ing.grid

RESEARCH ARTICLE

plotID - a toolkit for connecting research data and visualization

Martin Hock [1]
Hannes Mayr [1]
Manuela Richter [1]
Jan Lemmer
Peter F. Pelz [1]

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

Date Published:

Reviewer:

Licenses:

Keywords:
research data management, visualization, figure, plot, mapping, referencing, ID

Data availability:
Data can be found here:
example.py

Software availability:
Software can be found here:
`git.rwth-aachen.de/plotid/plotid_python`

Abstract. While visualizations can carry a vast amount of information compared to text and are often used for validation, references to data and metadata resulting in these visualizations are not common. To provide such references, the software plotID provides two key modules that strive to seamlessly integrate into a generic, Python-based research workflow. The module *tagplot* generates or accepts a unique ID and anchors it (visibly) as a reference to a figure or picture. The module publish exports the figure along with the data, code and parameters used in its creation into folders named by the reference id for later reuse. The tools work to provide aid in research data management with simple base functionality as opposed to encompassing management frameworks. Later features and improvements will expand the scope and applicability to other programming environments.

1 Statement of need

Scientific results are published in the form of hypotheses, axioms and equations as well as text and diagrams. Likewise, research software is being published more and more frequently. The comprehensibility of scientific results is indispensable for scientific discourse and reproducibility. Hypotheses, axioms and equations are usually published in text form and can be referenced accordingly. Software can be made traceable and referencable through the use of version control software. But what about diagrams? A diagram published in a paper is difficult to trace because the (raw) data is usually not available. However, the traceability of diagrams and the data they contain is not only a challenge in publication but also in everyday research. Diagrams are used for visualization and are therefore often produced for interim results. While the researcher continues the research process with investigations, experiments or simulations, volatile but important information like metadata, background information and details of the data processing are lost.

To be able to reconstruct the complete path, a treasure map is needed, starting from a publication, marking important waypoints of the process back to the raw data and metadata. This map needs to be provided along with the product that will be reviewed the most - the created diagram. If diagrams - regardless of whether they are published later or only serve as interim results - are

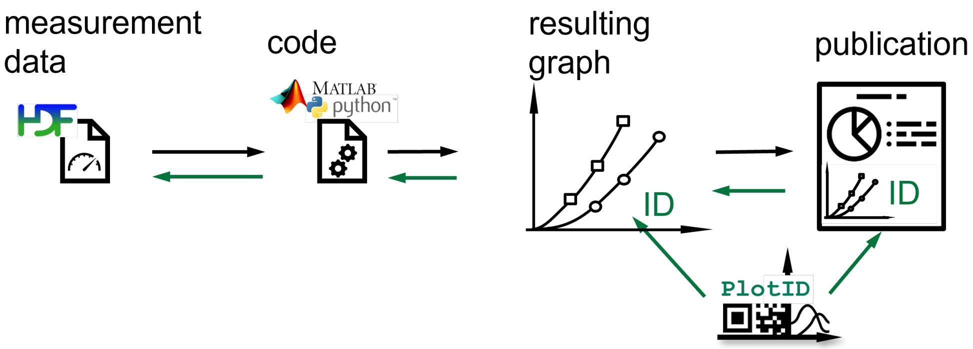


Figure 1: Research workflow from left to right; Afterwards following the chain of references from right to left
'plotID-referencing' by Martin Hock, licensed under CC-BY-SA 4.0

provided with an identifier, that connects to previous steps, traceability can be ensured. Figure 1 shows the order in which important elements are created and how the reference chain tracks back.

A tool designed to meet these needs must meet the following requirements:

- Diagrams must have a unique identifier.
- The identifier must reference the raw data, relevant metadata and the code used to process the data.
- The method must be easy to implement into the existing research workflow.

To reduce the effort of organizing figures along with all necessary data and metadata for later review and reuse, the tool plotID was developed. plotID meets all the above-mentioned requirements and its implementation is described in this paper. The tool is limited to usage in an existing python environment, but investigations on enabling independent installation and execution or offering plotID as a web-based service are ongoing. The software depends on multiple python libraries and is currently limited to visualizations from the [7] and general picture files such as PNG and JPG.

Researchers often tend to keep an Excel table, noting down manually which data file corresponded with which result along with input parameters. Sometimes an ID system is used (counting up or using the date), but interim results like visualizations - used to verify results - are usually not included. The authors could not find any other tool or software that aims at this specific task. Few research data management frameworks strive to organize the full workflow via snapshots by version controlling all software and data. Software execution commands have to be run through the framework so the parameters can be captured and recorded in a snapshot. DataLad[6] works like this. Labelling the figure with a corresponding ID, however, is unique to plotID. The scope of plotID is also a different, limited one and integrating into frameworks with a broad scope like DataLad is entirely possible as a future extension of functionality.

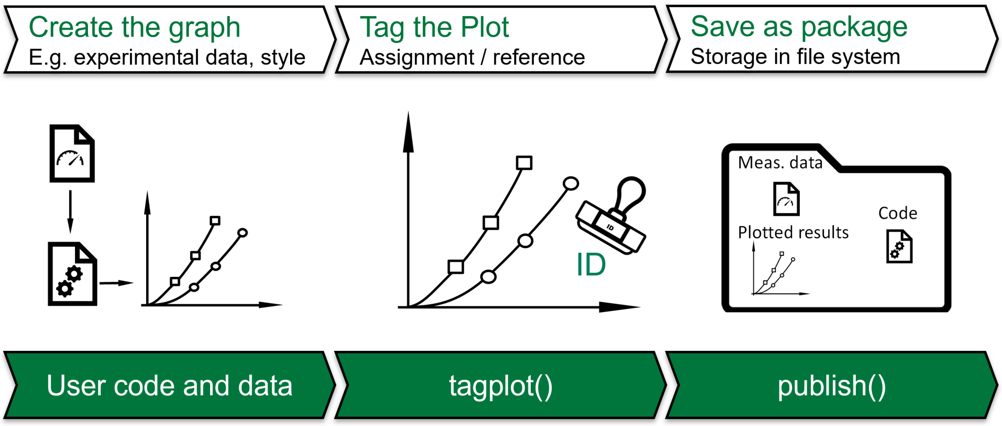


Figure 2: Workflow integrating the plotID core functions
'plotID-workflow' by Martin Hock, licensed under CC-BY-SA 4.0 🅭🅯🄎

2 Methodology

The developed tool plotID is a software solution which covers the above-mentioned needs. The concepts and methods underlying the software are independent of the programming language. The aim of the software is to support the research workflow shown in figure 2 and to enable traceability. plotID aims to help during the early research process to decrease the work of making publications reproducible later on. To ensure ease of use, the tool has been designed to be integrated seamlessly into existing scripts. For this purpose, a GUI has been omitted. Instead, two main functions (building blocks) are provided, which can be inserted into existing user scripts as a one-liner. They are the core of plotID. The first module creates a (unique) ID and stamps this ID onto an object containing a visualization, while the second module helps organize all relevant code, software, and data that went into creating this graphic, into one complete package. Furthermore, connectivity to existing identifiers is ensured. If a specific visualization is later chosen to be included in a publication, the ID can be replaced by a permanent identifier like a DOI and the package of code, software and data can be published at the location referenced by the DOI. The ID on in the published paper will then directly reference the data, software and code used to create it, hence curating reproducibility. In the following, plotID is presented in more detail using the Python implementation.

3 Python - Implementation

The first version of plotID was implemented in Matlab since this is the most widely used programming language in the local working environment and the language the authors had the highest familiarity with. After reaching a usable state, the focus shifted to rewriting the tool in Python, the second most used programming language (locally). The goal was to make plotID accessible to a broader audience. Moreover, in contrast to Matlab, Python better fulfils the requirements for reusable software in the sense of the FAIR principles. In addition to being widely used in the engineering and research community, Python is non-proprietary, open source, easy to install or even shipped along many operating systems. Python also offers a package index (*PyPI*) and installer (*pip*) for easy distribution of software packages.

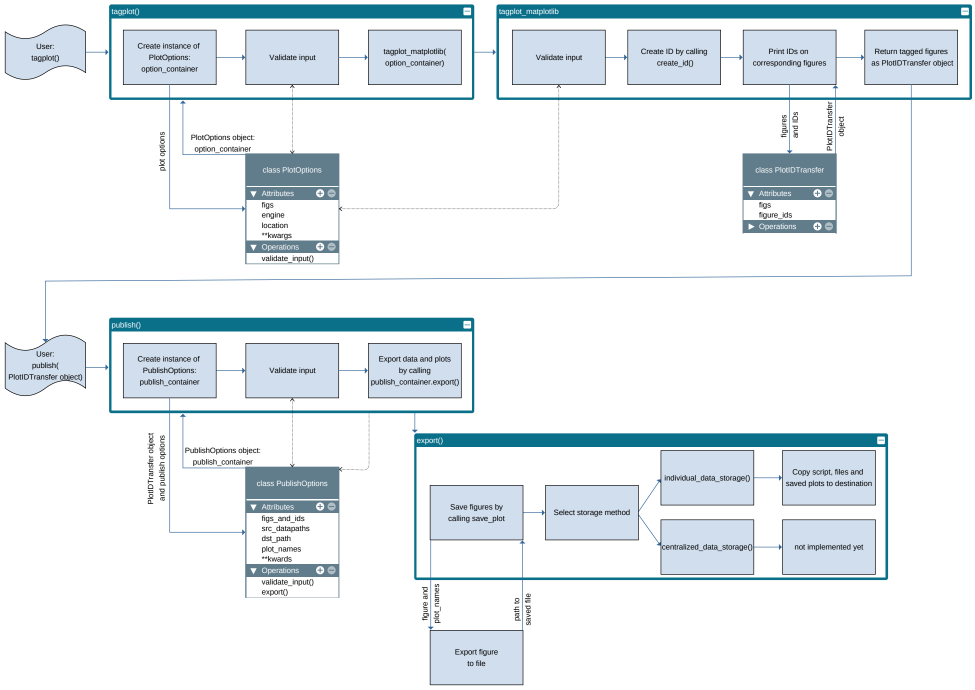

User: tagplot()
tagplot()
Create instance of PlotOptions: option_container
Validate input
tagplot_matplotlib(option_container)
tagplot_matplotlib
Validate input
Create ID by calling create_id()
Print IDs on corresponding figures
Return tagged figures as PlotIDTransfer object
plot options
PlotOptions object: option_container
class PlotOptions
Attributes
figs
engine
location
**kwargs
Operations
validate_input()
figures and IDs
PlotIDTransfer object
class PlotIDTransfer
Attributes
figs
figure_ids
Operations
User: publish(PlotIDTransfer object)
publish()
Create instance of PublishOptions: publish_container
Validate input
Export data and plots by calling publish_container.export()
PlotIDTransfer object and publish options
PublishOptions object: publish_container
class PublishOptions
Attributes
figs_and_ids
src_datapaths
dst_path
plot_names
**kwards
Operations
validate_input()
export()
export()
Save figures by calling save_plot
Select storage method
individual_data_storage()
Copy script, files and saved plots to destination
centralized_data_storage()
not implemented yet
figure and plot_names
path to saved file
Export figure to file

Figure 3: System architecture diagram
'plotID-system-architecture' by Martin Hock, licensed under CC-BY-SA 4.0 🅭🅯🄎

4 Core functions

The core functions of plotID are *tagplot()* and *publish()*. *tagplot()* generates an ID and adds this ID to the figure object. *publish()* saves the figure object and image file, along with the script file, plotID was called from – the essential elements necessary to recreate the visualization from scratch. At this time, plotID does not yet export the python environment or the imported modules. Refer to the subsection 'Script' for more information about the planned implementation. Additional functionality might bring additional steps with the further development of plotID and a widening of its scope.

4.1 tagplot()

The *tagplot()* function creates an ID and tags the figure object with this ID.

4.1.1 ID

tagplot() creates a unique ID (unique in a restricted system), that consists of a static part and a generated part. The static part is handed over as a parameter and is meant to be used to identify a project or organizational unit to which the figure is assigned. The generated part is by default created from the UNIX-Time stamp in hexadecimal form. As an alternative option, a random number generator can be used. The implementation of the ID is modular, enabling easy implementation of individual needs or sources for IDs.

4.1.2 Tagging

In Python, there are multiple available packages that can produce visualizations from data. Adding an ID needs to be implemented for many of these engines separately. For now plotID supports figures created with *matplotlib* and raw image files. The ID is added as an attribute to the object and the graphical, visible item.

4.1.3 Arguments

Necessary input arguments for *tagplot(figs, engine[, prefix, id_method, location])*:

- *figs*: the figure object or a list of objects, that is to be tagged
- *engine*: the plot/image engine to be used (currently only 'matplotlib' and 'image' (for plain image files) are supported)

Optional input arguments are:

- *prefix*: to define a static part of each created ID (prefix='Ing.grid-'). Type: string.
- *id_method*: to define how the unique part of the ID is created ('random', 'time'). Type: string.
- *location*: to define the position the ID is displayed in, relative to the full graphical object (cardinal directions like 'west', custom inputs for rotation and position are to be implemented). Type: string.

Output arguments are the tagged object and ID, if a list of objects was input, then the output is a list as well.

At this point the figure object can still be modified, for example, to adjust colours or positioning or recreate the full plot before exporting a final version.

4.2 publish()

This function starts the export process. The source files of the processed data, the visualization (including the tagged ID), and the script hosting the call to the publish function are copied together into a destination folder.

4.2.1 Script

A function in Python has access to the file path of the script which it was called from. With this, the code for calculations can easily be gathered. For this reason, *publish()* cannot be called from the command line or from within a script that has been started with the 'python -m' flag.

For dependent packages, the 'import' lines of the script can be compared with the output of 'pip freeze' which returns all installed packages including their version. The overlap of these lists can be written into a requirements.txt, which is added to the exported folder. Using the Python package installer *pip* the original package versions can be reinstalled. This has not yet been implemented. Furthermore, the user has to take care of including additional function files as data paths, that have not been imported but are still accessed by the executed script.

4.2.2 Data files

Data files are handed over as a list of file or folder paths. Ideally, the script already manages a list of all files that are read during the execution of the script. It is up to the user to control this. By default, the data files are copied to each exported package. For large data files, the *centralized* flag is intended.

The data files are copied to a central folder, relative to the export packages. For further exports, the data files are compared to the ones already copied and only copied if new data files are present. With this, a publication on a data repository could encompass the data files in addition to multiple ”satellite” folders containing the specific script, parameters and graphics. For HDF5 files, each package can contain an empty HDF5 file that only contains a link to the ”real” central data file. While this has proven to be helpful in the Matlab implementation, the Python version aims to include the ’centralized’ option in a future release.

4.2.3 Arguments

Necessary input arguments for *publish(src_datapath, dst_path, figure, plot_name[, ...])* are:

- *src_datapath*: This can be a single or a list of file or folder paths for source data and additional function files. The type is a string or a list of strings.
- *dst_path*: This is the destination folder path. If it does not exist, the folder will be created. The type is a string.
- *figure*: This is a figure object, the exact class depends on the plot engine used. This object will be turned into an image file.
- *plot_name*: This is the name for the graphics objects. The type is a string or list of strings. If a single name is passed for multiple objects, a raising number will be added.

Optional input arguments:

- *data_storage*: Currently only ’individual’ and ’centralized’ are available. ’Individual’ will store all data in each exported package, while ’centralized’ stores the data files in a central folder separate from the packages containing script and image files.

5 Example script

The following script shows how plotID is used.

```python
# %% Import modules
import numpy as np
import matplotlib.pyplot as plt
from plotid.tagplot import tagplot
from plotid.publish import publish

# %% Set Project ID
PROJECT_ID = "MR04_"

```

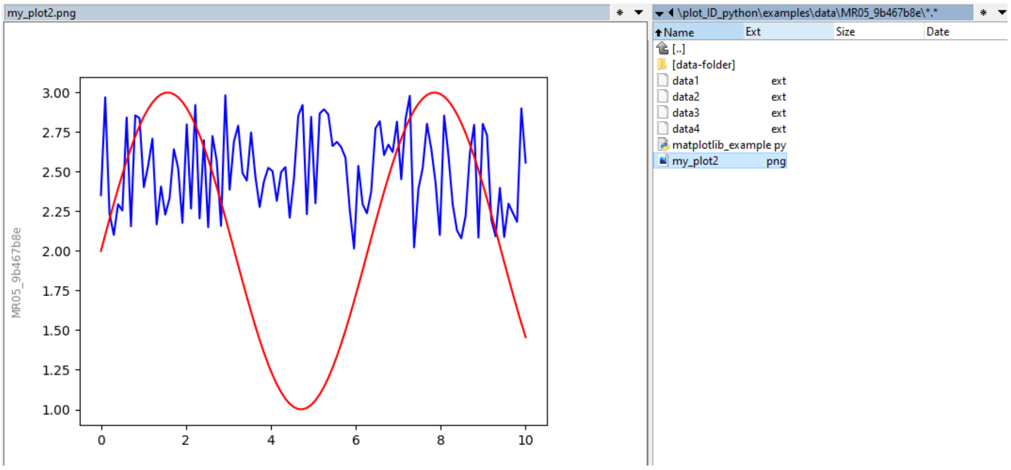
```python
# %% Create sample data
x = np.linspace(0, 10, 100)
y = np.random.rand(100) + 2
y_2 = np.sin(x) + 2

# %% Create sample figures

# 1. figure
IMG1 = 'image1.png'
FIG1 = plt.figure()
plt.plot(x, y, color='black')
plt.plot(x, y_2, color='yellow')
plt.savefig(IMG1)

# 2. figure
IMG2 = 'image2.png'
FIG2 = plt.figure()
plt.plot(x, y, color='blue')
plt.plot(x, y_2, color='red')
plt.savefig(IMG2)
```

In this part, the plotID modules and those necessary to create figures and images are imported. The variable *PROJECT_ID* is set to provide a static part of the ID. Random data is used to create two figures with matplotlib and their image files.

```python
# %% TagPlot

# If multiple figures should be tagged, they must be provided as list.
FIGS_AS_LIST = [FIG1, FIG2]
IMGS_AS_LIST = [IMG1, IMG2]

# Example for how to use tagplot with matplotlib figures
[TAGGED_FIGS, ID] = tagplot(FIGS_AS_LIST, 'matplotlib',
    prefix=PROJECT_ID, id_method='time', location='west')

# Example for how to use tagplot with image files
# [TAGGED_FIGS, ID] = tagplot(IMGS_AS_LIST, 'image', prefix=PROJECT_ID,
#              id_method='random', location='west')
```

Both matplotlib objects are tagged with a generated ID in one line of code. Tagging the image files has been commented out in this case.

```python
# %% Publish
```



Name	Ext	Size	Date
[..]			
[data-folder]			
data1	ext		
data2	ext		
data3	ext		
data4	ext		
matplotlib_example	py		
my_plot2	png		

Figure 4: Example export folder and tagged plot
'plotID-example-export' by Martin Hock, licensed under CC-BY-SA 4.0 🅭🅯🄎

Files (README.md and LICENSE) and a folder from the code repository are used in place of research data files. The folder ending with '-exports' is the destination, and 'testimage' is a freely chosen name for the exported image files.

This also shows that the workflow does not depend on any kind of file format or pre-organized structures. Any kind of data can be used, and even if the library creating the visualization is not (yet) supported, the resulting image file can still be tagged.

Figure 4 shows the resulting export folder with (renamed) data files, the script file and the tagged plot.

6 Distribution

Providing easy ways to acquire and use the software is important for adoption. The code is open source under the Apache-v2.0 license. plotID requires a Python version ≥3.10 and is OS-independent. The current release version is v0.2.1 showing an alpha state.

At this time, the following distribution methods are available and described in the repository's[5] README file.

6.1 Source Code

The plain source code is publicly available on a GitLab repository located under git.rwth-aachen.de/plotID/plotID_python/[5] and can be directly downloaded or cloned with git.

```
git clone https://git.rwth-aachen.de/plotid/plotid_python.git
cd plotid_python
pip install -r requirements.txt
pip install .
```

6.2 Python Package

plotID is listed in the official Python Package Index (PyPI)[10]. The installation is done with the following command:

```
pip install plotid
```

Distributing plotID independently from an existing Python installation is one of the aims of later versions. Possible ways to achieve this are providing compiled executables or a central web-hosted service.

7 Ensuring good software quality

To ensure continuous good software quality, we adhere to best practices and the style guide PEP-8[9]. This includes comments, docstrings and code formatting. To ensure adherence to these guidelines, automated tests on the code are implemented.

7.1 Unit tests

Python offers various libraries for unit testing. plotID is using the *unittest* module[11], which is delivered with Python by default. Tests for each function are defined in the *tests* folder, along with the *runner_test.py* script which organizes the execution of the tests, by discovering the test files based on their location. The *unittest* module also measures the code covered by the tests, and total coverage of less than 95% is considered as failed. The tests are executed by a GitLab CI/CD pipeline[3] with every commit and merge request. Additional Jobs in the pipeline execute Pylint[2] and Flake8[1] to check against coding style, programming errors and cyclomatic complexity. Commits that fail the pipeline tests cannot be merged into the main branch and will not make it into a release version. In the future, additional tests e.g. against security risks introduced by dependencies and more detailed reports are planned.

7.2 Documentation

To ensure easy access and understanding of the code, Python docstrings[8] have been implemented in the source code from the beginning. The docstrings are compiled into HTML using the Sphinx[12] python package and GitLab CI-CD[3] creating an automatically generated API reference. The documents are hosted using GitLab Pages[4]. This documentation[13] will be improved by adding the readme, example code, example use cases and an introductory text until version 1.0.

8 Conclusion

The idea of plotID is simple yet. As with most research data management operations, the benefit for the additional work presents at a later time – although in this case, it presents itself for the creator of data or visualizations and not only for later reuse. The code and open-source implementation are still work-in-progress, but the core functionality is present. There are many ideas to improve and add functionality present already. Bug reports, merge requests with code, ideas for features and all feedback are welcome and best voiced in the GitLab repository.

9 Acknowledgements

We acknowledge the help from Jan Stifter and Benjamin Hermann with testing the software and feedback on the user interface.

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.

10 Roles and contributions

Martin Hock: Conceptualization, Methodology, Coding, Tests, Writing – original draft

Hannes Mayr: Coding, Tests, Methodology

Manuela Richter: Conceptualization, Methodology, Coding

Jan Lemmer: Conceptualization, Methodology

Peter F. Pelz: Project administration, Supervision, Funding Acquisition

References

[1] GitHub. *flake8/index.rst at main · PyCQA/flake8*. 2022. URL: https://github.com/PyCQA/flake8 (visited on 08/29/2022).

[2] GitHub. *PyCQA/pylint: It's not just a linter that annoys you!* 2022. URL: https://github.com/PyCQA/pylint (visited on 08/29/2022).

[3] *GitLab CI/CD | GitLab*. 2022. URL: https://docs.gitlab.com/ee/ci/ (visited on 08/19/2022).

[4] *GitLab Pages | GitLab*. 2022. URL: https://docs.gitlab.com/ee/user/project/pages/ (visited on 08/29/2022).

[5] GitLab RWTH Aachen. *PlotID / plotID_python · GitLab*. 2022. URL: https://git.rwth-aachen.de/plotid/plotid_python (visited on 08/19/2022).

[6] Yaroslav O. Halchenko et al. "DataLad: distributed system for joint management of code, data, and their relationship". In: *Journal of Open Source Software* 6.63 (2021), p. 3262. DOI: 10.21105/joss.03262. URL: https://doi.org/10.21105/joss.03262.

[7] *Matplotlib - Visualizations with python*. 2022. URL: https://matplotlib.org/ (visited on 10/10/2022).

[8] *PEP 257 – Docstring Conventions | peps.python.org*. 2022. URL: https://peps.python.org/pep-0257/ (visited on 08/29/2022).

[9] *PEP 8 – Style Guide for Python Code | peps.python.org*. 2022. URL: https://peps.python.org/pep-0008/ (visited on 10/11/2022).

[10] PyPI. *PyPI · The Python Package Index*. 2022. URL: https://pypi.org/ (visited on 08/19/2022).

[11] *unittest — Unit testing framework — Python 3.10.6 documentation*. 2022. URL: https://docs.python.org/3/library/unittest.html (visited on 08/29/2022).

[12] *Welcome — Sphinx documentation*. 2022. URL: https://www.sphinx-doc.org/en/master/ (visited on 08/29/2022).

[13] *Welcome to PlotID's documentation! — plotID 0.1.2 documentation*. 2022. URL: https://plotid.pages.rwth-aachen.de/plotid_python/ (visited on 08/29/2022).