# MDAKits: Supporting and promoting the development of community packages leveraging the MDAnalysis library

Version 0.1.0b

Irfan Alibay[1], Jonathan Barnoud[2], Oliver Beckstein[3], Richard Gowers[4], Fiona Naughton[5], Lily Wang[4]

[1] *Department of Biochemistry, The University of Oxford, United Kingdom*
[2] *Centro Singular de Investigación en Tecnoloxías Intelixentes, Santiago de Compostela, Spain*
[3] *Department of Physics, Arizona State University, Tempe, AZ, USA*
[4] *Open Molecular Software Foundation, Irvine, CA, USA*
[5] *Cardiovascular Research Institute, University of California, San Francisco, San Francisco, CA, USA*

**Abstract**

The open sharing of code that abides by the basic principles of FAIR (findability, accessibility, interoperability, and reusability) is essential to robust, reproducible, transparent science. However, scientists typically are not supported in making the substantial effort required to make software FAIR-compliant, or incentivised with academic recognition or reward. Here we propose a framework to support a broad ecosystem of MDAnalysis toolkits, or “MDAKits”. We envision that MDAKits will be independent add-on packages building on MDAnalysis that meet a set of software package standards to be listed on a centralized MDAKit registry. We will continually assess packages based on criteria such as the presence of tests and documentation with the aim of encouraging continuous improvement. To lower the barrier of entry for new developers, we will provide tools such as cookiecutter templates and assist with technical support towards fulfilling these criteria. We will also work with journals such as the Journal of Open Source Software to streamline the pathway to publication, creating academic incentive for researchers to publish code. Through the MDAKits framework, we aim to foster the creation of a diverse ecosystem of sustainable community-driven downstream tools.

## TABLE OF CONTENTS

## 1. Introduction

### *1.1 Scientific code frequently fails to meet FAIR tenets, impeding scientific progress*

Software has become increasingly essential to research. In many areas, it underlies fundamental tasks such as generating, processing, analyzing, storing, visualizing, and communicating the key results and insights ultimately published. Despite this, software is typically not central to the publication peer review process in many scientific fields. Consequently, scientific code frequently fails to meet the basic tenets of FAIR: findability, accessibility, interoperability, and reusability [1], [2].

With the publication of "The FAIR Guiding Principles for scientific data management and stewardship" in 2016 and the follow-up FAIR Principles for Research Software in 2022, it has become increasingly acknowledged that abiding by the principles of FAIR is crucial to promoting robust, reproducible, and efficient scientific discovery and innovation [1], [2]. We believe that extending FAIR principles to include open-source release not only significantly advances that goal, but furthermore is necessary for transparent research. Open sharing of code brings a number of substantial benefits to the scientific community. For example, scientists can accurately replicate a given methodology or re-use previous code, reducing duplication of effort and reducing the risk of implementation errors. Indeed, the molecular simulation community in particular has made a concerted effort over recent years to encourage the open sharing of scientific codes [3]. For example, as of July 2022, over 4700 GitHub repositories containing Python code that makes use of MDAnalysis have been made publically available.

However, simply sharing code is not sufficient to fulfill FAIR guidelines. In fact, making software FAIR compliant requires significant investment and often expert knowledge on the part of the developers, especially if the code was written specifically for a particular research project. For example, the Python ecosystem is so dynamic that it is common for research code to rapidly become obsolete or unusable if a new version of a key library is released. To fulfill the Reusability tenet of FAIR alone, code should include documentation, version control, and dependency management. Ideally, it would also include unit tests, examples, and packaging. Even when code is released in reference to a publication, it often falls short of ideal FAIR standards. A short survey of publications in Scopus [4] and the Journal of Open Source Software [5] over 2017 – 2021 identified that out of a total 720 papers citing MDAnalysis, only 43 linked to code available on a version control platform such as GitHub, GitLab, or Bitbucket. Of these, only 18 met the requirements of best practices: they implemented unit tests, comprehensive documentation, and some means of installation.

Two major factors contribute to the lack of open-source FAIR compliant code. Firstly, code is typically written by scientists with no formal training or support in programming, for whom implementing FAIR principles can pose an intimidating and tedious barrier. Secondly, despite the substantial investment of effort and time required to implement best practices, publishing FAIR software is not typically appreciated with academic recognition or reward. Fostering a culture of open-source FAIR software requires addressing both.

## *1.2 Centralized open-source packages such as MDAnalysis offer a limited solution*

One solution is to consolidate scientific code around a small number of large, central packages. MDAnalysis is a widely-used open-source Python library for molecular simulation data. With over 16 years of development by more than 160 developers, MDAnalysis has refined its code base to offer a mature, robust, flexible API that offers a range of high-performance tools to extract, manipulate, and analyze data from the majority of common simulation formats. MDAnalysis tools have been used for a variety of scientific applications ranging from exploring protein-ligand interactions [6]–[8], to understanding lipid behavior [9], [10], to assessing the behavior of novel materials [11], [12].

Until recently, MDAnalysis has encouraged users to contribute their code back into the library to make it available to others. Notable examples of this include the waterdynamics and ENCORE [13] analysis modules. This approach, also successfully taken by packages such as cpptraj [14] and the gromacs tools [15], has a number of key advantages:

- MDAnalysis can ensure that the code follows best practices (including documentation and tests)
- Code is promoted and made freely accessible to all MDAnalysis users
- The burden of maintenance, support, and potential updates becomes primarily the responsibility of MDAnalysis, rather than the contributing developer

However, the many costs of this approach can, under some conditions, result in unsustainable, untenable disadvantages:

- Ensuring that the code follows best practices often requires long review periods and strict code-style adherence

- The necessity of keeping the API stable between major releases precludes quick releases of breaking changes. In general, MDAnalysis has a slow release cycle, so new features and bug fixes can take months to become available in new releases
- As MDAnalysis implicitly agrees to maintain any code that has been added, a certain level of understanding and expertise is required from the maintainers. If the core developer team lacks expertise in a specific discipline or subdiscipline, adding new code in these areas introduces a significant maintenance burden should the original code contributors not be available to help with maintenance. Consequently, it is impractical to include recently released or cutting-edge techniques in the core library
- Introducing new package dependencies incurs heavier costs for many users who may not require this additional code
- Code contributors lose complete ownership of their code

The many disadvantages listed above can severely limit the usefulness of centralizing code around one monolithic package. Indeed, encountering these issues when attempting to expand the core MDAnalysis library attests that this approach is not the most suited to the MDAnalysis community.

### *1.3 An ecosystem of downstream packages may yield more sustainable progress*

We believe that a sustainable alternative solution is for communities such as MDAnalysis to encourage and foster researchers in their efforts towards developing individual software. We propose to overcome the difficulties of implementing FAIR best practices through the provision of structured technical assistance. Specifically, we envision that suitable tooling and documentation could be provided to ease the development of new packages, as well as a platform (which we refer to as a “registry”) where packages that meet certain standards can be advertised to the community. This idea is not novel; it is reminiscent of other successful ecosystems such as PLUMED’s PLUMED-NEST [16] or AiiDA’s plugin registry [17], which list available tools that are known to work to their respective user communities.

With the help of tooling such as cookiecutter templates and example repositories, we can model best practices, promote the use of helpful tools e.g. for checking code coverage, and reduce the work required to set up processes such as continuous integration, versioned documentation, packaging and deployment. Developers can also reach out to the MDAnalysis community for feedback, technical assistance, or even make connections with new co-developers and potential users. Decoupled from MDAnalysis's release cycle, developers would be able to introduce new changes as required, keeping complete control over their code-base. Joining an MDAnalysis registry would allow for frequent and streamlined communication between MDAnalysis and downstream developers, allowing developers to be efficiently forewarned about potential breaking changes.

Although establishing such an ecosystem of MDAnalysis-supported packages would likely require substantial investment from MDAnalysis developers, this approach is nonetheless likely to be far more sustainable than centralizing around a super-package. Offering technical assistance to individual developers in implementing best practices will constitute a large part of the effort; however, we believe that this would remain lower than the effort associated with adding additional functionality to the core MDAnalysis library. Furthermore, once the ecosystem has been established, we hope that a growing portion of the community will participate in taking care of the registry and developers; and that the culture of following best practices and publishing code will gain momentum in itself.

In part, we hope that this momentum will be driven by users and user expectations. Users of the MDAnalysis ecosystem would gain huge benefit from the provision of a package registry. They would be able to see new software as it gets added, rather than having to comb through literature or rely on developers advertising the code themselves. They would also be able to easily verify the current development status of a package – e.g. the registry could contain information about the health of a given codebase, such as whether it contains unit tests, sufficient documentation, and which versions of MDAnalysis it is compatible with. Packages on the registry would also come with easy-to-find instructions on how to easily install and run a given package, significantly lowering the technical barrier to use and experimentation. As the maintenance remains the burden of the package owners, unfortunately the risk remains that packages on the registry may eventually become out-of-date, which is indeed one of the major disadvantages of this approach. However, the registry significantly increases the likelihood that packages will reach users who will become sufficiently motivated to contribute or take over their maintenance and development.

In the rest of this document we detail how MDAnalysis proposes to implement such an ecosystem of toolkits, which we will henceforth call "MDAKits" (MDAnalysis Toolkits). We detail our expectations for MDAKits in terms of best practices and how their registration and continuous validation is anticipated to work.

## 2. The MDAKit framework

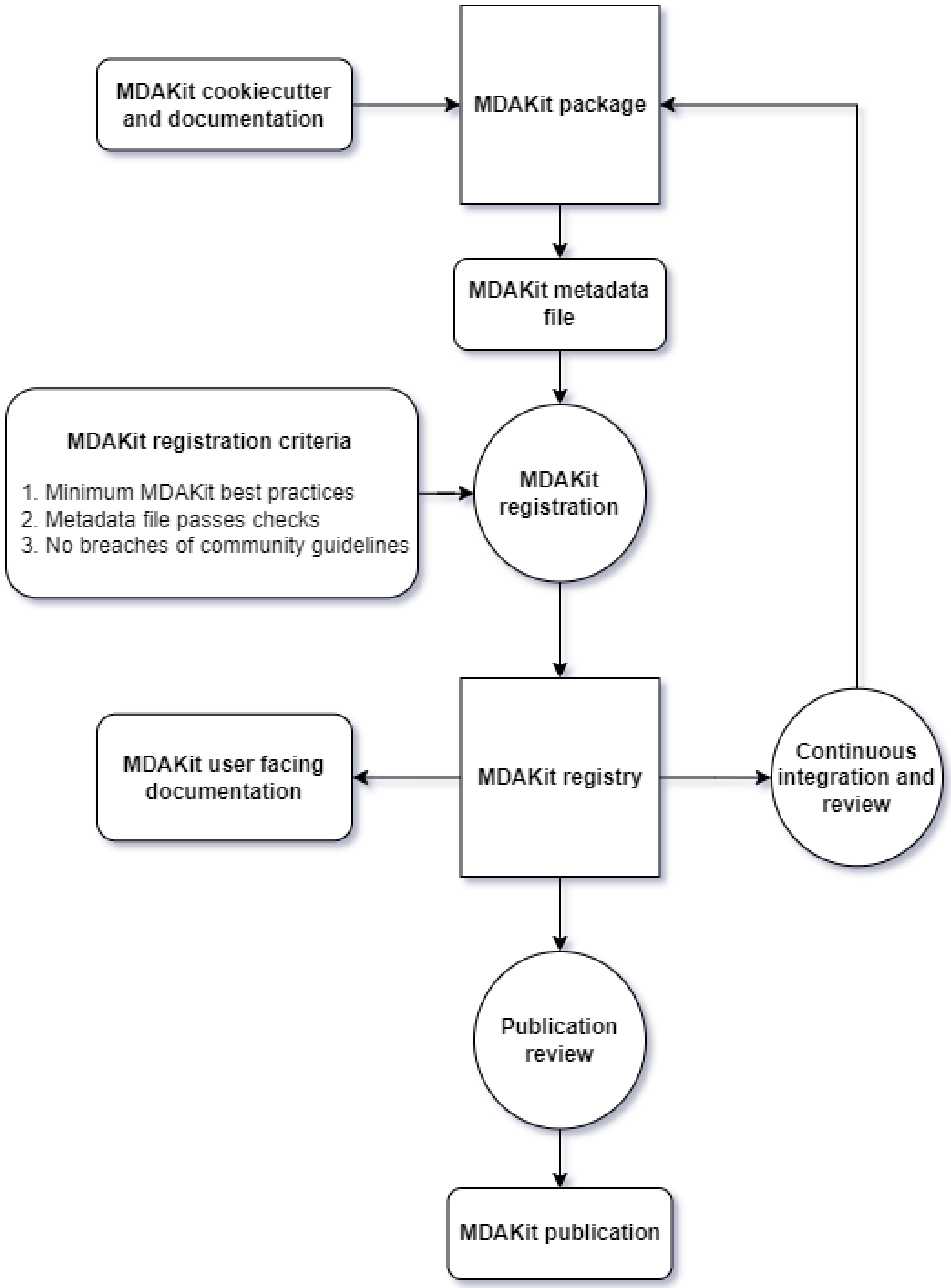

MDAKit cookiecutter and documentation
MDAKit package
MDAKit metadata file
MDAKit registration criteria
1. Minimum MDAKit best practices
2. Metadata file passes checks
3. No breaches of community guidelines
MDAKit registration
MDAKit user facing documentation
MDAKit registry
Continuous integration and review
Publication review
MDAKit publication

**Figure 1.** Workflow diagram of the MDAKit framework. Starting from the creation of an MDAKit package, with the help of documentation and the MDAKit cookiecutter, the

package then goes through the process of being added to the MDAKit registry, undergoing continuous validation and review and eventually reaching the stage of publication.

The MDAKit framework (**Fig. 1**) is designed to be a complete workflow to help and incentivise developers to go from the initial stages of package development all the way through to the long term maintenance of a mature codebase, while adhering to best practices. As such, the main goals of this framework are:

i. To help as many packages as possible implement best practices and develop user communities.

ii. To ensure that members of the MDAnalysis community can easily identify new packages of interest and know to what extent they are suitable for production use.

iii. To improve contacts between MDAnalysis core library developers and those developing packages using MDAnalysis.

iv. To encourage participation from the community at all steps of the process.

We wish to state two main points that the framework is *not* designed for early on:

i. The MDAKit framework is not intended to restrict the packages which can participate. It is our view that all packages at any stage of their development are of value to the community. As such, we aim for framework components to be as non-blocking as possible.

ii. It is not the intention of any parts of this framework to take ownership of the packages which participate within it. The original code developers retain full ownership and responsibility for their packages and may optionally participate in any part of this framework.

iii. We also do not want to block future contributions to the core library. If new code in MDAKits prove particularly popular, and the MDAKit developers are amenable to contributing these back into the core library, the MDAnalysis team will work with them to integrate additional functionality into MDAnalysis itself

## *2.1 Overview of the framework*

The MDAKit framework, **Fig. 1**, is a multi-step process. In the first step of the MDAKit framework, developers create an initial package which is intended to achieve a set purpose of their choice. To help with this process, MDAnalysis provides a cookiecutter template specifically for MDAKits [18], alongside documentation on best practices and how to optimally use the MDAnalysis API. An overview of what we consider to be best practices for the contents of MDAKit packages is included in Section 3. We note that at this point MDAKits are not expected to fully adhere to best practices, but should at least meet the minimum requirements defined in Section 3 before moving to the next step along this process.

Once a package is suitably developed, code owners are encouraged to add the details of their code to the "MDAKit registry" which will advertise their package to the MDAnalysis community and offer continual validation and review tools to help with package maintenance. Section 4 contains more information about the MDAKit registry, including the registration process (Section 4.2). Briefly, the registration process involves submitting a metadata file to the registry that contains essential information about the MDAKit, such as where the source code is provided, who the code authors are, and how to install the MDAKit. The contents of this metadata file will be reviewed both by automatic code checks and the MDAnalysis developer team before being added to the registry. We want to highlight that this process does not include checks on scientific validity or code health. In fact, none of the processes in this framework account for the scientific validity of the MDAKits. While members of the community are free to offer help, scientific or technical validity is beyond the scope of what is feasible with the MDAnalysis registry.

Upon registration, the MDAKit will be automatically advertised to the MDAnalysis community (see Section 4.3). In the first instance this will amount to a set of auto-generated pages which will expose the details in the metadata file provided in the registration step. Additional tags and badges will also be included which reflect the current status and health of the package. Examples include:

- whether or not it is compatible with the latest versions of MDAnalysis
- what percentage of the codebase is covered by unit tests
- what type or extent of documentation is provided
- what Python versions are currently supported.

This status information will be provided as part of checks done during the continual validation and review steps (see Sections 4.4 and 4.5) of the framework. These steps will involve a mix of regularly scheduled automatic (e.g. linters and unit test execution) checks and more infrequent manual (e.g. code reviews) processes. It is our intention that code health analysis will help developers maintain and improve their codes, as well as suitably warn potential users about issues they may encounter when using a given codebase.

Where possible, the framework will encourage a code review process to be carried out by members of the MDAnalysis community. The aim here is to work with developers in identifying potential areas of improvements for both MDAKits and the core MDAnalysis library (see Sections 4.5 and 4.6). We aim to tie this process closely to the review processes of journals such as the Journal of Open Source Software, which would help lower the barrier towards and encourage an eventual publication (Section 4.7).

## 3. Defining MDAKits: best practice package features

Here we list requirements that we believe MDAKits should strive to fulfill in order to meet best practices in Python package usability and maintenance. To help with implementing these, a cookiecutter is provided which offers a template for potential MDAKits to follow [18]. We want to emphasize again that the aim of the MDAKit project is to encourage best practices whilst also minimizing barriers to sharing code where possible. Therefore, only a minimal set of requirements listed here as “required”l are necessary for MDAKits to be included in the MDAKit registry. Similarly, we do not mean to enforce the label of MDAKit on any package; the process is fully optional and the code owners reserve the right to associate themselves with it.

### *3.1 Code using MDAnalysis (required)*

This is the base requirement of all MDAKits. The intent of the MDAKit framework is to support packages existing downstream from the MDAnalysis core library. MDAKits should therefore contain code using MDAnalysis components which are intended by the package authors to address the MDAKit’s given purpose.

### *3.2 Open source code under an OSI approved license (required)*

The core aim of MDAKits is to encourage the open sharing of codes to potential users within the MDAnalysis community and beyond. To achieve this, we require that codes under this framework be released as open source. Here we define open source as being under an Open Source Initiative (OSI) approved license.

As of writing, the MDAnalysis library is currently licensed under GPLv2+ [19]. Due to limitations with this license type, we cannot currently recommend other licenses than GPLv2+ for codes importing MDAnalysis. However, we hope to relicense to a less restrictive license. In this event, MDAKits will be able to adopt a wider range of OSI approved licenses.

### *3.3 Versioning and provision under an accessible version-controlled repository (required)*

The ability to clearly identify changes in a codebase is crucial to enabling reproducible science. By referencing a specific release version, it is possible to trace back any bug fixes or major changes which could lead to a difference in results obtained with a later version of the same codebase. Whilst we encourage the use of semver [20], any PEP440 [21] compliant versioning specification, would be suitable for MDAKits.

Beyond versioning releases, it is also crucial to be able to develop code in a sustainable and collaborative manner. The most popular way of achieving this is through the use of version control through Git [22]. We require all MDAKits to be held in a publicly facing version controlled repository such as GitHub [23], GitLab [24], or Bitbucket [25].

### *3.4 Designated code authors and maintainers (required)*

In order for users to be able to contact the code owners and maintainers, all MDAKits should clearly list their authors and a means of contacting the persons responsible for maintaining the codebase. To incentivise and recognise contributors throughout the life of a project, we recommend the use of a version controlled "authors" file which lists the authors to a codebase over time.

### *3.5 Documentation (required)*

Describing what a given code does and how to use it is a key component of open sharing. Ideally a package would include a complete description of the entire codebase, including both API documentation and some kind of user guide with worked examples on how the code could be used in certain scenarios. Whilst this is recommended as best practices for an MDAKit, we recognise that this is not  always feasible, especially in the early stages of development. Therefore, the minimum requirement for MDAKits is to have a readme file which details the key aspects of the MDAKit, such as what it is intended to do, how to install it, and a basic usage example.

For best practices, we strongly recommend using docstrings (see PEP 257 [26]) to document code components and using a tool such as ReadTheDocs [27] to build, version and host documentation in a user-friendly manner. We also recommend using duecredit [28] to provide the correct attributions to a given method if it has been published previously.

## *3.6 Tests and continuous integration (required)*

Testing a critical component to ensure that code behaves as intended. Not only does it prevent erroneous coding, but it also assures users that the code they rely on is working as intended. We require at least a single regression test for major functionality to qualify for the registry (i.e. if a toolkit implements a new analysis method, at least one test that checks to see if the analysis code yields the expected value on provided data; regression tests can often double as example documentation).

Ideally one should do full unit testing of the contents of a code, ensuring that not only a specific outcome is reached, but also that each smaller component works. As part of best practices, we highly recommend implementing tests using a framework such as pytest [29] for executing tests and codecov [30] to capture which lines are covered by the tests. We strongly encourage that a minimum of at least 80% of the code lines be covered by tests.

To ensure that tests are run regularly, best practices would be to implement a continuous integration pipeline that tests every time new code is introduced. We encourage the use of free pipelines such as GitHub Actions [31] to achieve this.

## *3.7 Packaging*

Providing a standard means of installing code as a package is important to ensuring that the others can correctly link to (i.e. *import* in the case of Python) and use its contents. Whilst it can be easy to expect users to simply read a Python script, look at its required dependencies, and install them manually, this can quickly become unreasonable should the code grow beyond a single file. Additionally, the lack of clearly defined versions, including the intended Python versions, can lead to inoperable code.

As best practices we heavily encourage the use of setuptools [32] or an alternative such as poetry [33] for package installation. We also encourage that packages be available on common package repositories such as PyPi [34] and conda-forge [35]. The use of such repositories and their respective package managers can significantly lower the barrier to installing a package, enabling new users to rapidly get started using it.

### *3.8 Bug reporting, user discussions, and community guidelines*

To help maintain and grow the project, it is important to specify where users can raise any issues they might have about the project or simply ask questions about its operation. To achieve this, we recommend at the very least adding documentation that points users to an issue tracker.

Key to successfully building a user community is ensuring that there are proper guidelines in place for how users will interact with a project. As best practices we recommend making a code of conduct available that defines how users should interact with developers and each other within a project. It is also advised to provide information on how users can contribute to the project as part of its documentation.

## 4. The MDAKit registry

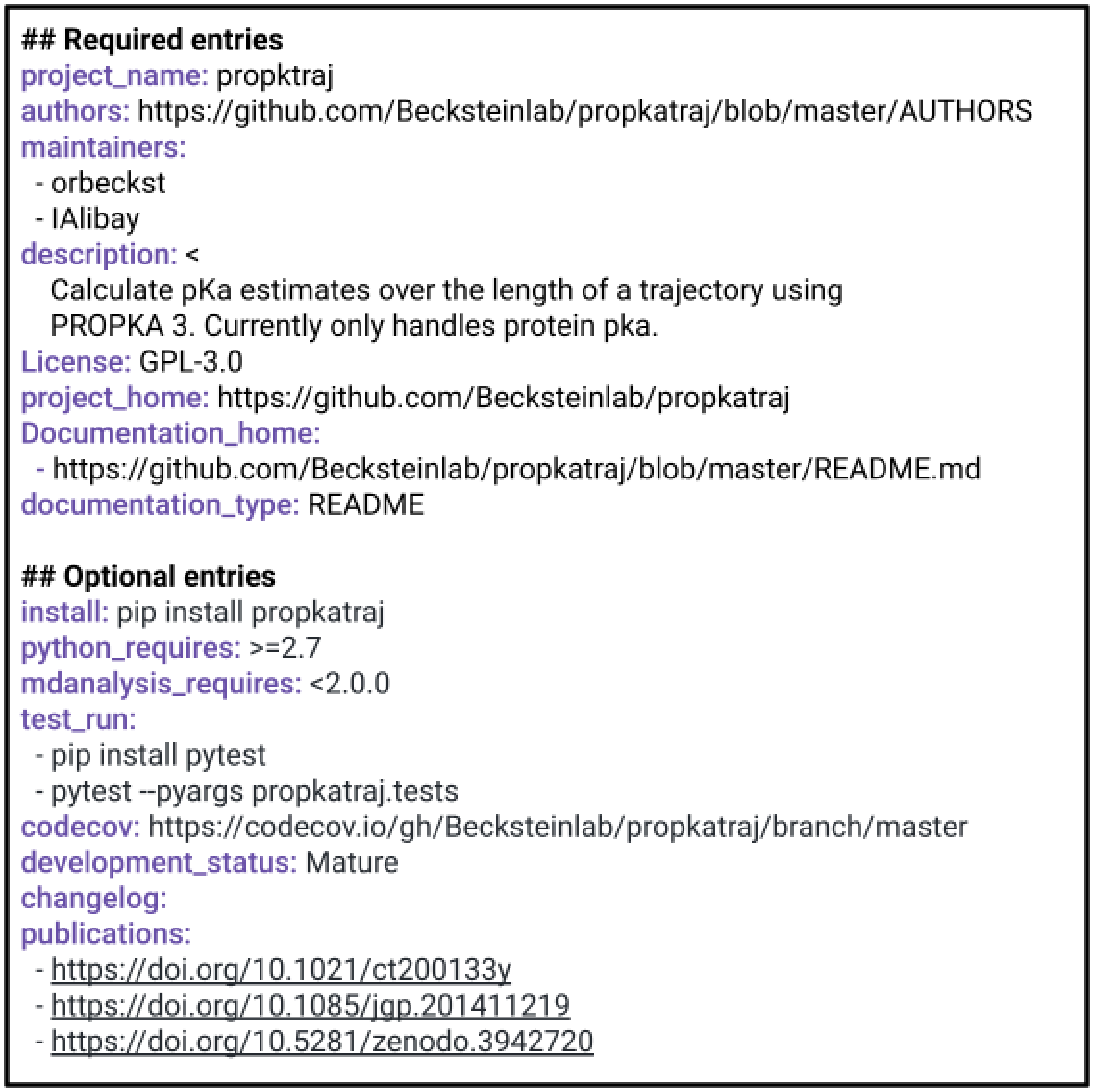
As defined in Section 2, once MDAKits are created, we encourage that they be added to the MDAKit registry. The registry not only provides a platform to advertise MDAKits to the MDAnalysis user community, but also offers tools and workflows to help packages improve and continue to be maintained. Here we describe the various processes which will occur within the registry. We note that we expect the exact details of how these processes will be implemented to evolve over time based on feedback from MDAKit developers and other members of the MDAnalysis community.

### *4.1 MDAKit registry contents*

The main aim of the registry is to hold information about MDAKits. The contents of the registry will therefore center around a list of packages and the metadata associated with each MDAKit. This metadata will take the form of two files: one containing user-provided information on the package contents (see section 4.2), and the other a set of mostly auto-generated details indicating the code health of the package (see section 4.3).

This metadata will be used for two purposes: continuous integration testing and documentation. Continuous testing, helper methods and workflows will be used to regularly install MDAKits and run their test suite (if available) to check if they still work as intended. Should the tests fail, package maintainers will be automatically contacted and failure information will be recorded in the code health metadata to inform users. For the registry documentation, the metadata will be used to provide user-facing information about the various MDAKits in the registry, their contents, how to install them, and their current status as highlighted by continuous integration tests. The registry will also include further information and user guides on the MDAKit framework, helping developers implement the contents of this whitepaper.

*4.2 Registering MDAKits*



```
## Required entries
project_name: propktraj
authors: https://github.com/Becksteinlab/propkatraj/blob/master/AUTHORS
maintainers:
  - orbeckst
  - IAlibay
description: <
   Calculate pKa estimates over the length of a trajectory using
   PROPKA 3. Currently only handles protein pka.
License: GPL-3.0
project_home: https://github.com/Becksteinlab/propkatraj
Documentation_home:
  - https://github.com/Becksteinlab/propkatraj/blob/master/README.md
documentation_type: README

## Optional entries
install: pip install propkatraj
python_requires: >=2.7
mdanalysis_requires: <2.0.0
test_run:
  - pip install pytest
  - pytest --pyargs propkatraj.tests
codecov: https://codecov.io/gh/Becksteinlab/propkatraj/branch/master
development_status: Mature
changelog:
publications:
  - https://doi.org/10.1021/ct200133y
  - https://doi.org/10.1085/jgp.201411219
  - https://doi.org/10.5281/zenodo.3942720
```

**Figure 2.** YAML metadata file for an MDAKit entry of the propkatraj package.

A key feature of the MDAKit framework is the process of adding MDAKits to the registry. As previously defined, our intent is to offer a low barrier to entry and have packages be registered early in their development cycles. This allows developers to benefit from the MDAKit registry validation and review processes early on, hopefully lowering the barrier to further improvements and encouraging early user interactions and feedback.

From an MDAKit developer standpoint, the registration process involves opening a pull request against the MDAKit registry adding a new YAML file (Fig. 2) with metadata about the project. The metadata, as detailed in Figure 2, contains information such as the MDAKit description, source code location, install instructions, how to run tests, and where to find usage documentation. Complete details about the metadata file specification will be provided in the MDAKit registry documentation.

After a pull request is opened, the MDAnalysis developers will review the contents of the submission based on the following criteria:

1. If the required features for MDAKits are met (Section 3), that is:
    a. Does the MDAKit contain code using MDAnalysis?
    b. Is the MDAKit license appropriate?
    c. Is the MDAKit code offered through a suitable version-controlled platform?
    d. Are the MDAKit authors and maintainers clearly designated in the metadata file?
    e. Is there at least minimal documentation in place detailing the MDAKit and its functionality?
    f. Are there at least minimal regression tests available within the MDAKit code?

2. If the metadata file passes linting and integration checks

3. That there are no potential breaches of community guidelines

Once the criteria is passed the metadata will be merged and the MDAKit will be considered registered. Updates to the MDAKit metadata can be carried out at any time after registration by opening pull requests to change the metadata file contents.

## *4.3 Advertising MDAKits*

Registered MDAKits will be automatically added to the registry's public facing documentation. This involves an indexable list of entries for all registered MDAKits. Each entry will display available information from the provided metadata, e.g. what the MDAKit does, any relevant keywords, how to obtain the source code, how to install the package, and where to find relevant documentation. Alongside this information will also be a set of badges which describe the current health of the codebase, allowing users to rapidly identify which packages are currently active, and their level of code maturity. This will include information such as; which MDAnalysis library versions the package is compatible with, how much test coverage does the package have, what type of

MDAnalysis API extensions are provided (e.g. using base classes such as; AnalysisBase or ReaderBase), and whether integration tests are current failing.

Information about MDAKits will be continually updated, either through automatic checks or manual additions provided by package owners updating the metadata files. As we aim for the MDAKit registry to be immutable (aside from special cases covered by Section 4.8), should an MDAKit stop being maintained, it will not be removed from the index but instead labeled as abandoned.

## *4.4 Continual validation*

The MDAKit registry will implement workflows to validate the code health of registered packages. This will mostly center around a test matrix that will regularly run to check if the latest MDAKit release can be installed and if unit tests pass with both the latest release of MDAnalysis and the development version. Should tests fail regularly, an issue will be automatically raised on the MDAKit registry issue tracker contacting the package maintainers and letting them know of the failure. The auto-generated code health metadata for the MDAKit will also be updated to reflect whether or not the tests are currently failing or passing.

In the future we will hope to expand these tests to include more historical releases of the MDAKits and the MDAnalysis library, checks for different architectures (non-x86), and operating systems. We may also expand the checks to consider the cross-compatibility of MDAKits with each other, offering insights on which packages can be safely used together.

## *4.5 Continual review*

To help package growth and improvements, it is our goal for the registry to become a platform that allows members of the MDAnalysis community to offer feedback on MDAKits over the lifetime of their inclusion on the registry. Unfortunately, as MDAnalysis developers can only devote limited time towards the registry, offering regularly scheduled comprehensive reviews of packages is too large an undertaking to be practical.

Instead, we aim to use a system of badges and achievements to push packages towards gradual improvements. For example, we may offer an achievement that encourages MDAKits to use high performance PBC-aware distance routines defined in `MDAnalysis.lib.distances` instead of relying on NumPy's `linalg` method to find the distance between two points. Once MDAKit owners believe that they have suitably updated their code to match this, they can open a pull request highlighting these changes and have developers review these smaller, more focused updates.

MDAKit users will also be encouraged to provide feedback, request improvements, and report bug fixes. However, this should happen outside the scope of the registry; instead, we will ask for users to use the MDAKit's own issue tracker for these.

### *4.6 Feeding back into the MDAnalysis library*

The existence of the MDAKits framework does not preclude the addition of new codes and methods to the core MDAnalysis library. The MDAKit registry, and especially the ongoing review process, will provide a platform for MDAnalysis and MDAKit developers to interact and work together to identify common goals and areas of improvements for both upstream and downstream packages. In particular, MDAnalysis developers will work with MDAKit developers to see if any popular MDAKit methods, components or other means to improve core method performance and lower the barrier to downstream package development can and should be implemented back into the core MDAnalysis library.

### *4.7 Towards publication*

We have laid out a number of best practices here that we encourage MDAKits to fulfill. These essentially amount to the majority of the contribution criteria for submissions to software-focused journals such as the Journal Open Source Software [36]. In order to incentivise developers, we will heavily encourage MDAKits to consider submission to a journal such as JOSS once they meet the required levels of best practices. To aid in this process, the MDAnalysis developers will in the first instance work with journal editors at JOSS to create a streamlined process to submit MDAKits as JOSS entries. The details of this process are still under development.

### *4.8 Raising issues, concerns, and paths to registry removal*

If community members (users, developers or otherwise) have concerns about an MDAKit, we primarily encourage them to raise issues on the MDAKit's issue tracker.

However, in situations where the MDAKit maintainers cannot respond, or if the concern relates to code of conduct breaches, MDAnalysis developers may step in. If an MDAKit has systemic issues with its correctness, the MDAKit may be given special annotations warning users about the issues before using the code. We generally view the MDAKit registry as a permanent record, and will avoid removing packages after registration even if they become fully obsolete. However, we reserve the right to remove packages at our discretion in specific cases, notably code of conduct breaches and violation of the GitHub terms of service [37].

### *4.9 Long term registry maintenance and support*

As with most MDAnalysis projects, long-term support for the MDAKit framework and especially the registry is expected to be carried out by contributors from the MDAnalysis community. Members of the MDAnalysis core development team will lead the maintenance of the registry and also be responsible for passing judgment on serious events such as code of conduct breaches. In the long term, we hope that any gains in popularity of the MDAKits framework will be accompanied by an increase in community involvement in reviews and other maintenance tasks.

## 5. Conclusion

In this document we have outlined MDAnalysis' plans to implement a framework, termed MDAKits, to enable and incentivize the creation of FAIR compliant packages which use and extend MDAnalysis' core functionality. Having identified a large sustainability gap in the sharing of codes using MDAnalysis, we explore the different possibilities which could be taken to resolve this issue. We conclude that attempting to take a single package approach, by adding more methods to MDAnalysis, has shown to be too unsustainable in the past. Instead we propose the MDAKits framework as a means of enabling developers to create new packages and fostering them through the process of achieving best practices in FAIR compliance.

In section 3 we detail the idea of an MDAKit and the various requirements which we think should be reached in order to meet best practices. These include; using a suitable open-source license, the use of version control, the implementation of comprehensive documentation, implementing unit tests, and using best practices for Python packaging. In section 4 we then detail the basis of the MDAKit registry, a public facing repository which exposes MDAKits to the MDAnalysis community, offering regular checks and reviews in order to help improve and maintain the MDAKits it holds. We describe the complete set of processes involved in the registry, ranging from the initial registration of MDAKits all the way through to eventual publication in software-focused journals such as JOSS.

With the MDAKits idea still being in its infancy, we are yet to fully evaluate the way in which the MDAnalysis community will engage with the process. Nevertheless, it is our firm belief that this process will have a positive impact on the quality of packages created and used by scientists within the community. Please note that we anticipate that there will be further revisions of this document as the MDAKit framework continues to be implemented.

## 6. Acknowledgements

We acknowledge the 163 developers and countless community members which have contributed to the MDAnalysis project over the last 16 years and NumFOCUS for its support as our fiscal sponsor.

This work was possible thanks to a grant by the Chan Zuckerberg Initiative, supporting MDAnalysis and the MDAKit project under an EOSS4 award.

## 7. References

[1] N. P. C. Hong *et al.*, “FAIR Principles for Research Software (FAIR4RS Principles),” Mar. 2022, doi: 10.15497/RDA00065.

[2] M. D. Wilkinson *et al.*, “The FAIR Guiding Principles for scientific data management and stewardship,” *Sci. Data*, vol. 3, no. 1, Art. no. 1, Mar. 2016, doi: 10.1038/sdata.2016.18.

[3] W. P. Walters, “Code Sharing in the Open Science Era,” *J. Chem. Inf. Model.*, vol. 60, no. 10, pp. 4417–4420, Oct. 2020, doi: 10.1021/acs.jcim.0c01000.

[4] “Scopus.” https://www.scopus.com/

[5] “Journal of Open Source Software.” https://joss.theoj.org

[6] I. Alibay, “IAlibay/MDRestraintsGenerator: MDRestraintsGenerator 0.1.0.” Zenodo, Mar. 01, 2021. doi: 10.5281/zenodo.4570556.

[7] D. B. Kokh, B. Doser, S. Richter, F. Ormersbach, X. Cheng, and R. C. Wade, “A workflow for exploring ligand dissociation from a macromolecule: Efficient random acceleration molecular dynamics simulation and interaction fingerprint analysis of ligand trajectories,” *J. Chem. Phys.*, vol. 153, no. 12, p. 125102, Sep. 2020, doi: 10.1063/5.0019088.

[8] C. Bouysset and S. Fiorucci, “ProLIF: a library to encode molecular interactions as fingerprints,” *J. Cheminformatics*, vol. 13, no. 1, p. 72, Sep. 2021, doi: 10.1186/s13321-021-00548-6.

[9] K. A. Wilson, L. Wang, Y. C. Lin, and M. L. O’Mara, “Investigating the lipid fingerprint of SLC6 neurotransmitter transporters: a comparison of dDAT, hDAT, hSERT, and GlyT2,” *BBA Adv.*, vol. 1, p. 100010, Jan. 2021, doi: 10.1016/j.bbadva.2021.100010.

[10] “LiPyphilic: A Python Toolkit for the Analysis of Lipid Membrane Simulations | Journal of Chemical Theory and Computation.” https://pubs.acs.org/doi/10.1021/acs.jctc.1c00447

[11] R. Gowers, M. Matta, and H. Nguyen, “kugupu/kugupu: v0.1.2.” Zenodo, Feb. 17, 2021. doi: 10.5281/zenodo.4545322.

[12] A. Schlaich and P. Loche, “MAICoS / MAICoS · GitLab,” *GitLab*. https://gitlab.com/maicos-devel/maicos

[13] M. Tiberti, E. Papaleo, T. Bengtsen, W. Boomsma, and K. Lindorff-Larsen, “ENCORE: Software for Quantitative Ensemble Comparison,” *PLOS Comput. Biol.*, vol. 11, no. 10, p. e1004415, Oct. 2015, doi: 10.1371/journal.pcbi.1004415.

[14] D. R. Roe and T. E. Cheatham, “PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data,” *J. Chem. Theory Comput.*, vol. 9, no. 7, pp. 3084–3095, Jul. 2013, doi: 10.1021/ct400341p.

[15] M. J. Abraham *et al.*, “GROMACS: High performance molecular simulations through

multi-level parallelism from laptops to supercomputers," *SoftwareX*, vol. 1–2, pp. 19–25, Sep. 2015, doi: 10.1016/j.softx.2015.06.001.

[16] M. Bonomi *et al.*, "Promoting transparency and reproducibility in enhanced molecular simulations," *Nat. Methods*, vol. 16, no. 8, Art. no. 8, Aug. 2019, doi: 10.1038/s41592-019-0506-8.

[17] "AiiDA plugin registry." https://aiidateam.github.io/aiida-registry/

[18] "Cookiecutter for MDAnalysis-based packages." MDAnalysis, Jul. 29, 2022. [Online]. Available: https://github.com/MDAnalysis/cookiecutter-mdakit

[19] "GNU General Public License v2.0 - GNU Project - Free Software Foundation." https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html

[20] T. Preston-Werner, "Semantic Versioning 2.0.0," *Semantic Versioning*. https://semver.org/

[21] "PEP 440 – Version Identification and Dependency Specification | peps.python.org." https://peps.python.org/pep-0440/

[22] "Git." https://git-scm.com/

[23] "GitHub," *GitHub*. https://github.com

[24] "GitLab." https://about.gitlab.com/

[25] Atlassian, "Bitbucket | Git solution for teams using Jira," *Bitbucket*. https://bitbucket.org/product

[26] "PEP 257 – Docstring Conventions | peps.python.org." https://peps.python.org/pep-0257/

[27] "Read the Docs." https://readthedocs.org/

[28] Y. O. Halchenko *et al.*, "duecredit/duecredit: 0.9.1." Zenodo, Apr. 13, 2021. doi: 10.5281/zenodo.4685131.

[29] H. Krekel, B. Oliveira, R. Pfannschmidt, F. Bruynooghe, B. Laugher, and F. Bruhin, "pytest." 2004. [Online]. Available: https://github.com/pytest-dev/pytest

[30] "Codecov," *Codecov*. https://about.codecov.io/

[31] "GitHub Actions," *GitHub*. https://github.com/features/actions

[32] P. P. Authority, "setuptools: Easily download, build, install, upgrade, and uninstall Python packages." [Online]. Available: https://github.com/pypa/setuptools

[33] "Poetry - Python dependency management and packaging made easy." https://python-poetry.org/

[34] "PyPI · The Python Package Index," *PyPI*. https://pypi.org/

[35] “conda-forge | community driven packaging for conda.” https://conda-forge.org/

[36] “JOSS - Submission requirements.” https://joss.readthedocs.io/en/latest/submitting.html

[37] “GitHub Terms of Service,” *GitHub Docs*. https://ghdocs-prod.azurewebsites.net/en/site-policy/github-terms/github-terms-of-service