CleanGraph: Human-in-the-loop Knowledge Graph Refinement and Completion

Tyler Bikaun, Michael Stewart and **Wei Liu**
The University of Western Australia
tyler.bikaun@research.uwa.edu.au

Abstract

This paper presents **CleanGraph**, an interactive web-based tool designed to facilitate the refinement and completion of knowledge graphs. Maintaining the reliability of knowledge graphs, which are grounded in high-quality and error-free facts, is crucial for real-world applications such as question-answering and information retrieval systems. These graphs are often automatically assembled from textual sources by extracting semantic triples via information extraction. However, assuring the quality of these extracted triples, especially when dealing with large or low-quality datasets, can pose a significant challenge and adversely affect the performance of downstream applications. CleanGraph allows users to perform Create, Read, Update, and Delete (CRUD) operations on their graphs, as well as apply models in the form of plugins for graph refinement and completion tasks. These functionalities enable users to enhance the integrity and reliability of their graph data. A demonstration of CleanGraph and its source code can be accessed at `https://github.com/nlp-tlp/CleanGraph` under the MIT License.

1 Introduction

In the current data-centric era, where data is frequently referred to as the 'new oil', knowledge graphs (KGs)[1]—structured representations of facts, encompassing semantically defined entities and relations—stand at the forefront of harnessing its value across various tasks, including question-answering (Huang et al., 2019), information retrieval (Wise et al., 2020), recommendation (Guo et al., 2020), and reasoning (Chen et al., 2020a).

Text-based knowledge graphs, a product of information extraction methods such as entity recognition, relation extraction, and entity linking (Ji et al., 2022), have seen immense growth through automation, leading to expansive, encyclopedic graphs predominantly from high-quality web data (Bollacker et al., 2008; Mitchell et al., 2018) using techniques like distance supervision (Mintz et al., 2009). Despite these advances, creating domain-specific knowledge graphs from diverse sources, including user-generated content or specialised fields like medicine, law enforcement, and engineering, has received less focus (Abu-Salih, 2021). These domains present unique challenges due to resource scarcity, potential data quality issues, and the need for expert verification of factual accuracy. These difficulties have hindered the production of reliable, cost-effective knowledge graphs in these domains, leaving potential downstream applications underexplored compared to general-domain counterparts.

While the need for high-quality and complete knowledge graphs predates the advent of large language models (LLMs) (Paulheim, 2017), the widespread availability and capabilities of LLMs for zero or few-shot learning (Brown et al., 2020) have further underscored this necessity. LLMs have democratised knowledge graph construction from various text corpora (Ye et al., 2022), thus lowering the barriers to creating domain-specific graphs; however, they are not guaranteed to be factually correct. Consequently, maintaining the quality and completeness of these graphs remains paramount, especially for downstream applications like question-answering and recommendation, where factuality is crucial. As a result, there has been a concerted effort among researchers to improve knowledge graph quality and coverage via knowledge graph refinement (KGR) and completion (KGC) (Paulheim, 2017; Rossi et al., 2021; Ji et al., 2022).

However, there remains a deficiency in task-specific software capable of interactive, human-in-the-loop operations for these tasks. Presently, the available software predominantly supports visuali-

[1]Alongside knowledge bases.

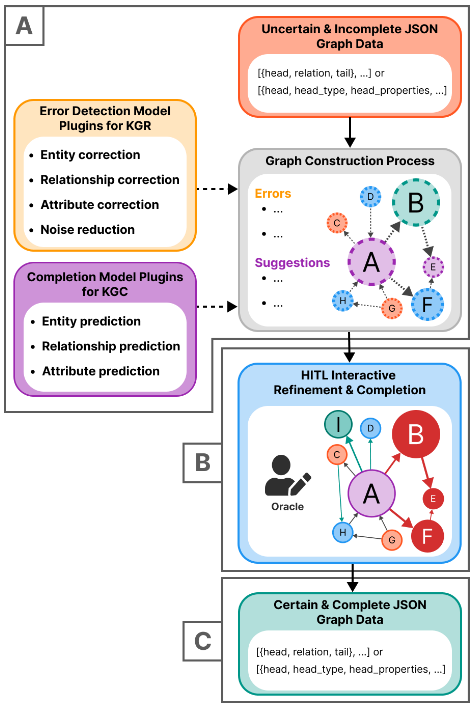

A
Uncertain & Incomplete JSON Graph Data
[{head, relation, tail}, ...] or
[{head, head_type, head_properties, ...]
Error Detection Model Plugins for KGR
Entity correction
Relationship correction
Attribute correction
Noise reduction
Completion Model Plugins for KGC
Entity prediction
Relationship prediction
Attribute prediction
Graph Construction Process
Errors
Suggestions
B
HITL Interactive Refinement & Completion
Oracle
C
Certain & Complete JSON Graph Data
[{head, relation, tail}, ...] or
[{head, head_type, head_properties, ...]

Figure 1: Schematic overview of the CleanGraph tool illustrating (A) graph data input, along with the use of optional model plugins for knowledge graph refinement (KGR) and completion (KGC), (B) the inclusion of human-in-the-loop (HITL) operations in the process, and (C) graph data output.

sation and querying but lacks open accessibility or the capacity to incorporate KGR and KGC models easily. As a solution, such software could employ domain experts—or oracles—for quality assurance and validation, enhancing data accuracy before integration into downstream applications.

To address this gap, we introduce **CleanGraph**, a powerful yet intuitive web-based tool developed for introspection of, and interaction with, knowledge graphs[2] and results of KGR and KGC models (Figures 1 and 2). CleanGraph simplifies knowledge graph verification and refinement, making it user-friendly for technical and non-technical users. With built-in features that support visualisation and interaction, along with customisable KGR and KGC model plugins, CleanGraph assures an effortless quality control experience (Figure 1). This paper details the design, functionality, and architecture of CleanGraph, demonstrating its potential to enhance the integrity and reliability of knowledge graphs.

[2]Represented as untyped and typed property graphs.

2 Preliminaries - Knowledge graph refinement and completion

Knowledge graph refinement (KGR) and completion (KGC) are distinct, essential techniques used to enhance the quality and comprehensiveness of knowledge graphs (Paulheim, 2017; Chen et al., 2020b). Despite extensive research in this domain, there remains a scarcity of accessible software allowing the direct involvement of an oracle in these processes. This section outlines both techniques, their subtasks, and their latest advancements.

2.1 Knowledge graph refinement (KGR)

KGR enhances the accuracy and relevance of the *existing* facts within a knowledge graph (Paulheim, 2017). This includes modifying or deleting incorrect or outdated entities (nodes), relationships (edges), or properties. The goal of KGR is to maintain the graph's accuracy, reliability, and overall quality. KGR tasks encompass:

- **Entity correction:** Checking and rectifying incorrect entity representations (e.g., resolving ambiguities, synonyms, etc.) and types.
- **Relationship correction:** Examining and fixing erroneous relationships between entities, as well as types and directions.
- **Property correction:** Validating and correcting the property values associated with entities or relationships.
- **Noise reduction:** Removing irrelevant or redundant entities and relationships.

KGR challenges stem from the dynamic nature of knowledge leading to outdated information, misinterpretations, data input errors, and lack of context. Extensive research effort has been devoted to KGR, resulting in the development of statistical, machine learning, and deep learning techniques (Paulheim, 2017). Current advancements in KGR focus on utilising knowledge representations like knowledge graph embeddings (Ji et al., 2022).

2.2 Knowledge graph completion (KGC)

In contrast, KGC aims to *fill in the missing gaps* within the knowledge graph (Chen et al., 2020b). This involves predicting and adding missing entities, relationships, or properties, enhancing the graph's comprehensiveness. KGC tasks include:

- **Entity prediction:** Inferring and adding new missing entities from the graph.

- **Relationship prediction:** Discovering and adding missing relationships between existing entities.

- **Property prediction:** Predicting and adding missing entity or relationship properties.

KGC challenges arise from incomplete source data, the complexity of inferring unknown relations or properties, and the inherent uncertainty of predictions. Ensuring the accuracy of new information to prevent polluting the graph with incorrect data is also a major concern. Similar to KGR, KGC has received significant research attention, with recent advancements made in applying graph embeddings (Rossi et al., 2021; Choudhary et al., 2021; Ji et al., 2022).

3 Key features

The key features of CleanGraph, designed to simplify the introspection of knowledge graphs, are elaborated upon as follows.

3.1 Graph creation and data structure

Building a graph in CleanGraph is a user-friendly process. Users simply upload or input a JSON array of semantic triples, minimally in the `{head, relation, tail}` format[3]. Models for error detection (KGR) and completion (KGC) can optionally be selected at this stage. To create more intricate property graphs, additional optional fields such as `types` and `properties` can be supplied (Figure 1A). Once the initial graph is constructed, any selected error detection and completion models are applied. This process populates the `errors` and `properties` fields of the relevant nodes and edges in the graph which are subsequently rendered in the user interface for human interaction (Figure 2).

3.2 Graph visualisation and interaction

Visualisation While viewing graph data in a tabular structure like `(head, relation, tail)` is sufficient for simple, untyped graphs, this approach is inefficient for larger, complex graphs. To address this, CleanGraph adopts a visually intuitive approach by representing graph data in its native format: as frequency-weighted nodes and edges (Figure 2).

Graphs are displayed as directed acyclic graphs using a simulated force-directed layout, facilitated by the d3 visualisation library[4]. This layout, compared to tabular formats, allows users to visually parse the graph, identify patterns, and interact with nodes and edges easily through mouse actions and shortcuts. Users can quickly grasp the frequencies and types of nodes and edges, aided by colour coding.

To manage information overload, CleanGraph partitions the entire graph into focused subgraphs; each centred on a single node. Users can then 'paginate' through these subgraphs, effectively managing large graph data (Figure 3). In addition, errors and suggestions identified by the EDM and CM plugins are displayed adjacent to the relevant nodes and edges, offering instant feedback to users (Figure 2).

Interaction CleanGraph enables intuitive interaction with the graph. Users can manipulate the entire graph view through panning and zooming. Direct engagement with nodes and edges is possible via mouse clicks and shortcuts. Clicking on a node or edge reveals its details, properties, and any errors and suggestions (Figure 2). Users can toggle the `reviewed` state of these elements or remove them from the graph entirely.

3.3 Comprehensive graph CRUD operations

CleanGraph is designed to provide comprehensive Create, Read, Update, and Delete (CRUD) operations across graphs, specifically accommodating the property graph format[5]. It effectively handles nodes and edges, permitting them to support arbitrary key-value pair properties of any data type except objects and arrays.

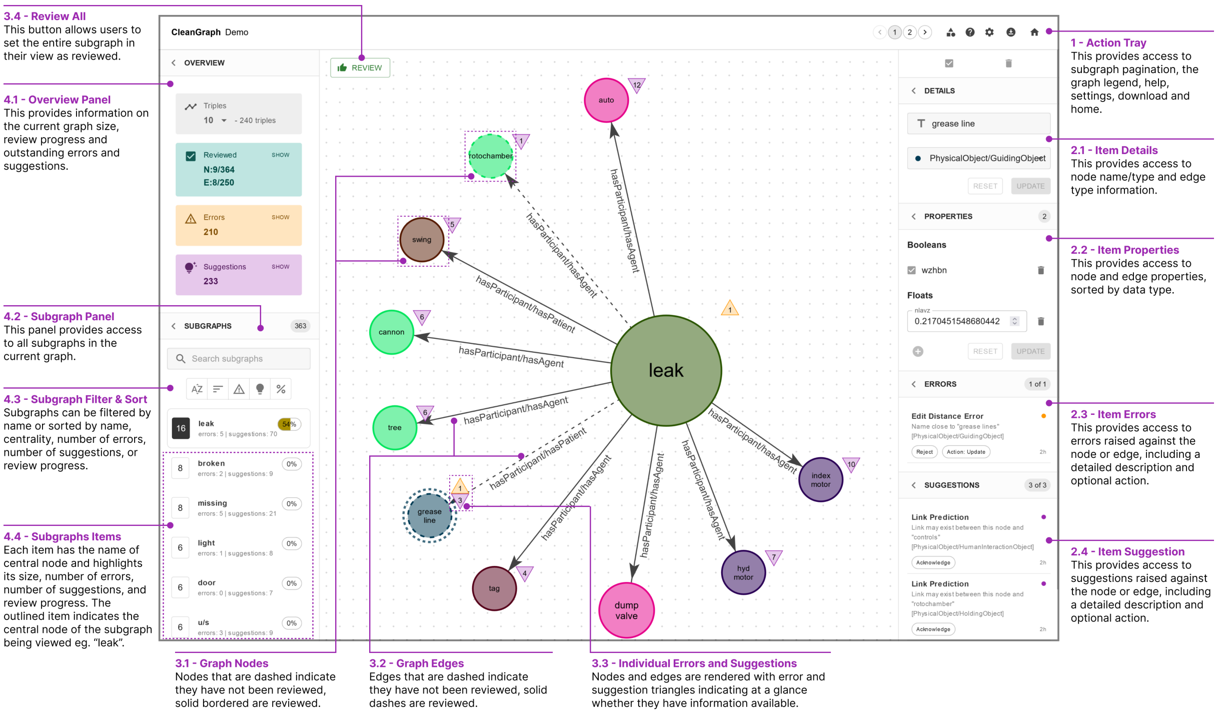
The core CRUD operations that are supported to make the manipulation of graph data[6] seamless include (1) the Addition of new nodes or edges to the graph, (2) Item review for error detection and graph refinement (Figure 2), (3) Subgraph merging (section 3.3.2), (4) Acknowledgement and action on detected errors and suggested modifications (Figure 2), (5) Reversal of edge direction to correct relational flow, and (6) Item deletion (section

[3]For an untyped graph.

[4]Implemented via *react-force-graph*.

[5]This allows any arbitrary graph structure to be supported, even RDF-graphs, where the minimum amount of information is a head, relation and tail.

[6]*item(s)* refer to nodes and edges.



3.4 - Review All
This button allows users to set the entire subgraph in their view as reviewed.

4.1 - Overview Panel
This provides information on the current graph size, review progress and outstanding errors and suggestions.

4.2 - Subgraph Panel
This panel provides access to all subgraphs in the current graph.

4.3 - Subgraph Filter & Sort
Subgraphs can be filtered by name or sorted by name, centrality, number of errors, number of suggestions, or review progress.

4.4 - Subgraphs Items
Each item has the name of central node and highlights its size, number of errors, number of suggestions, and review progress. The outlined item indicates the central node of the subgraph being viewed eg. "leak".

1 - Action Tray
This provides access to subgraph pagination, the graph legend, help, settings, download and home.

2.1 - Item Details
This provides access to node name/type and edge type information.

2.2 - Item Properties
This provides access to node and edge properties, sorted by data type.

2.3 - Item Errors
This provides access to errors raised against the node or edge, including a detailed description and optional action.

2.4 - Item Suggestion
This provides access to suggestions raised against the node or edge, including a detailed description and optional action.

3.1 - Graph Nodes
Nodes that are dashed indicate they have not been reviewed, solid bordered are reviewed.

3.2 - Graph Edges
Edges that are dashed indicate they have not been reviewed, solid dashes are reviewed.

3.3 - Individual Errors and Suggestions
Nodes and edges are rendered with error and suggestion triangles indicating at a glance whether they have information available.

Figure 2: User interface of CleanGraph: Starting clockwise from the top right, (1) the action tray and subgraph pagination, (2) a secondary sidebar showing details, properties, errors, and suggestions for the chosen node or edge, (3) an interactive graph visualisation, and finally, (4) a primary sidebar displaying a progress overview and subgraphs.

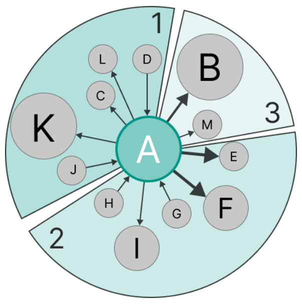


Figure 3: Illustration of CleanGraph's subgraph pagination process: A subgraph centred on the node (A) with 12 connected edges is split into 3 'pages' of 5 triples (size) for manageable viewing.

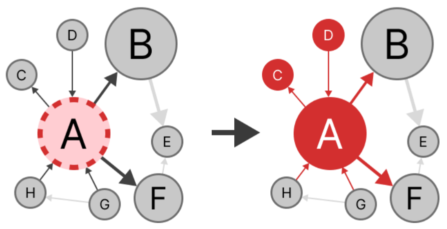


Figure 4: CleanGraph's 1-hop Item Deletion Illustrated: The removal of node (A) consequently eliminates all its corresponding edges and any nodes (C, D) that would become orphaned due to this operation.

3.3.1). These operations facilitate the integration of detected errors by KGR models and predictions made by KGC models into the graph. The subgraph merging and item deletion functionalities are in further detail below.

3.3.1 Item deletion

CleanGraph facilitates KGR by enabling users to perform item deletions. To maintain the directed acyclic graph structure, it automatically propagates 1-hop edge removal and orphan node elimination when nodes or edges are deleted (Figure 4).

3.3.2 Subgraph merging

CleanGraph's subgraph merging feature resolves entity errors. When a node's name or type changes, the system checks for potential conflicts with existing nodes. On detecting a conflict, it prompts the user to approve a proposed node merge. If approved, the respective subgraphs of the conflicting nodes merge, incorporating not only nodes and edges, but also associated properties, errors, suggestions, and frequencies for a comprehensive and accurate unification of the node data (Figure 5).

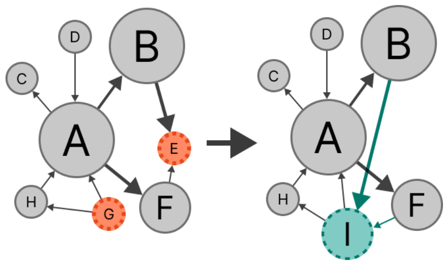


Figure 5: CleanGraph's Node Merge Illustrated: The merging of node (E) into (G) increments the node frequency and redistributes corresponding edges, resulting in a new node (I).

3.4 Plugin architecture

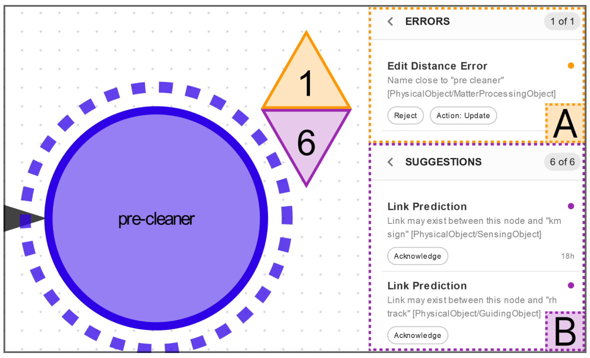
CleanGraph's plugin architecture enables flexible support for KGR and KGC tasks. By accommodating a broad spectrum of Error Detection Models (EDMs for KGR) and Completion Models (CMs for KGC), users can integrate any model that complies with the plugin interface, enabling seamless integration with the CleanGraph user interface (Figure 2). Note: plugins are optional and only expedite the graph quality assurance process.

3.4.1 Authoring plugins

CleanGraph's plugin authoring process is designed to promote collaboration and encourage the exchange of plugins via its open-source repository. These plugins may span heuristic, statistical, and deep-learning models, which we hope to evolve over time, leading to a shared resource community. To create a plugin, authors simply add a Python script that complies with the predefined plugin interface and adheres to the specified Pydantic input/output data models, into the server's `/plugins` directory. The input model is fed the graph as a set of semantic triples, and the output model returns a list of errors or suggestions, each containing relevant information and potential actions (defined and validated as Pydantic data models). These results are then rendered interactively on the user interface. Plugins are easily accessible from the user interface due to CleanGraph's plugin manager, ensuring reproducible usage across different graph construction processes.

3.4.2 Error detection models

Error detection model outputs, 'errors', are displayed in the UI as easily identifiable 'notifications' (Figures 2 and 6A) attached to graph nodes and edges. Each notification can optionally include a corrective action, such as recommendations to



ERRORS 1 of 1

Edit Distance Error
Name close to "pre cleaner" [PhysicalObject/MatterProcessingObject]
Reject | Action: Update

SUGGESTIONS 6 of 6

Link Prediction
Link may exist between this node and "km sign" [PhysicalObject/SensingObject]
Acknowledge 18h

Link Prediction
Link may exist between this node and "rh track" [PhysicalObject/GuidingObject]
Acknowledge

Figure 6: Display of CleanGraph's Error and Suggestion Features: (A) shows errors associated with a particular item (node), offering an optional corrective action (yellow triangle), while (B) presents informational suggestions (purple triangle). Both errors and suggestions can be acknowledged by the user.

merge nodes, update node types or labels, reverse edge directions, modify edge types, and update or delete node/edge properties. These actions assist the user in undertaking pertinent actions during the quality assurance process. CleanGraph defines a finite set of actions — `Update` and `Delete` — embodied in Pydantic data models, which can be freely used by EDM plugins. Nevertheless, actions are not mandatory; reviewers can simply 'acknowledge' them within the UI.

3.4.3 Completion models

Analogous to EDMs, completion model outputs, 'suggestions', are also displayed in the UI similarly as 'notifications' on nodes and edges (Figures 2 and 6B). Suggestions notifications can have `Create` and `Update` actions, corresponding to the addition of new nodes/edges (`Create`) or updated node/edge properties (`Update`).

4 System architecture

CleanGraph is built using the FARM[7] web application stack and comprises a server and a client. The server, implemented using FastAPI in Python, collaborates seamlessly with the client developed using a blend of HTML, Javascript (React), and CSS. The choice of a Python-based server is so that there is native support for KGR and KGC models typically implemented in Python libraries such as PyTorch and TensorFlow.

A NoSQL database, MongoDB, is used for data management, and stores graph data in four collections (`Graphs`, `Triples`, `Nodes`, and `Edges`).

[7]FARM - FastAPI, React, MongoDB.

Table 1: Comparative analysis of CleanGraph with existing tools for knowledge graph management and manipulation. Abbreviations: *HITL*, *KGR*, and *KGC* refer to Human-in-the-Loop, Knowledge Graph Refinement, and Knowledge Graph Completion, respectively.

Tool	Visualisation	Querying	HITL CRUD	HITL KGR	HITL KGC	Open-Source	RDF-based	Property Graph-based
AllegroGraph	✓	✓	-	-	-	-	✓	-
Blazegraph	-	✓	-	-	-	-	✓	-
Neo4J	✓	✓	-	-	-	✓	-	✓
JanusGraph	✓	✓	-	-	-	✓	-	✓
TigerGraph	✓	✓	-	-	-	-	-	✓
CleanGraph	✓	-	✓	✓	✓	✓	-	✓

The choice of NoSQL over graph databases such as Neo4J owes to its support for graph-structured data and its ability to store arbitrarily structured data on nodes and edges. This flexibility circumvents the need for intricate representations often required by graph databases when dealing with complex data, such as nested arrays of complex objects for errors, suggestions, and properties.

5 Comparison with existing tools

Table 1 compares CleanGraph with existing tools for knowledge graph management. While most tools, such as AllegroGraph and Neo4J, are designed to handle large-scale knowledge graphs and lack Human-in-the-Loop (HITL) support, CleanGraph focuses on managing smaller text-based knowledge graphs with robust HITL functionalities.

Existing tools excel in visualising, storing, and querying extensive graph structures, but their support for HITL refinement and completion is often limited. Integration with external environments like Python is possible (e.g. Py2Neo for Neo4J), but these solutions generally fall short of a seamless iterative workflow involving human interaction. CleanGraph, however, offers visualisation, property graph-based support, and comprehensive HITL CRUD, KGR, and KGC functionalities. Its unique design emphasises flexibility and adaptability, enabling continuous human-involved refinement and completion.

It's important to recognize that this comparison may not be entirely fair, as the tools serve different purposes. Large graph databases aim to efficiently handle substantial data, while CleanGraph prioritises interactive, iterative development. This distinction, along with variations in open-source availability and feature sets, reflects the diverse goals and target audiences of these tools.

6 Conclusion and future work

This paper presented CleanGraph, a versatile and interactive tool designed for the refinement and completion of knowledge graphs. CleanGraph's unique contributions include providing comprehensive capabilities for graph CRUD operations and review that does not require any programming experience, alongside arbitrary knowledge graph refinement and completion models through its plugin infrastructure.

While CleanGraph is a robust, ready-to-use tool, it continues to evolve. Future enhancements include: i) expanding the range of available Error Detection and Completion plugins to incorporate graph embedding models, ii) supporting graph queries, iii) optimising performance for large-scale knowledge graphs, and iv) accommodating RDF-based semantic knowledge graphs.

Acknowledgements

This research is supported by the Australian Research Council through the Centre for Transforming Maintenance through Data Science (grant number IC180100030), funded by the Australian Government. Additionally, Bikaun acknowledges funding from the Mineral Research Institute of Western Australia. Bikaun and Liu acknowledge the support from ARC Discovery Grant DP150102405.

References

Bilal Abu-Salih. 2021. Domain-specific knowledge graphs: A survey. *Journal of Network and Computer Applications*, 185:103076.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Xiaojun Chen, Shengbin Jia, and Yang Xiang. 2020a. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*, 141:112948.

Zhe Chen, Yuehan Wang, Bin Zhao, Jing Cheng, Xin Zhao, and Zongtao Duan. 2020b. Knowledge graph completion: A review. *Ieee Access*, 8:192435–192456.

Shivani Choudhary, Tarun Luthra, Ashima Mittal, and Rajat Singh. 2021. A survey of knowledge graph embedding and their applications. *arXiv preprint arXiv:2107.07842*.

Qingyu Guo, Fuzhen Zhuang, Chuan Qin, Hengshu Zhu, Xing Xie, Hui Xiong, and Qing He. 2020. A survey on knowledge graph-based recommender systems. *IEEE Transactions on Knowledge and Data Engineering*, 34(8):3549–3568.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 105–113.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S Yu. 2022. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Trans Neural Netw Learn Syst*, 33(2):494–514.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011.

T Mitchell, W Cohen, E Hruschka, P Talukdar, B Yang, J Betteridge, A Carlson, B Dalvi, M Gardner, B Kisiel, J Krishnamurthy, N Lao, K Mazaitis, T Mohamed, N Nakashole, E Platanios, A Ritter, M Samadi, B Settles, R Wang, D Wijaya, A Gupta, X Chen, A Saparov, M Greaves, and J Welling. 2018. Never-ending learning. *Commun. ACM*, 61(5):103–115.

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

Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. 2021. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49.

Colby Wise, Vassilis N Ioannidis, Miguel Romero Calvo, Xiang Song, George Price, Ninad Kulkarni, Ryan Brand, Parminder Bhatia, and George Karypis. 2020. Covid-19 knowledge graph: Accelerating information retrieval and discovery for scientific literature. *arXiv preprint arXiv:2007.12731*.

Hongbin Ye, Ningyu Zhang, Hui Chen, and Huajun Chen. 2022. Generative knowledge graph construction: A review. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1–17.