DrugEx: deep learning models and tools for exploration of drug-like chemical space

Sicho, M.; Luukkonena, S.; Maagdenberg, H.W. van den; Schoenmaker, L.; Bequignon, O.J.M.; Westen, G.J.P. van

DrugEx: Deep Learning Models and Tools for Exploration of Drug-like Chemical Space

Martin Šicho[a],[†,‡] Sohvi Luukkonen[a],[†] Helle W. van den Maagdenberg[a],[†] Linde Schoenmaker[a],[†] Olivier J. M. Béquignon[a],[†] and Gerard J. P. van Westen[*,†]

†*Leiden Academic Centre for Drug Research, Leiden University, 55 Einsteinweg, 2333 CC Leiden, The Netherlands*

‡*CZ-OPENSCREEN: National Infrastructure for Chemical Biology, Department of Informatics and Chemistry, Faculty of Chemical Technology, University of Chemistry and Technology Prague, Technická 5, 166 28, Prague, Czech Republic*

E-mail: gerard@lacdr.leidenuniv.nl

[a]These authors have contributed equally to this work.

Abstract

The discovery of novel molecules with desirable properties is a classic challenge in medicinal chemistry. With the recent advancements of machine learning, there has been a surge of *de novo* drug design tools. However, few resources exist that are both user-friendly as well as easily customisable. In this application note, we present the new versatile open-source software package DrugEx for multi-objective reinforcement learning. This package contains the consolidated and redesigned scripts from the prior DrugEx papers including multiple generator architectures and a variety of scoring tools and multi-objective optimisation methods. It has a flexible application programming interface and can readily be used via the command line interface or the graphical user interface GenUI. The DrugEx package is publicly available at https://github.com/CDDLeiden/DrugEx.

1 Introduction

Drug discovery is a tedious and resource-intensive process taking up to tens of years and costing millions of dollars on average.[1] Early discontinuation of compounds with poor prognosis using computer-aided drug design can decrease the number of experiments needed, thus helping reduce such costs. *De novo* drug design (DNDD) aims at exploring the vastness of the drug-like chemical space ($\sim 10^{63}$ molecules)[2] to identify hit or lead compounds to be optimised into novel drug candidates.

Rapid technological improvements over the last decades have led to the rising popularity of advanced machine learning methods. These developments have also greatly influenced the field of DNDD with state-of-the-art methods including population-based metaheuristics, recurrent neural networks, generative adversarial networks, variational autoencoders and transformers.[3,4] Moreover, concepts such as transfer, conditional and reinforcement learning are often applied to generate molecules with desired properties.

Typical objectives guiding the drug discovery process are maximisation of predicted efficiencies, synthetic accessibility or drug-likeness of the compounds, and minimising off-target effects and toxicity. Even without optimisation towards favourable physicochemical and pharmacokinetic properties, DNDD is inherently a multi-objective optimisation (MOO) problem.[4,5]

In this application note, we present the new open-source software library DrugEx, a tool for de

novo design of small molecules with deep learning generative models in a multi-objective reinforcement learning (RL) framework. This comprehensive tool represents the consolidation of the original work of Liu et al.'s multiple scripts based DrugEx releases. The first version of DrugEx[6] consisted of a recurrent neural network (RNN) single-task agent of gated recurrent units (GRU) which were updated to long short-term memory (LSTM) units in the second version,[7] also introducing MOO-based RL and an updated exploitation-exploration strategy. In its third version,[8] generators based on a variant of the transformer[9,10] and a novel graph-based encoding allowing for the sampling of molecules with specific substructures were introduced. These developments were built on the work of Olivecrona et al.[11] for the use of reinforcement learning, that of Arus-Pous et al.[12] and of Yang et al.'s *SynthaLinker*[13] for the recurrent neural network and transformer architectures respectively. Here all these versions and capabilities have been consolidated in a single package.

In subsection 2.1, we describe the currently available generator algorithms, the different training modes, and data preprocessing steps and amend the recently introduced graph encoding of molecules from Ref. 8. In subsection 2.2, we present the three steps to score compounds for the RL, the computation and scaling of scores per objective and the multi-objective optimisation, and detail some predefined options. Furthermore, to facilitate usage, this work is supplemented with a rich Python application programming interface (API), a command line interface (CLI) and a graphical user interface (GUI) that are described in section 3. Finally, pre-trained models are made publicly available to ease the *de novo* design of molecules.

2 Application overview

2.1 Molecular generator

2.1.1 Algorithms

The original DrugEx articles describe six different generator architectures.[6–8] The current DrugEx package includes four of these models: two SMILES-based recurrent neural network (RNN) using GRU or LSTM units, and sequence- and graph-based transformers using fragments as starting blocks. The fragment-based LSTM models with and without attention from Ref. 8 have been discontinued as they were outperformed by the other models. The available models are shortly introduced below and the detailed model architectures are described in section S1.

Recurrent neural networks RNNs are used to create molecules without the use of input fragments. These molecules are generated in the form of tokenized SMILES sequences. The RNNs are built from long short-term memory (LSTM) units[14] or gated recurrent units (GRUs).[15] The RNN model consists of the following layers: an embedding layer, three recurrent layers, a linear layer and a softmax activation layer. These building blocks are trained to predict the most likely next output token. Compared to the transformer models this generator does not require inputs, is quick to train and still has a relatively low error rate.

Transformers In addition to the token-based RNNs, DrugEx includes two fragment-based models that are variants of the transformer model using either graphs or sequences as molecular representations. For the fragment-based modelling, molecules are constructed from building blocks (detailed in section 2.1.2). These fragments are combined to create fragmented scaffolds, which form the input for the model and are grown into novel molecules.

Sequence-based transformer The sequence-based transformer model is a decoder-only transformer that applies a multi-headed attention architecture and position-wise feed-forward layers followed by a linear layer and an activation function to predict the most likely output token.[9,10] In contrast to the RNN, the transformer models allow for user-defined inputs in the form of fragments. Furthermore, contrary to the graph-based transformer, the sequence-based transformer allows for easy incorporation of stereochemistry.

Graph-based transformer The Graph-based transformer variant deals with the positional encodings differently from the more classical sequence-based transformer encodings.[9] As with a graph representation the atom index cannot directly be used, the encoding is a combination of the atom index (current position) and the connected atom index (previous position).[8] For more details on the graph representation of the molecules see section 2.1.2). The graph-based model consists of a transformer encoder and a GRU-based decoder. The Graph transformer has some considerable advantages compared to the sequence-based transformer as it only creates valid molecules and has a higher incorporation rate of fragments.

2.1.2 Data preprocessing

The following section describes the default implementation of molecular preparation *i.e.* standardisation and fragmentation in DrugEx (detailed information in section S2). Nevertheless, custom steps can easily be implemented with the provided Python API (subsection 3.1).

In short, standardisation is applied, ensuring that only small organic molecules are kept. Then for the transformer models, fragmentation is performed using the BRICS[16] or RECAP[17] algorithms. Combinations of the obtained fragments are made for the model to be pretrained or fine-tuned on hybridising these fragments to form the original molecules. For the RNN, no fragmentation is performed as the model creates molecules from an empty solution. Encoding of the inputs differs between sequence and graph-based models. Figure Figure 1 illustrates the encoding types per generator algorithm and gives a detailed description of the graph-based encoding as it amends that described in Ref. 8. Details on the encoding of SMILES sequences are available in section S2.

2.1.3 Training

Pretraining & Transfer learning Before guiding a generator to create compounds with specific properties, it needs to be pretrained to learn the language of drug-like molecules and be able to generate reasonable molecules. This involves training a generator to reproduce compounds from a large set of (fragmented) drug-like molecules. We have shared pretrained RNN-based and transformer-based generators (pretrained on ChEMBL27, ChEMBL31[18] and Papyrus v5.5[19]) on Zenodo.[20] Further details about the pretrained models are given in section S1 and section S4.

Furthermore, a generator can be directed towards the desired chemical space by 'fine-tuning' a pretrained generator *via* transfer learning with a set of molecules occupying the desired chemical space.

During training, the loss on a separate test set is assessed at each training epoch to select the best model epoch and allow for early stopping. In brief, for all models, the loss is calculated by taking the average negative log-likelihood (softmax) of the predicted outputs. For the SMILES-based model, it is also possible to use SMILES validity for this purpose.

Reinforcement learning During reinforcement learning (RL), the 'desirability' of generated molecules is quantified by the environment based on various properties and used as a reward (subsection 2.2). The generator is optimised using the policy gradient scheme.[21]

In order to control the exploration rate, molecules are generated based on the output of two generators: a generator that is updated based on the reward function and saved as the final generator (the 'exploitation network' or 'agent') and a static generator (the 'exploration network' or 'prior'). The fraction of outputs coming from each generator mimics the mutation rate in evolutionary algorithms and is tuneable. Currently, this has been tested with the fine-tuned model as exploitation network and the pretrained model as exploration network.[6–8] To improve exploration, the exploitation network can use the outputs from two networks, of which one is constantly updated based on the reward function and the other is only updated every 50 epochs by default (as is shown by Figure 3 in the original paper[7]). This is the default for the RNN-based generator, but due to the higher computation costs of transformer-based generators, we advise against using this periodically updated generator.

Multiple metrics are computed at each epoch:

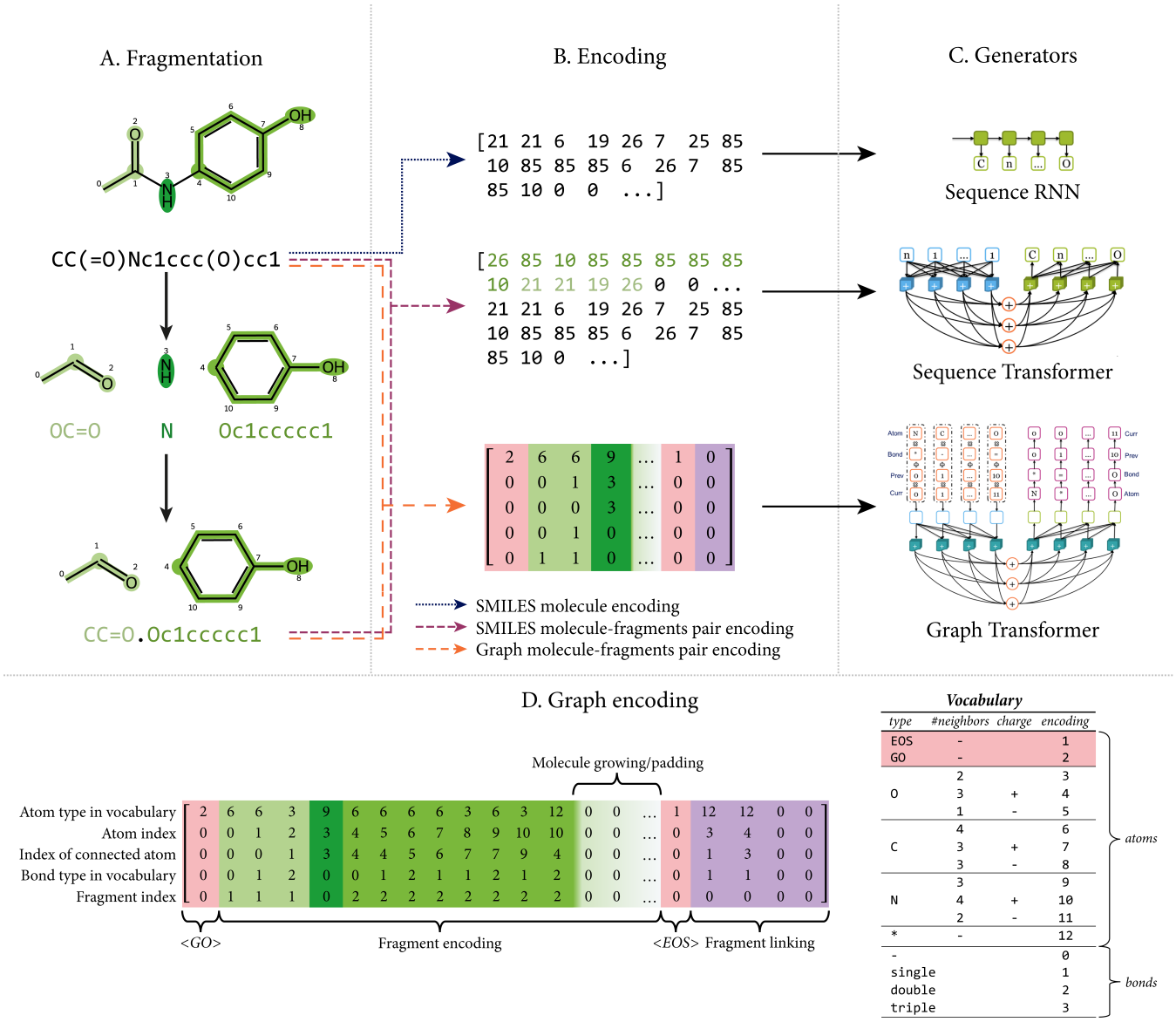


Figure 1: **Correspondence of input and encoding types with generator models.** Input molecules are fragmented for sequence and graph transformers (A), and then input molecules or molecule-fragment pairs are encoded (B) before being used for training and/or sampling by the three generator architectures available (C). Graph encoding matrix of acetaminophen (D) based on the vocabulary next to it. To be encoded as a graph, the molecule is split into three fragments by the BRICS algorithm along the bonds on both sides of the nitrogen atom. Based on the atom-type and bond-type vocabulary encodings a graph matrix is constructed. This matrix consists of 5 rows: (*i*) the current atom type as encoded by the vocabulary, (*ii*) the 0-based atom index in the molecule, (*iii*) the index of one of its neighbouring atoms, (*iv*) the bond type as encoded by the vocabulary and (*v*) the 1-based index of the fragment being encoded respectively. The matrix consists of four major column blocks, from left to right, the start token block (<GO>, pink), the columns used for the encoding of fragments (green), the end token block (<EOS>, pink) and columns indicating the linking between fragments (purple). The dimension of the graph matrix is $5 \times D$ with $D = d - 2 - n_{fragments}$, where $n_{fragments}$ is the number of fragments encoded in the molecule and d the width of the block encoding fragments. Should d be greater than the number of columns required to encode all fragments of a molecule, the remaining columns are filled with zeros, as exemplified by the sub-block used for padding. During sampling, this sub-block is used to grow the molecule. By default, the graph matrix has dimensions 5×400.

the ratio of valid, accurate (only for fragment-based models) unique or desired molecules, and the average arithmetic and geometric mean score per objective; and in the API users can define customised metrics. A compound is "desired" if it fulfils all objectives as defined in section 2.2.2. One of the metrics, by default the desirability ratio is used to select the best model epoch and to allow for early stopping. For fragment-based sampling, the inputs can either be a specific scaffold or the unique fragment combinations of a given dataset.

2.2 Molecule scoring

The scoring of molecules with the environment at each reinforcement learning epoch is done in three stages: *(i)* obtaining raw scores for each of the selected objectives (section 2.2.1), *(ii)* scaling of the raw scores with modifier functions (section 2.2.2) and *(iii)* a multi-objective optimisation step (section 2.2.3) to obtain a final reward per molecule.

2.2.1 Objectives

The API gives a large flexibility to the user to use custom scoring methods that take SMILES as inputs and give a numeric score as output. Moreover, DrugEx is coupled with the QSPRpred package to allow the use of a wide range of ML models and offers a range of other predefined objective functions.

QSPRpred To optimize the binding affinity for one or more targets or other molecular properties, DrugEx users can choose to add one or more quantitative-structure activity/property (QSAR/QSPR) models as objectives for the reinforcement learning reward. To this end, DrugEx is compatible with any python script that receives SMILES as input and produces a score as an output through the API. A separate package, QSPRpred (https://github.com/CDDLeiden/QSPRPred), was developed to simplify the development of QSAR models. The setup of QSPRpred is very similar to DrugEx, using the same structure of the API and command-line interface. It also comes with tutorials to help users get started. QSPRpred has a selection of scikit-learn[22] models and a PyTorch[23] fully-connected neural network available through the API, so the user can train a wide variety of QSAR models. Furthermore, due to its modularity, QSPRpred is customizable; users can for example add new model types and molecular descriptors.

Predefined objectives The DrugEx package offers a set of predefined property calculations that can be used as objectives in the scoring environment. These components are summarised in section S3 and include functions to compute ligand or lipophilic efficiencies from affinity predictions, a variety of similarity measures to a reference structure, estimations of (retro)synthetic accessibility and a plethora of physicochemical descriptors.

2.2.2 Modifiers

To ensure the optimisation, each objective is coupled with a modifier function that transforms it into a maximisation task and scales all raw scores between 0 and 1. Custom modifiers are easily implemented with the API, but DrugEx offers a variety of predefined modifiers for both monotonic (*eg.* `ClippedScore`) and non-monotonic objectives (*eg.* `Gaussian`). Some of these modifiers do not normalise or do not transform the objectives to maximisation tasks in all cases, and should be used with caution, especially when using an aggregation-based multi-objective optimisation scheme. All modifiers are summarised in section S3. Each objective-modifier couple is associated with a desirability threshold set between 0 and 1 to determine if a compound fulfils the desirability criteria on that objective or not. A compound is considered desired if for all objectives its modifier scores are above their corresponding thresholds.

2.2.3 Multi-objective optimisation

Since version 2, DrugEx enables multi-objective optimisation during RL.[7] DrugEx offers three different MOO schemes: a parametric aggregation method and Pareto ranking-based schemes. The aggregation method is the parametric weighted

sum (WS) which uses dynamic weights for each objective to especially reward compounds that perform well on the worst-performing objective(s) at each iteration. Pareto-based schemes do not combine multiple objectives into one but rather search for the best trade-off between them and the initial ranking of molecules is done based on the ranking of the Pareto frontiers. After assigning each molecule to a front, the compounds in each frontier are ranked based on a distance metric to increase the diversity of solutions. DrugEx proposes two distance metric formulations: the crowding distance (PRCD) and the Tanimoto distance (PRTD). For the latter we propose several subtly different ranking schemes all based on the Tanimoto distance. Detailed descriptions of three schemes are given in section S3. The WS is slightly faster than the Pareto-based methods, and more stable for many-objective optimisation as the number of Pareto-equivalent solutions increases with the number of objectives. On the other hand, the Pareto-based schemes enforce diversity in the ranking which is not the case for the WS.

3 Implementation

Aside from the addition of several new features for the generation and scoring of molecular structures, a significant part of the development was dedicated to creating a flexible and scalable software architecture. We have extensively revised the original Python source code of all published DrugEx models[6–8] and transformed it into a self-contained open-source Python package with a clear structure and API. In addition, a simple command line interface (CLI) was also implemented that allows quick invocation of the main DrugEx functions and improves the management of inputs and outputs. The package supports many monitoring utilities that log training progress and result in easy-to-read machine-readable formats such as TSV (Tab Separated Values) and JSON (JavaScript Object Notation) files. When using the CLI, these files are backed up after each (even unsuccessful) run so older results and settings are not lost and can be retrieved at any time. These and other modifications should empower users to quickly explore different scenarios when building their generative models and also ensure reproducible results by keeping track of the set parameters. Both the CLI and Python API are documented and we also created easy-to-follow Jupyter notebooks tutorials to help users get started. Finally, we have performed significant optimisations in multiple parts of the workflow by utilizing multiprocessing where possible.

3.1 Python Package

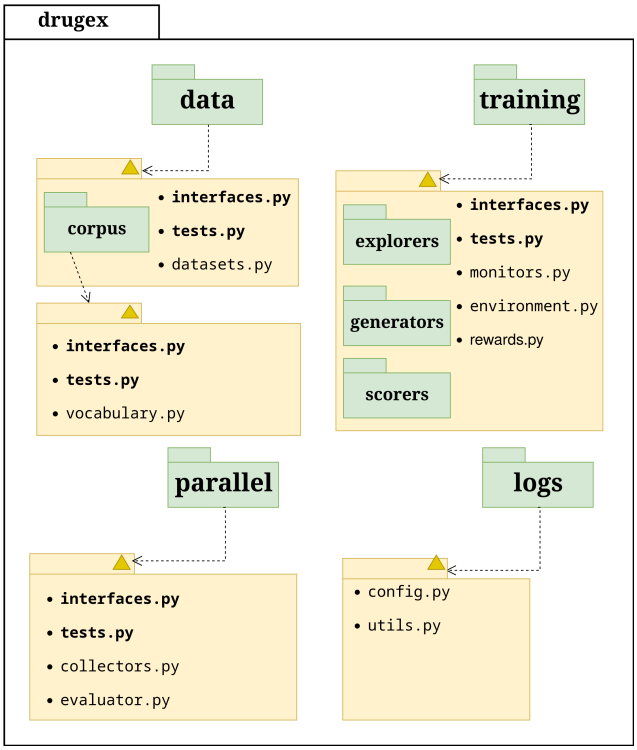


Figure 2: A simplified diagram of the open source Python package architecture. The two main sub-packages of the `drugex` package are `data` and `training`. The `data` package handles the preparation of data sets that can then be used to instantiate and train models in `training`. Aside from the model classes themselves (`generators`), the `training` package also contains data structures needed train model with reinforcement learning (`explorers`) using a scoring environment based on scoring functions (`scorers`).

The software package is divided into intuitively organized sub-packages and modules, each handling either preprocessing of the data, model training or generation of new molecules by loading the

model for sampling (Figure 2). The code is organised with modularity, extensibility, and testability in mind. Each larger subpackage contains a clear definition of its interfaces in the `interfaces.py` module and also unit tests in `tests.py`. Interface definitions are most of the time facilitated through abstract classes for which each subpackage provides default implementations that can be automatically tested with the provided unit tests. These classes form the core of the DrugEx Python API that users can exploit to make modifications to their workflows and/or interact with the DrugEx models programmatically.

Application programming interface The Python API exposes many functions from preprocessing, model training and sampling of molecular structures. Users can mix and match the necessary objects or create custom classes to accomplish their goals. For example, it is possible to apply customized fragmentation strategies by implementing the fragmentation API or use a modified training monitor class to change progress and result tracking during model training.

Command line interface If no customisation is required, the package also offers a command line interface for quick setup of experiments with default implementations of the most common tasks. The package contains three main executable scripts: (1) `dataset.py`, (2) `train.py` and (3) `generate.py`. These scripts are usually executed in order to preprocess input data, train new models and generate a virtual library of compounds. These scripts are installed with the package `drugex`. Each script also automatically logs standard input and output, tracks the history of executed commands and stores generated data outputs so that they can be retrieved later which adds to the reproducibility of experiments.

Documentation Both API and CLI usage is documented and we have tried to provide sufficient description of each interface, class and function. This Sphinx-generated documentation is available at https://cddleiden.github.io/DrugEx/docs and is updated with each new DrugEx release.

Tutorials Aside from source code documentation, the DrugEx web page also provides descriptions of command line arguments and usage examples for the CLI. In addition, we also compiled a collection of Jupyter notebooks that provides a comprehensive introduction to the Python API. The tutorials feature more advanced concepts and are a good starting point for any users who require more customization or any future contributors to familiarise themselves with the code.

3.2 Graphical User Interface

We also added support for the new DrugEx features to our GenUI platform,[24] which provides a graphical user interface (GUI) for molecular generators. GenUI is an open-source web-based application built with the Django web framework[25] and the RDKit cheminformatics toolkit.[26] GenUI provides features for easy integration of cheminformatics tasks commonly used in *de novo* generation of molecules (*i.e.* management of a compound database, QSAR modelling and chemical space visualization). As of now most of the features available through the DrugEx Python package are also exposed in this GUI to allow quick creation and management of generative workflows from the import of the training data to interactive visual analysis of the generated and real chemical space. One notable feature of the new GUI is the interactive creation of scoring environments, which makes the setup of desirability modifier functions for the multi-objective optimization more intuitive (Figure 3).

4 Conclusion

In this paper, we have described the DrugEx open-source software package that facilitates the training of a diverse set of generative models for *de novo* design of small molecules. The package is based on the original Python scripts previously introduced by Liu et al. that were used to develop and validate these models.[6–8] It includes the following new features: early stopping in all training modes, additional predefined scoring functions, and improved QSPR modelling (hyperparameter optimisation, new input features, etc.) with the

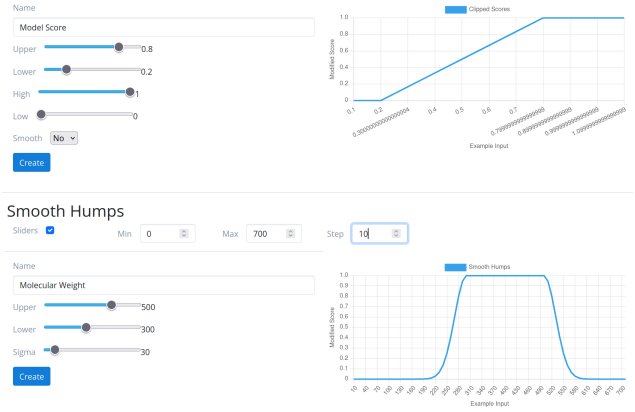


Figure 3: An impression of GenUI, showing the interactive interface for visualising and creating desirability modifiers by simply adjusting sliders or inputting parameter values directly. In the shown example, a Clipped Score modifier is used with an activity classifier to give a maximum reward to structures with probability scores higher than 0.8, and a Smooth Hump function is used to reward structures with molecular weight in between 200 and 500 Daltons).

separate QSPRpred package. The performance has also been enhanced by utilizing parallel processing where possible in both the DrugEx and QSPRpred packages. Furthermore, the current implementation features major revisions of the original API source code of which most notable are the addition of a command line interface (CLI) and Python API. A graphical user interface (GUI) is also provided via the GenUI web application.

We envisage that the new DrugEx software package and its GenUI integration should be suitable for a diverse set of users. On one side, the package provides a quick and easy way to set up experiments and build models via the CLI and GUI, but on the other side, it also enables more advanced alterations to the workflow through the new Python API. The documentation was also significantly improved and we now provide an easy-to-follow tutorial for new users. Finally, all software presented in this work is provided as open-source software and accessible at https://github.com/CDDLeiden/DrugEx.

We regard the publication of this package as an important step in the development of DrugEx that will be the basis for many research projects and innovations yet to come. In fact, we believe that groundbreaking approaches are only possible when developers of generative models for chemistry undertake such open-source software development initiatives, to facilitate prospective validation and testing of their new methods and most importantly their application. Additionally, providing rich documentation and tutorial helps enhancing the models' usability and integration potential, allowing for faster adoption and feedback leading to the development of better AI-powered models and tools.

In future developments of the DrugEx package, we will not only focus on the integration of novel objectives from the drug discovery toolbox (i.e. molecular docking or retrosynthesis prediction), but also on increasing the range of possible inputs to alternative linear representations of compounds (i.e. SELFIES[27]) in sequence-based models or adding support for encoding stereochemistry. Moreover, we would like to focus on the development of user-centric features such as providing an even better learning platform for teaching the underlying concepts of AI-based molecular generation and improving the GenUI integration. The potential of artificial intelligence in drug discovery is tremendous, but integrating these novel tools in current workflows still remains a challenge and we hope that our software package will help to overcome at least some of those challenges.

Author information MS (orcid:0000-0002-8771-1731); SL (orcid:0000-0001-9387-1427); HWvdM (orcid:0000-0002-9718-7806); LS (orcid:0000-0001-9879-1004); OJMB (orcid:0000-0002-7554-9220); GJPvW (orcid:0000-0003-0717-1817)

Author contributions **MS**: Methodology, Software, Data Curation, Writing – Original Draft, Writing - Review & Editing, Visualization **SL**: Methodology, Software, Data Curation, Writing – Original Draft, Writing - Review & Editing, Visualization, Project administration **HWvdM**: Methodology, Software, Data Curation, Writing – Original Draft, Writing - Review & Editing, Visualization **LS**: Methodology, Software, Data Curation, Writing – Original Draft, Writing - Review & Editing, Visualization **OJMB**: Methodol-

ogy, Software, Data Curation, Writing – Original Draft, Writing - Review & Editing, Visualization **GJPvW**: Resources, Writing - Review & Editing, Supervision, Funding acquisition

Funding SL received funding from the Dutch Research Council (NWO) in the framework of the Science PPP Fund for the top sectors, and acknowledges the Dutch Research Council (NWO ENPPS.LIFT.019.010). MŠ was supported by Czech Science Foundation Grant No. 22-17367O and by the Ministry of Education, Youth and Sports of the Czech Republic (project number LM2023052).

Acknowledgement The authors thank Xuhan Liu, the author of the original idea to develop the DrugEx models and code, we are happy for his continuous support of the project; Roelof van der Kleij for his help using the university IT infrastructure; our Master student Yorick van Aalst for testing of the code; and Alan K. Hassen and Andrius Bernatavicius for fruitful discussions. Some of the used computational resources were provided by the e-INFRA CZ project (ID:90140), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

(1) Wouters, O. J.; McKee, M.; Luyten, J. Estimated Research and Development Investment Needed to Bring a New Medicine to Market, 2009-2018. *Journal of the American Medical Association* **2020**, *323*, 844–853, DOI: 10.1001/jama.2020.1166.

(2) Kirkpatrick, P.; Ellis, C. Chemical space. *Nature* **2004**, *432*, 823, DOI: 10.1038/432823a.

(3) Liu, X.; IJzerman, A. P.; van Westen, G. J. P. In *Artificial Neural Networks*; Cartwright, H., Ed.; Springer US: New York, NY, 2021; Vol. 2190; pp 139–165, DOI: 10.1007/978-1-0716-0826-5$_6$, Series Title: Methods in Molecular Biology.

(4) Luukkonen, S.; van den Maagdenberg, H. W.; Emmerich, M. T.; van Westen, G. J. Artificial Intelligence in Multi-objective Drug Design. *Current Opinion in Structural Biology* **2023**, *79*, DOI: 10.1016/j.sbi.2023.102537.

(5) Fromer, J. C.; Coley, C. W. Computer-aided multi-objective optimization in small molecule discovery. *Patterns* **2023**, *4*, 100678, DOI: 10.1016/j.patter.2023.100678.

(6) Liu, X.; Ye, K.; van Vlijmen, H. W. T.; IJzerman, A. P.; van Westen, G. J. P. An exploration strategy improves the diversity of de novo ligands using deep reinforcement learning: a case for the adenosine A2A receptor. *Journal of Cheminformatics* **2019**, *11*, 35, DOI: 10.1186/s13321-019-0355-6.

(7) Liu, X.; Ye, K.; van Vlijmen, H. W. T.; Emmerich, M. T. M.; IJzerman, A. P.; van Westen, G. J. P. DrugEx v2: de novo design of drug molecules by Pareto-based multi-objective reinforcement learning in polypharmacology. *Journal of Cheminformatics* **2021**, *13*, 85, DOI: 10.1186/s13321-021-00561-9.

(8) Liu, X.; Ye, K.; van Vlijmen, H. W. T.; IJzerman, A. P.; van Westen, G. J. P. DrugEx v3: scaffold-constrained drug design with graph transformer-based reinforcement learning. *Journal of Cheminformatics* **2023**, *15*, 24, DOI: 10.1186/s13321-023-00694-z.

(9) Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention is All You Need. 2017.

(10) Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I., et al. Improving language understanding by generative pre-training. **2018**,

(11) Olivecrona, M.; Blaschke, T.; Engkvist, O.; Chen, H. Molecular de-novo design through deep reinforcement learning. *Journal of Cheminformatics* **2017**, *9*, 48, DOI: 10.1186/s13321-017-0235-x.

(12) Arús-Pous, J.; Patronov, A.; Bjerrum, E. J.; Tyrchan, C.; Reymond, J.-L.; Chen, H.; Engkvist, O. SMILES-based deep generative scaffold decorator for de-novo drug design. *Journal of Cheminformatics* **2020**, *12*, 38, DOI: 10.1186/s13321-020-00441-8.

(13) Yang, Y.; Zheng, S.; Su, S.; Zhao, C.; Xu, J.; Chen, H. SyntaLinker: automatic fragment linking with deep conditional transformer neural networks. *Chemical Science* **2020**, *11*, 8312–8322, DOI: 10.1039/D0SC03126G.

(14) Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Computation* **1997**, *9*, 1735–1780, DOI: 10.1162/neco.1997.9.8.1735.

(15) Cho, K.; van Merrienboer, B.; Bahdanau, D.; Bengio, Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR* **2014**, *abs/1409.1259*.

(16) Degen, J.; Wegscheid-Gerlach, C.; Zaliani, A.; Rarey, M. On the Art of Compiling and Using 'Drug-Like' Chemical Fragment Spaces. *ChemMedChem* **2008**, *3*, 1503–1507, DOI: https://doi.org/10.1002/cmdc.200800178.

(17) Lewell, X. Q.; Judd, D. B.; Watson, S. P.; Hann, M. M. *Journal of Chemical Information and Computer Sciences* **1998**, *38*, 511–522, DOI: 10.1021/ci970429i.

(18) Gaulton, A. et al. The ChEMBL database in 2017. *Nucleic Acids Research* **2016**, *45*, D945–D954, DOI: 10.1093/nar/gkw1074.

(19) Béquignon, O. J. M.; Bongers, B. J.; Jespers, W.; IJzerman, A. P.; van der Water, B.; van Westen, G. J. P. Papyrus: a large-scale curated dataset aimed at bioactivity predictions. *Journal of Cheminformatics* **2023**, *15*, 3, DOI: 10.1186/s13321-022-00672-x.

(20) (a) Béquignon, O. J. M. DrugEx RNN-GRU pretrained model (ChEMBL31). 2023; https://doi.org/10.5281/zenodo.7550739; (b) Béquignon, O. J. M. DrugEx RNN-GRU pretrained model (Papyrus 05.5). 2023; https://doi.org/10.5281/zenodo.7550792; (c) Liu, X. DrugEx v2 pretrained model (ChEMBL27). 2022; https://doi.org/10.5281/zenodo.7096837; (d) Béquignon, O. J. M. DrugEx v2 pretrained model (ChEMBL31). 2022; https://doi.org/10.5281/zenodo.7378916; (e) Schoenmaker, L.; Béquignon, O. J. M. DrugEx v2 pretrained model (Papyrus 05.5). 2022; https://doi.org/10.5281/zenodo.7378923; (f) Sicho, M. DrugEx pretrained model (SMILES-based; Papyrus 05.5). 2023; https://doi.org/10.5281/zenodo.7635064; (g) Béquignon, O. J. M. DrugEx pretrained model (SMILES-based; RECAP; Papyrus 05.5). 2023; https://doi.org/10.5281/zenodo.7622774; (h) Liu, X. DrugEx v3 pretrained model (graph-based; ChEMBL27). 2022; https://doi.org/10.5281/zenodo.7096823; (i) Béquignon, O. J. M. DrugEx v3 pretrained model (graph-based; Papyrus 05.5). 2022; https://doi.org/10.5281/zenodo.7085421; (j) Béquignon, O. J. M. DrugEx pretrained model (graph-based; RECAP; Papyrus 05.5). 2023; https://doi.org/10.5281/zenodo.7622738.

(21) Sutton, R. S.; McAllester, D.; Singh, S.; Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. Advances in Neural Information Processing Systems 12. 1999.

(22) Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **2011**, *12*, 2825–2830.

(23) Paszke, A. et al. *Advances in Neural Information Processing Systems 32*; Curran Associates, Inc., 2019; pp 8024–8035.

(24) Sicho, M.; Liu, X.; Svozil, D.; van Westen, G. J. P. GenUI: interactive and extensible open source software platform for de novo molecular generation and cheminformatics. *Journal of*

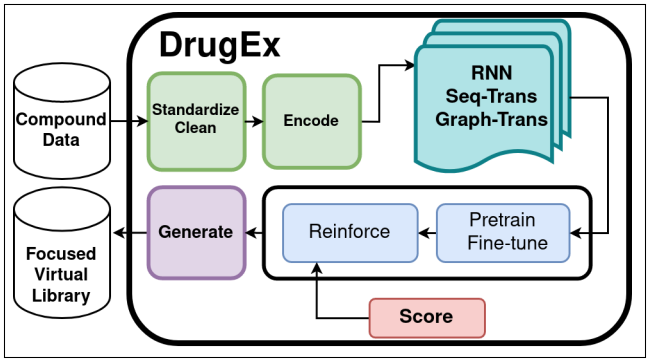
Cheminformatics **2021**, *13*, 73, DOI: 10.1186/s13321-021-00550-y.

(25) Django Software Foundation, Django Web Framework. https://djangoproject.com.

(26) RDKit, RDKit: Open-source cheminformatics. https://www.rdkit.org.

(27) Krenn, M.; Häse, F.; Nigam, A.; Friederich, P.; Aspuru-Guzik, A. Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Machine Learning: Science and Technology* **2020**, *1*, 045024, DOI: 10.1088/2632-2153/aba947, Publisher: IOP Publishing.

TOC Graphic


DrugEx
Compound Data
Standardize Clean
Encode
RNN
Seq-Trans
Graph-Trans
Focused Virtual Library
Generate
Reinforce
Pretrain
Fine-tune
Score