SOFTWARE DESCRIPTOR

# Plot Serializer – A Tool for Creating FAIR Data for Scientific Figures

Michaela Leštáková [1], Ning Xia [1], Julius Florstedt [1]

1. Chair of Fluid Systems, Technische Universität Darmstadt, Darmstadt, Germany.

**Abstract.** This software descriptor introduces Plot Serializer, a Python package for supporting researchers in creating FAIR datasets corresponding to the figures of their manuscript. Fitting into existing workflows, Plot Serializer enables effortless export of data plotted in scientific figures into interoperable datasets with customizable metadata for improved reusability and thus facilitates research data management practices. Besides a clear description of Plot Serializer's scope and functionality, a minimal example of its usage and output is given. Finally, its limitations and future plans are outlined.

**Date Submitted:**
2025-03-31

**Keywords:**
research data management, figure, plot, FAIR data, metadata

**Data availability:**
This article does not use data.

**Software availability:**
Plot Serializer GitLab Repository
Plot Serializer DOI
Plot Serializer Docs

**Corresponding Author:**
Michaela Leštáková
michaela.lestakova@tu-darmstadt.de

## 1 Introduction

Research objects such as data and code are ubiquitous in scientific work. To fight the reproducibility crisis in science, more and more researchers are adopting the practice of sharing research objects associated with publications or even as standalone research output. This practice is sometimes also required by journals, conferences and funding bodies. The research objects are of best use for the scientific community if they are findable, accessible, interoperable and reusable, i.e. FAIR [1], [2]. However, making research objects FAIR is not only challenging but often also time-consuming. Plot Serializer has been developed as a Python package that helps researchers create FAIR datasets corresponding to the figures of their manuscript with little effort. This leads to enabling the reader to understand the interconnections between different research objects, such as which data is depicted in a certain figure in the manuscript and with which code it was created, which is an important part of the "R" in FAIR: reusability.

In scientific articles, data visualizations or figures can be seen as "windows" to the data space behind the article: they are an essential result of scientific work and serve as a link between the text and the data that it is based on. However, probably every researcher knows the struggle of getting their hands on the data depicted in a figure. In most cases, it is still necessary to contact the authors of the paper to obtain the data. Fortunately, it is becoming more common that scientific articles contain a data availability statement with a reference to an openly available dataset [3]. However, even then the data may be poorly documented or not follow the FAIR principles: despite being findable and accessible, they may lack interoperability and reusability. Plot Serializer has been developed as a tool to address these issues, aiming to lower the threshold for creating comprehensible, datasets corresponding to the figures in a scientific publication.

## 2 Scope

Plot Serializer is a Python package that enables effortless export of data plotted in scientific figures into interoperable datasets with customizable metadata for improved reusability. As the name indicates, Plot Serializer utilizes serialization: the process of converting a Python object or data structure into a format that can be easily stored or transmitted [4]. The current version of Plot Serializer provides APIs for figure creation using `matplotlib`, the most popular plotting package among Python users. Other plotting packages such as plotly are currently not supported but the modular architecture of Plot Serializer allows to include them in the future.

Using a `Proxy` class, Plot Serializer wraps the plotting functions of `matplotlib` and captures the data immediately after being passed to the plotting function, hence ensuring consistency between the plotted data and exported data. Important metadata are gathered in the process of plotting. It is possible to differentiate between two kinds of metadata in the context of figures: semantic metadata that carry information about the content and meaning of the data (for example axis labels or plot title) and formatting metadata that describe the plot style (for example axis scaling, line thickness or colors). Plot Serializes prioritizes semantic information to formatting information, as its focus lies on supporting research data management (RDM). Plot Serializer uses its own metadata model that loosely follows the conventions of `matplotlib`. The data models have been implemented using `Pydantic` [5].

Currently, Plot Serializer covers the most widely used types of 2D and 3D figures, namely:

- line plot 2D
- line plot 3D
- scatter 2D
- scatter 3D
- surface 3D
- bar plot
- error bar
- box plot
- pie
- histogram

Each of these figure types has slightly different requirements regarding data formatting and metadata modelling. We are continuously working on expanding the list.

As not all semantic metadata are by default provided through the figure (for example certain parameter values may instead be provided via the figure caption or through a text box), Plot Serializer offers the possibility to add custom metadata in the form of key-value pairs to each element of the plot. This enables customizability to a broad range of use-cases across disciplines.

Once the figure has been finalized, Plot Serializer allows the export to a JSON file which is easily human and machine readable, as well as to Research Object Crate (RO-crate), a newly established format for storing research objects based on JSON-LD [6]. The idea behind it is to improve reusability of research objects by packaging them along with their metadata, which can capture identifiers, provenance, relations and annotations, in a machine readable manner [6].

Plot Serializer also includes tools for deserializing its output, i.e. the JSON files, to recreate the figures. This is where the the formatting metadata play an important role. As the formatting metadata in Plot Serializer contain only a limited selection of all formatting information that a matplotlib figure would provide, the focus lies on comprehensible rather than identical representation of the original figure.

To summarize, serializing figures with Plot Serializer offers researchers a simple but efficient tool for creating FAIR datasets that correspond to the figures in their scientific articles. This may ultimately help readers find the dataset corresponding to a certain figure and vice versa while guaranteeing to include essential semantic and formatting metadata.

## 3 Related Work

Because of the important role data visualization plays in scientific articles, several tools exist for creating figures in most programming languages. In Python, the most well-known and most widely used one is `matplotlib` [7]. Using the `pyplot` module in this package, users can create a broad spectrum of figure types and perform advanced formatting. The Python APIs provided by matplotlib are well documented and easy to use, making them easy to integrate into any workflow. As the name suggests, matplotlib's main focus lies on the visualization of the data, with the final product being the figure. The data depicted in the figure is not comprehensively stored in the corresponding Python object, and matplotlib does not contain any function for serializing the figure objects it creates.

`plotly` [8] is another popular plotting package that provides Python APIs. `plotly` is originally a JavaScript library `plotly.js` with the main purpose of creating interactive plots for websites. `plotly` by default enables to serialize the figure objects into JSON files, similarly to Plot Serializer. However, focusing on visualization rather than RDM, `plotly` prioritizes formatting metadata to semantic metadata.

The most widely used package for serialization of objects in Python is `pickle` [9]. Using `pickle`, however, the object hierarchy is kept upon serialization, which ultimately means its main focus lies on formatting requirements of matplotlib. To find data and add relevant semantic metadata to it would be very challenging for the user. Moreover, the data format pickle uses is Python-specific. While this brings advantages regarding the serialization, it also means reduced interoperability from the perspective of the FAIR criteria.

Recently, some authors have demonstrated RDM workflows that include creating and publishing data for each figure with the aim of improving reusability of their data [10], [11]. In their workflows, a JSON file is created for each figure in the article which contains the data as well as semantic metadata. These files are published in a data repository and linked in the article.

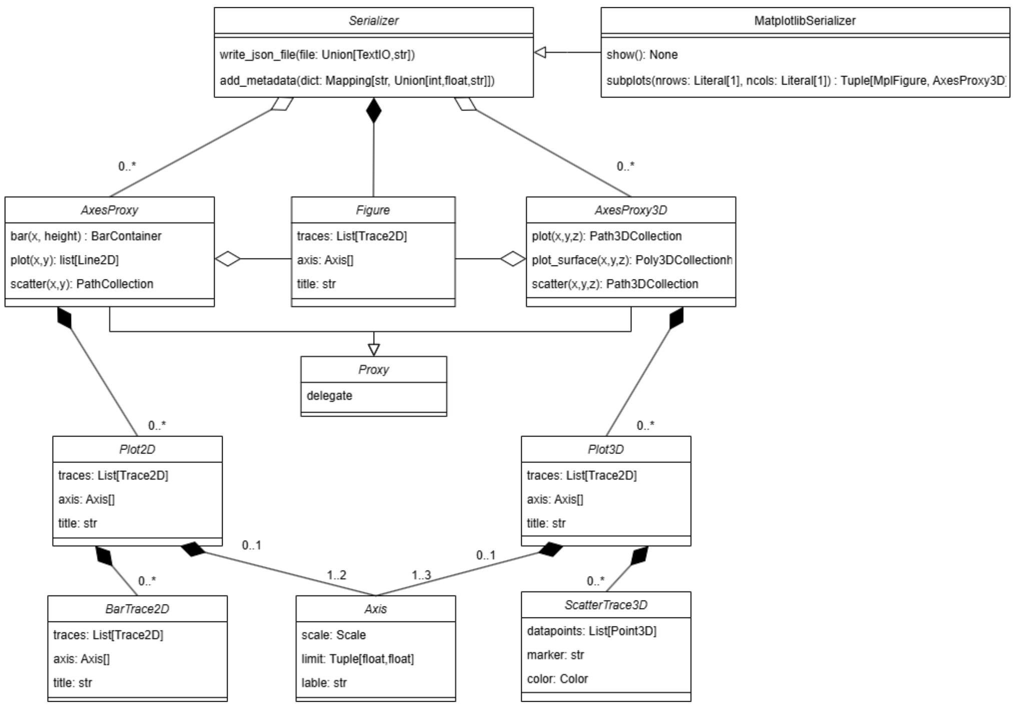

**Figure 1:** Simplified class diagram for two figure types in Plot Serializer: a 2D bar plot and a 3D scatter plot.

## 4 Implementation

Plot Serializer is implemented as a library, mirroring the most common API calls of `matplotlib` while supplementing its functionality with generating the JSON format out of the plotting data. Instead of starting the plotting process via the `matplotlib.pyplot` [7] object, the user instead creates an instance of Plot Serializer's `MatplotlibSerializer` class which acts as the main API for Plot Serializer.

The API of `MatplotlibSerializer` follows the one of matplotlib.pyplot.subplots(). Upon execution, `MatplotlibSerializer.subplots()` creates a Figure object like its matplotlib counterpart but, crucially, its own `AxesProxy` object rather than `matplotlib`'s `Axes` object. The `AxesProxy` class contains functions that enable serialization and can thus be seen as the core of the Plot Serializer architecture.

The aim of `AxesProxy` is to mimic the functionality of matplotlib's Axis class but to enable gathering data along with all necessary metadata handed over by the user during the plotting process. The data is captured in the initial step of the execution of the plotting functions such as `plot()` or `scatter()`. Metadata is gathered all throughout the plotting process: a part of it may come from arguments passed to the plotting functions, such as `marker` or `label` in the minimal example in Section 5, while others are gathered from other functions executed on the object, such as `xlabel` and `ylabel` ibid. Last but not least, using `AxesProxy` allows Plot Serializer to easily differentiate between errors raised in `matplotlib` from its own.

The class hierarchy of Plot Serializer is strongly tailored to the one of `matplotlib` with some changes for better understandability in the scientific community, see Figure 1. It is modelled using

`Pydantic` [5], a state-of-the-art Python package for data validation which supports conversion to JSON. Each scientific figure is thus represented using a `Figure` class. Each `Figure` can contain multiple `Plots`. Depending on their dimensionality, each `Plot` can have two or three `Axes`, corresponding to the coordinate lines of the figure. The `Axes` form the coordinate system of the `Plot`. The `Plot` can contain multiple `Traces`, which are sets of `Datapoints` related in a way that separates them from other datapoints. The minimal example in Section 5 contains two `Traces`: one for children and one for adults. The terminology of the classes and their properties has been selected with a focus on good human readability of the resulting JSON.

Besides writing the figure into a JSON file, Plot Serializer supports adding the JSON to an RO-Crate or create a new one containing the serialized figure [6].

To facilitate better usability of data serialized using Plot Serializer, the package contains a so-called `Deserializer` which enables to convert a JSON file created by Plot Serializer back into the corresponding `Pydantic` class to be ultimately used by `matplotlib` to recreate the original figure. As previously discussed, the focus of Plot Serializer lies on RDM and thus semantic rather than formatting metadata, which means that `Deserializer` will not be able to perfectly reproduce highly individualized figures. However, it should be able to deliver comprehensible representations of the underlying data in most cases.

To assure code quality, Plot Serializer uses both static and dynamic code analysis.
For static code analysis, Plot Serializer relies on the linter Ruff which allows it to improve code-structure, readability and maintainability. Code and functionality independent from the `matplotlib` API are typed and type-checked via `MyPy`.

The dynamic analysis consists primarily of testing. The plotting functions for each of the covered figure types are first tested manually with multiple input sets to ensure that the output matches the expectation. If correct, the resulting JSON files are used as a benchmark in subsequent unit tests and compared after each commit. Additionally, Plot Serializer uses automatic testing (mostly fuzzing), testing a variety of inputs with hypothesis strategies. The testing is performed with `pytest` and achieves a code coverage of 83% , not counting hypothesis testing.

Plot Serializer is well documented. The documentation has been created using Sphinx and is available under https://plot-serializer.readthedocs.io/en/latest/. Each version comes with a thorough general and API documentation.

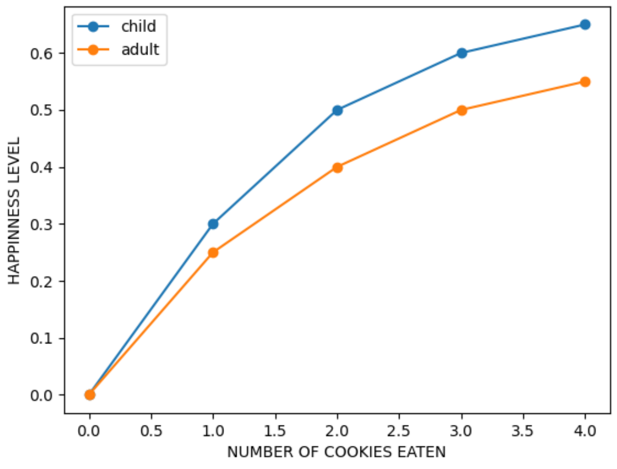

**Figure 2:** Example figure

## 5 Minimal Example

The example figure in Figure 2 was created using the following code:

```
from plot_serializer.matplotlib.serializer import MatplotlibSerializer

serializer = MatplotlibSerializer()
fig, ax = serializer.subplots()

x = [0, 1, 2, 3, 4]
y_child = [0, 0.3, 0.5, 0.6, 0.65]
y_adult = [0, 0.25, 0.4, 0.5, 0.55]

ax.plot(x, y_child, marker="o", label="child")
ax.plot(x, y_adult, marker="o", label="adult")

ax.set_xlabel("NUMBER OF COOKIES EATEN")
ax.set_ylabel("HAPPINNESS LEVEL")
ax.legend()

fig.savefig("cookies.png")
serializer.write_json_file("./test_plot.json")
```

The command `write_json_file` from line 18 of the above code will produce a JSON file `test_plot.json` with the following contents:

```json
{
  "plots": [
    {
      "type": "2d",
      "title": "",
      "x_axis": {
        "label": "NUMBER OF COOKIES EATEN",
        "scale": "linear"
      },
      "y_axis": {
        "label": "HAPPINNESS LEVEL",
        "scale": "linear"
      },
      "traces": [
        {
          "type": "line",
          "linewidth": 1.5,
          "linestyle": "-",
          "marker": "o",
          "label": "child",
          "datapoints": [
            {
              "x": 0,
              "y": 0.0
            },
            {
              "x": 1,
              "y": 0.3
            },
            {
              "x": 2,
              "y": 0.5
            },
            {
              "x": 3,
              "y": 0.6
            },
            {
              "x": 4,
              "y": 0.65
            }
          ]
        },
        {
          "type": "line",
          "linewidth": 1.5,
          "linestyle": "-",
          "marker": "o",
          "label": "adult",
          "datapoints": [
            {
              "x": 0,
              "y": 0.0
            },
            {
              "x": 1,
              "y": 0.25
            },
            {
              "x": 2,
              "y": 0.4
            },
            {
              "x": 3,
              "y": 0.5
            },
            {
              "x": 4,
              "y": 0.55
            }
          ]
        }
      ]
    }
  ]
}
```

The JSON file provides the essential information about the figure and the data shown in it. The user does not have to provide any additional information that goes beyond good scientific data visualization practices, such as providing axis descriptions – all information stems from what has been passed to the `ax` object via the corresponding functions.

The figure is the first and only element of the `"plots"` list. Under the keyword `"traces"`, the two traces, i.e. sets of data points depicted in the diagram can be found. Hence, there are two traces, each consisting of 4 data points, which depict the relationship between `"NUMBER OF COOKIES EATEN"` and `"HAPPINNESS LEVEL"` for children and adults.

Plot Serializer also allows users to add custom metadata to each figure element – the figure itself, the plot (for figure with multiple plots, referred to in `matplotlib` as subplots), the axes, the traces and the individual datapoints:

```python
serializer.add_custom_metadata_figure({"date_created": "10.01.2025", "author": "Michaela Lestakova"})
serializer.add_custom_metadata_plot(
    {"description": "the figure describes the relationship between number of cookies eaten and happinness level"}
)
serializer.add_custom_metadata_axis({"unit": "percent"}, axis="y")
serializer.add_custom_metadata_trace({"definition": "child is a person of age 0-17.99"}, trace_selector=0)
serializer.add_custom_metadata_trace({"definition": "adult is a person of age 18+"}, trace_selector=0)
serializer.add_custom_metadata_datapoints(
    {"information": "you may have something important to say about this point"}, trace_selector=0, point_selector=1
)
```

## 6 Plot Serializer and the FAIR Principles for Research Software

As a Python package, Plot Serializer follows the FAIR principles for research software [1] in the following aspects:

| | |
|---|---|
| **Findable & Accessible** | • Plot Serializer has a DOI and is versioned (F1, A2)<br>• Plot Serializer is listed on PyPI where all relevant metadata can be found (A1, F2) |
| **Interoperable** | • Plot Serializer exports to JSON, a format that performs well in terms of human and machine readability (I1) |
| **Reusable** | • Plot Serializer has a detailed and openly available documentation (R1)<br>• Plot Serializer is published under an open source license – MIT (R1)<br>• A list of dependencies of Plot Serializer is provided. Plot Serializer does not depend on proprietary software (R2)<br>• The software quality of Plot Serializer is guaranteed through rigorous testing and continuous integration (R3) |

**Table 1:** Specification of how Plot Serializer aligns with the FAIR principles for research software. The concrete criteria are named in parentheses in the left column.

## 7 Conclusion and Outlook

This software descriptor introduces Plot Serializer, a Python package for supporting researchers in creating FAIR datasets corresponding to the figures of their manuscript. It enables effortless export of data plotted in scientific figures into interoperable datasets with customizable metadata for improved reusability, facilitating research data management practices. Plot Serializer fits well into established plotting workflows and can be easily adopted by anybody familiar with the popular plotting package `matplotlib`. In this software descriptor, we have briefly introduced the architecture of Plot Serializer as well as the underlying data models and provided a minimal example of its usage. We have also described its scope and limitations and provided information about code quality assurance.

Plot Serializer is under continuous development. In the near future, we aim to extend its scope to more figure types. Moreover, we aim to standardize its metadata model into a metadata schema building upon existing ontologies. The metadata schema will be published to ensure comprehensiveness of the metadata terminology across domains. In long term, Plot Serializer may be expanded to other popular plotting packages in Python.

## 8 Acknowledgements

The authors would like to thank the Federal Government and the Heads of Government of the Länder, as well as the Joint Science Conference (GWK), for their funding and support within the framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) – project number 442146713.

We would like to thank the student group consisting of Jan Groen, Jonas Jahnel, Max Troppmann, Thomas Wu and, of course, the co-author of this paper Julius Florstedt who developed the first version of Plot Serializer as a student project.

The original idea about storing plot data in human and machine readable form, out of which Plot Serializer was born, stems from Kevin T. Logan and Tim M. Buchert. Many thanks for the inspiring discussions.

## 9 Roles and contributions

**Michaela Leštáková:** Conceptualization, Software, Writing – original draft, Supervision

**Ning Xia:** Conceptualization, Software, Writing – original draft, Supervision

**Julius Florstedt:** Conceptualization, Software, Writing – original draft

## References

[1] M. Barker, N. P. Chue Hong, D. S. Katz, *et al.*, “Introducing the FAIR principles for research software,” *Scientific data*, vol. 9, no. 1, p. 622, 2022. DOI: `10.1038/s41597-022-01710-x`.

[2] M. D. Wilkinson, M. Dumontier, I. J. J. Aalbersberg, *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific data*, vol. 3, p. 160 018, 2016. DOI: `10.1038/sdata.2016.18`.

[3] L. Tedersoo, R. Küngas, E. Oras, *et al.*, "Data sharing practices and data availability upon request differ across scientific disciplines," *Scientific data*, vol. 8, no. 1, p. 192, 2021. DOI: `10.1038/s41597-021-00981-0`.

[4] M. Pilgrim, "Serializing python objects," in *Dive Into Python 3*. Berkeley, CA: Apress, 2009, pp. 205–223, ISBN: 978-1-4302-2416-7. DOI: `10.1007/978-1-4302-2416-7_13`. [Online]. Available: `https://doi.org/10.1007/978-1-4302-2416-7_13`.

[5] S. Colvin, E. Jolibois, H. Ramezani, *et al.*, *Pydantic*, version v2.10.6, Jan. 2025. [Online]. Available: `https://github.com/pydantic/pydantic`.

[6] S. Soiland-Reyes, P. Sefton, M. Crosas, *et al.*, "Packaging research artefacts with RO-Crate," *Data Science*, vol. 5, no. 2, pp. 97–138, 2022. DOI: `10.3233/DS-210053`. eprint: `https://doi.org/10.3233/DS-210053`. [Online]. Available: `https://doi.org/10.3233/DS-210053`.

[7] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: `10.1109/MCSE.2007.55`.

[8] N. Kruchten, A. Seier, and C. Parmer, *An interactive, open-source, and browser-based graphing library for Python*, version 5.24.1, Sep. 2024. DOI: `10.5281/zenodo.14503524`. [Online]. Available: `https://github.com/plotly/plotly.py`.

[9] Python documentation, *Pickle — Python object serialization*, 2025. [Online]. Available: `https://docs.python.org/3/library/pickle.html` (visited on 03/17/2025).

[10] K. T. Logan, J. M. Stürmer, T. M. Müller, and P. F. Pelz, *Comparing approaches to distributed control of fluid systems based on multi-agent systems*, 2023. arXiv: `2212.08450 [eess.SY]`. [Online]. Available: `https://arxiv.org/abs/2212.08450`.

[11] T. Müller and P. Pelz, "Algorithmisch gestützte Planung dezentraler Fluidsysteme," Dissertation, Technische Universität Darmstadt and Shaker Verlag, 2022.