Scikit-fingerprints: easy and efficient computation of molecular fingerprints in Python

Jakub Adamczyk[1,*], Piotr Ludynia[2]

AGH University of Krakow, Department of Computer Science, Cracow, Poland

Abstract

In this work, we present *scikit-fingerprints*, a Python package for computation of molecular fingerprints for applications in chemoinformatics. Our library offers an industry-standard scikit-learn interface, allowing intuitive usage and easy integration with machine learning pipelines. It is also highly optimized, featuring parallel computation that enables efficient processing of large molecular datasets. Currently, *scikit-fingerprints* stands as the most feature-rich library in the open source Python ecosystem, offering over 30 molecular fingerprints. Our library simplifies chemoinformatics tasks based on molecular fingerprints, including molecular property prediction and virtual screening. It is also flexible, highly efficient, and fully open source.

Keywords: molecular fingerprints, chemoinformatics, molecular property prediction, Python, machine learning, scikit-learn
2000 MSC: 92-04, 92-08, 92E10, 68N01

Metadata

1. Motivation and significance

Molecules are the basic structures processed in computational chemistry. They are most commonly represented as molecular graphs, which need to be converted into multidimensional vectors for the majority of processing algorithms, most prominently for machine learning (ML) applications. This is typically done with molecular fingerprints, which are feature extraction algorithms that encode structural information about molecules as vectors [1].

---

*Corresponding author

Email address: `jadamczy@agh.edu.pl` (Jakub Adamczyk)

[1]ORCID 0000-0003-4336-4288
[2]ORCID 0009-0004-0749-9569

Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.6.1
C2	Permanent link to code/repository used for this code version	`https://github.com/scikit-fingerprints/scikit-fingerprints/tree/SoftwareX_submission_v1.6.1`
C3	Permanent link to Reproducible Capsule	N/A
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python 3.9 or newer, RDKit
C7	Compilation requirements, operating environments & dependencies	Linux, Windows, MacOS
C8	If available Link to developer documentation/manual	`https://scikit-fingerprints.github.io/scikit-fingerprints/`
C9	Support email for questions	jadamczy@agh.edu.pl

Table 1: Code metadata

They are widely used in chemoinformatics, e.g. for chemical space diversity measurement [2, 3, 4] and visualization [5, 6], clustering [7, 8, 9, 10], virtual screening [11, 12], molecular property prediction [13, 14, 15], and many more [16, 17, 18, 19, 20, 21]. These chemoinformatics tasks, which often rely on machine learning methods, are important for many real-life applications, particularly drug design. For properly assessing the performance of predictive models, train-test splitting is crucial, and molecular fingerprints can also be used there [22, 23, 24, 25, 26]. The performance of fingerprint-based models remains very competitive, even compared to state-of-the-art graph neural networks (GNNs) [14]. Hybrid molecular property prediction models are also a subject of recent research, combining molecular fingerprints with GNNs [27, 28, 29, 30], transformers [31, 32], or autoencoders [33].

The selection of the optimal fingerprint representation for a given application is nontrivial. It typically requires the computation of many different fingerprints [14], and may also require tuning their hyperparameters [34, 35]. Using multiple fingerprints at once often improves results, e.g. via concatenation [15] or data fusion [36, 37]. Processing large molecular datasets necessitates efficient implementations that leverage modern multicore CPUs. Python, the most popular language in chemoinformatics today, includes the scikit-learn library [38], which has become the de facto standard tool for tabular machine learning tasks, and deep learning frameworks like PyTorch [39]. Scikit-learn
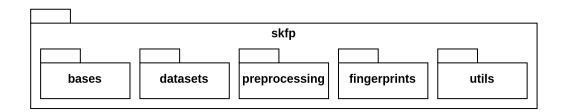
Figure 1: Package diagram of *scikit-fingerprints*.

in particular is renowned for its intuitive and widely adopted API [40].

Popular open source tools for computing molecular fingerprints, such as Chemistry Development Kit (CDK) [41], Open Babel [42], and RDKit [43], are written in Java or C++. None of them are compatible with the scikit-learn API, and their Python wrappers can be cumbersome to work with. They also offer no or very limited support for parallel computation.

Here, we present *scikit-fingerprints*, a new Python library for easy and efficient computation of molecular fingerprints. It is fully scikit-learn compatible, enabling easy integration into ML pipelines as a feature extractor for molecular data. It offers optimized parallel computation of fingerprints, enabling the processing of large datasets and experiments with multiple algorithms. We implemented over 30 different fingerprints, making it the most feature-rich library in the open source Python ecosystem for molecular fingerprinting. Those include those based only on molecular graph topology (2D), as well as those utilizing graph conformational structure (3D, spatial). It is fully open source, publicly available on PyPI [44] and on GitHub at https://github.com/scikit-fingerprints/scikit-fingerprints.

2. Software description

2.1. Software architecture

scikit-fingerprints is a Python package for computing molecular fingerprints, designed for chemoinformatics and ML workflows. Its interface is fully compatible with scikit-learn API [40], ensured by proper inheritance from scikit-learn base classes and comprehensive tests.

The package structure is shown in Figure 1. All functionality is contained in the `skfp` package, allowing easy imports. The base classes are in `skfp.bases` package, and they can be used to extend the functionality with new or customized fingerprints. `skfp.datasets` has functions to load popular datasets for easy benchmarking. `skfp.preprocessing` contains classes for preprocessing molecules before computing fingerprints, as described in Section 2.2.1.

Fingerprints are represented as classes in package `skfp.fingerprints`. Lastly, `skfp.utils` contains additional utilities, such as input type validators.

2.2. Software functionalities

User-facing functionalities can be divided into preprocessing and fingerprint calculation. It also supports loading popular datasets. In addition, in contrast to existing software, we support efficient parallelism and implement multiple measures for ensuring high code quality and security.

2.2.1. Preprocessing

Fingerprints take RDKit `Mol` objects as input to the `.transform()` method. However, for convenience, all 2D-based fingerprints also take the SMILES input, converting them internally. If done multiple times, this entails a small performance penalty, so *scikit-fingerprints* offers `MolFromSmiles` and `MolToSmiles` classes for easier conversions.

SMILES representation for a molecule is not unique, and there are various non-standard extensions to this format [45, 46, 47]. In particular, incorrect or very unlikely molecules can be written in SMILES form. For example, string "H=H" is a syntactically correct SMILES, but is not a chemically valid molecule. `MolFromSmiles` by design performs only basic sanitization checks, to enable reading arbitrary data. For expanded checks, we implement the `MolStandardizer` class. Since there is no one-size-fits-all solution for molecular standardization, we use the most widely used standardization steps, recommended by RDKit [48]. This helps ensure high data quality at the beginning of the pipeline.

All fingerprints utilizing conformational (3D, spatial) information require `Mol` input, with conformers calculated using RDKit, with `conf_id` property set. Conformer generation can be troublesome, with multiple different algorithms and settings available. `ConformerGenerator` class in *scikit-fingerprints* greatly simplifies this process, offering reasonable defaults. It attempts to maximize efficiency for easy molecules and minimize the chance of failure for complex compounds, based on the ETKDGv3 algorithm [49], known to give excellent results [50].

2.2.2. Fingerprints calculation

Different molecular fingerprints are represented as classes, all inheriting from `BaseFingerprintTransformer`, and further from `BaseSubstructureFingerprint` for substructure fingerprints such as Klekota-Roth [51] (see Figure 2). They are used as stateless transformers in scikit-learn and used mainly via the `.transform()` method. It takes a list of SMILES strings or RDKit `Mol` objects, and outputs a dense NumPy array [52] or a sparse SciPy array in
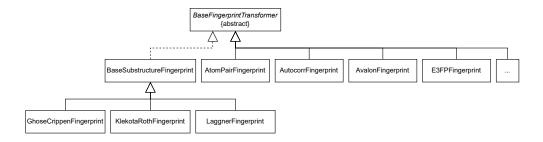
Figure 2: Class diagram for fingerprint classes. Some classes omitted for readability.

CSR format [53]. Various options, such as vector length for hashed fingerprints (e.g. ECFP [54]), binary/count variant, dense/sparse output etc. are specified by constructor parameters. This ensures full composability with scikit-learn constructs like pipelines and feature unions.

We implement more than 30 different fingerprints of various types, e.g. circular ECFP [54] and SECFP [55], path-based Atom Pair [56] and Topological Torsion [57], substructure-based MACCS [58] and Klekota-Roth [51], physicochemical descriptors such as EState [59] and Mordred [60], and more. We used efficient RDKit subroutines, written in C++, e.g. for matching SMARTS patterns. A complete list of implemented fingerprints is available in *scikit-fingerprints* online documentation.

2.2.3. Parallelism

Since molecules can be processed independently when computing fingerprints, the task is embarrassingly parallel [61]. This means that we can efficiently utilize all available CPU cores. To minimize inter-process communication, by default, input molecules are split into as many chunks as there are cores available, and processed in parallel by Python workers. We utilize Joblib [62], with the Loky executor, which uses memory mapping to efficiently pass the resulting arrays between processes. Furthermore, by using sparse arrays and smaller chunk sizes, users can minimize memory utilization for large datasets and fingerprints that yield long output vectors [35].

Furthermore, we support distributed computing with Dask [63], used as a Joblib executor. This way, *scikit-fingerprints* can take advantage of large high-performance computing (HPC) clusters. Connecting to the Dask cluster only requires setting a single parameter in the Joblib configuration [64].

2.2.4. Datasets loading

Fingerprints are often used in the context of molecular property prediction on standardized benchmarks. In particular, they constitute strong baselines, often outperforming complex graph neural networks (GNNs) [13, 17, 14].

Therefore, their easy usage is important for a fair evaluation of advances in graph classification.

We utilized HuggingFace Hub [65, 66] to host datasets. It offers easy downloading, caching, and loading datasets, with automated compression to Parquet format. Currently, the most widely used MoleculeNet [67] benchmark has been integrated, and additional datasets can be easily added with the unified interface. Users can load data sets as in scikit-learn. For example, loading the MoleculeNet BBBP dataset uses the function `load_bbbp()`.

2.2.5. Code quality and CI/CD

We ensure high code quality and security with multiple measures. The code is versioned using Git and GitHub. New features have to be submitted through Pull Requests and undergo code review. We use pre-commit hooks [68] to verify code quality before each commit:

- `bandit` [69], `safety` [70] - security analysis and dependency vulnerability scanning, following security recommendations [71, 72]

- `black` [73], `flake8` [74], `isort` [75], `pyupgrade` [76] - code style, following reproducibility and readability guidelines [77]

- `mypy` [78] - type checking; our entire code is statically typed, following security recommendations [79, 80]

- `xenon` [81] - cyclomatic complexity

We implemented a comprehensive suite of 196 integration and unit tests. They use the PyTest framework [82], and are run automatically on GitHub Runners as a part of the CI/CD process. Passing all tests is required to merge the code into the master branch. We run tests on a full matrix of operating systems (Linux, Windows, MacOS) and Python versions (from 3.9 to 3.12), ensuring proper execution in different environments.

Any changes to the documentation are automatically deployed to the GitHub Pages. New package versions are deployed to PyPI by using GitHub Releases, with new changes description. Internally, this uses a GitHub Actions workflow and creates a Git tag on the commit used in the given release. *scikit-fingerprints* can be installed via pip by running `pip install scikit-fingerprints`.

3. Illustrative examples

3.1. Parallel computation

Since computing molecular fingerprints is an embarrassingly parallel task, it can very effectively utilize modern multicore CPU architectures, e.g. for
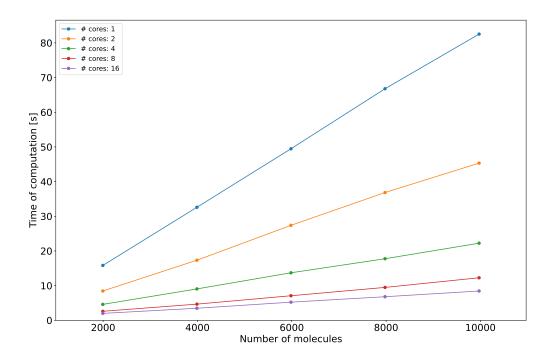
Figure 3: Computation time for PubChem fingerprint.

large databases in molecular property prediction or virtual screening. To illustrate the capability of *scikit-fingerprints* in this regard, we calculate fingerprints for the popular HIV dataset from the MoleculeNet benchmark [67]. It contains a wide variety of molecules from medicinal chemistry, including organometallics, small and large molecules, some atoms with very high numbers of bonds, etc. For this experiment, we limit the data to 10 thousand molecules, due to the high computational time required to run the benchmark multiple times for many data sizes and fingerprints. The code is available in the GitHub repository, in `benchmarking` directory.

As an example, we present the timings for the PubChem fingerprint [83], commonly used for virtual screening, in Figure 3. Speedup for all fingerprints [3] is shown in Figure 4, when using 16 cores and 10 thousand molecules. Speedup is defined as a ratio of sequential to parallel computation time. We calculate those times as an average of 5 runs, using a machine with Intel Core i7-13700K 3.4 GHz CPU. For 3D fingerprints, we do not include the conformer generation time.

---

[3]We omit Pharmacophore fingerprint due to excessive computation time. Due to the checking of multiple SMARTS patterns for all atoms, it is by far the slowest fingerprint.
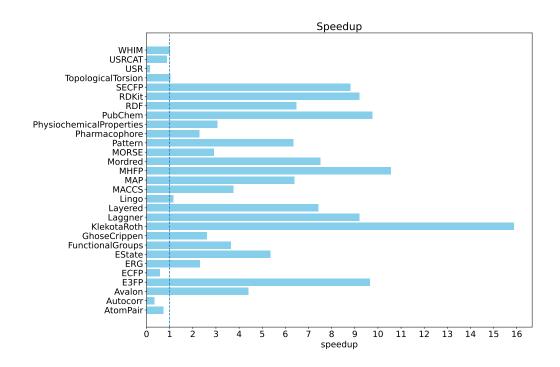
Figure 4: Speedup for fingerprints when using 16 cores.

PubChem fingerprint clearly benefits from parallelism, with time clearly decreasing when using more cores. This behavior is typical for more computationally heavy fingerprints, like substructure-based ones, which have to check numerous SMARTS patterns for each molecule. In particular, as visible in Figure 3, this gain appears for many data sizes. Even for just 2000 molecules, the time decreases from about 15 seconds to just about 2 seconds, which is much more convenient for interactive analyses and ad hoc queries, like searching for similar molecules.

High speedup values indicate that a significant majority of fingerprints benefit from parallelism, with Klekota-Roth achieving the greatest improvement. In general, computationally expensive ones like SECFP or Mordred gain the most. Only the fastest fingerprints, like ECFP or Atom Pair, have a speedup less than 1, meaning slower computation than the sequential one. However, we did not tune the number of cores, and using 6 or 8 could be enough for some fingerprints given this amount of data.

Lastly, in Figure 5 we provide a detailed speedup plot for six commonly used fingerprints of different types: hashed (ECFP [54] and RDKit [84]), substructural (MACCS [58] and PubChem [83]), and descriptors (EState [59] and Mordred [60]). Here, we could use the entire HIV dataset (about 41 thousand molecules), since the computational cost was much lower for only
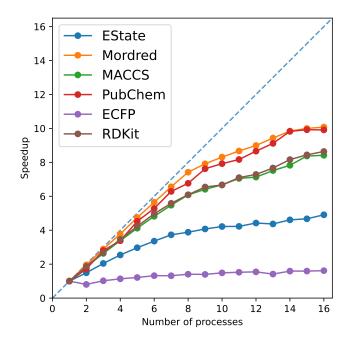
8

Figure 5: Speedup plot for selected fingerprints.

six fingerprints. Overall, fingerprints are well scalable with number of cores, particularly heavier ones like Mordred or PubChem. They achieve almost perfect linear speedup up to about 8 cores. Only the extremely fast ECFP fingerprint seems to be better suited for sequential computation, at least for a dataset of this size.

3.2. Sparse matrix support

Molecular fingerprints are often extremely sparse. Therefore, using proper representation can result in large memory savings, compared to dense arrays. Differences are particularly significant for large datasets, which are typical for virtual screening or similarity searching.

scikit-fingerprints has full support for sparse matrix computations, using SciPy. As an example, we calculated the memory usage of the resulting fingerprint arrays for PCBA dataset from MoleculeNet [67], consisting of almost 440 thousand molecules. In Table 2, we report memory usage of dense and sparse representations. We also report memory savings, defined as how many times the sparse representation reduced the memory usage. For brevity, we show the results of 5 fingerprints with the largest reduction. Code to produce results for all fingerprints is available in the GitHub repository, in `benchmarking` directory.

Fingerprint name	Dense array size (MB)	Sparse array size (MB)	Memory savings
Klekota-Roth	2029	23	88.2x
FCFP	855	15	57x
Physiochemical Properties	855	17	50.3x
ECFP	855	19	45x
Topological Torsion	855	19	45x

Table 2: Memory usage of fingerprints in dense and sparse versions.

Clearly, fingerprints greatly benefit from sparse representations, with a density of arrays around just 1-2%. In particular, popular ECFP and FCFP fingerprints [54] are among those that benefit the most. The Klekota-Roth fingerprint [51], which is quite long for a substructure-based fingerprint, obtains a reduction from almost 2 GB RAM to just 23 MB, i.e. 88.2 times. Those savings would be even more important during the hyperparameter tuning of downstream classifiers when many copies of the data matrix are created. Using a sparse representation did not negatively impact computation time, compared to the dense one.

3.3. Molecular property prediction

scikit-fingerprints can greatly simplify the process of classifying molecules. We show a part of a pipeline in Listing 3.3, responsible for computing ECFP fingerprints from SMILES strings and their classification. For brevity, we omit loading the data, which is just standard Pandas code.

Inputs can be any sequences that consist of SMILES strings or RDKit `Mol` objects, e.g. Python lists or Pandas series. Since `ECFPFingerprint` is a stateless transformer class, it uses an empty `.fit()` method in the pipeline. The code is also parallelized, requiring only the `n_jobs` parameter.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline

from skfp.fingerprints import ECFPFingerprint

pipeline = make_pipeline(
    ECFPFingerprint(n_jobs=-1),
    RandomForestClassifier(n_jobs=-1, random_state=0)
)
pipeline.fit(smiles_train, y_train)

y_pred = pipeline.predict(smiles_test)
```

Fingerprint	Dataset AUROC and tuning gain			Average tuning AUROC gain
	BACE	BBBP	HIV	
GhoseCrippen	84.0 (+2.9)	73.3 (+4.9)	76.0 (+4.3)	+4.0
RDKit	83.0 (+1.2)	73.0 (+5.8)	76.7 (+0.6)	+2.5
Laggner	80.1 (+3.1)	73.8 (+0.7)	76.1 (+1.0)	+1.6
Avalon	83.8 (+2.3)	71.3 (+0.6)	78.0 (+1.7)	+1.5
EState	82.3 (+1.7)	71.7 (+1.0)	76.4 (+0.0)	+0.9

Table 3: Molecular property prediction performance using different fingerprints and gain from tuning their hyperparameters.

3.4. Fingerprint hyperparameter tuning

Most papers in the literature neglect hyperparameter tuning for molecular fingerprints, only tuning downstream classifiers. We conjecture that this is also due to the lack of easy-to-use and efficient software for computing fingerprints. The works that perform such tuning [34, 35] indicate that it is indeed beneficial.

We perform hyperparameter tuning for all 2D fingerprints on MoleculeNet single-task classification datasets [67], using the scaffold split provided by OGB [85]. Only the pharmacophore fingerprint was omitted due to the excessive computation time for some molecules. A Random Forest classifier with default hyperparameters was used, in order to isolate the tuning improvements to just fingerprints. In Table 3, we report the area under receiver operating characteristic curve (AUROC) values obtained when using tuned hyperparameters, improvement from tuning compared to the default parameters, and average gain over all datasets. Due to space limitations, we present the results for 5 fingerprints that had the highest average gain. They can therefore be considered as the methods with the highest tunability [86]. The hyperparameter grids and code are available in the GitHub repository, in `benchmarking` directory.

Tuning fingerprints results in considerable gains, as high as 5.8% AUROC in case of RDKit fingerprint [84] on BBBP dataset. Notably, substructure-based Ghose-Crippen fingerprint [87] gains 4% AUROC on average, using feature counts instead of binary indicators. This signifies that further research in this area, using *scikit-fingerprints*, would be highly beneficial.

3.5. Complex pipelines for 3D fingerprints

For tasks requiring 3D information, i.e. fingerprints based on conformers, the whole processing pipeline becomes more complex. Conformers need to be generated and often post-processed with force field optimization, and resulting fingerprints may have missing values. Additionally, using more than

one fingerprint is often beneficial, especially for virtual screening, as they take into account different geometry features. In Listing 3.5, we present an example of how to create such a pipeline to vectorize molecules for screening, calculating the GETAWAY [88] and WHIM [89] descriptors. This short example would require well over 100 lines of code in RDKit, even without parallelization.

```
from sklearn.impute import SimpleImputer

from skfp.fingerprints import (
    GETAWAYFingerprint, WHIMFingerprint
)
from skfp.preprocessing import ConformerGenerator
from sklearn.pipeline import make_pipeline, make_union


pipeline = make_pipeline(
    ConformerGenerator(
        optimize_force_field="MMFF94", n_jobs=-1
    ),
    make_union(
        GETAWAYFingerprint(n_jobs=-1),
        WHIMFingerprint(n_jobs=-1)
    ),
    SimpleImputer(strategy="mean"),
)
```

3.6. Comparison with existing software

We compare *scikit-fingerprints* with existing libraries for chemoinformatics, which also support the computation of molecular fingerprints. Differences are summarized in Table 4.

In terms of Python support, we provide the first Python-native solution, with other libraries relying on various wrappers. It is also installable with `pip` from PyPI, and can be easily managed with modern dependency managers such as Poetry [90]. We implement the largest number of fingerprints, including both all those available in other libraries, and new ones like MAP4 [91] or E3FP [92]. Another advantage of *scikit-fingerprints* is the full support of parallelism and even distributed computing, which is nonexistent or very limited elsewhere. It is also the only library utilizing pre-commit hooks, dedicated security tools, and offering a fully scikit-learn compatible interface.

	CDK	Open Babel	RDKit	scikit-fingerprints
Language	Java	C++	C++	Python
pip-installable	No	Yes	Yes	Yes
Last PyPI update	Never	2020	2024	2024
Number of fingerprints	13	7	22	31
scikit-learn compatible	No	No	No	Yes
Parallelism	No	No	Very limited	Yes
Pre-commit hooks	No	No	No	Yes
Code quality tools	Yes	No	Yes	Yes
Security tools	No	No	No	Yes
Integrated datasets	No	No	No	Yes
Easy proprietary usage	Yes (LGPL-2.1)	No (GPL-2.0)	Yes (BSD-3)	Yes (MIT)

Table 4: Comparison of scikit-fingerprints with other solutions.

4. Impact

scikit-fingerprints is a comprehensive library for computing molecular fingerprints. Leveraging fully scikit-learn compatible interfaces, researchers can easily integrate it with complex pipelines for processing molecular data. Comprehensive capabilities, with over 30 fingerprints, both 2D and 3D, with efficient conformer generation, enable using varied solutions for molecular property prediction, virtual screening, and other tasks. Intuitive and unified APIs make it easy to use for domain specialists with less programming expertise, like computational chemists, chemoinformaticians, or molecular biologists. We also put strong emphasis on code quality, security, and automated checks and analyzers.

The lack of efficient parallelism is a major downside of existing solutions. Modern molecular databases can easily encompass millions of molecules, especially for virtual screening [11, 12]. Our solution, utilizing all available cores, results in significant speedups, enabling efficient processing of large datasets. This is also beneficial for hyperparameter tuning [34, 35], fingerprint concatenation [15], data fusion [36, 37], and other computationally complex tasks.

Simple class hierarchy and high code quality make our solution easily extensible. New fingerprints can be easily added, automatically benefiting from parallelization and scikit-learn compatibility. GitHub repository had 7 contributors to date, showing a good reception by the community and an easy learning curve. The first issue by an external researcher has been made in a week of making the library public, highlighting the need for modern software in this area.

Research shows that fingerprint-based molecular property prediction remains competitive compared to graph neural networks [13, 16, 14], justifying further

research in this area. In particular, they should be applied as baselines for a fair evaluation of the impact of novel approaches, which is particularly easy with our library. *scikit-fingerprints* has already been applied to molecular chemistry research. In [93], it was used to implement ECFP fingerprint as a baseline algorithm, ensuring fair comparison of various approaches on the MoleculeNet benchmark. It is also actively applied to predict the toxicity of pesticides for honey bees, using the recently proposed ApisTox dataset [94]. Furthermore, numerous research projects at the Faculty of Computer Science at AGH University of Krakow are currently utilizing it.

Finally, *scikit-fingerprints* is constantly evolving, with new fingerprints being added. We are also working on expanding the functionality, e.g. implementing data splitting functions based on fingerprints, or adding molecular filters like Lipinski's Rule of 5 [95] for preprocessing. Therefore, its impact in chemoinformatics will be even greater in the future.

5. Conclusions

We have developed *scikit-fingerprints*, an open source Python library for computing molecular fingerprints. It is simple to use, fully compatible with the scikit-learn API, and easily installable from PyPI. It is also the most feature-rich and highly efficient library available in the open source Python ecosystem, allowing parallel computation of more than 30 different fingerprints. Multiple mechanisms have been implemented to ensure high code quality, maintainability, and security. It fills the gap for a single, definitive software in the Python ecosystem for molecular fingerprints. It facilitates quicker, more efficient, and more comprehensive experiments in the fields of chemoinformatics, drug design, and computational molecular chemistry.

References

[1] R. Todeschini, V. Consonni, Molecular Descriptors for Chemoinformatics, John Wiley & Sons, 2009. `doi:https://doi.org/10.1002/9783527628766`.

[2] A. Koutsoukas, S. Paricharak, W. R. J. D. Galloway, D. R. Spring, A. P. IJzerman, R. C. Glen, D. Marcus, A. Bender, How Diverse Are Diversity Assessment Methods? A Comparative Analysis and Benchmarking of Molecular Descriptor Space, Journal of Chemical Information and Modeling 54 (1) (2014) 230–242. `doi:10.1021/ci400469u`.

[3] R. Sayle, 2D similarity, diversity and clustering in RDKit, RDKit UGM (2019).

[4] A. Bender, How similar are those molecules after all? Use two descriptors and you will have three different answers, Expert Opinion on Drug Discovery 5 (12) (2010) 1141–1151. `doi:10.1517/17460441.2010.517832`.

[5] S. Riniker, G. A. Landrum, Similarity maps - a visualization strategy for molecular fingerprints and machine-learning methods, Journal of Cheminformatics 5 (1) (2013) 43. `doi:10.1186/1758-2946-5-43`.

[6] M. Lovrić, T. Đuričić, H. T. N. Tran, H. Hussain, E. Lacić, M. A. Rasmussen, R. Kern, Should We Embed in Chemistry? A Comparison of Unsupervised Transfer Learning with PCA, UMAP, and VAE on Molecular Fingerprints, Pharmaceuticals 14 (8) (2021). `doi:10.3390/ph14080758`.

[7] S. Hernández-Hernández, P. J. Ballester, On the Best Way to Cluster NCI-60 Molecules, Biomolecules 13 (3) (2023). `doi:10.3390/biom13030498`.

[8] D. Butina, Unsupervised Data Base Clustering Based on Daylight's Fingerprint and Tanimoto Similarity: A Fast and Automated Way To Cluster Small and Large Data Sets, Journal of Chemical Information and Computer Sciences 39 (4) (1999) 747–750. `doi:10.1021/ci9803381`.

[9] M. G. Malhat, H. M. Mousa, A. B. El-Sisi, Improving Jarvis-Patrick algorithm for drug discovery, in: 2014 9th International Conference on Informatics and Systems, 2014, pp. DEKM–61–DEKM–66. `doi:10.1109/INFOS.2014.7036710`.

[10] R. Taylor, Simulation Analysis of Experimental Design Strategies for Screening Random Compounds as Potential New Drugs and Agrochemicals, Journal of Chemical Information and Computer Sciences 35 (1) (1995) 59–67. `doi:10.1021/ci00023a009`.

[11] S. Riniker, G. A. Landrum, Open-source platform to benchmark fingerprints for ligand-based virtual screening, Journal of Cheminformatics 5 (1) (2013) 26. `doi:10.1186/1758-2946-5-26`.

[12] I. Muegge, P. Mukherjee, An overview of molecular fingerprint similarity search in virtual screening, Expert Opinion on Drug Discovery 11 (2) (2016) 137–148. `doi:10.1517/17460441.2016.1117070`.

[13] B. Zagidullin, Z. Wang, Y. Guan, E. Pitkänen, J. Tang, Comparative analysis of molecular fingerprints in prediction of drug combination effects, Briefings in Bioinformatics 22 (6) (2021) bbab291. `doi: 10.1093/bib/bbab291`.

[14] D. Jiang, Z. Wu, C.-Y. Hsieh, G. Chen, B. Liao, Z. Wang, C. Shen, D. Cao, J. Wu, T. Hou, Could graph neural networks learn better molecular representation for drug discovery? A comparison study of descriptor-based and graph-based models, Journal of Cheminformatics 13 (1) (2021) 12. `doi:10.1186/s13321-020-00479-8`.

[15] L. Xie, L. Xu, R. Kong, S. Chang, X. Xu, Improvement of Prediction Performance With Conjoint Molecular Fingerprint in Deep Learning, Frontiers in Pharmacology 11 (2020). `doi:10.3389/fphar.2020.606668`.

[16] N. M. O'Boyle, R. A. Sayle, Comparing structural fingerprints using a literature-based similarity benchmark, Journal of Cheminformatics 8 (1) (2016) 36. `doi:10.1186/s13321-016-0148-0`.

[17] D. Baptista, J. Correia, B. Pereira, M. Rocha, Evaluating molecular representations in machine learning models for drug response prediction and interpretability, Journal of Integrative Bioinformatics 19 (3) (2022) 20220006. `doi:doi:10.1515/jib-2022-0006`.

[18] Y. Song, S. Chang, J. Tian, W. Pan, L. Feng, H. Ji, A Comprehensive Comparative Analysis of Deep Learning Based Feature Representations for Molecular Taste Prediction, Foods 12 (18) (2023). `doi:10.3390/foods12183386`.

[19] Y. Long, H. Pan, C. Zhang, H. T. Song, R. Kondor, A. Rzhetsky, Molecular Fingerprints Are a Simple Yet Effective Solution to the Drug–Drug Interaction Problem, in: The 2022 ICML Workshop on Computational Biology, 2022.

[20] D. Boldini, D. Ballabio, V. Consonni, R. Todeschini, F. Grisoni, S. A. Sieber, Effectiveness of molecular fingerprints for exploring the chemical space of natural products, Journal of Cheminformatics 16 (1) (2024) 35. `doi:10.1186/s13321-024-00830-3`.

[21] B. Ran, L. Chen, M. Li, Y. Han, Q. Dai, Drug-Drug Interactions Prediction Using Fingerprint Only, Computational and Mathematical Methods in Medicine 2022 (1) (2022) 7818480. `doi:https://doi.org/10.1155/2022/7818480`.

[22] J. Deng, Z. Yang, H. Wang, I. Ojima, D. Samaras, F. Wang, A systematic study of key elements underlying molecular property prediction, Nature Communications 14 (1) (2023) 6395. `doi:10.1038/s41467-023-41948-6`.

[23] M. Ashton, J. Barnard, F. Casset, M. Charlton, G. Downs, D. Gorse, J. Holliday, R. Lahana, P. Willett, Identification of Diverse Database Subsets using Property-Based and Fragment-Based Molecular Descriptions, Quantitative Structure-Activity Relationships 21 (6) (2002) 598–604. `doi:https://doi.org/10.1002/qsar.200290002`.

[24] R. Kpanou, P. Dallaire, E. Rousseau, J. Corbeil, Learning self-supervised molecular representations for drug-drug interaction prediction, BMC Bioinformatics 25 (1) (2024) 47. `doi:10.1186/s12859-024-05643-7`.

[25] J. Adamczyk, J. Poziemski, P. Siedlecki, ApisTox: a new benchmark dataset for the classification of small molecules toxicity on honey bees, arXiv preprint arXiv:2404.16196 (2024).

[26] G. A. Landrum, M. Beckers, J. Lanini, N. Schneider, N. Stiefl, S. Riniker, SIMPD: an algorithm for generating simulated time splits for validating machine learning approaches, Journal of Cheminformatics 15 (1) (2023) 119. `doi:10.1186/s13321-023-00787-9`.

[27] T. Wang, J. Sun, Q. Zhao, Investigating cardiotoxicity related with hERG channel blockers using molecular fingerprints and graph attention mechanism, Computers in Biology and Medicine 153 (2023) 106464. `doi:https://doi.org/10.1016/j.compbiomed.2022.106464`.

[28] Z. Chen, L. Zhang, J. Sun, R. Meng, S. Yin, Q. Zhao, DCAMCP: A deep learning model based on capsule network and attention mechanism for molecular carcinogenicity prediction, Journal of Cellular and Molecular Medicine 27 (20) (2023) 3117–3126. `doi:https://doi.org/10.1111/jcmm.17889.`

[29] H. Zhang, J. Wu, S. Liu, S. Han, A pre-trained multi-representation fusion network for molecular property prediction, Information Fusion 103 (2024) 102092.

[30] B. Zhao, W. Xu, J. Guan, S. Zhou, Molecular property prediction based on graph structure learning, Bioinformatics 40 (5) (2024) btae304. `doi:10.1093/bioinformatics/btae304.`

[31] N. Wen, G. Liu, J. Zhang, R. Zhang, Y. Fu, X. Han, A fingerprints based molecular property prediction method using the bert model, Journal of Cheminformatics 14 (1) (2022) 71. `doi:10.1186/s13321-022-00650-3.`

[32] J. Li, X. Jiang, Mol-BERT: An Effective Molecular Representation with BERT for Molecular Property Prediction, Wireless Communications and Mobile Computing 2021 (1) (2021) 7181815. `doi:https://doi.org/10.1155/2021/7181815.`

[33] A. Ilnicka, G. Schneider, Compression of molecular fingerprints with autoencoder networks, Molecular Informatics 42 (6) (2023) 2300059. `doi:https://doi.org/10.1002/minf.202300059.`

[34] S. Cui, Q. Li, D. Li, Z. Lian, J. Hou, Hyper-Mol: Molecular Representation Learning via Fingerprint-Based Hypergraph, Computational Intelligence and Neuroscience 2023 (1) (2023) 3756102. `doi:https://doi.org/10.1155/2023/3756102.`

[35] L. Pattanaik, C. W. Coley, Molecular Representation: Going Long on Fingerprints, Chem 6 (6) (2020) 1204–1207. `doi:https://doi.org/10.1016/j.chempr.2020.05.002.`

[36] C. M. Ginn, P. Willett, J. Bradshaw, Combination of molecular similarity measures using data fusion, Springer Netherlands, Dordrecht, 2002, pp. 1–16. `doi:10.1007/0-306-46883-2_1.`

[37] G. M. Sastry, V. S. S. Inakollu, W. Sherman, Boosting Virtual Screening Enrichments with Data Fusion: Coalescing Hits from Two-Dimensional Fingerprints, Shape, and Docking, Journal of Chemical Information and Modeling 53 (7) (2013) 1531–1542. `doi:10.1021/ci300463g.`

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, Édouard Duchesnay, Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research 12 (85) (2011) 2825–2830.

[39] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in PyTorch, NIPS 2017 Autodiff Workshop (2017).

[40] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, G. Varoquaux, API design for machine learning software: experiences from the scikit-learn project, in: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013, pp. 108–122.

[41] C. Steinbeck, Y. Han, S. Kuhn, O. Horlacher, E. Luttmann, E. Willighagen, The Chemistry Development Kit (CDK): An Open-Source Java Library for Chemo- and Bioinformatics, Journal of Chemical Information and Computer Sciences 43 (2) (2003) 493–500. `doi:10.1021/ci025584y`.

[42] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, G. R. Hutchison, Open Babel: An open chemical toolbox, Journal of Cheminformatics 3 (1) (2011) 33. `doi:10.1186/1758-2946-3-33`.

[43] RDKit: Open-source Cheminformatics, `https://www.rdkit.org`, accessed: 2024-05-08. `doi:10.5281/zenodo.10633624`.

[44] PyPI: Python Package Index (PyPI) is a repository of software for the Python programming language, `https://pypi.org/`, accessed: 2024-05-08.

[45] R. G. A. Bone, M. A. Firth, R. A. Sykes, SMILES Extensions for Pattern Matching and Molecular Transformations: Applications in Chemoinformatics, Journal of Chemical Information and Computer Sciences 39 (5) (1999) 846–860. `doi:10.1021/ci990422w`.

[46] OpenEye OEChem Toolkit documentation: SMILES Line Notation, `https://docs.eyesopen.com/toolkits/python/oechemtk/SMILES.html`, accessed: 2024-09-06.

[47] Open Babel documentation: Radicals and SMILES extensions, `https://openbabel.org/docs/Features/Radicals.html`, accessed: 2024-09-06.

[48] The RDKit Book: Molecular Sanitization, `https://www.rdkit.org/docs/RDKit_Book.html#molecular-sanitization`, accessed: 2024-05-08.

[49] S. Wang, J. Witek, G. A. Landrum, S. Riniker, Improving Conformer Generation for Small Rings and Macrocycles Based on Distance Geometry and Experimental Torsional-Angle Preferences, Journal of Chemical Information and Modeling 60 (4) (2020) 2044–2058. `doi:10.1021/acs.jcim.0c00025`.

[50] A. T. McNutt, F. Bisiriyu, S. Song, A. Vyas, G. R. Hutchison, D. R. Koes, Conformer Generation for Structure-Based Drug Design: How Many and How Good?, Journal of Chemical Information and Modeling 63 (21) (2023) 6598–6607. `doi:10.1021/acs.jcim.3c01245`.

[51] J. Klekota, F. P. Roth, Chemical substructures that enrich for biological activity, Bioinformatics 24 (21) (2008) 2518–2525. `doi:10.1093/bioinformatics/btn479`.

[52] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al., Array programming with NumPy, Nature 585 (7825) (2020) 357–362. `doi:10.1038/s41586-020-2649-2`.

[53] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al., SciPy 1.0: fundamental algorithms for scientific computing in Python, Nature Methods 17 (3) (2020) 261–272. `doi:10.1038/s41592-019-0686-2`.

[54] D. Rogers, M. Hahn, Extended-Connectivity Fingerprints, Journal of Chemical Information and Modeling 50 (5) (2010) 742–754. `doi:10.1021/ci100050t`.

[55] D. Probst, J.-L. Reymond, A probabilistic molecular fingerprint for big data settings, Journal of Cheminformatics 10 (1) (2018) 66. `doi:10.1186/s13321-018-0321-8`.

[56] R. E. Carhart, D. H. Smith, R. Venkataraghavan, Atom pairs as molecular features in structure-activity studies: definition and applications,

Journal of Chemical Information and Computer Sciences 25 (2) (1985) 64–73. `doi:10.1021/ci00046a002`.

[57] R. Nilakantan, N. Bauman, J. S. Dixon, R. Venkataraghavan, Topological torsion: a new molecular descriptor for SAR applications. Comparison with other descriptors, Journal of Chemical Information and Computer Sciences 27 (2) (1987) 82–85. `doi:10.1021/ci00054a008`.

[58] J. L. Durant, B. A. Leland, D. R. Henry, J. G. Nourse, Reoptimization of MDL keys for use in drug discovery, Journal of Chemical Information and Computer Sciences 42 (6) (2002) 1273–1280. `doi:10.1021/ci010132r`.

[59] L. H. Hall, L. B. Kier, Electrotopological State Indices for Atom Types: A Novel Combination of Electronic, Topological, and Valence State Information, Journal of Chemical Information and Computer Sciences 35 (6) (1995) 1039–1045. `doi:10.1021/ci00028a014`.
URL `https://doi.org/10.1021/ci00028a014`

[60] H. Moriwaki, Y.-S. Tian, N. Kawashita, T. Takagi, Mordred: a molecular descriptor calculator, Journal of Cheminformatics 10 (1) (2018) 4. `doi:10.1186/s13321-018-0258-y`.

[61] M. Herlihy, N. Shavit, The Art of Multiprocessor Programming, Revised Reprint, Elsevier, 2012, p. 14.

[62] Joblib: running Python functions as pipeline jobs, `https://joblib.readthedocs.io/en/stable/`, accessed: 2024-05-08.

[63] M. Rocklin, Dask: Parallel Computation with Blocked algorithms and Task Scheduling, in: SciPy, 2015, pp. 126–132. `doi:10.25080/Majora-7b98e3ed-013`.

[64] Dask documentation: Scikit-Learn & Joblib, `https://ml.dask.org/joblib.html`, accessed: 2024-05-08.

[65] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., HuggingFace's Transformers: State-of-the-art Natural Language Processing, arXiv preprint arXiv:1910.03771 (2019).

[66] HuggingFace Hub: scikit-learn organization, `https://huggingface.co/scikit-fingerprints`, accessed: 2024-05-08.

[67] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, V. Pande, MoleculeNet: a benchmark for molecular machine learning, Chemical Science 9 (2) (2018) 513–530. `doi:10.1039/C7SC02664A`.

[68] pre-commit: A framework for managing and maintaining multi-language pre-commit hooks, `https://pre-commit.com`, accessed: 2024-05-08.

[69] bandit: a tool designed to find common security issues in Python code, `https://bandit.readthedocs.io/en/latest/`, accessed: 2024-05-08.

[70] Safety: Python dependency vulnerability scanner, `https://github.com/pyupio/safety`, accessed: 2024-05-08.

[71] S. Peng, P. Liu, J. Han, A Python Security Analysis Framework in Integrity Verification and Vulnerability Detection, Wuhan University Journal of Natural Sciences 24 (2) (2019) 141–148. `doi:10.1007/s11859-019-1379-5`.

[72] M. Alfadel, D. E. Costa, E. Shihab, Empirical analysis of security vulnerabilities in Python packages, Empirical Software Engineering 28 (3) (2023) 59. `doi:10.1007/s10664-022-10278-4`.

[73] black: The uncompromising Python code formatter, `https://black.readthedocs.io/en/stable/`, accessed: 2024-05-08.

[74] flake8: Your Tool For Style Guide Enforcement, `https://flake8.pycqa.org/en/latest/`, accessed: 2024-05-08.

[75] isort: A Python utility / library to sort imports, `https://pycqa.github.io/isort/`, accessed: 2024-05-08.

[76] pyupgrade: A tool to automatically upgrade syntax for newer versions of the language, `https://github.com/asottile/pyupgrade`, accessed: 2024-05-08.

[77] C. T. Hoyt, B. Zdrazil, R. Guha, N. Jeliazkova, K. Martinez-Mayorga, E. Nittinger, Improving reproducibility and reusability in the Journal of Cheminformatics, Journal of Cheminformatics 15 (1) (2023) 62. `doi:10.1186/s13321-023-00730-y`.

[78] mypy: Optional static typing for Python, `https://mypy-lang.org/`, accessed: 2024-05-08.

[79] F. Khan, B. Chen, D. Varro, S. McIntosh, An Empirical Study of Type-Related Defects in Python Projects, IEEE Transactions on Software Engineering 48 (8) (2022) 3145–3158. `doi:10.1109/TSE.2021.3082068`.

[80] H. Gulabovska, Z. Porkoláb, Survey on Static Analysis Tools of Python Programs, in: SQAMIA, 2019.

[81] xenon: Monitoring tool based on radon, `https://github.com/rubik/xenon`, accessed: 2024-05-08.

[82] pytest: Helps you write better programs, `https://pytest.org/`, accessed: 2024-05-08.

[83] PubChem Subgraph Fingerprint, `https://ftp.ncbi.nlm.nih.gov/pubchem/specifications/pubchem_fingerprints.pdf`, accessed: 2024-05-08.

[84] The RDKit Book: RDKit Fingerprints, `https://www.rdkit.org/docs/RDKit_Book.html#rdkit-fingerprints`, accessed: 2024-05-08.

[85] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, J. Leskovec, Open Graph Benchmark: Datasets for Machine Learning on Graphs, in: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20, Curran Associates Inc., Red Hook, NY, USA, 2020.

[86] P. Probst, A.-L. Boulesteix, B. Bischl, Tunability: Importance of Hyper-parameters of Machine Learning Algorithms, Journal of Machine Learning Research 20 (53) (2019) 1–32.

[87] A. K. Ghose, G. M. Crippen, Atomic Physicochemical Parameters for Three-Dimensional Structure-Directed Quantitative Structure-Activity Relationships I. Partition Coefficients as a Measure of Hydrophobicity, Journal of Computational Chemistry 7 (4) (1986) 565–577. `doi:https://doi.org/10.1002/jcc.540070419`.

[88] V. Consonni, R. Todeschini, M. Pavan, Structure/Response Correlations and Similarity/Diversity Analysis by GETAWAY Descriptors. 1. Theory of the Novel 3D Molecular Descriptors, Journal of Chemical Information and Computer Sciences 42 (3) (2002) 682–692. `doi:10.1021/ci015504a`.

[89] R. Todeschini, P. Gramatica, New 3D molecular descriptors: the WHIM theory and QSAR applications, Perspectives in Drug Discovery and Design 9 (0) (1998) 355–380. `doi:10.1023/A:1027284627085`.

[90] Poetry: Python packaging and dependency management made easy, `https://python-poetry.org`, accessed: 2024-05-08.

[91] A. Capecchi, D. Probst, J.-L. Reymond, One molecular fingerprint to rule them all: drugs, biomolecules, and the metabolome, Journal of Cheminformatics 12 (1) (2020) 43. `doi:10.1186/s13321-020-00445-4`.

[92] S. D. Axen, X.-P. Huang, E. L. Cáceres, L. Gendelev, B. L. Roth, M. J. Keiser, A Simple Representation of Three-Dimensional Molecular Structure, Journal of Medicinal Chemistry 60 (17) (2017) 7393–7409. `doi:10.1021/acs.jmedchem.7b00696`.

[93] J. Adamczyk, W. Czech, Molecular Topological Profile (MOLTOP) – Simple and Strong Baseline for Molecular Graph Classification (2024). `arXiv:2407.12136`.

[94] J. Adamczyk, J. Poziemski, P. Siedlecki, Apistox: a new benchmark dataset for the classification of small molecules toxicity on honey bees (2024). `arXiv:2404.16196`.

[95] C. A. Lipinski, F. Lombardo, B. W. Dominy, P. J. Feeney, Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings, Advanced Drug Delivery Reviews 23 (1) (1997) 3–25. `doi:https://doi.org/10.1016/S0169-409X(96)00423-1`.