

Scikit-Mol brings cheminformatics to Scikit-Learn

Esben Jannik Bjerrum^{1*}, Rafał Adam Bachorz^{2,3}, Adrien Bitton⁴, Oh-hyeon Choung⁵, Ya Chen⁶, Carmen Esposito⁷, Son Viet Ha⁸, Andreas Pöhlmann⁴

Scikit-Mol is an open-source toolkit that aims to bridge the gap between two well-established toolkits, RDKit and Scikit-Learn, in order to provide a simple interface for building cheminformatics models. By leveraging the strengths of both RDKit and Scikit-Learn, Scikit-Mol provides a powerful platform for creating predictive modeling in drug discovery and materials design. Unlike other toolkits that often integrate both chemistry and machine learning, Scikit-Mol rather aims to be a simple bridge between the two, reducing the maintenance effort required to keep up with changes and new features in e.g. Scikit-Learn. A simple example of Scikit-Mol's functionality is provided, demonstrating its compatibility with Scikit-Learn pipelines. Overall, Scikit-Mol provides a useful and flexible package for building self-contained and self-documented cheminformatics models with minimal maintenance required.

Introduction

Building machine learning models from molecular datasets plays a vital role in predictive modelling for drug discovery and materials design.[1] It enables the estimation of molecule properties at the design stage, which can aid in the prioritization and filtering of virtual datasets. Recently, predictive models have also been employed to guide de novo generative algorithms for designing molecules with desirable properties.[2] Therefore, it is not surprising that several toolkits and frameworks have been proposed to facilitate this process. These toolkits have different focuses, strengths, and levels of popularity, as evidenced in Table 1.

Two widely used projects are Scikit-Learn[3] and RDKit[4]. The former aims to be a Python/Numpy-based machine learning library, while the latter specializes in cheminformatics calculations and processing, although it also offers some machine learning models. RDKit is a general-purpose toolkit for cheminformatics, with a well-supported interface in Python, and Scikit-Learn is a general-purpose machine-learning framework for Python, with numerous machine learning models and utilities for dimensionality reduction, clustering, or building predictive models.

The Open Drug Discovery Toolkit (ODDT) is a modular toolkit that combines existing open-source toolkits such as RDKit and Scikit-Learn, making it easier to calculate descriptors such as protein-ligand fingerprints and build scoring

models of protein-ligand information.[5] Scikit-Chem delivers a more pythonic interface to RDKit molecular objects and functionality, but it appears largely unmaintained, with the last code commit made in 2020, and the majority from 2016.[6] DeepChem is a framework that aims to provide a framework for doing deep learning in chemistry.[7] PySMILESUtils is a specialized package that augments, tokenizes, and



Figure 1: The Scikit-Mol logo draws inspiration from the two prominent open-source projects that are integrated and linked.

1. *Odyssey Therapeutics, Cambridge, MA, USA*
 2. *Simulations-Plus Inc., Lancaster, CA, USA*
 3. *Polish Academy of Sciences, Łódź, Poland*
 4. *Machine Learning Research, Data Science & AI, Bayer AG, Berlin, Germany*
 5. *ETH, Zürich, Switzerland*
 6. *Department of Pharmaceutical Sciences, Division of Pharmaceutical Chemistry, Faculty of Life Sciences, University of Vienna, 1090 Vienna, Austria*
 7. *Aptuit, an Evotec company, Verona, Italy*
 8. *Johannes Gutenberg University, Mainz, Germany*
- * Corresponding author: esben@odysseytx.com

vectorizes SMILES strings[8] for use with deep learning models, with the purpose of building cheminformatics models. [9] Although it does not appear to be widely used, it was recently featured and used in a paper by Jürgen Schmidhuber and colleagues.[10] OpenChem is a deep learning toolkit for Computational Chemistry with a PyTorch[11] backend. [12] It handles various vectorizations and provides deep learning models. However, the latest commit was made two years ago, except for the License, which was updated last year.

A new project that seems ambitious is the Oloren Chemengine, which aims to provide tools for creating a state-of-the-art molecular property prediction engine.[13] It supports various molecular formats, descriptor calculations, and deep learning embeddings and offers numerous models. The code currently appears to be rising in popularity and is developing fast.

Table 1: Example cheminformatics and machine learning projects, data collected March 2023

Project	Year released (estimated)	Google Scholar Citations	GitHub Stars	GitHub Contributors
RDKit[4]	2006	N/A	2000	155
Scikit-Learn[3]	2007	70717	53400	2610
Open Drug Discovery Toolkit (ODDT)[5]	2015	129	316	10
Scikit-Chem[6]	2016	N/A	57	3
DeepChem[7]	2017	20	4200	152
PySMILESUtills[9]	2017	2	48	5
OpenChem[12]	2020	37	506	2
Scikit-Mol	2022	N/A	22	7
OlorenChemEngine[13]	2022	0	77	5

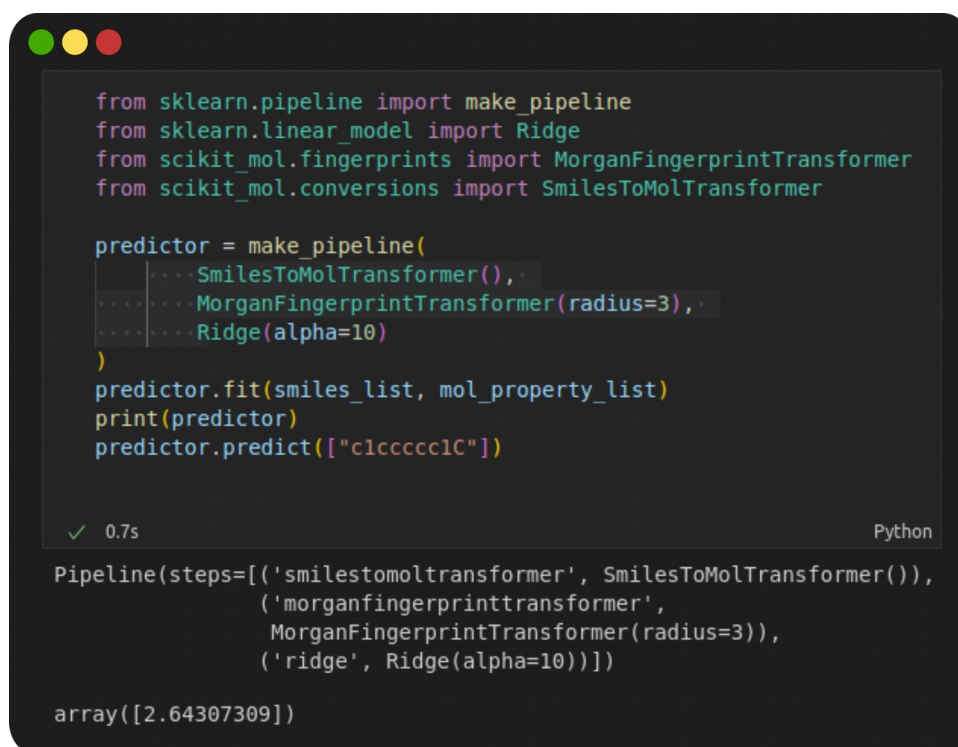
So why do we need another toolkit? Why Scikit-Mol? While some of the toolkits mentioned above, such as RDKit and Scikit-Learn, aim to be general-purpose with each focusing on their respective domains, they are highly successful in their own right. Other toolkits try to integrate both chemistry and machine/deep learning into a single framework, often tailored to the specific needs of the authors. However, this approach can lead to a lot of maintenance effort as the underlying toolkits evolve and develop. For example, when new models become available in Scikit-Learn, a new wrapper may need to be written to include them in the framework, or if PyTorch receives updates with potential API changes, a major update may be required. Consequently, some projects become unmaintained and stale, moreover as their main developers may move on to new positions after the project has served its purpose in a PhD or post-doc project.

Scikit-Mol, on the other hand, aims to be a relatively simple bridge between two already successful and established open-source toolkits, RDKit and Scikit-Learn, both with proven continuous impact and support. It provides support for building simple and largely self-documented picklable models. By bridging these two successful open-source projects, the project becomes less dependent on updates in other projects that necessitate code maintenance. For instance, a new Scikit-Learn machine learning model is likely to be supported without any code changes in Scikit-Mol. Our hope is to create a relevant and helpful package that can remain useful, flexible, and relevant with a minimum of maintenance needed.

A simple Scikit-Mol example

Figure 2 provides a simple example that demonstrates the package's primary use case and objective. The fingerprint transformers are Scikit-Learn compatible and can be integrated into Scikit-Learn pipelines within the Scikit-Learn framework. Once the pipeline is created and trained on existing data, the predictor is self-contained, without the need

for external feature calculation. Instead, the input is in a standard molecular information format, specifically SMILES strings (assuming they are RDKit-parseable). The pipeline is serializable and easily pickled and unpickled, as well as self-documented with easily inspectable parameters. This approach avoids potential uncertainty or misunderstandings about the fingerprint size, radius, and type, should the model later be needed. If imports, print statements, and in-function call newlines are disregarded, a molecular predictive model that uses SMILES strings as input can be created and fitted in just two lines of Python code.



```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Ridge
from scikit_mol.fingerprints import MorganFingerprintTransformer
from scikit_mol.conversions import SmilesToMolTransformer

predictor = make_pipeline(
    SmilesToMolTransformer(),
    MorganFingerprintTransformer(radius=3),
    Ridge(alpha=10)
)
predictor.fit(smiles_list, mol_property_list)
print(predictor)
predictor.predict(["ClCCCClC"])

✓ 0.7s Python

Pipeline(steps=[('smilestomoltransformer', SmilesToMolTransformer()),
                ('morganfingerprinttransformer',
                 MorganFingerprintTransformer(radius=3)),
                ('ridge', Ridge(alpha=10))])

array([2.64307309])
```

Figure 2: A simple example of the tight integration between Scikit-Mol and Scikit-Learn. Assuming data is available as a list of RDKit parsable SMILES strings (*smiles_list*) and a list of properties (*mol_property_list*), a self-contained, serializable and self-documented predictor is easily created.

Installation

The tagged versions of the code from GitHub are automatically updated on the Python Package Index and can be easily installed in a Python environment via pip. To install Scikit-Mol, simply run the command: `pip install scikit-mol`. You can use Conda or virtualenv environments.

Implementation

Scikit-Mol provides a range of classes that subclass the Scikit-Learn transformer mixin and adhere to the requirements and API of Scikit-Learn. Most notably, the transformer classes have a `.transform` method that takes an iterable (e.g., List, NumPy[14] array, or Pandas[15] series) and transforms the members into a new list or array of similar length. For example, the `MorganFingerprintTransformer` will transform a list of RDKit molecular objects into a dense NumPy array that can be directly used in other transformers, scalars, or models from Scikit-Learn.

There are several requirements for a Scikit-Mol class:

- It must subclass `BaseEstimator` and `TransformerMixin` from Scikit-Learn.
- Parameters used in `__init__()` must be keywords with a default value and be set as equally named properties on the object.
- `__init__()` should only be used to set properties. The object must be reconfigurable by changing the properties directly or by using the `set_params` method. Thus, no logic, input sanitation, or other

manipulation should be done in the `__init__` method, as this would interfere with the expectations of the `.set_params` methods used by other classes of Scikit-Learn.

- The class must have a `transform` method that takes data and returns the transformed data. The `transform` method must, for Scikit-Learn compatibility, accept a second optional argument for the y-data, even though it is not used or returned.

Full notes on subclass requirements are available in the Scikit-Learn documentation for developers (<https://scikit-learn.org/stable/developers/develop.html>).

The classes are created around different available RDKit fingerprints and descriptors, but we have grouped the count and bit-based versions of the fingerprints, which can be configured at object instantiation. An overview of the implemented fingerprints and classes can be seen in Table 2.

We have strived to keep the keyword arguments close to the original RDKit options for the fingerprint functions. Conflicts have been resolved via properties. For example, for the RDKit fingerprint, the size of the dense arrays in the hashed fingerprints is determined via the `fpSize` keyword, whereas for the Morgan, atompair, and topological torsion fingerprints, they are set via the `nBits` keyword. Thus, the `RDKitFingerprint` class uses a property setter for the `fpSize` property to instead set the `nBits` property on the object, which is then used by the abstract class for preparation of the dense array.

Table 2: Currently available transformers

Fingerprints	Module
AtomPair	<code>scikit_mol.fingerprints.AtomPairFingerprintTransformer</code>
AvalonFP	<code>scikit_mol.fingerprints.AvalonFingerprintTransformer</code>
MACCSKeys	<code>scikit_mol.fingerprints.MACCSKeysFingerprintTransformer</code>
MHFPPFingerprint	<code>scikit_mol.fingerprints.MHFingerprintTransformer</code>
Morgan	<code>scikit_mol.fingerprints.MorganFingerprintTransformer</code>
RDKitFP	<code>scikit_mol.fingerprints.RDKitFingerprintTransformer</code>
SECFPMol	<code>scikit_mol.fingerprints.SECFingerprintTransformer</code>
TopologicalTorsion	<code>scikit_mol.fingerprints.TopologicalTorsionFingerprintTransformer</code>
Descriptors	
Molecular Descriptors	<code>scikit_mol.descriptors.MolecularDescriptorTransformer</code>
Standardization	
Based on <code>rdMolStandardize</code>	<code>scikit_mol.standardizer.Standardizer</code>
Conversions	
Smiles to RDKit Mol	<code>scikit_mol.conversions.SmilesToMol</code>

The `MolecularDescriptorTransformer` is a re-packaging of the `MolecularDescriptorCalculator` from RDKit which can be configured with a list of the desired descriptors. Additionally, for input standardization purposes, there is a class which implements a basic standardization routine based on RDKit's `rdMolStandardize` module. This transforms the provided RDKit mol objects into uncharged parent molecules.

One possible weakness of the Scikit-Learn pipelines and transformers is that they expect valid input. As a transformer class cannot manipulate both the input features, X , and the target values, y , at the same time, a transformer class for preprocessing or filtering away, e.g., unsanitizable and unparsable SMILES on-the-fly cannot be created. This could lead to unwanted effects and hidden errors in both model-building and prediction. Instead, a pre-sanitization class is provided in the utilities module, which can help filter dataframes or SMILES lists into sanitizable and unsanitizable groups for inspection.

Parallel transformation

Many modern computers have multiple processors, allowing for parallel execution of the Scikit-Mol transformers. The communication of the objects with child threads is implemented by recreating the transformer object in the child process. Instead of pickling the object itself to be sent to the child process, information on the transformer class and the parameter settings of the object is sent and used to recreate a clone of the object in the child process. Some of the fingerprints and descriptors use instantiated RDKit objects that interface with C++ backends to transform molecule objects. In this case, it was deemed safer to recreate these objects from scratch rather than serialize an object that may contain pointers to C++ objects in memory.

However, the creation of child processes differs on various platforms. By default, Linux systems use the fork method, while Windows and Mac use the spawn method. Mac can be set to use fork, but this is not available on Windows systems. The fork method uses copy-on-change, which allows child processes to directly access objects in the parent process's scope without copying them. This reduces both the computational and memory overhead of large read-only datasets. In contrast, the spawn process requires the main thread to be rerun in the child processes, which unfortunately prevents direct usage of the transformers in parallel mode in Jupyter notebooks on Windows platforms and on Mac in the default configurations.

Figure 3 illustrates the speedups that can be achieved for descriptor calculation using the Desc2DTransformer on a workstation with 32 cores (16 physical) using a dataset of 7228 small molecule binders to the serotonin transporter (Genesymbol SLC6A4).[16] As the dataset size and the number of cores used increase, the calculation becomes faster compared to single-threaded performance. When using 16 cores with the full dataset, the calculation only takes 7% of the time taken by a single thread, which is equivalent to a 14-fold speedup. However, for other fingerprint calculations with parallel execution, the time taken may be longer. Figure 4 shows an example with the MorganFingerprintTransformer, where only the largest datasets show a moderate increase in calculation performance.

Generally, the faster the fingerprint calculation itself, the larger the dataset needs to be for parallelism to be worthwhile, as illustrated in the plot in Figure 5. Only the largest datasets (>10,000 samples) would make it worthwhile to parallelize Morgan, Atompairs, and Topological Torsions. SECfingerprint, MACCS keys, and RDKitFP are intermediate and would benefit from parallelism when the dataset size is larger, say >500. Descriptors, on the other hand, benefit almost immediately, even for the smallest datasets (>100 samples).

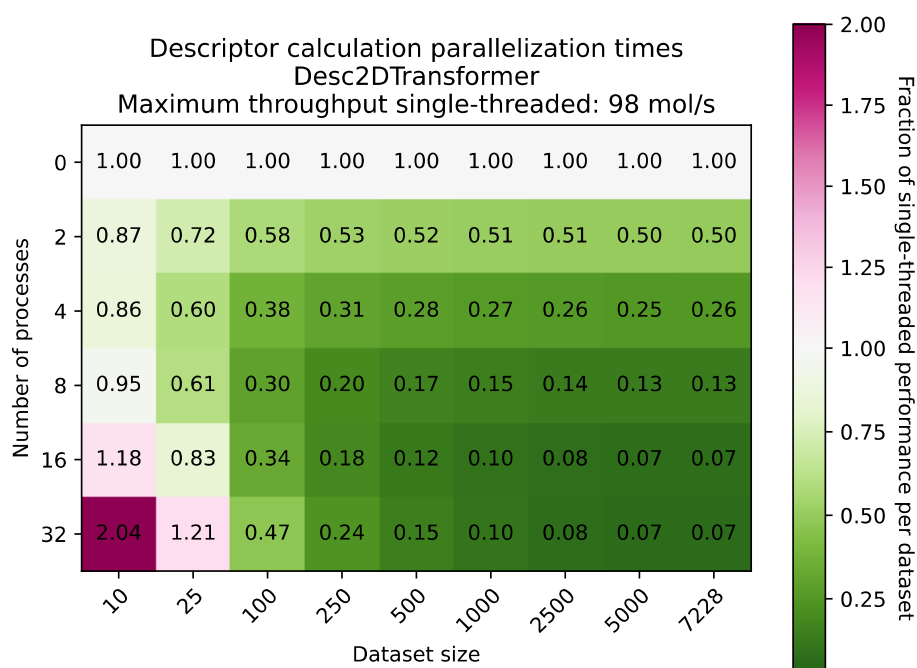


Figure 3: Speedup of descriptor calculation using parallel calculation on different dataset sizes of a dataset of 7228 small molecule binders to the serotonin transporter (Genesymbol SLC6A4) using a workstation with 32 CPUs (16 physical cores)

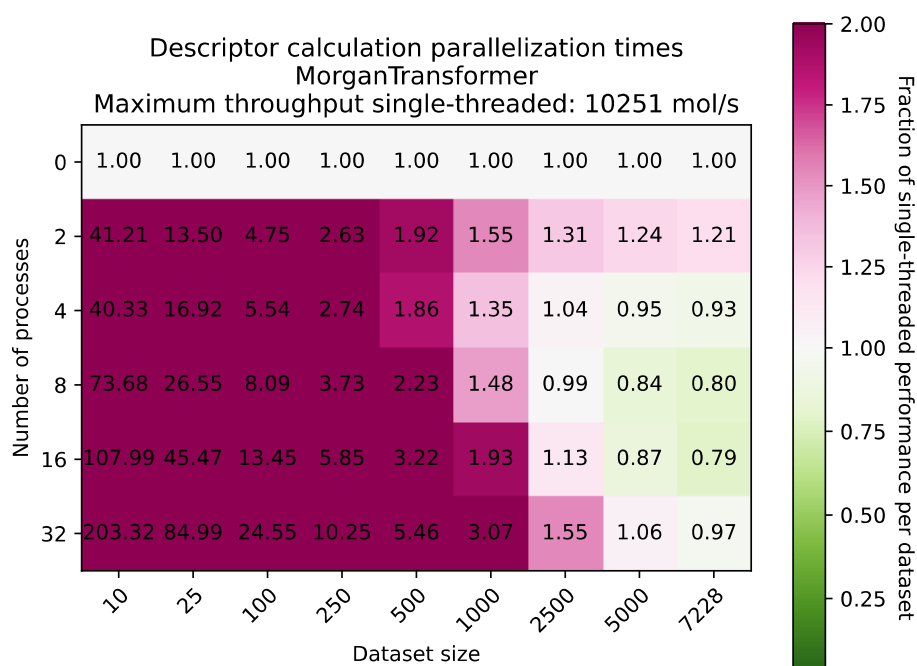


Figure 4: Speedup of Morgan fingerprint calculation using parallel calculation on different dataset sizes of a dataset of 7228 small molecule binders to the serotonin transporter (Genesymbol SLC6A4) using a workstation with 32 CPUs (16 physical cores)

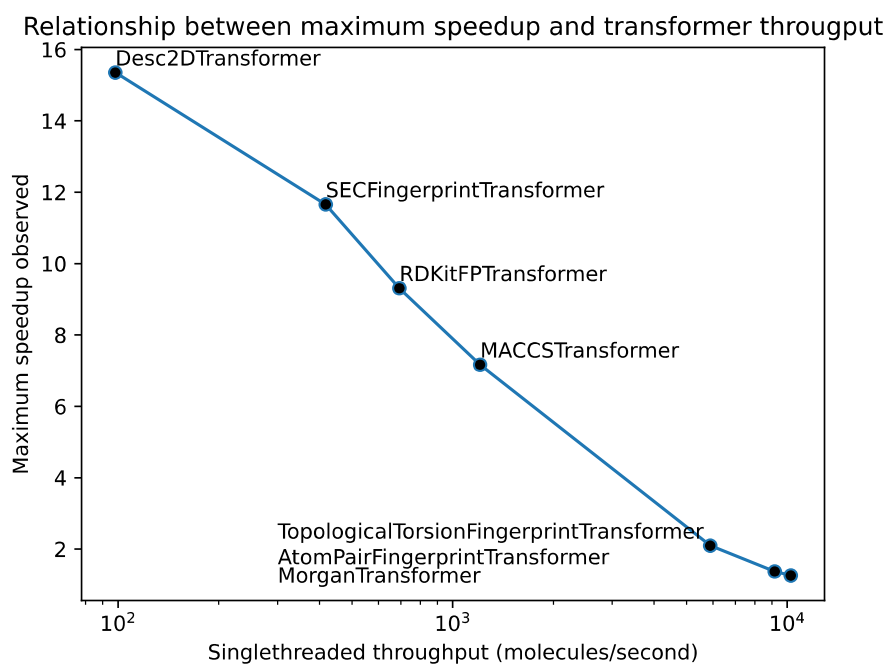


Figure 5: Relationship between parallel performance and the calculation of features of a single molecule.

Documentation

The Scikit-Mol package offers extensive documentation, including several notebooks that illustrate various use-cases of the transformers, such as integration in pipelines, hyperparameter optimization, and standardization. These notebooks are available at the repository (<https://github.com/EBjerrum/scikit-mol/tree/main/notebooks>). Additionally, the classes themselves are well-documented via docstrings that can be accessed in interactive Python through the `help()` function.

Conclusion

In conclusion, Scikit-Mol is a powerful software package that seamlessly integrates RDKit molecular featurization with Scikit-Learn. By leveraging the strengths of these open-source projects, Scikit-Mol provides a simple, yet comprehensive solution for the development of molecular predictors. We believe that by avoiding the creation of a standalone package and instead building on top of well-established resources, Scikit-Mol will remain useful, flexible, and relevant for years to come.

Acknowledgements

Ekaterina V. G. Bjerrum from Productivista.com is acknowledged for the nice Logo design

Conflicts of interest

The authors declare no conflicts of interest

Data Availability Statement

All source code is available via the projects github repository at <https://github.com/EBjerrum/scikit-mol>

References

- [1] Y.-C. Lo, S. E. Rensi, W. Torng, and R. B. Altman, “Machine learning in chemoinformatics and drug discovery,” *Drug Discov. Today*, vol. 23, no. 8, pp. 1538–1546, Aug. 2018, doi: 10.1016/j.drudis.2018.05.010.
- [2] A. Zhavoronkov *et al.*, “Deep learning enables rapid identification of potent DDR1 kinase inhibitors,” *Nat. Biotechnol.*, vol. 37, no. 9, Art. no. 9, Sep. 2019, doi: 10.1038/s41587-019-0224-x.
- [3] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, no. null, pp. 2825–2830, Nov. 2011.
- [4] “RDKit: Open source cheminformatics.” Accessed: Sep. 08, 2022. [Online]. Available: <http://www.rdkit.org>
- [5] M. Wójcikowski, P. Zielenkiewicz, and P. Siedlecki, “Open Drug Discovery Toolkit (ODDT): a new open-source player in the drug discovery field,” *J. Cheminformatics*, vol. 7, no. 1, p. 26, Jun. 2015, doi: 10.1186/s13321-015-0078-2.
- [6] “scikit-chem: simple cheminformatics for Python — scikit-chem 0.0.6 documentation.” <https://scikit-chem.readthedocs.io/en/latest/> (accessed May 01, 2023).
- [7] B. Ramsundar, “Molecular Machine Learning with DeepChem,” Stanford University, 2018. Accessed: May 01, 2023. [Online]. Available: <https://www.proquest.com/openview/9c0e06a343233b48d962991d19873ed8/1?pq-origsite=gscholar&cbl=18750&diss=y>
- [8] D. Weininger, “SMILES, a Chemical Language and Information System: 1: Introduction to Methodology and Encoding Rules,” *J. Chem. Inf. Comput. Sci.*, vol. 28, no. 1, pp. 31–36, 1988, doi: 10.1021/ci00057a005.
- [9] E. Bjerrum, T. Rastemo, R. Irwin, C. Kannas, and S. Genheden, “PySMILESUtils – Enabling deep learning with the SMILES chemical language.” ChemRxiv, Jun. 29, 2021. doi: 10.26434/chemrxiv-2021-kzhbs.
- [10] M. Andronov, V. Voinarovska, N. Andronova, M. Wand, D.-A. Clevert, and J. Schmidhuber, “Reagent prediction with a molecular transformer improves reaction data quality,” *Chem. Sci.*, vol. 14, no. 12, pp. 3235–3246, Mar. 2023, doi: 10.1039/D2SC06798F.
- [11] B. Steiner *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” *Adv. Neural Inf. Process. Syst. NIPS*, no. NeurIPS, 2019.
- [12] M. Korshunova, B. Ginsburg, A. Tropsha, and O. Isayev, “OpenChem: A Deep Learning Toolkit for Computational Chemistry and Drug Design,” *J. Chem. Inf. Model.*, 2021, doi: 10.1021/acs.jcim.0c00971.
- [13] D. Huang *et al.*, “A Unified System for Molecular Property Predictions: Oloren ChemEngine and its Applications.” ChemRxiv, Oct. 20, 2022. doi: 10.26434/chemrxiv-2022-zz776.
- [14] C. R. Harris *et al.*, “Array Programming with NumPy,” *Nature*, vol. 585, no. September, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.
- [15] W. McKinney, “Data Structures for Statistical Computing in Python,” in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 56–61. doi: 10.25080/Majora-92bf1922-00a.
- [16] E. Bjerrum, “SLC6A4_active_escape_export.csv.” figshare, Dec. 10, 2020. doi: 10.6084/m9.figshare.13360199.v1.