# ExeKGLib: Knowledge Graphs-Empowered Machine Learning Analytics

Antonis Klironomos[1,2], Baifan Zhou[3], Zhipeng Tan[1,4], Zhuoxun Zheng[1,5], Gad-Elrab Mohamed[1], Heiko Paulheim[2], and Evgeny Kharlamov[1,3]

[1] Bosch Center for Artificial Intelligence, Germany
[2] University of Mannheim, Germany
[3] University of Oslo, Norway
[4] RWTH Aachen, Germany
[5] Oslo Metropolitan University, Norway

**Abstract.** Many machine learning (ML) libraries are accessible online for ML practitioners. Typical ML pipelines are complex and consist of a series of steps, each of them invoking several ML libraries. In this demo paper, we present `ExeKGLib`, a Python library that allows users with coding skills and minimal ML knowledge to build ML pipelines. `ExeKGLib` relies on knowledge graphs to improve the transparency and reusability of the built ML workflows, and to ensure that they are executable. We demonstrate the usage of `ExeKGLib` and compare it with conventional ML code to show `ExeKGLib`'s benefits.

**Keywords:** Machine learning · Knowledge graphs · Python library

## 1 Introduction

Due to the significant advancements in the realm of computer science, particularly in the field of machine learning (ML), there is a plethora of ML algorithms and corresponding libraries publicly accessible [10,2,9,3]. The use of ML is steadily rising in both academic and industrial settings [11]. Experts in various domains are also learning ML for the sake of applying it to solve domain-specific challenges, e.g. biologists [7,5], oncologists [6,1], and engineers in the industry [8,12,4]. The development of functional and useful ML workflows can be complex and time-consuming, which can pose a barrier for non-ML experts. Thus, there is a need for a user-friendly approach that neither requires excessive knowledge nor training in ML. While, existing tools such as Amazon Sage Maker[6] or Google AutoML[7] provide convenient graphical user interfaces (GUI) and application programming interfaces (API), yet do not provide open-source code libraries.

In this paper, we introduce `ExeKGLib`, an easily-extendable Python library that supports a variety of methods for data visualization, data preprocessing and feature engineering, and ML modeling. `ExeKGLib` works in two steps: (1)

[6] https://aws.amazon.com/sagemaker

[7] https://cloud.google.com/automl

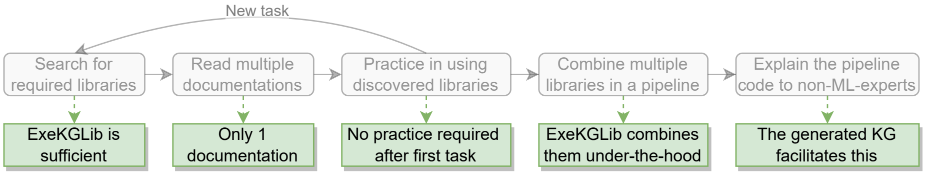
New task
Search for required libraries
Read multiple documentations
Practice in using discovered libraries
Combine multiple libraries in a pipeline
Explain the pipeline code to non-ML-experts
ExeKGLib is sufficient
Only 1 documentation
No practice required after first task
ExeKGLib combines them under-the-hood
The generated KG facilitates this

**Fig. 1.** Improvements on conventional data science workflow

Generate executable ML pipelines using knowledge graphs (KGs), (2) Convert generated pipelines into functional Python scripts, and execute these scripts. We rely on KGs for expressing the created pipelines to make them more understandable and reusable, and to verify that they are executable [13]. `ExeKGLib` can be used by a wide range of users and in a variety of scenarios: from domain experts that want to do ML to teachers and students for teaching and learning ML.

In the following sections, we start with demonstrating `ExeKGLib`'s usage. Then, we describe the used KG schemata and discuss the underlying details of KG construction and pipeline generation in Section 3.

## 2 Usage Demonstration

Our target user can generate an ML pipeline either by importing `ExeKGLib`'s `ExeKG` Python module or by interacting with the provided Typer CLI without writing code [8]. We demonstrate the former usage with three sample Python files [9]. The pipelines represented by the generated sample KGs are briefly explained below:

1. **ML pipeline**: Loads features and labels from an input CSV dataset, splits the data, trains and tests a k-NN model, and visualizes the prediction errors.
2. **Statistics pipeline**: Loads a feature from an input CSV dataset, normalizes it, and plots its values (before and after normalization) using a scatter plot.
3. **Visualization pipeline**: Loads a feature from an input CSV dataset and plots its values using a line plot.

The above pipelines (in form of executable KGs) can be executed using the provided Typer CLI [10]. To exhibit the pipelines' transparency, we have visualized the sample pipelines using Neo4j [11]. The script to perform this visualization for any executable KG is also provided.

Experimentation with the offered resources can verify the benefits of `ExeKGLib` on the traditional data science process (Figure 1). In particular, using our tool to solve a task reduces the overhead prior to the implementation, reduces the effort during the code development, and increases the explainability of the resulting ML pipeline. A brief display of the tool's practical advantages for a generic classification task is illustrated in Table 1. In a conventional setting (table's middle

[8] https://github.com/boschresearch/ExeKGLib#usage
[9] https://github.com/boschresearch/ExeKGLib/tree/main/examples
[10] https://github.com/boschresearch/ExeKGLib#executing-an-ml-pipeline
[11] https://bit.ly/exe-kg-lib-visualizations

**Table 1.** Comparison between conventional code and `ExeKGLib` for a classification task

| **Pipeline steps** | **Conventional code** | **Code using ExeKGLib** |
|---|---|---|
| 1. Load data | `pd.read_csv()` + convert to `numpy` | `ExeKG.create_data_entity()`<br>`ExeKG.create_pipeline_task()` |
| 2. Split data | `sklearn...train_test_split()` | `ExeKG.add_task()` |
| 3. Train | `sklearn...Classifier().fit()` | `ExeKG.add_task()` |
| 4. Evaluate | `sklearn...Classifier().predict()` | `ExeKG.add_task()` |
| 5. Visualize | `matplotlib.pyplot...()` | `ExeKG.add_task()` |

column), the user needs to separately import three different libraries (*i.e.* `pandas`, `scikit-learn`, `matplotlib`) and use five of their modules. On the other hand, when using `ExeKGLib` (table's right column), the user needs a limited number of libraries and modules, and thus learning is easier and faster by skipping reading extensive documentation of various libraries.

## 3 System Design

`ExeKGLib` relies on KG schemata to construct executable KGs (representing an ML pipeline) and execute them. Both of these processes use the `rdflib` Python library combined with SPARQL queries to find and create KG components.

### 3.1 Underlying KG Schemata

`ExeKGLib` utilizes an upper-level KG schema (Data Science – namespace: `ds`) that describes data science concepts such as data entity, task, and method. The supported tasks and methods are separated into bottom-level KG schemata [12]:

- **Visualization** tasks schema, which includes two types of methods: (1) The plot canvas methods that define the plot size and layout. (2) The various kinds of plot methods (e.g. line plot, scatter plot, or bar plot).
- **Statistics and Feature Engineering** tasks schema including methods such as Interquartile Range calculation, mean and standard deviation calculation, etc., which can also form more complex methods like outlier detection and normalization.
- **ML** tasks schema representing ML algorithms like Linear Regression, MLP, and k-NN and helper functions that perform e.g. data splitting and ML model performance calculation.

`ExeKGLib`'s Python implementations of the above methods utilize common libraries such as `matplotlib` and `scikit-learn`.

### 3.2 Executable KG Construction

As shown in Figure 2, the internal process of creating an executable KG starts with extracting the columns from the input dataset (CSV file). `ExeKGLib` populates the KG with data entities representing the target columns. Data entities are then used as input to the ML pipeline tasks.

[12] https://github.com/boschresearch/ExeKGLib#kg-schemata

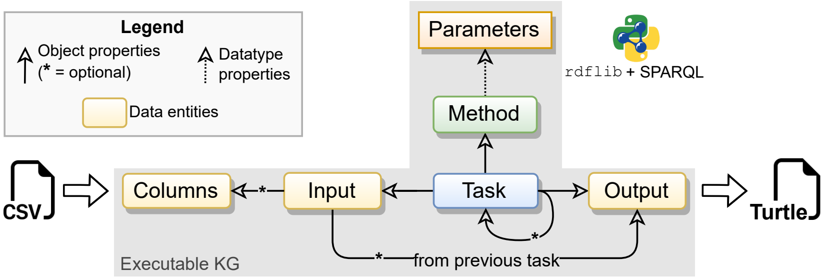

**Fig. 2.** Executable KG construction phase

Afterward, ExeKGLib adds to the KG the entities representing the user-specified task type (e.g. classification) and method type (e.g. k-NN), which are taken from the provided bottom-level KG schemata; and links the current task with the chosen method, input data entities, datatype properties, and the next task. Throughout the process, the compatibility of the aforementioned KG components is ensured by ExeKGLib based on the KG schemata. Finally, the created KG is serialized and saved on the disk in Turtle.

### 3.3 ML Pipeline Execution

To execute a given KG, ExeKGLib parses the KG with the help of the above KG schemata (Section 3.1). After that, the pipeline's *Tasks* (owl:Individuals) are sequentially traversed using the object property ds:hasNextTask. Based on the IRI of the next *Task* (owl:Individual), the *Task*'s type and properties are retrieved and mapped dynamically to a Python object. Such mapping allows for extending the library without modifying the KG execution code. Finally, for each *Task*, the Python implementation of the selected method type is invoked.

## 4 Future Work

We plan to add additional algorithms to ExeKGLib to support a wider variety of ML-related tasks, which can be conveniently done due to its good extendability. In the future, we will build a system by integrating ExeKGLib with a graph-based database. This will allow for easier management of the produced executable KGs, quick visualization, and more convenient reuse.

**Acknowledgements** : The work was partially supported by EU projects Dome 4.0 (GA 953163), OntoCommons (GA 958371), DataCloud (GA 101016835), Graph Massiviser (GA 101093202), and EnRichMyData (GA 101093202).

## References

1. Abreu, P.H., Santos, M.S., Abreu, M.H., Andrade, B., Silva, D.C.: Predicting Breast Cancer Recurrence Using Machine Learning Techniques: A Systematic Review. ACM Computing Surveys **49**(3), 52:1–52:40 (Oct 2016). https://doi.org/10.1145/2988544

2. Bartschat, A., Reischl, M., Mikut, R.: Data Mining Tools. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **9**(4), e1309 (2019). https://doi.org/10.1002/widm.1309
3. Heidrich, B., Bartschat, A., Turowski, M., Neumann, O., Phipps, K., Meisenbacher, S., Schmieder, K., Ludwig, N., Mikut, R., Hagenmeyer, V.: pyWATTS: Python Workflow Automation Tool for Time Series. arXiv preprint arXiv:2106.10157 (2021). https://doi.org/10.48550/arXiv.2106.10157
4. Huang, Z., Fey, M., Liu, C., Beysel, E., Xu, X., Brecher, C.: Hybrid Learning-Based Digital Twin for Manufacturing Process: Modeling Framework and Implementation. Robotics and Computer-Integrated Manufacturing **82**, 102545 (2023). https://doi.org/10.1016/j.rcim.2023.102545
5. Kim, J., Ahn, I.: Infectious Disease Outbreak Prediction Using Media Articles with Machine Learning Models. Scientific Reports **11**(1), 4413 (Feb 2021). https://doi.org/10.1038/s41598-021-83926-2
6. Kourou, K., Exarchos, T.P., Exarchos, K.P., Karamouzis, M.V., Fotiadis, D.I.: Machine Learning Applications in Cancer Prognosis and Prediction. Computational and Structural Biotechnology Journal **13**, 8–17 (Jan 2015). https://doi.org/10.1016/j.csbj.2014.11.005
7. Libbrecht, M.W., Noble, W.S.: Machine Learning Applications in Genetics and Genomics. Nature Reviews Genetics **16**(6), 321–332 (Jun 2015). https://doi.org/10.1038/nrg3920
8. Meng, L., McWilliams, B., Jarosinski, W., Park, H.Y., Jung, Y.G., Lee, J., Zhang, J.: Machine Learning in Additive Manufacturing: A Review. JOM **72**(6), 2363–2377 (Jun 2020). https://doi.org/10.1007/s11837-020-04155-y
9. Mikut, R., Bartschat, A., Doneit, W., Ordiano, J.Á.G., Schott, B., Stegmaier, J., Waczowicz, S., Reischl, M.: The MATLAB Toolbox SciXMiner: User's Manual and Programmer's Guide. arXiv preprint arXiv:1704.03298 (2017). https://doi.org/10.48550/arXiv.1704.03298
10. Obulesu, O., Mahendra, M., ThrilokReddy, M.: Machine Learning Techniques and Tools: A Survey. In: 2018 International Conference on Inventive Research in Computing Applications (ICIRCA). pp. 605–611. IEEE (2018). https://doi.org/10.1109/ICIRCA.2018.8597302
11. Sarker, I.H.: Machine Learning: Algorithms, Real-World Applications and Research Directions. SN Computer Science **2**(3), 160 (Mar 2021). https://doi.org/10.1007/s42979-021-00592-x
12. Zeng, L., Al-Rifai, M., Chelaru, S., Nolting, M., Nejdl, W.: On the Importance of Contextual Information for Building Reliable Automated Driver Identification Systems. In: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC). pp. 1–8. IEEE (2020). https://doi.org/10.1109/ITSC45102.2020.9294439
13. Zheng, Z., Zhou, B., Zhou, D., Zheng, X., Cheng, G., Soylu, A., Kharlamov, E.: Executable Knowledge Graphs for Machine Learning: A Bosch Case of Welding Monitoring. In: The Semantic Web – ISWC 2022, vol. 13489, pp. 791–809. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-19433-7_45