Decide: Knowledge-Based Version Incompatibility Detection in Deep Learning Stacks

Zihan Zhou
The University of Hong Kong
Hong Kong, Hong Kong
zihan2@connect.hku.hk

Zhongkai Zhao
National University of Singapore
Singapore, Singapore
zhongkai.zhao@u.nus.edu

Bonan Kou
Purdue University
West Lafayette, USA
koub@purdue.edu

Tianyi Zhang
Purdue University
West Lafayette, USA
tianyi@purdue.edu

ABSTRACT

Version incompatibility issues are prevalent when reusing or reproducing deep learning (DL) models and applications. Compared with official API documentation, which is often incomplete or out-of-date, Stack Overflow (SO) discussions possess a wealth of version knowledge that has not been explored by previous approaches. To bridge this gap, we present Decide, a web-based visualization of a knowledge graph that contains 2,376 version knowledge extracted from SO discussions. As an interactive tool, Decide allows users to easily check whether two libraries are compatible and explore compatibility knowledge of certain DL stack components with or without the version specified. A video demonstrating the usage of Decide is available at https://youtu.be/wqPxF2ZaZo0.

CCS CONCEPTS

• **Software and its engineering → Software organization and properties**.

KEYWORDS

Version Compatibility, Knowledge Graph, Deep Learning

ACM Reference Format:
Zihan Zhou, Zhongkai Zhao, Bonan Kou, and Tianyi Zhang. 2024. Decide: Knowledge-Based Version Incompatibility Detection in Deep Learning Stacks. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3663529.3663796

1 INTRODUCTION

Deep learning (DL) has been widely applied in diverse domains, such as computer vision [12], natural language processing [14], and autonomous driving [8]. However, most DL applications are built on a complex and heterogeneous DL stack [4], including libraries, runtime systems, drivers, operating systems, and hardware components. The intricate dependencies among these components make version issues hard to detect and resolve and a significant cause of failed builds in DL projects [2].

Several techniques [3, 6, 9, 10, 13] have been proposed to detect dependency issues. However, they primarily focus on detecting dependency issues among Python packages, with limited consideration for issues related to drivers, operating systems, and hardware components [10]. Moreover, these techniques heavily rely on documented version constraints and dependencies specified in PyPI and official API documentation [9], often overlooking undocumented issues that developers encounter in practice. In contrast, Q&A platforms such as Stack Overflow (SO) offer up-to-date and comprehensive information on dependency issues and their solutions in real-world scenarios. This wealth of version knowledge has not been explored by previous work.

In this paper, we present Decide, an interactive web tool that visualizes a knowledge graph containing 2,376 version knowledge extracted from SO discussions. Here, version knowledge refers to (in)compatibility relationships between any two DL stack components (e.g., `Python 3.7` is compatible with `TensorFlow 1.5.0`). These relationships are extracted from 355,000 SO posts that contain version knowledge via UnifiedQA [5]. The extracted knowledge was further consolidated into a knowledge graph among 48 popular DL stack components in Decide. Our evaluation of 343 version (in)compatibility relations shows relations in the knowledge graph are highly accurate (83.7% of the sampled relations are correct).

Specifically, Decide provides three major functionalities:

(1) **Visualization**. Decide visualizes a knowledge graph with (in)compatibility relations between DL components.
(2) **Search Features**. Decide utilizes the function calling ability of GPT-4 to parse natural language search queries of the users, enabling smooth access to any components or relations.
(3) **Citations**. Decide cites source SO posts, allowing users to validate the extracted knowledge.

In our technical paper [15], we showed this knowledge graph can be used to effectively detect version issues in DL projects. For experiment results, the code of Decide, and more information on our technical paper, please check https://github.com/LexieZhou/Decide.

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663796

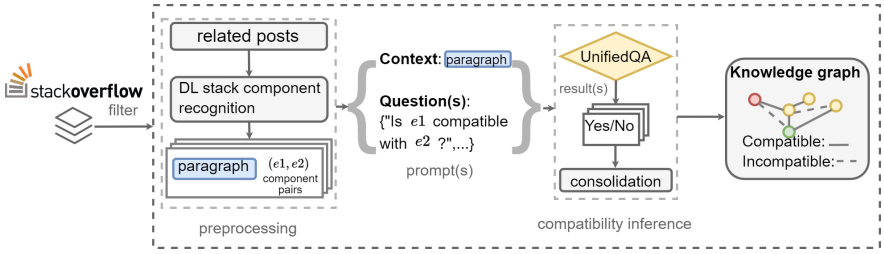

Figure 1: An overview of the knowledge graph construction and incompatibility detection process

Context: `tensorflow 1.13` doesn't work with `cuda 10.1` because of the following: *"ImportError: libcublas.so.10.0: cannot open shared object file: No such file or directory"*. `tensorflow` is looking for `libcublas.so.10.0` whereas cuda provides `libcublas.so.10.1.0.105`.
Question: Does `tensorflow 1.13` work with `cuda 10.1`?

Answer from UnifiedQA: No.

Figure 2: QA examples for version compatibility inference

Table 1: Version matching patterns

Regex Pattern	Matched Examples
`v{0,1}\d+(\.d+){1,2}`	`3.7, 2.4.3, v2.3, v1.13.5`
`v{0,1}\d+(\.\d+){0,1}(\.x){0,1}`	`3.x, 1.3.x, v1.x, v2.2.x`
`(COMPONENT)(-| |_)v{0,1}\d+`	`python v3, cuda-8, Windows 64`

2 KNOWLEDGE GRAPH CONSTRUCTION

This section presents how DECIDE extracts version compatibility knowledge from SO posts to build a knowledge graph. Figure 1 provides an overview of the approach. Please refer to our technical paper [15] for more details.

2.1 Data Collection and Filtering

We downloaded the Stack Exchange Data Dump [1] with 53 million Stack Overflow posts from July 31, 2008 to September 5, 2021. We manually analyzed 798 popular SO tags to identify 46 DL-related tags. [1] We filtered the SO posts to only retain posts tagged with at least one of these tags. 4.9M posts remained after this step. Further refinement using 22 linguistic patterns summarized from 150 posts with version incompatibility knowledge[2] related to version compatibility issues narrowed the selection to 549K posts. Finally, we removed unaccepted answer posts to ensure the quality of our dataset. After this step, 355K posts remain. A random sample of 384 posts (CI=95%) showed that 84.9% of them contain version incompatibility knowledge.

2.2 DL Stack Component Recognition

Not all paragraphs in a version-related SO post mentioned version compatibility information. We designed a filtering mechanism to locate paragraphs containing version compatibility information to improve knowledge extraction efficiency.DECIDE only selected paragraphs that mention at least two different versioned components. In the current implementation, DECIDE supports the recognition of 48 popular components[3] across five DL stack layers. These components were manually identified from all DL components appearing in the 200 posts with the highest vote score (i.e., upvotes minus downvotes). The authors also added synonyms or aliases for these components to improve recognition accuracy.

Furthermore, we designed three regex patterns to identify versions mentioned in a paragraph, as shown in table 1. Finally, DECIDE matches each component with the closest version in the dependency tree of a sentence using a weighted stable matching algorithm [11].

2.3 Compatibility Inference via a Pre-trained QA Model

DECIDE infers the compatibility relationship between components based on the paragraph information. In this work, we propose a novel approach to reframe the relationship classification task as a Question-Answering (QA) task and use UnifiedQA [5] to infer the compatibility between components. UnifiedQA is a large model with 3 billion parameters pre-trained on eight datasets. It has been demonstrated to understand deep semantics in natural language and achieve state-of-the-art performance in multiple QA benchmarks [5]. UnifiedQA takes two inputs—a *question* and a *context document* from which the answer is extracted. DECIDE uses the paragraph as a context document and asks UnifiedQA a yes-or-no question to infer the compatibility between the two components. Figure 2 illustrates a QA example from the real SO post—[Post 55028552]. Considering prompt design has a noticeable impact on model performance [7], we experimented with eight question templates designed based on the 22 linguistic patterns identified in the post-filtering procedure and found the best prompt[4]. If conflicts exist among SO posts, DECIDE chooses the relationship supported by the most posts.

To evaluate the accuracy, we randomly sampled 343 relations from a total of 2,376. For each relation, two authors independently verified whether the relation is true by searching online or performing experiments. Then, they compared their verification results and resolved any disagreement. The Cohen's Kappa score was 0.89.

[1]The complete list of tags can be found at https://github.com/KKZ20/DECIDE/blob/main/DECIDE/docs/SO_tags.json
[2]The complete list of linguistic patterns can be found in the supplementary material.
[3]A complete list of 48 DL stack components can be found at https://github.com/KKZ20/DECIDE/blob/main/DECIDE/docs/DL_Stack_Components.txt
[4]Both the prompts and our experiment results can be found at https://github.com/KKZ20/DECIDE/blob/main/DECIDE/docs/DL_Stack_Components.txt

After independent verification by two authors, 287 relations were confirmed as correct, resulting in an overall accuracy of 83.7%.

3 KNOWLEDGE GRAPH VISUALIZATION

Decide is built with `React.js` for web application and `Node.js` for server. Decide consists of four key components (Figure 3): (1) the Compatibility Visualizer (A), (2) the Information Panel (B), (3) the Search Bar (C), and (4) the Statistical Panel (D).

3.1 The Compatibility Visualizer

The Compatibility Visualizer (Figure 3 A) shows the knowledge graph that visualizes 2,376 compatibility relationships between DL components.

Knowledge Graph is defined as $G = \langle N, L \rangle$, where:

$$N = \{n_i \mid i \geq 0, \forall n_i \in \{l, r, d, c, h\}\}$$
$$L = \{l_j \mid j \geq 0, \forall l_j \in \{comp, incomp\}\}$$

N is the set of nodes denoting DL components with their version numbers. Decide supports the recognition of DL components across five different DL stack layers [4]: (1) *library layer* (l) that contains the popular frameworks (e.g., PyTorch) and other libraries (e.g., NumPy, SciPy), (2) *runtime layer* (r) that contains the execution interpreters or virtual machines of programming languages (e.g., JVM), (3) *driver layer* (d) that includes hardware drivers and accelerated SDKs (e.g., CUDA, cuDNN), (4) *OS/container layer*(c) that includes the operating systems and other containers or virtual environments (e.g., Anaconda, Docker), (5) *hardware layer* (h) that includes the hardware and chips (e.g., CPU, GPU).

L is the set of links representing the compatible (*comp*) or incompatible (*incomp*) relationship between two components, denoted by the symbols $\leftrightarrow$ and $\nleftrightarrow$ respectively. For a pair of versioned components, let *#Compatible* represent the number of posts that Decide infers a compatible relationship between them, and *#Incompatible* denote the number of posts that Decide infers an incompatible relationship. We define the confidence score of the relationship between two versioned components as follows: $\textit{confidence score} = \frac{\#Compatible - \#Incompatible}{\#Compatible + \#Incompatible}$. If the *confidence score* is a positive number, it implies a compatible relationship. Otherwise, it implies an incompatible relationship. Relationships with a neutral confidence score (*confidence score* = 0) are discarded.

Once we had acquired version knowledge from SO posts, we restructured the version knowledge data into node data (N) and link data (L) and utilized the JavaScript library `D3.js` [5] to construct the knowledge graph in the Compatibility Visualizer.

3.2 The Information Panel

The Information Panel (Figure 3 B) displays detailed information regarding the DL components themselves as well as the (in)compatibility relationships between them.

3.2.1 DL Library Statistics. We extracted all the statistics about DL libraries from `Libraries.io`, a website that provides comprehensive library information. This retrieved information [6] includes details such as keywords, licenses, and dependencies. When a node is clicked, Decide retrieves the corresponding component. If the component is a DL library, Decide will display detailed information using the extracted statistics.

3.2.2 Relationship Details. When a link is clicked, the information panel will show details of the (in)compatibility relationship between the two DL components. Upon clicking, Decide retrieves the link data and displays their predicted (in)compatibility, the confidence score, all the relevant SO posts, and votes of posts. Additionally, we utilize the post IDs stored in the link data to associate the post title and URL further. This allows users to obtain an overview of the posts and access the original SO discussions for further exploration.

3.3 The Search Bar

The search bar (Figure 3 C) provides users three query options to interact with the knowledge graph. When a query is received, Decide applies filters to the node data and link data, resulting in a smaller dataset that includes relevant nodes and links. The filtered data is then used to generate a focused and concise knowledge graph for exploration.

The first query option allows users to quickly check the compatibility between two different DL components using natural language queries. For example, a user can inquire, *"Does* `Python 3.6.8` *work with* `Ubuntu 16.04.6`*?"*. To facilitate this process, Decide employs the function calling ability of GPT-4 or regular expression pattern matching (Table 1) to identify the two versioned components. Decide then filters the node data and link data. Subsequently, the knowledge graph shows only the mentioned components and their (in)compatibility relationships.

Second, users can search for a versioned DL component to see all the associated version knowledge. Using keyword matching, Decide identifies the versioned component in the search query. Subsequently, Decide searches through the available node data to gather all possible versioned nodes matching the search criteria, which are then presented in the search results, allowing users to select. Once users confirm their search selection, Decide further filters the knowledge graph to include the selected component, its connected DL components, and their respective (in)compatible relationships. By doing so, Decide provides users with a comprehensive view of the version knowledge surrounding the chosen component. For example, searching for `Python 3.5` will display `Python 3.5` and its related (in)compatible DL components.

Third, users can search for a DL stack component without a specified version. Decide will show all versions of that component and their version knowledge. Similarly, Decide uses keyword matching to identify the mentioned component and further queries the knowledge graph.

3.4 The Statistical Panel

To provide easy access, Decide also includes the Statistical Panel (Figure 3 D), which displays the five most popular components with common version issues from each DL stack layer. We calculated the number of version knowledge for each component and identified the top five most discussed components in each DL stack layer. Since version issues about these components are very common, we create shortcuts for these components in the statistical panel.

[5] Details about `D3.js` can be found at https://d3js.org/

[6] The retrieved stats data can be found at https://github.com/LexieZhou/Decide/blob/main/stats/stats.json

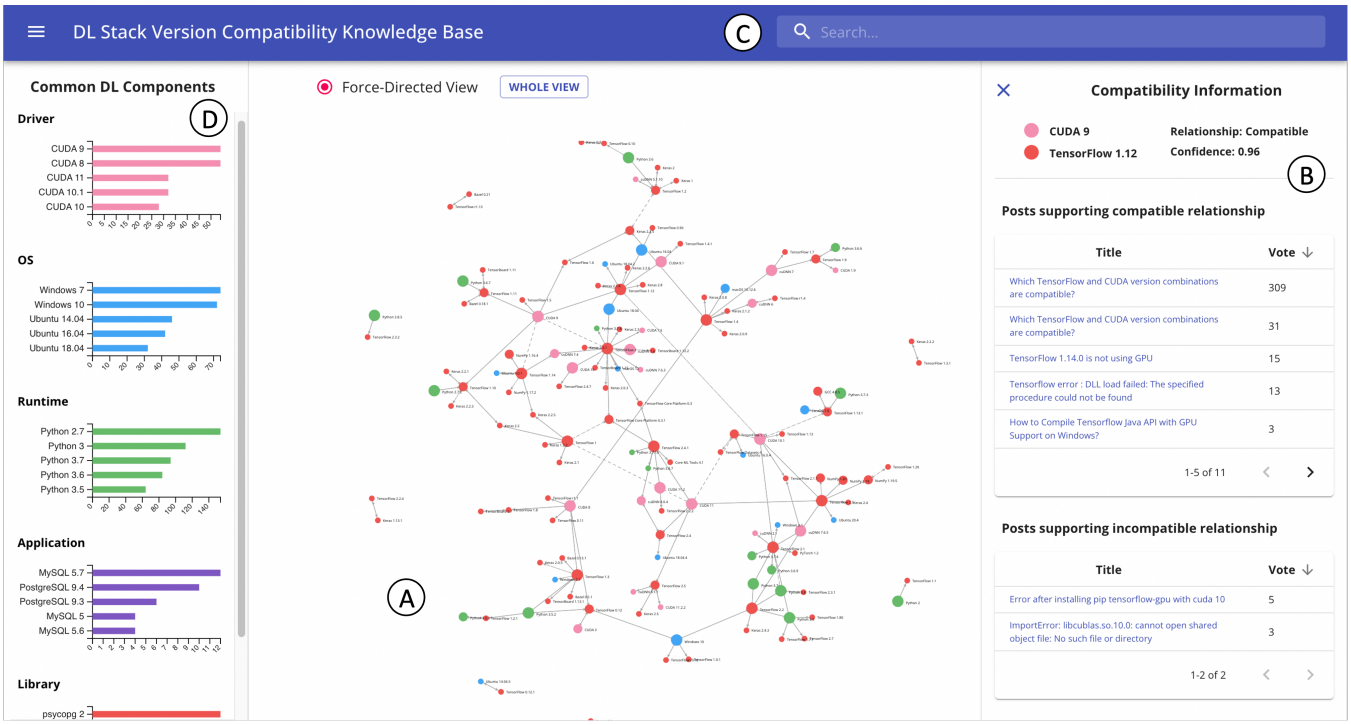

Figure 3: DECIDE, an interactive knowledge-based tool for detecting and identifying version (in)compatibility in deep learning stacks. (A) The Compatibility Visualizer. (B) The Information Panel. (C) The Search Bar. (D) The Statistical Panel.

Table 2: Accuracy of version incompatibility detection

	Precision	Recall	F1 Score
Watchman [10]	16.7%	5.9%	8.7%
PyEGo [13]	33.3%	29.4%	31.2%
DECIDE	91.7%	64.7%	75.9%

4 EVALUATION

Our technical research paper [15] shows DECIDE can be used to detect version issues in real DL projects by creating a benchmark consisting of 10 popular projects from GitHub. We first searched for DL projects on GitHub with at least 100 stars and a *requirements.txt* file. We manually reproduced them on our local machine until we found ten projects with at least one version issue on our local machine. [7] For each experiment, we manually resolved the incompatibility issue. Among the total 17 issues, 4 issues involved components at the same layer, while 13 issues involved components between different DL stack layers.

We compared DECIDE against two state-of-the-art approaches, PyEGo [13] and Watchman [10]. Table 2 shows the precison, recall, and F1 score of DECIDE, Watchman, and PyEGo. Overall, Decide achieves 91.7% precision and 64.7% recall, significantly outperforming Watchman and PyEgo. The comparison method detail can be found in our technical paper [15].

[7]The 10 benchmark project statistics can be found at https://github.com/LexieZhou/Decide/blob/main/stats/RQ1/project_statistics.md

5 DISCUSSION

Threats to validity. The DL component recognition algorithm employed in our research introduces potential identification errors into our dataset. Additionally, the performance of DECIDE is dependent on the accuracy and limitations of UnifiedQA. Another potential threat arises from the relatively small benchmark that we used to evaluate DECIDE. Due to the large volume of SO posts, it was not feasible to manually validate each post for accuracy. Instead, we inspected random samples, which may lead to imprecise estimations.

Limitations. DECIDE only extracted knowledge from a limited scope from SO, which may limit its effectiveness to only those scenarios and incompatibilities discussed on SO without adapting to new updates. Also, the current version of DECIDE focuses only on version issues of 48 Python-based DL components. Future research endeavors can expand the knowledge graph by incorporating knowledge from other online documents or apply it to other tasks by adding more diverse SO posts.

6 CONCLUSION

Deep learning has found applications in diverse domains, but version incompatibility issues often lead to build failures in DL projects. In this paper, we introduced DECIDE, an interactive web-based visualization of a knowledge graph containing 2,376 version knowledge extracted from Stack Overflow discussions. DECIDE empowers users to confidently reuse or deploy deep learning projects on their local machines, minimizing the likelihood of version-related failures.

REFERENCES

[1] 2022. Stack Exchange Data Dump. Accessed on June 09, 2022. https://archive.org/details/stackexchange

[2] Junxiao Han, Shuiguang Deng, David Lo, Chen Zhi, Jianwei Yin, and Xin Xia. 2020. An empirical study of the dependency networks of deep learning libraries. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 868–878. https://doi.org/10.1109/ICSME46990.2020.00116

[3] Eric Horton and Chris Parnin. 2019. DockerizeMe: automatic inference of environment dependencies for python code snippets. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 328–338. https://doi.org/10.1109/ICSE.2019.00047

[4] Kaifeng Huang, Bihuan Chen, Susheng Wu, Junmin Cao, Lei Ma, and Xin Peng. 2022. Demystifying dependency bugs in deep learning stack. *arXiv preprint arXiv:2207.10347* (2022). https://doi.org/10.48550/arXiv.2207.10347

[5] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UnifiedQA: Crossing Format Boundaries With a Single QA System. https://doi.org/10.48550/arXiv.2005.00700 arXiv:2005.00700 [cs.CL]

[6] Suchita Mukherjee, Abigail Almanza, and Cindy Rubio-González. 2021. Fixing dependency errors for Python build reproducibility. In *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis*. 439–451. https://doi.org/10.1145/3460319.3464797

[7] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*. PMLR, 8748–8763. https://doi.org/10.48550/arXiv.2103.00020

[8] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314. https://doi.org/10.1145/3180155.3180220

[9] Jiawei Wang, Li Li, and Andreas Zeller. 2021. Restoring execution environments of jupyter notebooks. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). https://doi.org/10.48550/arXiv.2103.02959

[10] Ying Wang, Ming Wen, Yepang Liu, Yibo Wang, Zhenming Li, Chao Wang, Hai Yu, Shing-Chi Cheung, Chang Xu, and Zhiliang Zhu. 2020. Watchman: Monitoring dependency conflicts for python library ecosystem. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 125–135. https://doi.org/10.1145/3377811.3380426

[11] Wikipedia contributors. 2023. Stable marriage problem. Accessed on January 10, 2024. https://en.wikipedia.org/wiki/Stable_marriage_problem

[12] Qing Wu, Yungang Liu, Qiang Li, Shaoli Jin, and Fengzhong Li. 2017. The application of deep learning in computer vision. In *2017 Chinese Automation Congress (CAC)*. IEEE, 6522–6527. https://doi.org/10.1109/CAC.2017.8243952

[13] Hongjie Ye, Wei Chen, Wensheng Dou, Guoquan Wu, and Jun Wei. 2022. Knowledge-based environment dependency inference for Python programs. In *Proceedings of the 44th International Conference on Software Engineering*. 1245–1256. https://doi.org/10.1145/3510003.3510127

[14] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2018. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine* 13, 3 (2018), 55–75. https://doi.org/10.1109/MCI.2018.2840738

[15] Zhongkai Zhao, Bonan Kou, Mohamed Yilmaz Ibrahim, Muhao Chen, and Tianyi Zhang. 2023. Knowledge-Based Version Incompatibility Detection for Deep Learning. *arXiv preprint arXiv:2308.13276* (2023). https://doi.org/10.1145/3611643.3616364

Received 2024-01-29; accepted 2024-04-15