GenUI: Interactive and Extensible Open Source Software Platform for *De Novo* Molecular Generation

M. Sicho[1,2,†], X. Liu[3,†], D. Svozil[1,2], G.J.P. van Westen[3,*]

[1]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

[2]CZ-OPENSCREEN: National Infrastructure for Chemical Biology, Institute of Molecular Genetics of the ASCR, v. v. i., Vídeňská 1083, 142 20 Prague 4, Czech Republic

[3]Drug Discovery and Safety, Leiden Academic Centre for Drug Research, Einsteinweg 55, Leiden, The Netherlands

[†]These authors contributed equally to this work

[*]corresponding author, e-mail: gerard@lacdr.leidenuniv.nl

Email addresses:

MŠ: martin.sicho@vscht.cz, ORCID: 0000-0002-8771-1731

XL: liu.x@lacdr.leidenuniv.nl, ORCID: 0000-0003-2368-4655

DS: daniel.svozil@vscht.cz, ORCID: 0000-0003-2577-5163

Abstract

Computer-aided *de novo* drug design holds promise to significantly accelerate the drug discovery process and bring down its costs. Thanks to this outlook, the field has thrived in the past few years and has seen a surge of new method development due to the proliferation of generative deep neural networks. However, the widespread adoption of new *de novo* drug design techniques has been slow in fields like medicinal chemistry or chemical biology. Such development is not surprising since in order to successfully integrate *de novo* drug design in existing processes and pipelines, a close collaboration between diverse groups of experimental and theoretical scientists needs to be established. Therefore, to accelerate the adoption of both modern and traditional *de novo* molecular generators, we developed GenUI (Generator User Interface), a software platform that makes it possible to integrate molecular generators within a feature-rich graphical user interface that is easy to use by experts of varying backgrounds. GenUI is implemented as a web service and its interfaces offer tools for data preprocessing, model building, molecule generation, and interactive chemical space visualization. Moreover, the platform is easy to extend with customizable frontend React.js components and backend Python extensions. GenUI is open source and a recently developed *de novo* molecular generator, DrugEx, was integrated as a proof of principle. In this work, we present the architecture and implementation details of the GenUI platform and discuss how it can facilitate collaboration in the disparate communities interested in *de novo* drug design and molecule generation.

Keywords

graphical user interface, de novo drug design, molecule generation, deep learning, web application

Introduction

Due to significant technological advances in the past decades, the body of knowledge on the effects and roles of small molecules in living organisms has grown tremendously [1, 2]. At present, we assume the number of entries across all databases to be in the range of hundreds of millions or billions (10^8-10^9) [3-5] and a large portion of this data has also accumulated in public databases such as ChEMBL [6, 7] or PubChem BioAssay [1]. Still, these numbers are rather small in comparison to 10^{33}, a recently reported estimation of the size of the drug like chemical space [8]. However, it should be noted that numerous studies in the past reported numbers both bigger and smaller depending on the definition used [8-11]. In addition, considering that only 1-2 measured biological activities per compound are available [12], the characterization of known compounds also needs to be expanded.

For a long time *de novo* molecular design algorithms for systematic and rational exploration of chemical space [13-15] and quantitative structure-activity relationship (QSAR) modeling [16] have been considered as tools that could broaden our horizons with less experimental costs and without the need to exhaustively evaluate as many as 10^{33} possible drug-like compounds to find the few of interest. The relevance of QSAR modeling and *de novo* molecular design for drug discovery has been discussed many times [13-21], but these approaches can be just as useful in the areas of chemical biology that require new tool compounds and chemical probes that might not be constrained to drug-like molecules only [22].

Thanks to the constant growth of bioactivity databases and widespread utilization of graphical processing units (GPUs) the application of powerful data-driven approaches based on deep neural networks (DNNs) has grown substantially. DNNs found many use cases in molecular virtual screening and *de novo* compound generation (**Figure 1**) [19]. This rapidly evolving class of algorithms has been influencing modern drug discovery by building more accurate QSAR models [12, 23], creating better molecular representations [24-26], predicting 3D protein structure with impressive accuracy [27] or achieving other promising results in many medicinal and clinical applications [3, 12, 17, 21, 28-30].

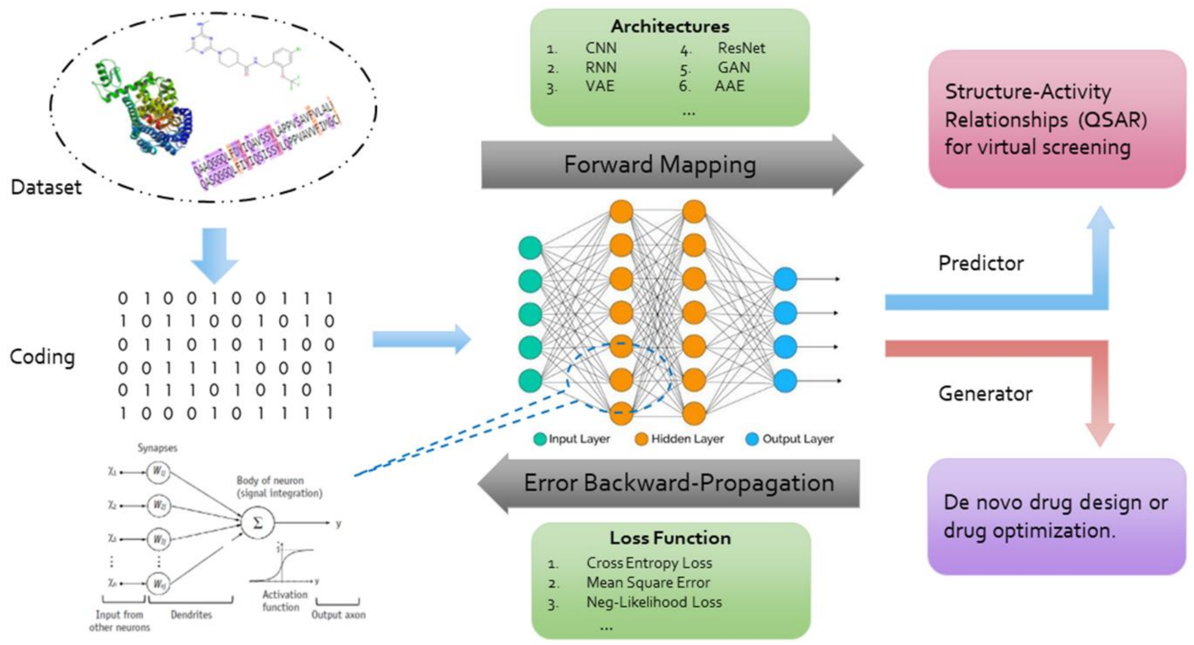


Figure 1 Schematic view of a typical cheminformatics workflow involving a DNN. First, a data set of compound structures and their measured activities on the desired target molecule (most often a protein) is compiled and encoded to suitable representation. Second, the encoded data is used as input of the neural network in forward mapping. A large number of architectures can be used with recurrent neural networks (RNNs) and convolutional neural networks (CNNs) as the most popular examples. Finally, the neural network is trained by backpropagating the error of a suitable loss function to adjust the activations inside the network so that the loss is minimized. Depending on the architecture, the network is trained either as a bioactivity predictor (e.g. a QSAR model) or as a molecular generator.

In the field of *de novo* drug design, the most attractive feature of DNNs is their ability to probabilistically generate compound structures [13, 31]. DNNs are able to take non-trivial structure-activity patterns into account, thereby increasing the potential for scaffold hopping and the diversity of designed molecules [32, 33]. A large number of generators based on DNNs were developed recently demonstrating the ability of various network architectures to generate compounds of given properties (biological activity included) [13, 31, 34-37].

Even though deep learning has been dominating *de novo* drug design in the recent years, it should be noted that the field also has a long history of evolutionary heuristic methods such as genetic algorithms on the forefront [20]. These traditional methods are still being investigated and developed [38-43] and it is yet to be established how they compare to the novel approaches based on deep learning [13]. Due to the simpler nature of these traditional approaches non-obvious relationships can be easily missed, which may affect the quality of the suggested chemical structures. However, simplicity can also be an advantage since interpretation of simpler methods is easier. This is especially problematic for deep learning models that can have more than thousands of parameters [44]. Moreover, a simpler method requires less training data [38] without sacrificing chemical space coverage [45].

One of the open questions for both traditional and deep learning molecular generators is also how they should be benchmarked, compared and interpreted [40]. Therefore, benchmarking studies of *de novo* drug design approaches are also the subject of ongoing research [46-48] and much needed to ensure that these methods have conclusive real impact on new ligand discovery [49, 50]. However, the ultimate test of a *de novo* drug design method should always be prospective application in real projects with experimental validation of the generated molecules.

Although *de novo* molecular design algorithms have been in development for multiple decades [51] and experimentally validated active compounds have been proposed [18, 52-59], these success stories are still far away from the envisaged performance of the ‘robot scientist’ [60-62]. Successful development of a completely automated and sufficiently accurate process has been elusive and hindered mostly by the computational expense and poor synthetic availability of the generated compounds [18]. Despite increasing efforts to automate the scientific process of decision making [18, 63-65], human insight and manual labor are still necessary to further refine the compounds generated by *de novo* molecular design algorithms. In particular, human intervention is of utmost importance in the process of compound scoring in which best candidates are prioritized for synthesis and experimental validation [18, 65].

Though many *in silico* compound generation and optimization tools are available for free [66], it is still an exception that these approaches are routinely used. The vast majority of methods described in the literature serve only as a proof of concept. Hence, they lack a proper graphical user interface (GUI) through which non-experts could easily access the algorithms and analyze their inputs and outputs in a convenient way. Even if such a GUI exists, it is often simplistic and intended to be used only with one particular method [41, 43, 67, 68]. Lack of easy to explain and auditable information systems is a factor leading to some level of disconnection between medicinal and computational chemists [69], which can hinder tighter collaboration that can stand in the way of effective utilization of many promising *de novo* drug design methods. Many molecular generators would also benefit from a comprehensive and easy to use application programming interface (API) that would enable easier integration with existing computational infrastructures. Recently a tool called Flame was presented that offers many of the aforementioned features in the field of predictive QSAR modeling [70], but while there are closed-source solutions like BRADSHAW [71] or Chemistry42 [72] to the best of our knowledge there is no such solution in the realm of open source software for *de novo* drug design. However, there has been effort to develop interactive databases of generated structures as evidenced by the most recent example, cheML.io [73].

In this work we present the development of GenUI, a software framework that provides a general-purpose GUI for molecular generators and enables easy integration of such algorithms with existing drug discovery pipelines as well. The GenUI framework integrates solutions for import, generation, storage and retrieval of compounds, visualization of the created molecular data sets and basic utilities for QSAR modeling. All features can be easily accessed through the web-based GUI or REST API to ensure that both human users and automated processes can interact with the application easily. Integration of new molecular generators and other features is facilitated by a Python API and GUI customization is possible via custom components implemented with the React.js JavaScript library. To demonstrate the features of the GenUI framework, our recently published molecular generator DrugEx [74] was integrated within the GenUI ecosystem. The source code of the GenUI platform is distributed under the MIT open-source license [75-77] and several Docker [78-80] images are also available online for quick deployment [81].

Implementation

Software Architecture

User interaction with GenUI happens through the frontend web client which issues REST API calls to the backend, which comprises five services (**Figure 2**). However, advanced users may also implement clients and automated processes that use the REST API directly.

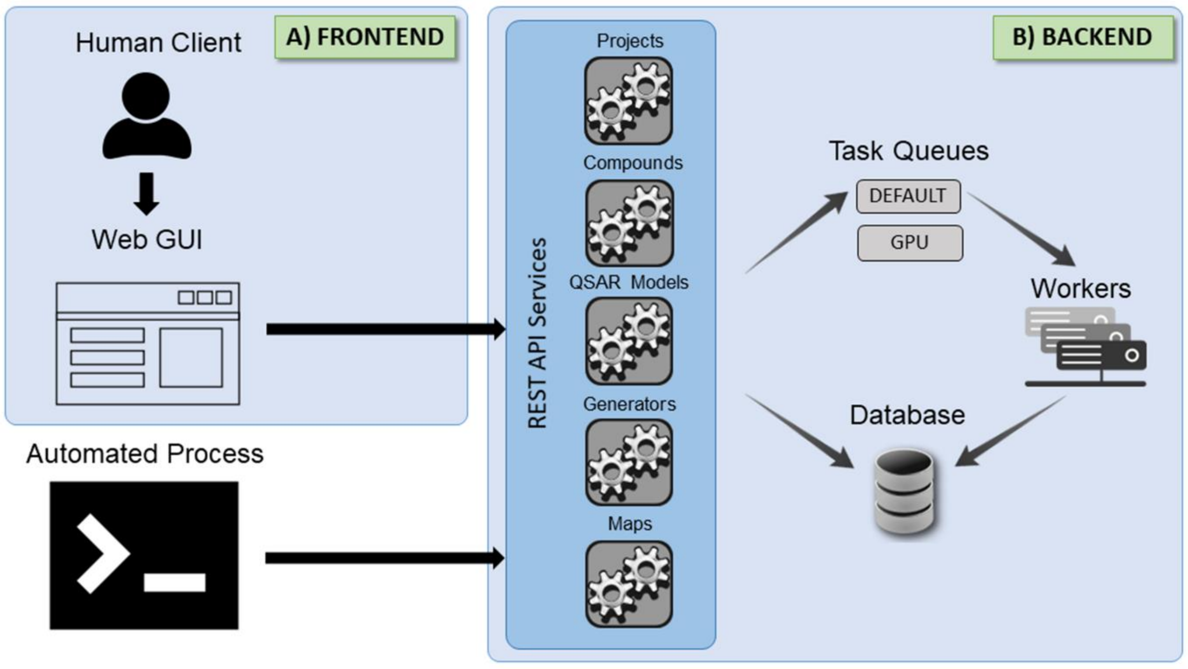


Figure 2 Schematic depiction of the GenUI platform. On the frontend (A), users interact with the web-based GUI to access the backend server services (B). All actions and data exchange are facilitated through REST API calls so that any automated process can also interact with GenUI. The backend application comprises five REST API services each of which has access to the data storage and task queue subsystems. The services can issue computationally intensive and long-running asynchronous tasks to backend workers to ensure sufficient responsiveness and scalability. In the current implementation, tasks can be submitted to two queues: (1) the default CPU queue, which handles all tasks by default, or (2) the GPU queue, intended for tasks that can be accelerated by the use of GPUs.

The five backend services form the core parts of GenUI and can be described as follows:

1. “Projects” service handles user account management, authorization, and workflows. It is used to log users in and organize their work into projects.
2. “Compounds” service manages the compound database including deposition, standardization, and retrieval of molecules and the associated data (i.e. bioactivities, physicochemical properties, or chemical identifiers).
3. “QSAR Models” service facilitates the training and use of QSAR models. They can be used to predict biological activities of the generated compounds, but they are also integral to training of many molecular generators.
4. “Generators” service is responsible for the integration of *de novo* molecular generators. It is meant to be used to set up and train generative algorithms whether they are based on traditional approaches or deep learning.
5. “Maps” service enables the creation of 2D chemical space visualizations and integration of dimensionality reduction algorithms.

In the following sections, the design and implementation of each part of the GenUI platform will be described in more detail.

Frontend

Graphical User Interface (GUI)

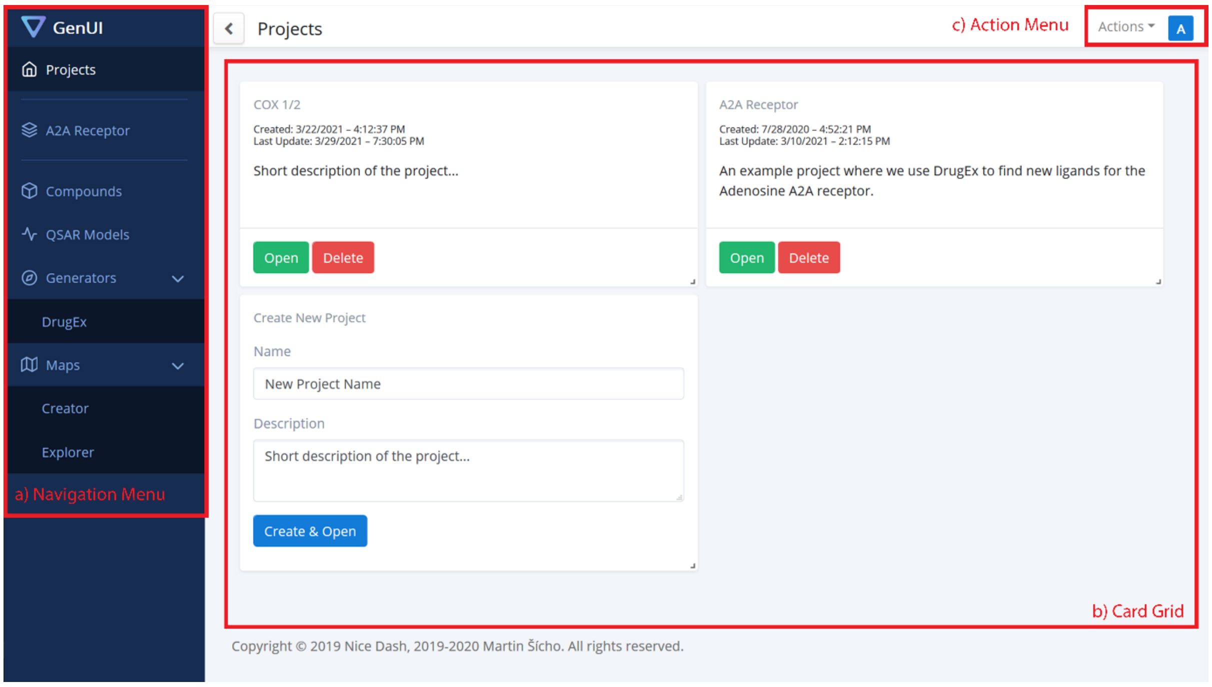
The GUI is implemented as a JavaScript application built on top of the React.js [82] web framework. The majority of graphical components is provided by the Vibe Dashboard open-source project [83], but the original collection of Vibe components was considerably expanded with custom components to fetch, send, and display data exchanged with the GenUI backend

REST API. In addition, frameworks Plotly.js [84], Charts.js [85] and ChemSpace.js [86] are used to provide helpful interactive figures.

The GUI reflects the structure of the GenUI backend services (**Figure 2** and **Figure 3**). Each backend service (Projects, Compounds, QSAR, Generators, and Maps) is represented as a separate item in the navigation menu on the left side of the interface (**Figure 3**a). Upon clicking a menu item, the corresponding page opens rendering a grid of cards (**Figure 3**b) that represent the objects corresponding to the selected backend service. Various actions related to the particular service can be performed from the action menu in the top right of the interface (**Figure 3**c).

Projects

The "Projects" interface serves as a simple way to organize user workflows. For example, a project can encapsulate a workflow for the generation of novel ligands for one protein target (**Figure 3**). Each project contains imported compounds, QSAR models, molecular generators and chemical space maps. The number of projects per user is not limited and they can be deleted or created as needed.



COX 1/2
Created: 3/22/2021 – 4:12:37 PM
Last Update: 3/29/2021 – 7:30:05 PM
Short description of the project...

A2A Receptor
Created: 7/28/2020 – 4:52:21 PM
Last Update: 3/10/2021 – 2:12:15 PM
An example project where we use DrugEx to find new ligands for the Adenosine A2A receptor.

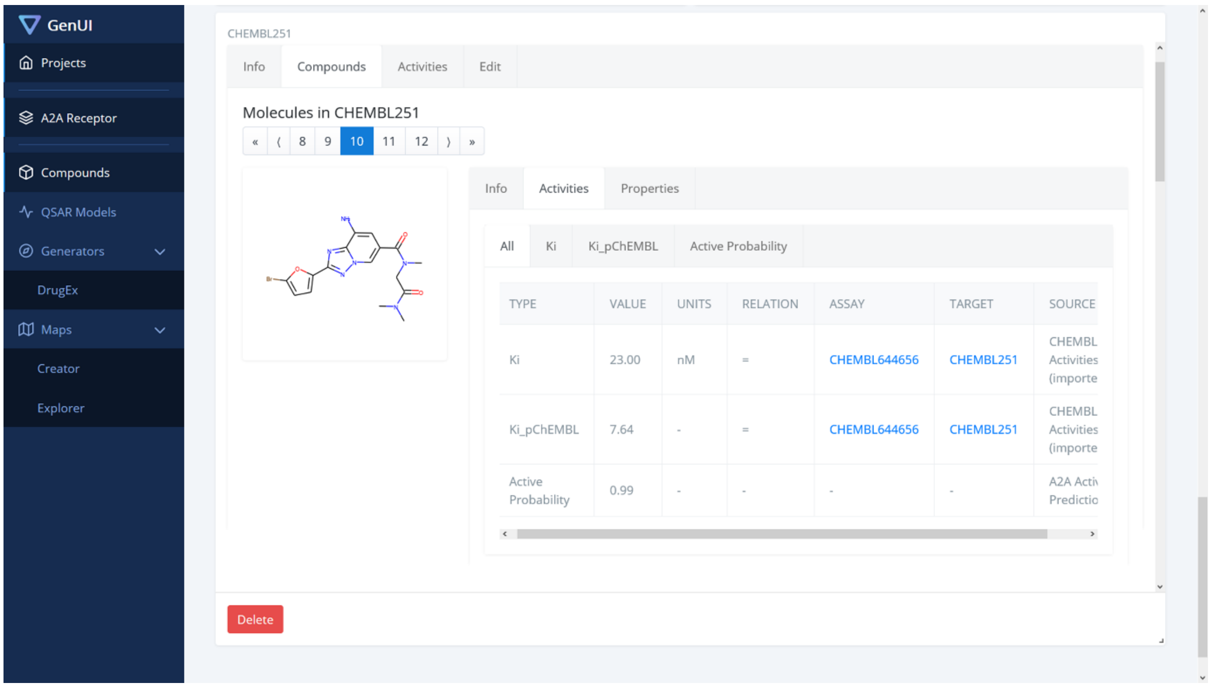
New Project Name
Short description of the project...

Copyright © 2019 Nice Dash, 2019-2020 Martin Šícho. All rights reserved.

Figure 3 A screenshot showing part of the GenUI web GUI. In the figure, the GUI is in a state where the "A2A Receptor" project is already open so the menu on the left can be used to access its data. The GUI consists of three main parts: a) navigation menu, b) card grid and c) action menu. The navigation menu is used to browse data associated with various GenUI services ("Projects" in this case). If a link is clicked in the navigation menu, the data of the selected service is displayed as a grid of interactive cards. Each card allows the users to manage particular data items (a project in this case). The action menu in the top right is also updated depending on the service selected in the navigation menu and performs actions not related to a particular data item. In this case, the action menu was used to bring up the project creation form on the bottom left of the card grid.

Compounds

Each project may contain any number of compound sets (**Figure 4**). Each set of compounds can have a different purpose in the project and come from a different source. Therefore, the contents of each card on the card grid depend on the type of compound set the card represents. Compounds can be generated by generators, but also imported from SDF files, CSV files or obtained directly from the ChEMBL database [6, 7]. New import filters can be easily added by extending the Python backend and customizing the components of the React API accordingly (see Python API and JavaScript API). For each compound in the compound set the interface can display its 2D representation (**Figure 4**), molecular identifiers (i.e. SMILES, InChI, and InChIKey), reported and predicted activities (**Figure 4**) and physicochemical properties (i.e. molecular weight, number of heavy atoms, number of aromatic rings, hydrogen bond donors, hydrogen bond acceptors, logP and topological polar surface area).

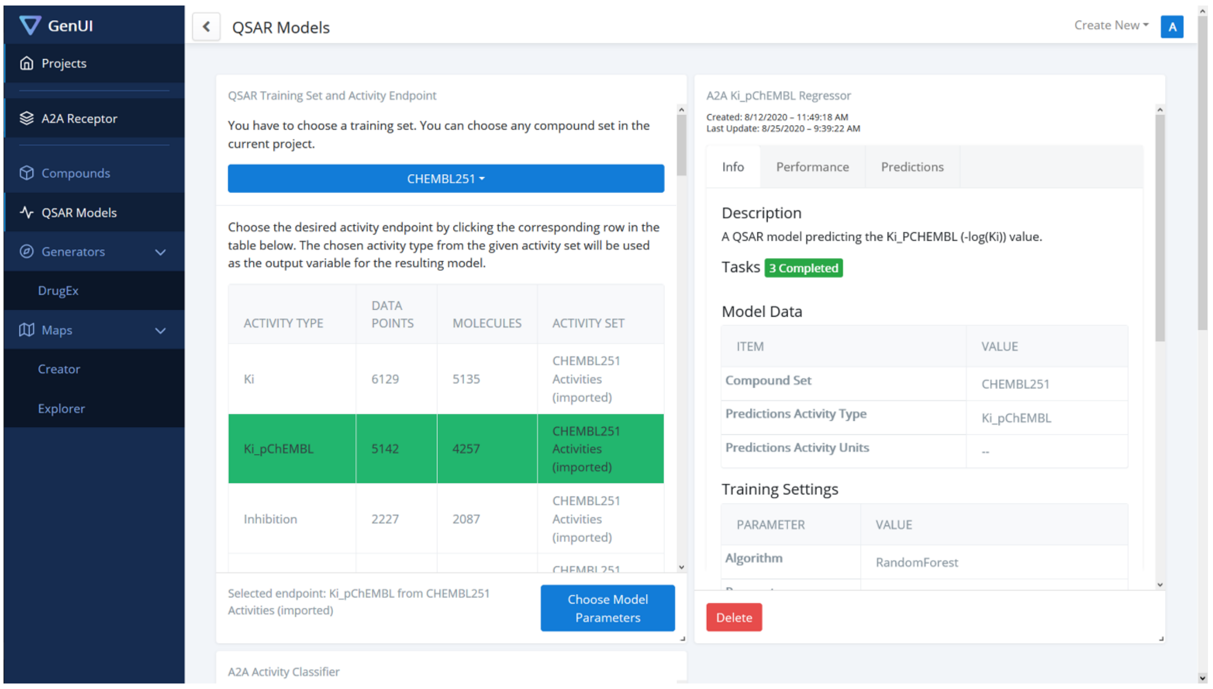


TYPE	VALUE	UNITS	RELATION	ASSAY	TARGET	SOURCE
Ki	23.00	nM	=	CHEMBL644656	CHEMBL251	CHEMBL Activities (importe
Ki_pChEMBL	7.64	-	=	CHEMBL644656	CHEMBL251	CHEMBL Activities (importe
Active Probability	0.99	-	-	-	-	A2A Acti Predictio

Figure 4 A screenshot showing part of the “Compounds” GUI. In this page, users can import data sets from various sources. A card representing an already imported data set from the ChEMBL database [7] is shown. The position and size of each displayed card can be modified by either dragging the card (reposition) or adjusting the bottom right corner (size change). The card shown is currently expanded over two rows of the card grid (**Figure 3**b) in order to accommodate the displayed data better. The “Activities” tab in the compound overview shows summary of the biological activity data associated with the compound. The activities are grouped by type and aside from experimentally determined activities the interface also displays activity predictions of available QSAR models. For example, in the view shown the “Active Probability” activity type is used to denote the output probability from a classification QSAR model. Each activity value also contains information about its origin (the “Source” column) so that it can be tracked back to its source.

QSAR Models

All QSAR models trained or imported in the given project are available from the “QSAR Models” page (**Figure 5**, **Figure 6**). Each QSAR model is represented by a card with several tabs. The “Info” tab contains model metadata, as well as a serialized model file to download (**Figure 5**). The “Performance” tab lists various performance measures of the QSAR model obtained by cross-validation or on an independent hold out test set (**Figure 6**). The validation procedure can be adjusted by the user during model creation (**Figure 5**). Making predictions with the model is possible under the “Predictions” tab. Each QSAR model can be used to make predictions for any compound set listed on the “Compounds” page and the calculated predictions will then become visible in that interface as well (**Figure 4**).



QSAR Training Set and Activity Endpoint

You have to choose a training set. You can choose any compound set in the current project.

Choose the desired activity endpoint by clicking the corresponding row in the table below. The chosen activity type from the given activity set will be used as the output variable for the resulting model.

ACTIVITY TYPE	DATA POINTS	MOLECULES	ACTIVITY SET
Ki	6129	5135	CHEMBL251 Activities (imported)
Ki_pChEMBL	5142	4257	CHEMBL251 Activities (imported)
Inhibition	2227	2087	CHEMBL251 Activities (imported)

Selected endpoint: Ki_pChEMBL from CHEMBL251 Activities (imported)

A2A Activity Classifier

A2A Ki_pChEMBL Regressor

Created: 8/12/2020 - 11:49:18 AM
Last Update: 8/25/2020 - 9:39:22 AM

Description

A QSAR model predicting the Ki_PCHEMBL (-log(Ki)) value.

Tasks 3 Completed

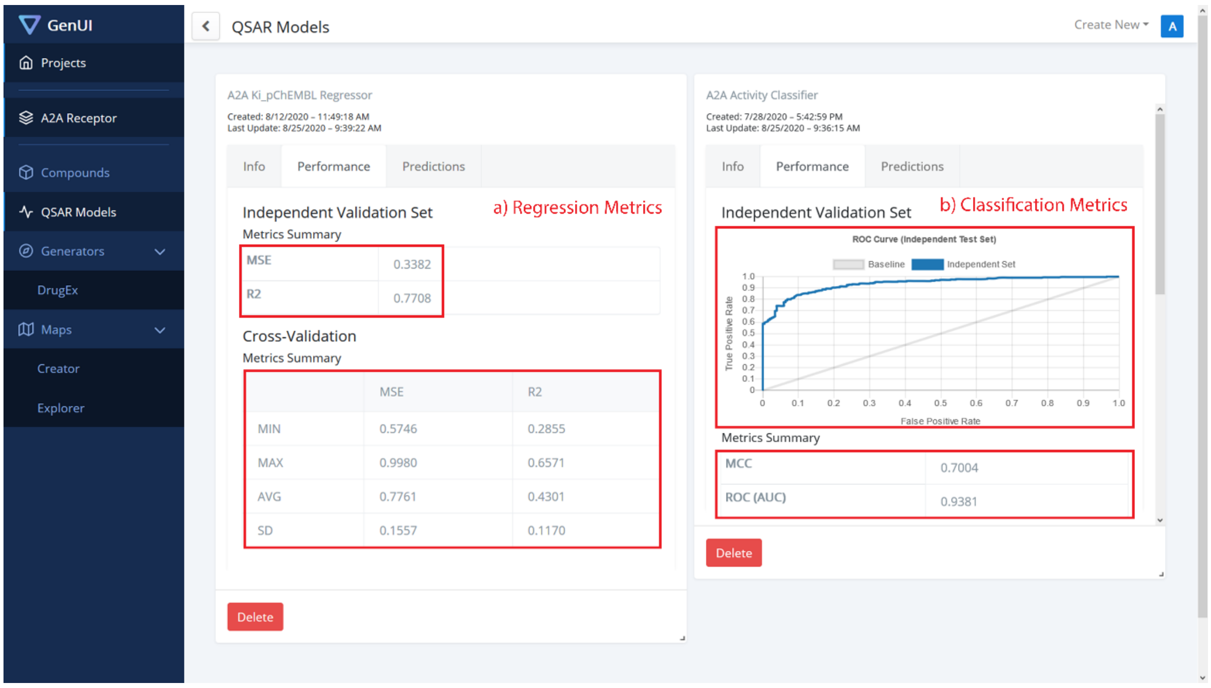
Model Data

ITEM	VALUE
Compound Set	CHEMBL251
Predictions Activity Type	Ki_pChEMBL
Predictions Activity Units	--

Training Settings

PARAMETER	VALUE
Algorithm	RandomForest

Figure 5 A screenshot showing part of the “QSAR Models” GUI. The card on the left side of the screen shows how training data is chosen for a new model while the card on the right shows metadata about an already trained model.



A2A Ki_pChEMBL Regressor
Created: 8/12/2020 – 11:49:18 AM
Last Update: 8/25/2020 – 9:39:22 AM

Independent Validation Set
Metrics Summary

MSE	0.3382
R2	0.7708

Cross-Validation
Metrics Summary

	MSE	R2
MIN	0.5746	0.2855
MAX	0.9980	0.6571
AVG	0.7761	0.4301
SD	0.1557	0.1170

A2A Activity Classifier
Created: 7/28/2020 – 5:42:59 PM
Last Update: 8/25/2020 – 9:36:15 AM

Independent Validation Set

Metrics Summary

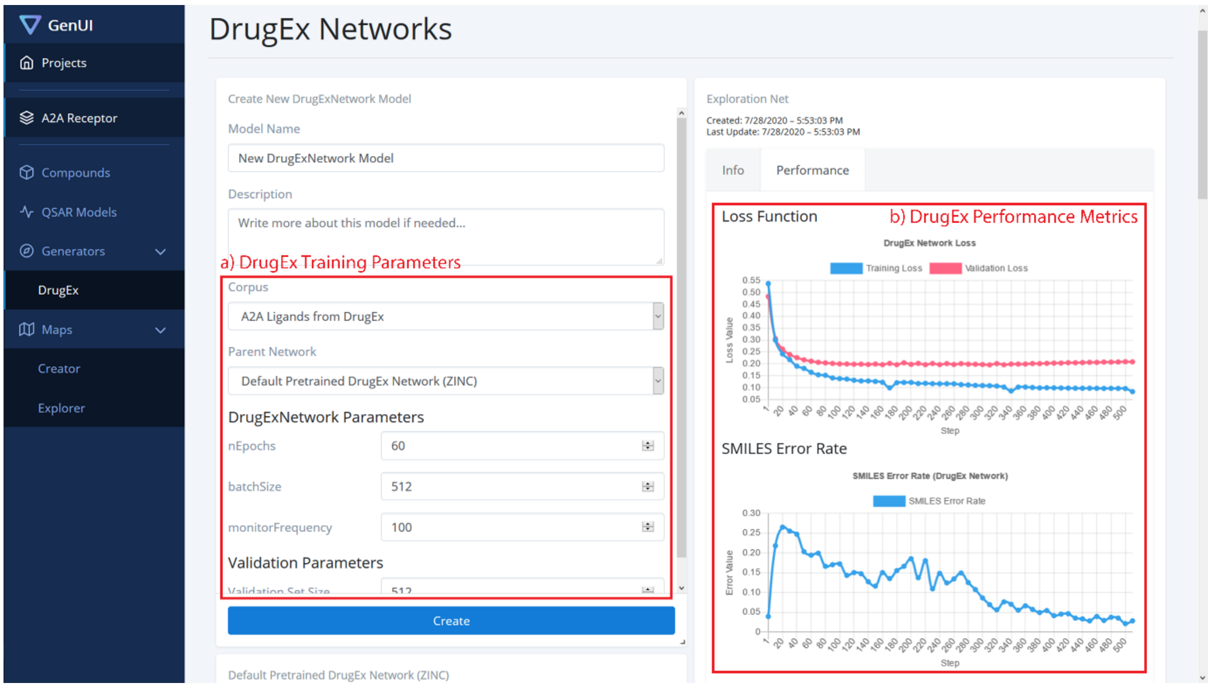
MCC	0.7004
ROC (AUC)	0.9381

Figure 6 Performance evaluation view for a) regression and b) classification QSAR model. In a) the mean-squared error (MSE) and the coefficient of determination (R2) are used as validation metrics. In b) the performance is measured on a hold out independent validation test set with the Matthews correlation coefficient (MCC) and the area under the receiver operating characteristic (ROC) curve (AUC). The ROC curve itself is also displayed above the metrics.

New QSAR models are submitted for training with a creation card (**Figure 5**) that helps users choose model hyperparameters and a suitable training strategy (i.e. the characteristics of the independent hold out validation set, the number of cross-validation folds or the choice of validation metrics). The "Info" tab of a trained model contains important metadata as well as a hyperlink to export the model and save it as a reusable Python object. This import/export feature enables users to archive and share their work, enhancing the reusability and reproducibility of the developed models [87]. The "Performance" tab can be used to observe model performance data according to the chosen validation scheme (**Figure 6**). This information is different depending on the chosen model type (regression vs. classification, **Figure 6**a vs. **Figure 6**b) and the parameters used (i.e. the choice of validation metrics). Additional performance measures and machine learning algorithms can be integrated with the backend Python API. Creation of such extensions does not even require editing of the GUI for many standard algorithms (see Python API).

Generators

Under the “Generators” menu item, the users find a list of individual generators implemented in the GenUI framework (**Figure 7**). Currently, only the DrugEx generator [74] is available, but other generators can be added easily by extending the Python backend and customizing the existing React components. In fact, the GUI for DrugEx is based on the same React components as the “QSAR Models” view.



DrugEx Networks

Create New DrugExNetwork Model

Model Name: New DrugExNetwork Model

Description: Write more about this model if needed...

Corpus: A2A Ligands from DrugEx

Parent Network: Default Pretrained DrugEx Network (ZINC)

DrugExNetwork Parameters

nEpochs	60
batchSize	512
monitorFrequency	100

Validation Parameters

Validation Set Size: 512

Default Pretrained DrugEx Network (ZINC)

Exploration Net

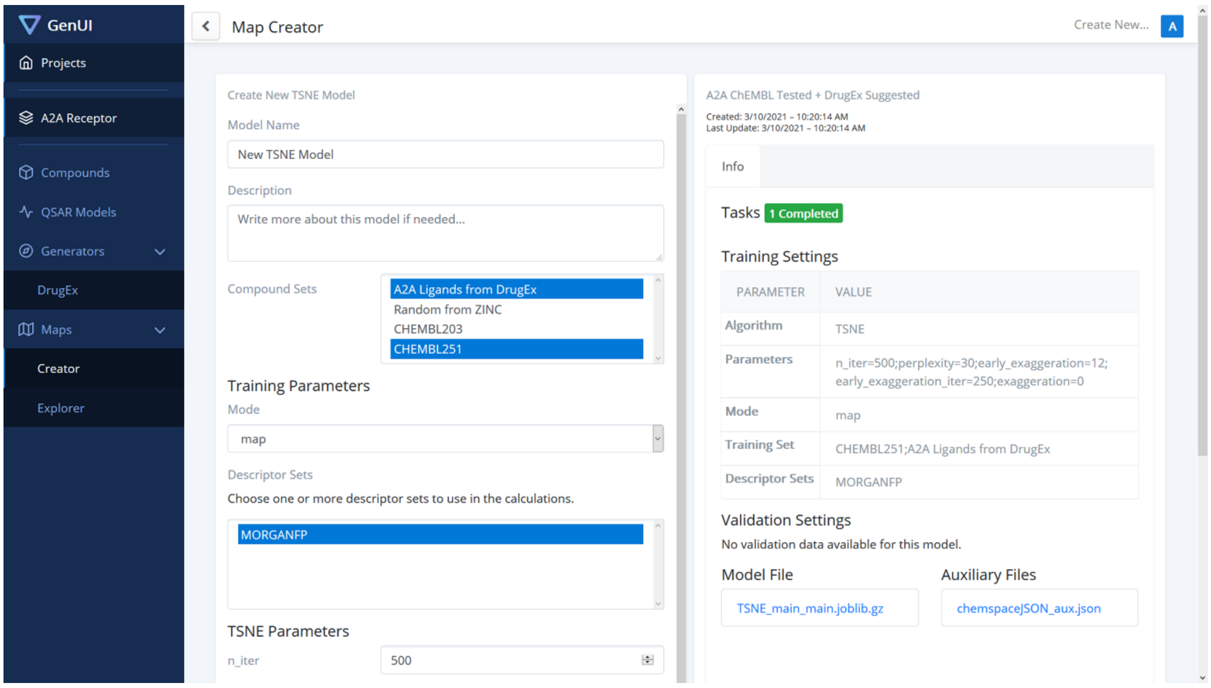
Created: 7/28/2020 - 5:53:03 PM
Last Update: 7/28/2020 - 5:53:03 PM

Figure 7 A screenshot showing part of the “DrugEx” GUI with a model creation card with a) DrugEx training parameters and b) performance overview of a trained DrugEx network. In a) the fields to define the compound set for the process of fine-tuning the ‘parent’ recurrent neural network trained on the ZINC data set [74] are shown. In addition, the form provides fields to set the number of learning epochs, training batch size, frequency of performance monitoring and size of the validation set. In b) the “Performance” tab tracks model performance. It shows values of the loss function on the training set and validation set (top) and the SMILES error rate (bottom) at each step of the training process. The performance view is updated according to the chosen monitoring frequency in real time as the model is being trained. Each model also has the “Info” tab which holds the same information as for QSAR models.

Like QSAR models, DrugEx networks can also be serialized and saved as files. For example, a cheminformatics researcher can build a DrugEx model outside of the GenUI ecosystem (i.e. using a script published with the original paper [74]) and provide the created model files to another researcher who can import and use the model from the GenUI web-based GUI. Therefore, it is easy to share work and accommodate various groups of users in this way.

Maps

Interactive visualization of chemical space is available under the “Maps” menu item. The menu separates the creation of the chemical space visualization, the “Creator” page (**Figure 8**), and its exploration, the “Explorer” page (**Figure 9**).



Create New TSNE Model

Model Name: New TSNE Model

Description: Write more about this model if needed...

Compound Sets: A2A Ligands from DrugEx (selected), Random from ZINC, CHEMBL203, CHEMBL251 (selected)

Training Parameters

Mode: map

Descriptor Sets

Choose one or more descriptor sets to use in the calculations.

MORGANFP (selected)

TSNE Parameters

n_iter: 500

A2A ChEMBL Tested + DrugEx Suggested

Created: 3/10/2021 – 10:20:14 AM
Last Update: 3/10/2021 – 10:20:14 AM

Info

Tasks 1 Completed

Training Settings

PARAMETER	VALUE
Algorithm	TSNE
Parameters	n_iter=500;perplexity=30;early_exaggeration=12; early_exaggeration_iter=250;exaggeration=0
Mode	map
Training Set	CHEMBL251;A2A Ligands from DrugEx
Descriptor Sets	MORGANFP

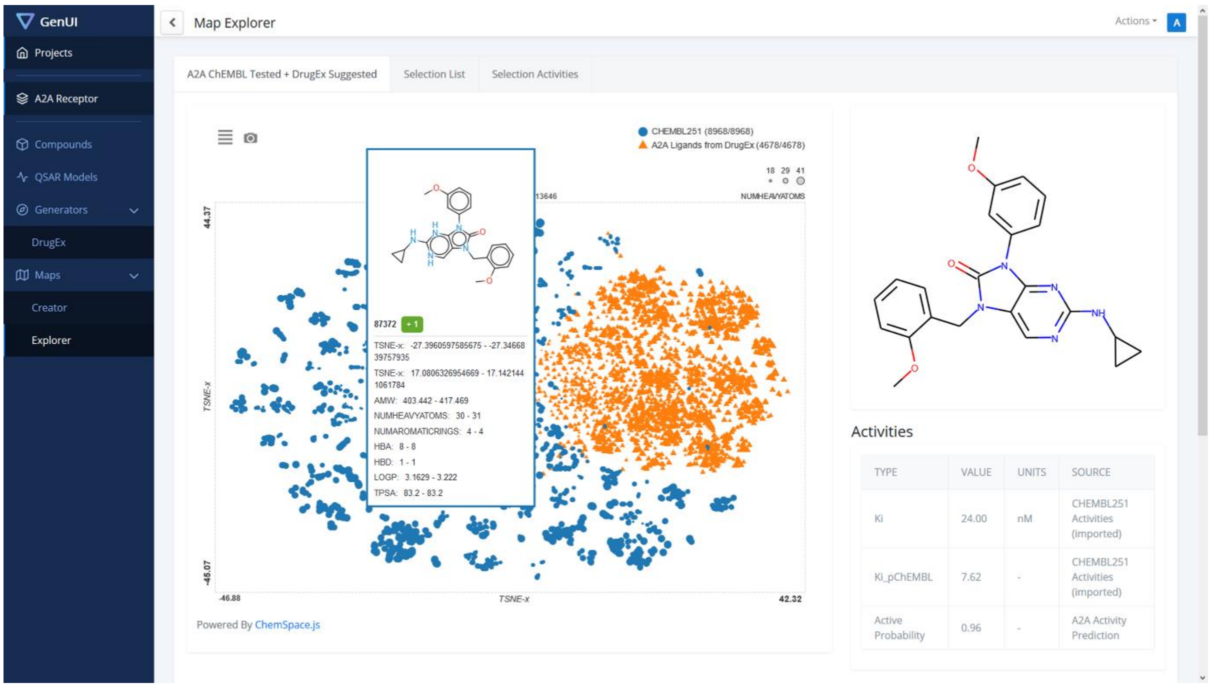
Validation Settings

No validation data available for this model.

Model File: TSNE_main_main.joblib.gz

Auxiliary Files: chemspaceJSON_aux.json

Figure 8 The “Creator” interface of GenUI “Maps” page. On the left a form to create a new t-SNE [88] mapping of two sets of compounds using Morgan fingerprints is shown while information about an existing map can be seen on the right.



87372 +1

TSNE-x: -27.3960597585675 - -27.3466839757935

TSNE-x: 17.0806326954669 - 17.1421441061784

AMW: 403.442 - 417.469

NUMHEAVYATOMS: 30 - 31

NUMAROMATICRINGS: 4 - 4

HBA: 8 - 8

HBD: 1 - 1

LOGP: 3.1629 - 3.222

TPSA: 83.2 - 83.2

Activities

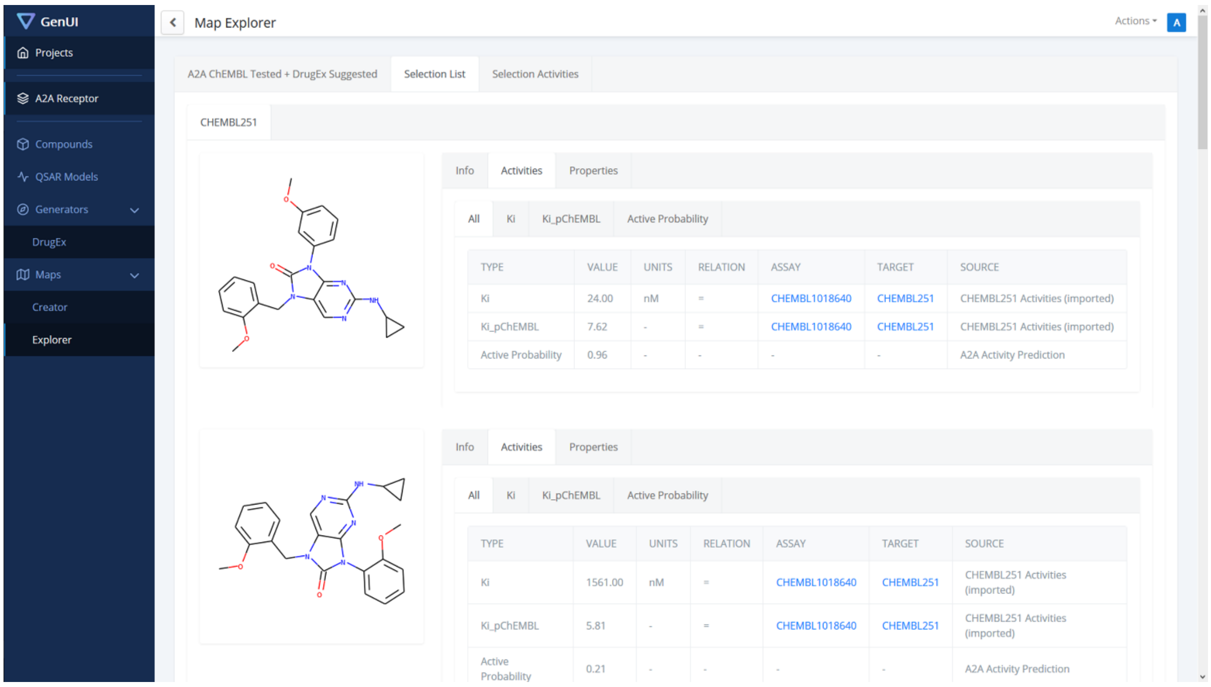
TYPE	VALUE	UNITS	SOURCE
Ki	24.00	nM	CHEMBL251 Activities (imported)
Ki_pChEMBL	7.62	-	CHEMBL251 Activities (imported)
Active Probability	0.96	-	A2A Activity Prediction

Figure 9 A screenshot showing the “Explorer” part of the “Maps” GUI. The interactive plot on the left side of the screen is provided by the ChemSpace.js library [86]. Each point in this visualization corresponds to one molecule.

In this particular configuration, the shapes and colors of the points indicate the compound set to which the compounds belong to. The color scheme of points can be changed with the menu in the top left corner of the plot. It is possible to color points by biological activities, physicochemical properties and other data associated with the compounds. The same can also be done with the size of the points. The points drawn in the map are interactive and hovering over a point shows a box with information about the compound inside and on the right side of the map. Groups of points can also be selected by drawing a rectangle over them in which case a list of selected compounds is shown in the "Selection List" tab (**Figure 10**) and their bioactivity data is summarized under the "Selection Activities" tab (**Figure 11**).

The "Creator" page is implemented as a grid of cards each of which represents an embedding of chemical compounds in 2D space (**Figure 8**). Implicitly, the GenUI platform enables t-SNE [88] embedding (provided by openTSNE [89]). However, new projection methods can be easily added to the backend through the GenUI Python API with no need to modify the GUI (see Python API) [90].

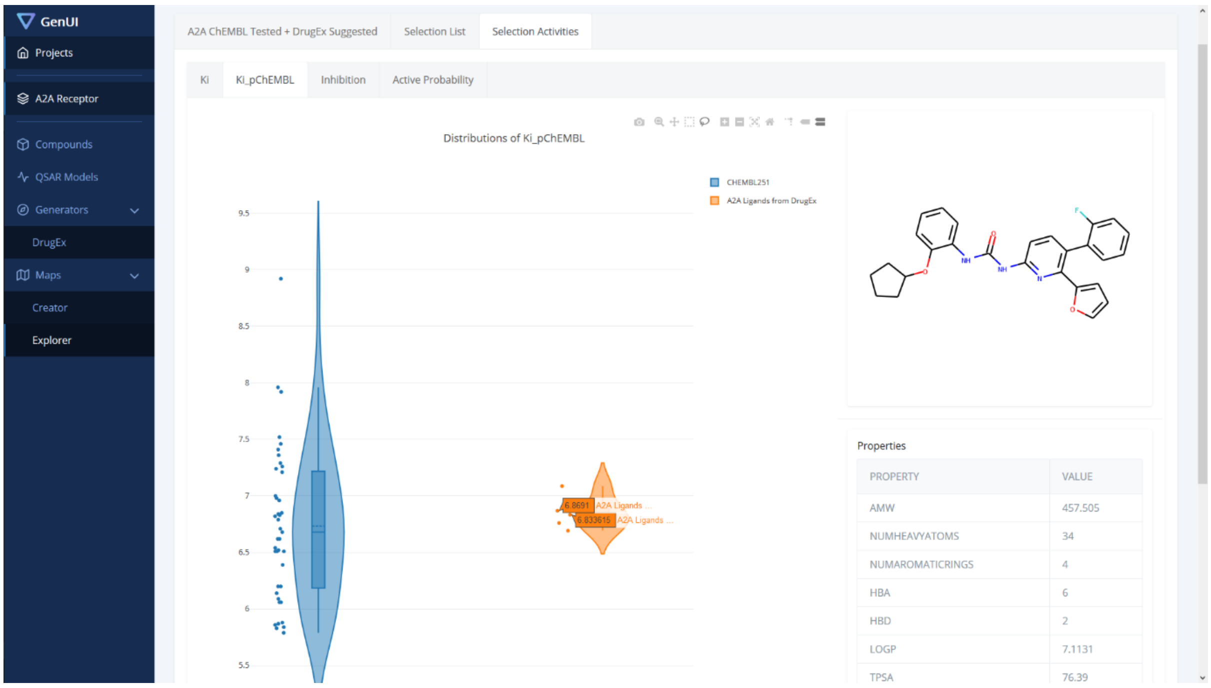
The purpose of the "Explorer" page is to interactively visualize chemical space embedding prepared in the "Creator" (**Figure 9**). In the created visualization the users can explore compound bioactivities, physicochemical properties, and other measurements for various representations and parts of chemical space. Thanks to ChemSpace.js [86] up to 5 dimensions can be shown in the map at the same time: X and Y coordinates, point color, point size and point shape. The map can be zoomed in by drawing a rectangle over a group of points. Such points form a selection and their detailed information is then displayed under the "Selected List" (**Figure 10**) and "Selected Activities" tabs (**Figure 11**).



TYPE	VALUE	UNITS	RELATION	ASSAY	TARGET	SOURCE
Ki	24.00	nM	=	CHEMBL1018640	CHEMBL251	CHEMBL251 Activities (imported)
Ki_pChEMBL	7.62	-	=	CHEMBL1018640	CHEMBL251	CHEMBL251 Activities (imported)
Active Probability	0.96	-	-	-	-	A2A Activity Prediction

TYPE	VALUE	UNITS	RELATION	ASSAY	TARGET	SOURCE
Ki	1561.00	nM	=	CHEMBL1018640	CHEMBL251	CHEMBL251 Activities (imported)
Ki_pChEMBL	5.81	-	=	CHEMBL1018640	CHEMBL251	CHEMBL251 Activities (imported)
Active Probability	0.21	-	-	-	-	A2A Activity Prediction

Figure 10 View of the "Selected List" tab of the "Explorer" page. The tab shows the selected molecules in the map as a list which is the same as the one used in the "Compounds" view (**Figure 4**). For easier navigation, the compounds are also grouped by the compound set they belong to and the view for each set can be accessed by switching tabs above the displayed list (only one compound set, CHEMBL251, is present in this case).



PROPERTY	VALUE
AMW	457.505
NUMHEAVYATOMS	34
NUMAROMATICRINGS	4
HBA	6
HBD	2
LOGP	7.1131
TPSA	76.39

Figure 11 View of the "Selection Activities" tab of the "Explorer" page. In this view, violin plots representing distributions of activities in the set of selected compounds are displayed. Each violin plot corresponds to one compound set and one activity type. The violin plots are also interactive and hovering over points updates the compound structure and its physicochemical properties are displayed on the right.

JavaScript API

Two main considerations in the development of GenUI are reusability and extensibility. Therefore, the frontend GUI comprises a large library of over 50 React components that are encapsulated in a standalone package (**Figure 12**). The package is organized into subpackages that follow the structure and hierarchy of design elements in the GenUI interface. In the following sections, we use the two most important groups of the React API components as case studies to illustrate how the frontend GUI can be extended. The presented components are "Model Components", used to add new trainable models, and the "REST API Components", used to fetch and send data between the frontend and the GenUI REST API services.

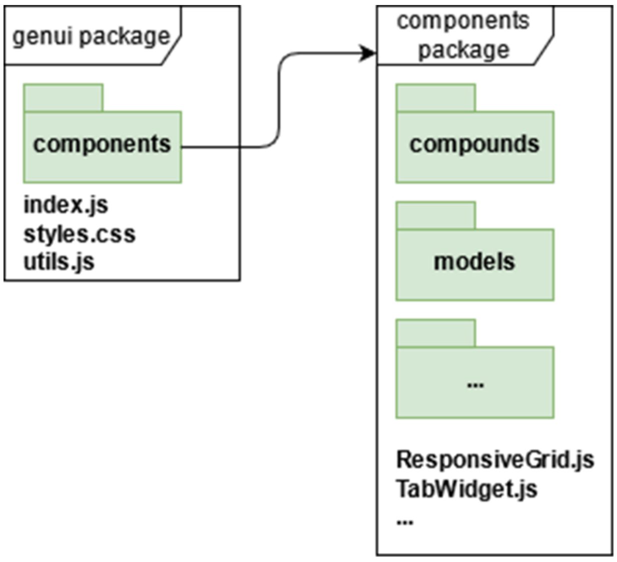


Figure 12 Schematic depiction of the GenUI React library which contains customized styles, utility functions and the components used in the GenUI web client. The "components" subpackage organizes the components into groups related to the structure of the GenUI interface. For example, components filed under the "models" subpackage are used in the creation of the "QSAR Models" (Figure 5), "DrugEx" (Figure 7) and "Maps" (Figure 8) interfaces while components under the "compounds" subpackage are used to implement the "Compounds" view (Figure 4). General purpose components (i.e. the card grid or the card tab widget) are in the root of the "components" subpackage.

Model Components

Much of the functionality of the GenUI platform is based on trained models. The "QSAR Models", "DrugEx" and "Maps" pages all borrow from the same library of reusable GenUI React components (**Figure 12**). At the core of the "models" component library (**Figure 12**) is the *ModelsPage* component (**Figure 13**). *ModelsPage* manages the layout and data displayed in model cards. When the users select to build a new model, the *ModelsPage* component is also responsible to show a card with the model creation form. The information that the *ModelsPage* displays can be customized through various React properties (**Figure 13**) that represent either data (data properties) or other components (component properties). Such an encapsulation approach and top-down data flow is one of the main strengths of the React framework. This design is very robust since it fosters appropriate separation of concerns by their encapsulation inside more and more specialized components. This makes the code easy to reuse and maintain.

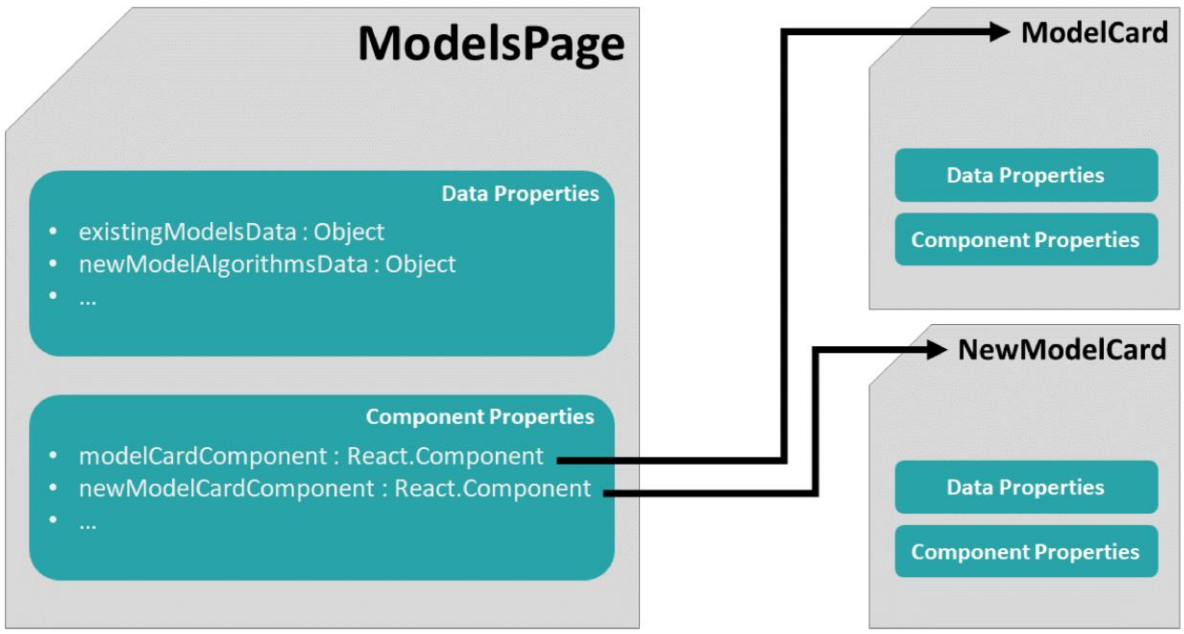


Figure 13 A simplified illustration of the high-level components in the GenUI React API for rendering model cards. The main *ModelsPage* component has two kinds of attributes (called "properties" in React): a) *data properties* and b) *component properties*. The values of data properties are used to display model data while the values of component properties are used as child components and injected into the GUI at appropriate places. If no component property is specified, default components are used as children instead (i.e. *ModelCard* and *NewModelCard*). The child components can accept data and component properties as well from their parent (i.e *ModelsPage*). This creates a hierarchy of reusable components that can be easily assembled and configured to accommodate the different needs of each model view in a standardized and consistent manner.

REST API Components

Because the GUI often needs to fetch data from the backend server, several React components were defined for that purpose. In order to use them, one just needs to provide the required REST API URLs as React component properties. For example, the *ComponentWithResources* component configured with the '/maps/algorithms/' URL will get all available embedding methods as JSON and converts the result to a JavaScript object. Many components can also periodically update the fetched data, which is useful for tracking information in real time. For paginated data there is also the *ApiResourcePaginator* component that only fetches a new page if a given event is fired (i.e. user presses a button).

This makes it convenient to create GUIs for larger data sets. In addition, user credentials are also handled automatically.

Many more specialized components are also available to fetch specific information. For example, the *TaskAwareComponent* tracks URLs associated with background asynchronous tasks and it regularly passes information about completed, running, or failed tasks to its child components. However, other specialized components exist that automatically fetch and format pictures of molecules, bioactivities, physicochemical properties or create, update and delete objects in the UI and the server [76].

Backend

The backend services are the core of the GenUI platform and the GenUI Python API provides a convenient way to write backend extensions (i.e. new molecular generators, compound import filters, machine learning algorithms for QSAR modeling, and dimensionality reduction methods for chemical space maps). All five backend services (**Figure 2**) are implemented with the Django web framework [91] and Django REST Framework [92]. For data storage, a freely available Docker [80] image developed by Informatics Matters Ltd. [93] is used. The Docker image contains an instance of the PostgreSQL database system with integrated database cartridge from the RDKit cheminformatics framework [94]. The integration of RDKit with the Django web framework is handled with the Django RDKit library [95]. All compounds imported in the database are automatically standardized with the current version of the ChEMBL structure curation pipeline [96].

Because the backend services also handle processing of long-running and computationally intensive tasks, the framework uses Celery distributed task queue [97] with Redis as a message broker [98] to dispatch them to workers. Celery workers are processes running in the background that consume tasks from the task queue and process them asynchronously.

Workers can either run on the same machine as the backend services or they can be distributed over an infrastructure of computers (see Deployment).

Python API

The GenUI backend codebase [77] is divided into multiple Python packages that each encapsulate a part of the GenUI Django project (**Figure 14**). Any package that resides in the root directory is referred to as the *root package*. Root packages facilitate many of the REST API endpoints (**Figure 2**), but they also contain reusable classes that are intended to be extended by extensions (see Generic Views and Viewsets, for example). In the following sections, some important features of the backend Python API are briefly highlighted. However, a much more detailed description with code examples is available on the documentation page of the project [90].

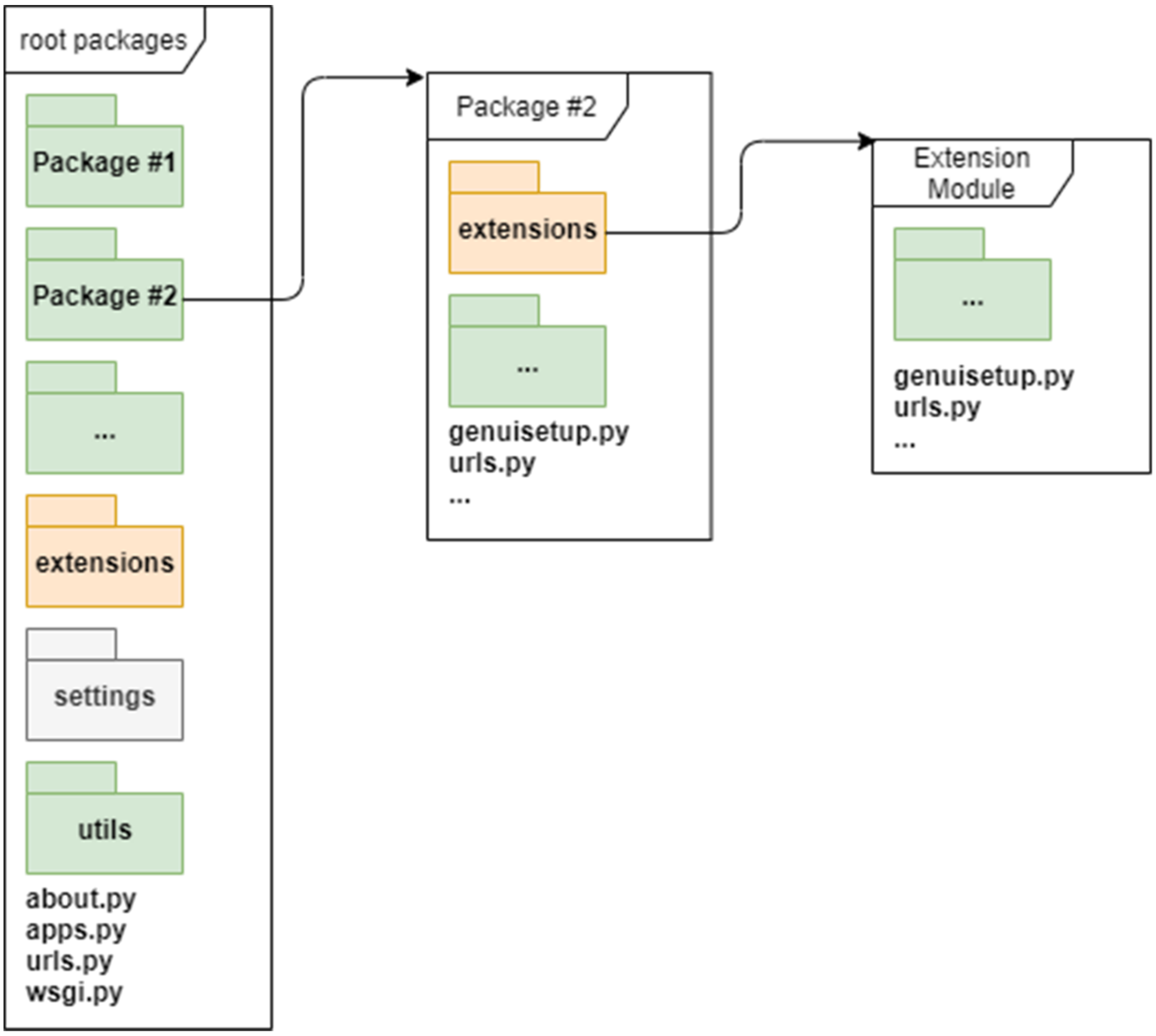


Figure 14 Schematic depiction of the GenUI backend Python code. The backend is formed by a single Django project which is designated by its *settings* package and the *urls* and *wsgi* modules. The GenUI code itself is divided into a number of *root packages*. Each root package has a predefined structure with the code of the package organized in its own modules and packages. Each root package of the GenUI framework also has the *extensions* subpackage, which is a collection of extension modules. GenUI extensions and packages can also define the *genuisetup* module, which is used to automatically configure the individual package or extension.

Extensions

Just like in the case of the GenUI React API, modularity and extensibility were also the main concerns during the design of the GenUI backend services. Each of the aforementioned root packages contains a Python package called *extensions* (**Figure 14**). The *extensions* package can contain any number of Django applications or Python modules, which ensures that the extending components of the GenUI framework are well-organized and loosely coupled.

Provided that GenUI extensions are structured a certain way they can take advantage of automatic configuration and integration (see Automatic Code Discovery). Before the Django project is deployed, GenUI applications and extensions are detected and configured with the *genuisetup* command, which makes sure that the associated REST API endpoints are exposed under the correct URLs. The *genuisetup* command is executed with the *manage.py* script (a utility script provided by the Django library).

Automatic Code Discovery

The root packages of the GenUI backend library define many abstract and generic base classes to implement and reuse in extensions. These classes either implement the REST API or define code to be run on the worker nodes inside Celery tasks. Automatic code discovery uses several introspection functions and methods to find the derived classes of the base classes found in the root packages. By default, this is done when the *genuisetup* command is executed (see Extensions).

For example, if the derived class defines a new machine learning algorithm to be used in QSAR modelling, automatic code discovery utilities make sure that the new algorithm appears as a choice in the QSAR modelling REST API and that proper parameter values are collected via the endpoint to create the model. Moreover, all changes also get automatically propagated to the web-based GUI because it uses the REST API to obtain algorithm choices for the model creation form. Thus, no JavaScript code has to be written to integrate a new machine learning algorithm. These mechanisms are also used when adding molecular generators, dimensionality reduction methods, or molecular descriptors.

Generic Views and Viewsets

When developing REST API services with the Django REST Framework, a common practice is using generic views and sets of views (called viewsets). In Django applications, views are functions or classes that handle incoming HTTP requests. Viewsets are classes defined by the Django REST Framework that bring functionality of several views (such as creation, update or deletion of objects) into one single class. Generic views and viewsets are then classes that usually do not stand on their own, but are designed to be further extended and customized.

The GenUI Python library embraces this philosophy and many REST API endpoints are encapsulated in generic views or viewsets. This ensures that the functionality can be reused and that no code needs to be written twice, as stated by the well-known DRY (“Don’t Repeat Yourself”) principle [99]. An example of such a generic approach is the *ModelViewSet* class that handles the endpoints for retrieval and training of machine learning models. This viewset is used by the *qsar* and *maps* applications, but also by the DrugEx extension. All these applications depend on some form of a machine learning model so they can take advantage of this interface, which automatically checks the validity of user inputs and sends model training jobs to the task queue.

Asynchronous Tasks

Many of the GenUI backend services take advantage of asynchronous tasks which are functions executed in the background without blocking the main application. Moreover, tasks do not even have to be executed on the same machine as the caller of the task, which allows for a great deal of flexibility and scalability (see Deployment).

The Celery task queue [97] makes creating asynchronous tasks as easy as defining a Python function [100]. In addition, some GenUI views already define their own tasks and no explicit task definition is needed in the derived views of the extensions. For example, the *compounds* root package defines a generic viewset that can be used to create and manage compound sets. The import and creation of compounds belonging to a new compound set is handled by implementing a separate initializer class, which is passed to the appropriate generic viewset class [90]. The initialization of a compound set can take a long time or may fail and, thus, should be executed asynchronously. Therefore, the viewset of the *compounds* application automatically executes the methods of the initializer class asynchronously with the help of an available Celery worker.

Deployment

Docker Images

Since the GenUI platform consists of several components with many dependencies and spans multiple programming languages, it can be tedious to set up the whole project on a new system. Docker makes deployment of larger projects like this easier by encapsulating different parts of the deployment environment inside Docker images [78-80]. Docker images are simply downloaded and deployed on the target system without the need to install any other tools beside Docker. GenUI uses many official Docker images available on the Docker image sharing platform Docker Hub [101]. The PostgreSQL database with built-in RDKit cartridge [93], Redis [102] and the NGINX web server [103, 104] are all obtained by this standard channel. In addition, we defined the following images to support the deployment of the GenUI platform itself [81]:

1. *genui-main*: Used to deploy both the frontend web application and the backend services.
2. *genui-worker*: Deploys a basic Celery worker without GPU support.
3. *genui-gpuworker*: Deploys a Celery worker with GPU support. It is the same as the *genui-worker,* but it has the NVIDIA CUDA Toolkit already installed.

The tools to build these images are freely available [81]. Therefore, developers can create images for extended versions of the GenUI that fit the needs of their organizations. In addition, the separation of the main application (*genui-main*) from workers also allows distributed deployment over multiple machines, which opens up the possibility to create a scalable architecture that can quickly accommodate teams of varying sizes.

Future Directions

Although the GenUI framework already implements much of the functionality needed to successfully integrate most molecular generators, there are still many aspects of the framework that can be improved. For instance, it would be beneficial if more sources of molecular structures and bioactivity information are integrated in the platform besides ChEMBL (i.e PubChem [105], ZINC [106], DrugBank [107], BindingDB [108] or Probes and Drugs [109]). Currently, GenUI also lacks features to perform effective similarity and substructure searches, which we see as a crucial next step to improve the appeal of the platform to medicinal chemists. The current version of GenUI would also benefit from extending the sets of descriptors, QSAR machine learning algorithms and chemical space projections since the performance of different methods can vary across data sets. Finally, the question of synthesizability of the generated structures should also be addressed and a system for predicting chemical reactions and retrosynthetic pathways could also be very useful to medicinal chemists if integrated in the GUI (i.e. by facilitating connection to a service such as the IBM RXN [110] or PostEra Manifold [111]).

Even though it is hard to determine the requirements of every project where molecular generators might be applied, many of the aforementioned features and improvements can be readily implemented with the GenUI React components (see JavaScript API) and the Python API (see Python API). In fact, the already presented extensions and the DrugEx interface are useful case studies that can be used as templates for integration of many other cheminformatics tools and *de novo* molecular generators. Therefore, we see GenUI as a flexible and scalable framework that can be used by organizations to quickly integrate tools and data the way it suits their needs the most. However, we would also like GenUI to become a new useful way to share the progress in the development of novel *de novo* drug design methods and other cheminformatics approaches in the public domain.

Conclusions

We implemented a full stack solution for integration of *de novo* molecular generation techniques in a multidisciplinary work environment. The proposed GenUI software platform provides a GUI designed to be easily understood by experts outside the cheminformatics domain, but it also offers a feature-rich REST API for programmatic access and straightforward integration with automated processes. The presented solution also provides extensive Python and JavaScript extension APIs for easy integration of new molecular generators and other cheminformatics tools. We envision that the field of molecular generation will likely expand in the future and that an open source software platform such as this one is a crucial step towards more widespread adoption of novel algorithms in drug discovery and related research. We also believe that GenUI can facilitate more engagement between different groups of users and inspire new directions in the field of *de novo* drug design.

Declarations

Authors' Contributions

GvW suggested the original idea of developing a graphical user interface for a molecular generator and supervised the study along with DS. MŠ extended the original idea and developed all software presented in this work. XL is the author of DrugEx and helped with its integration as a proof of concept. MŠ and XL also prepared the manuscript, which all authors proofread and agreed on.

Acknowledgements

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

XL thanks Chinese Scholarship Council (CSC) for funding.

Competing Interests

The authors declare that they have no competing interests.

Funding

This work was supported from the Ministry of Education of the Czech Republic (LM2018130).

Availability of Data and Materials

The complete GenUI codebase and documentation is distributed under the MIT license and located in three repositories publicly accessible on GitHub:

- https://github.com/martin-sicho/genui (backend Python code)
- https://github.com/martin-sicho/genui-gui (frontend React application)
- https://github.com/martin-sicho/genui-docker (Docker files and deployment scripts)

A reference application that was described in this manuscript can be deployed with Docker images that were uploaded to Docker Hub: https://hub.docker.com/u/sichom. However, the images can also be built with the available Docker files and scripts (archived at https://doi.org/10.5281/zenodo.4813625). The reference web application uses the following versions of the GenUI software:

- 0.0.0-alpha.1 for the frontend React application (archived at https://doi.org/10.5281/zenodo.4813608)
- 0.0.0.alpha1 for the backend Python application (archived at https://doi.org/10.5281/zenodo.4813586)

References

1. Wang Y, Cheng T, Bryant SH (2017) PubChem BioAssay: A Decade's Development toward Open High-Throughput Screening Data Sharing. SLAS DISCOVERY: Advancing the Science of Drug Discovery 22(6):655-666.

2. Tetko IV, Engkvist O, Koch U, Reymond J-L, Chen H (2016) BIGCHEM: Challenges and Opportunities for Big Data Analysis in Chemistry. Molecular Informatics 35(11-12):615-621.
3. Rifaioglu AS, Atas H, Martin MJ, Cetin-Atalay R, Atalay V, Doğan T (2019) Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases. Brief Bioinform 20(5):1878-1912.
4. Hoffmann T, Gastreich M (2019) The next level in chemical space navigation: going far beyond enumerable compound libraries. Drug Discov Today 24(5):1148-1156.
5. Tetko IV, Engkvist O, Chen H (2016) Does 'Big Data' exist in medicinal chemistry, and if so, how can it be harnessed? Future medicinal chemistry 8(15):1801-1806.
6. Davies M, Nowotka M, Papadatos G, Dedman N, Gaulton A, Atkinson F, Bellis L, Overington JP (2015) ChEMBL web services: streamlining access to drug discovery data and utilities. Nucleic Acids Research 43(W1):W612-W620.
7. Mendez D, Gaulton A, Bento AP, Chambers J, De Veij M, Félix E, Magariños María P, Mosquera Juan F, Mutowo P, Nowotka M *et al* (2019) ChEMBL: towards direct deposition of bioassay data. Nucleic Acids Research 47(D1):D930-D940.
8. Polishchuk PG, Madzhidov TI, Varnek A (2013) Estimation of the size of drug-like chemical space based on GDB-17 data. Journal of Computer-Aided Molecular Design 27(8):675-679.
9. Drew KLM, Baiman H, Khwaounjoo P, Yu B, Reynisson J (2012) Size estimation of chemical space: how big is it? Journal of Pharmacy and Pharmacology 64(4):490-495.
10. Walters WP, Stahl MT, Murcko MA (1998) Virtual screening—an overview. Drug Discovery Today 3(4):160-178.
11. Bohacek RS, McMartin C, Guida WC (1996) The art and practice of structure-based drug design: A molecular modeling perspective. Med Res Rev 16(1):3-50.
12. Lenselink EB, ten Dijke N, Bongers B, Papadatos G, van Vlijmen HWT, Kowalczyk W, IJzerman AP, van Westen GJP (2017) Beyond the hype: deep neural networks outperform established methods using a ChEMBL bioactivity benchmark set. Journal of Cheminformatics 9(1):45.
13. Liu X, IJzerman AP, van Westen GJP. Computational Approaches for De Novo Drug Design: Past, Present, and Future. In: *Artificial Neural Networks.* Edited by Cartwright H. New York, NY: Springer US; 2021: 139-165.
14. Coley CW (2021) Defining and Exploring Chemical Spaces. Trends in Chemistry 3(2):133-145.
15. Opassi G, Gesù A, Massarotti A (2018) The hitchhiker's guide to the chemical-biological galaxy. Drug Discovery Today 23(3):565-574.
16. Muratov EN, Bajorath J, Sheridan RP, Tetko IV, Filimonov D, Poroikov V, Oprea TI, Baskin II, Varnek A, Roitberg A *et al* (2020) QSAR without borders. Chemical Society Reviews 49(11):3525-3564.
17. Wang L, Ding J, Pan L, Cao D, Jiang H, Ding X (2019) Artificial intelligence facilitates drug design in the big data era. Chemometrics Intellig Lab Syst 194:103850.
18. Schneider G, Clark DE (2019) Automated De Novo Drug Design: Are We Nearly There Yet? Angew Chem Int Ed Engl 58(32):10792-10803.
19. Zhu H (2020) Big Data and Artificial Intelligence Modeling for Drug Discovery. Annual Review of Pharmacology and Toxicology 60(1):573-589.
20. Le TC, Winkler DA (2015) A Bright Future for Evolutionary Methods in Drug Design. ChemMedChem 10(8):1296-1300.
21. Lavecchia A (2019) Deep learning in drug discovery: opportunities, challenges and future prospects. Drug Discov Today 24(10):2017-2032.
22. Schreiber SL, Kotz JD, Li M, Aubé J, Austin CP, Reed JC, Rosen H, White EL, Sklar LA, Lindsley CW *et al* (2015) Advancing Biological Understanding and Therapeutics Discovery with Small-Molecule Probes. Cell 161(6):1252-1265.
23. Carpenter KA, Cohen DS, Jarrell JT, Huang X (2018) Deep learning and virtual drug screening. Future medicinal chemistry 10(21):2557-2567.

24. Chuang KV, Gunsalus LM, Keiser MJ (2020) Learning Molecular Representations for Medicinal Chemistry. Journal of Medicinal Chemistry 63(16):8705-8722.
25. Winter R, Montanari F, Noé F, Clevert D-A (2019) Learning continuous and data-driven molecular descriptors by translating equivalent chemical representations. Chemical Science 10(6):1692-1701.
26. Menke J, Koch O (2021) Using Domain-Specific Fingerprints Generated Through Neural Networks to Enhance Ligand-Based Virtual Screening. Journal of Chemical Information and Modeling 61(2):664-675.
27. Callaway E (2020) 'It will change everything': DeepMind's AI makes gigantic leap in solving protein structures. Nature 588(7837):203-204.
28. Ekins S, Puhl AC, Zorn KM, Lane TR, Russo DP, Klein JJ, Hickey AJ, Clark AM (2019) Exploiting machine learning for end-to-end drug discovery and development. Nat Mater 18(5):435-441.
29. Adam G, Rampášek L, Safikhani Z, Smirnov P, Haibe-Kains B, Goldenberg A (2020) Machine learning approaches to drug response prediction: challenges and recent progress. NPJ precision oncology 4:19-19.
30. Mayr A, Klambauer G, Unterthiner T, Steijaert M, Wegner JK, Ceulemans H, Clevert D-A, Hochreiter S (2018) Large-scale comparison of machine learning methods for drug target prediction on ChEMBL. Chem Sci 9(24):5441-5451.
31. Bian Y, Xie X-Q (2021) Generative chemistry: drug discovery with deep learning generative models. Journal of Molecular Modeling 27(3):71.
32. Zheng S, Lei Z, Ai H, Chen H, Deng D, Yang Y (2020) Deep Scaffold Hopping with Multi-modal Transformer Neural Networks.
33. Stojanović L, Popović M, Tijanić N, Rakočević G, Kalinić M (2020) Improved Scaffold Hopping in Ligand-Based Virtual Screening Using Neural Representation Learning. Journal of Chemical Information and Modeling 60(10):4629-4639.
34. Baskin II (2020) The power of deep learning to ligand-based novel drug discovery. Expert Opin Drug Discov 15(7):755-764.
35. Elton DC, Boukouvalas Z, Fuge MD, Chung PW (2019) Deep learning for molecular design—a review of the state of the art. Mol Syst Des Eng 4(4):828-849.
36. Xu Y, Lin K, Wang S, Wang L, Cai C, Song C, Lai L, Pei J (2019) Deep learning for molecular generation. Future Med Chem 11(6):567-597.
37. Jørgensen PB, Schmidt MN, Winther O (2018) Deep Generative Models for Molecular Science. Mol Inform 37(1-2).
38. Gantzer P, Creton B, Nieto-Draghi C (2020) Inverse-QSPR for de novo Design: A Review. Mol Inform 39(4):e1900087.
39. Yoshikawa N, Terayama K, Sumita M, Homma T, Oono K, Tsuda K (2018) Population-based De Novo Molecule Generation, Using Grammatical Evolution. Chem Lett 47(11):1431-1434.
40. Jensen JH (2019) A graph-based genetic algorithm and generative model/Monte Carlo tree search for the exploration of chemical space. Chem Sci 10(12):3567-3572.
41. Spiegel JO, Durrant JD (2020) AutoGrow4: an open-source genetic algorithm for de novo drug design and lead optimization. J Cheminform 12(1):25.
42. Leguy J, Cauchy T, Glavatskikh M, Duval B, Da Mota B (2020) EvoMol: a flexible and interpretable evolutionary algorithm for unbiased de novo molecular generation. J Cheminform 12(1):55.
43. Hoksza D, Skoda P, Voršilák M, Svozil D (2014) Molpher: a software framework for systematic chemical space exploration. J Cheminform 6(1):7.
44. Jiménez-Luna J, Grisoni F, Schneider G (2020) Drug discovery with explainable artificial intelligence. Nature Machine Intelligence 2(10):573-584.
45. Henault ES, Rasmussen MH, Jensen JH (2020) Chemical space exploration: how genetic algorithms find the needle in the haystack. PeerJ Phy Chem 2:e11.
46. Brown N, Fiscato M, Segler MHS, Vaucher AC (2019) GuacaMol: Benchmarking Models for de Novo Molecular Design. Journal of Chemical Information and Modeling 59(3):1096-1108.

47. Polykovskiy D, Zhebrak A, Sanchez-Lengeling B, Golovanov S, Tatanov O, Belyaev S, Kurbanov R, Artamonov A, Aladinskiy V, Veselov M *et al* (2020) Molecular Sets (MOSES): A Benchmarking Platform for Molecular Generation Models. Frontiers in Pharmacology 11:1931.
48. Bush JT, Pogany P, Pickett SD, Barker M, Baxter A, Campos S, Cooper AWJ, Hirst D, Inglis G, Nadin A *et al* (2020) A Turing Test for Molecular Generators. Journal of Medicinal Chemistry 63(20):11964-11971.
49. Walters WP, Murcko M (2020) Assessing the impact of generative AI on medicinal chemistry. Nature Biotechnology 38(2):143-145.
50. Zhavoronkov A, Aspuru-Guzik A (2020) Reply to 'Assessing the impact of generative AI on medicinal chemistry'. Nature Biotechnology 38(2):146-146.
51. Schneider G, Fechner U (2005) Computer-based de novo design of drug-like molecules. Nature Reviews Drug Discovery 4(8):649-663.
52. Li X, Xu Y, Yao H, Lin K (2020) Chemical space exploration based on recurrent neural networks: applications in discovering kinase inhibitors. J Cheminform 12(1):42.
53. Grisoni F, Neuhaus CS, Hishinuma M, Gabernet G, Hiss JA, Kotera M, Schneider G (2019) De novo design of anticancer peptides by ensemble artificial neural networks. J Mol Model 25(5):112.
54. Wu J, Ma Y, Zhou H, Zhou L, Du S, Sun Y, Li W, Dong W, Wang R (2020) Identification of protein tyrosine phosphatase 1B (PTP1B) inhibitors through De Novo Evoluton, synthesis, biological evaluation and molecular dynamics simulation. Biochem Biophys Res Commun 526(1):273-280.
55. Polykovskiy D, Zhebrak A, Vetrov D, Ivanenkov Y, Aladinskiy V, Mamoshina P, Bozdaganyan M, Aliper A, Zhavoronkov A, Kadurin A (2018) Entangled Conditional Adversarial Autoencoder for de Novo Drug Discovery. Molecular Pharmaceutics 15(10):4398-4405.
56. Merk D, Friedrich L, Grisoni F, Schneider G (2018) De Novo Design of Bioactive Small Molecules by Artificial Intelligence. Molecular Informatics 37(1-2):1700153.
57. Putin E, Asadulaev A, Vanhaelen Q, Ivanenkov Y, Aladinskaya AV, Aliper A, Zhavoronkov A (2018) Adversarial Threshold Neural Computer for Molecular de Novo Design. Molecular Pharmaceutics 15(10):4386-4397.
58. Sumita M, Yang X, Ishihara S, Tamura R, Tsuda K (2018) Hunting for Organic Molecules with Artificial Intelligence: Molecules Optimized for Desired Excitation Energies. ACS Central Science 4(9):1126-1133.
59. Zhavoronkov A, Ivanenkov YA, Aliper A, Veselov MS, Aladinskiy VA, Aladinskaya AV, Terentiev VA, Polykovskiy DA, Kuznetsov MD, Asadulaev A *et al* (2019) Deep learning enables rapid identification of potent DDR1 kinase inhibitors. Nature Biotechnology 37(9):1038-1040.
60. Sparkes A, Aubrey W, Byrne E, Clare A, Khan MN, Liakata M, Markham M, Rowland J, Soldatova LN, Whelan KE *et al* (2010) Towards Robot Scientists for autonomous scientific discovery. Autom Exp 2:1.
61. Coley CW, Eyke NS, Jensen KF (2020) Autonomous Discovery in the Chemical Sciences Part I: Progress. Angewandte Chemie International Edition 59(51):22858-22893.
62. Coley CW, Eyke NS, Jensen KF (2020) Autonomous Discovery in the Chemical Sciences Part II: Outlook. Angewandte Chemie International Edition 59(52):23414-23436.
63. Henson AB, Gromski PS, Cronin L (2018) Designing Algorithms To Aid Discovery by Chemical Robots. ACS Cent Sci 4(7):793-804.
64. Dimitrov T, Kreisbeck C, Becker JS, Aspuru-Guzik A, Saikin SK (2019) Autonomous Molecular Design: Then and Now. ACS Appl Mater Interfaces 11(28):24825-24836.
65. Schneider G (2018) Automating drug discovery. Nat Rev Drug Discov 17(2):97-113.
66. Willems H, De Cesco S, Svensson F (2020) Computational Chemistry on a Budget: Supporting Drug Discovery with Limited Resources. J Med Chem 63(18):10158-10169.

67. Chu Y, He X (2019) MoleGear: A Java-Based Platform for Evolutionary De Novo Molecular Design. Molecules 24(7).
68. Douguet D (2010) e-LEA3D: a computational-aided drug design web server. Nucleic Acids Research 38(suppl_2):W615-W621.
69. Griffen EJ, Dossetter AG, Leach AG (2020) Chemists: AI Is Here; Unite To Get the Benefits. Journal of Medicinal Chemistry 63(16):8695-8704.
70. Pastor M, Gómez-Tamayo JC, Sanz F (2021) Flame: an open source framework for model development, hosting, and usage in production environments. Journal of Cheminformatics 13(1):31.
71. Green DVS, Pickett S, Luscombe C, Senger S, Marcus D, Meslamani J, Brett D, Powell A, Masson J (2020) BRADSHAW: a system for automated molecular design. Journal of Computer-Aided Molecular Design 34(7):747-765.
72. Ivanenkov YA, Zhebrak A, Bezrukov D, Zagribelnyy B, Aladinskiy V, Polykovskiy D, Putin E, Kamya P, Aliper A, Zhavoronkov A (2021) Chemistry42: An AI-based platform for de novo molecular design. arXiv preprint arXiv:210109050.
73. Zhumagambetov R, Kazbek D, Shakipov M, Maksut D, Peshkov VA, Fazli S (2020) cheML.io: an online database of ML-generated molecules. RSC Advances 10(73):45189-45198.
74. Liu X, Ye K, van Vlijmen HWT, IJzerman AP, van Westen GJP (2019) An exploration strategy improves the diversity of de novo ligands using deep reinforcement learning: a case for the adenosine A2A receptor. J Cheminform 11(1):35.
75. MIT License. https://opensource.org/licenses/MIT. Accessed 2021-03-12.
76. GenUI Frontend Application. By Šícho M. https://github.com/martin-sicho/genui-gui. Accessed 2021-03-12.
77. GenUI Backend Application. https://github.com/martin-sicho/genui Accessed
78. Merkel D (2014) Docker: lightweight Linux containers for consistent development and deployment. Linux J 2014(239):Article 2.
79. Cito J, Ferme V, Gall HC: Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research. In: *Web Engineering: 2016// 2016; Cham*. Springer International Publishing: 609-612.
80. Docker. https://github.com/docker/docker-ce. Accessed
81. GenUI Docker Files. By Šícho M. https://github.com/martin-sicho/genui-docker. Accessed
82. React: A JavaScript Library for Building User Interfaces. By Facebook I. https://reactjs.org/. Accessed 2020-12-16.
83. Vibe: A beautiful react.js dashboard build with Bootstrap 4. By Salas J. https://github.com/NiceDash/Vibe. Accessed
84. Tétreault-Pinard ÉO (2019) Plotly JavaScript Open Source Graphing Library.
85. Chart.js: Simple yet flexible JavaScript charting for designers & developers. https://www.chartjs.org/. Accessed
86. ChemSpace JS. https://openscreen.cz/software/chemspace/home/. Accessed
87. Schaduangrat N, Lampa S, Simeon S, Gleeson MP, Spjuth O, Nantasenamat C (2020) Towards reproducible computational drug discovery. J Cheminform 12(1):9.
88. van der Maaten L, Hinton G (2008) Viualizing data using t-SNE. Journal of Machine Learning Research 9:2579-2605.
89. Poličar PG, Stražar M, Zupan B (2019) openTSNE: a modular Python library for t-SNE dimensionality reduction and embedding. bioRxiv:731877.
90. GenUI Python Documentation. https://martin-sicho.github.io/genui/docs/index.html. Accessed 2021-03-12.
91. Foundation DS (2019) Django (Version 2.2).
92. Encode OSS L (2019) Django REST Framework.
93. Debian-based images containing PostgreSQL with the RDKit cartridge. https://hub.docker.com/r/informaticsmatters/rdkit-cartridge-debian. Accessed
94. RDKit: Open-source cheminformatics toolkit. By http://www.rdkit.org/. Accessed
95. Django RDKit. https://github.com/rdkit/django-rdkit. Accessed

96. Bento AP, Hersey A, Félix E, Landrum G, Gaulton A, Atkinson F, Bellis LJ, De Veij M, Leach AR (2020) An open source chemical structure curation pipeline using RDKit. J Cheminform 12(1):51.
97. CELERY: Distributed Task Queue. https://github.com/celery/celery. Accessed
98. Redis: in-memory data structure store. By https://github.com/redis/redis. Accessed
99. Hunt A, Thomas D (2000) The Pragmatic Programmer: From Journeyman to Master. Addison-Wesley Longman Publishing Co. Inc.
100. Celery: Get Started. https://docs.celeryproject.org/en/stable/getting-started/introduction.html#get-started. Accessed 2020-12-16.
101. Docker Hub. https://hub.docker.com/. Accessed 2020-12-16.
102. Redis: Docker Official Images. By https://hub.docker.com/_/redis. Accessed
103. NGINX Web Server. By https://github.com/nginx/nginx. Accessed
104. NGINX: Official Docker Images. By https://hub.docker.com/_/nginx. Accessed
105. Kim S, Chen J, Cheng T, Gindulyte A, He J, He S, Li Q, Shoemaker BA, Thiessen PA, Yu B *et al* (2019) PubChem 2019 update: improved access to chemical data. Nucleic Acids Research 47(D1):D1102-D1109.
106. Irwin JJ, Sterling T, Mysinger MM, Bolstad ES, Coleman RG (2012) ZINC: A Free Tool to Discover Chemistry for Biology. Journal of Chemical Information and Modeling 52(7):1757-1768.
107. Wishart DS, Knox C, Guo AC, Shrivastava S, Hassanali M, Stothard P, Chang Z, Woolsey J (2006) DrugBank: a comprehensive resource for in silico drug discovery and exploration. Nucleic Acids Research 34(suppl_1):D668-D672.
108. Gilson MK, Liu T, Baitaluk M, Nicola G, Hwang L, Chong J (2016) BindingDB in 2015: A public database for medicinal chemistry, computational chemistry and systems pharmacology. Nucleic Acids Research 44(D1):D1045-D1053.
109. Skuta C, Popr M, Muller T, Jindrich J, Kahle M, Sedlak D, Svozil D, Bartunek P (2017) Probes & Drugs portal: an interactive, open data resource for chemical biology. Nature Methods 14(8):759-760.
110. IBM RXN for Chemistry. https://rxn.res.ibm.com/. Accessed 2021-03-12.
111. PostEra Manifold. https://postera.ai/manifold/. Accessed 2021-03-12.