Conference on Empirical Methods in Natural Language Processing, Edinburgh, UK, 27–31 July 2011; pp. 529–539.
19. Rocktäschel, T.; Riedel, S. End-to-End Differentiable Proving. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 3788–3800.
20. Yang, F.; Yang, Z.; Cohen, W.W. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 2319–2328.
21. Xiong, W.; Hoang, T.; Wang, W.Y. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. In Proceedings of the 22nd Conference on Empirical Methods in Natural Language Processing, Abu Dhabi, United Arab Emirates, 7–11 December 2017; pp. 564–573.
22. Das, R.; Dhuliawala, S.; Zaheer, M.; Vilnis, L.; Durugkar, I.; Krishnamurthy, A.; Smola, A.; McCallum, A. Go for a Walk and Arrive at the Answer: Reasoning over Paths in Knowledge Bases Using Reinforcement Learning. In Proceedings of the 6th International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
23. Williams, R.J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.* **1992**, *8*, 229–256.
24. Lin, X.V.; Socher, R.; Xiong, C. Multi-Hop Knowledge Graph Reasoning with Reward Shaping. In Proceedings of the 23rd Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 2–4 November 2018; pp. 3243–3253.
25. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780.
26. Jiang, C.; Zhu, T.; Zhou, H.; Liu, C.; Deng, T.; Hu, C.; Li, J. Path Spuriousness-Aware Reinforcement Learning for Multi-Hop Knowledge Graph Reasoning. In Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, Dubrovnik, Croatia, 2–6 May 2023; pp. 3173–3184.
27. Wan, G.; Du, B. GaussianPath: A Bayesian Multi-Hop Reasoning Framework for Knowledge Graph Reasoning. In Proceedings of the 35th AAAI Conference on Artificial Intelligence, Virtual, 2–9 February 2021; pp. 4393–4401.
28. Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; Taylor, J. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In Proceedings of the 34th ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 15–18 May 2008; pp. 1247–1250.
29. Carlson, A.; Betteridge, J.; Kisiel, B.; Settles, B.; Hruschka, E.; Mitchell, T. Toward an Architecture for Never-Ending Language Learning. In Proceedings of the 24th AAAI Conference on Artificial Intelligence, Atlanta, GA, USA, 11–15 July 2010; pp. 1306–1313.
30. Vrandečić, D.; Krötzsch, M. Wikidata: A Free Collaborative Knowledgebase. *Commun. ACM* **2014**, *57*, 78–85.

31. Toutanova, K.; Chen, D.; Pantel, P.; Poon, H.; Choudhury, P.; Gamon, M. Representing Text for Joint Embedding of Text and Knowledge Bases. In Proceedings of the 20th Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 17–21 September 2015; pp. 1499–1509.
32. Meilicke, C.; Chekol, M.W.; Ruffinelli, D.; Stuckenschmidt, H. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, China, 10–16 August 2019; pp. 3137–3143.
33. Lei, D.; Jiang, G.; Gu, X.; Sun, K.; Mao, Y.; Ren, X. Learning Collaborative Agents with Rule Guidance for Knowledge Graph Reasoning. In Proceedings of the 25th Conference on Empirical Methods in Natural Language Processing, Virtual, 16–20 November 2020; pp. 8541–8547.
34. Glorot, X.; Bengio, Y. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, Sardinia, Italy, 13–15 May 2010; pp. 249–256.
35. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.