

SOFTWARE

Open Access



Flame: an open source framework for model development, hosting, and usage in production environments

Manuel Pastor^{*} , José Carlos Gómez-Tamayo and Ferran Sanz

Abstract

This article describes Flame, an open source software for building predictive models and supporting their use in production environments. Flame is a web application with a web-based graphic interface, which can be used as a desktop application or installed in a server receiving requests from multiple users. Models can be built starting from any collection of biologically annotated chemical structures since the software supports structural normalization, molecular descriptor calculation, and machine learning model generation using predefined workflows. The model building workflow can be customized from the graphic interface, selecting the type of normalization, molecular descriptors, and machine learning algorithm to be used from a panel of state-of-the-art methods implemented natively. Moreover, Flame implements a mechanism allowing to extend its source code, adding unlimited model customization. Models generated with Flame can be easily exported, facilitating collaborative model development. All models are stored in a model repository supporting model versioning. Models are identified by unique model IDs and include detailed documentation formatted using widely accepted standards. The current version is the result of nearly 3 years of development in collaboration with users from the pharmaceutical industry within the IMI eTRANSafe project, which aims, among other objectives, to develop high-quality predictive models based on shared legacy data for assessing the safety of drug candidates.

Keywords: Modeling framework, Modeling tools, Reproducibility, Model management, Workflow, QSAR, Model integration, Web-interfaces, In-silico toxicology

Introduction

In the last years, biomedical data is becoming widely available, thanks to the creation of repositories like PubChem [1] and ChEMBL [2], databases resulting from public–private partnerships like eTOX [3, 4], as well as data policies like FAIR [5], which facilitate the access of existing data to the scientific community.

An interesting way of exploiting this vast amount of data is the development of mathematical models

connecting the chemical structure of the substances with their biological properties. Such models are not new. Quantitative Structure–Activity Relationships (QSAR) were first described in the 60 s [6, 7]. QSAR models use regression methods to identify the structural properties linked to quantitative biological properties or to predict these properties for new substances. For biological properties characterized using qualitative descriptions (e.g., positive or negative) conceptually similar approaches can be applied using classifiers. The first QSAR models were developed using small series of congeneric compounds, often synthesized and tested *ad-hoc* for the study. Nowadays, large series of structurally diverse compounds can be easily obtained from public repositories.

^{*}Correspondence: manuel.pastor@upf.edu
Research Programme on Biomedical Informatics (GRIB), Department of Experimental and Health Sciences, Hospital del Mar Medical Research Institute (IMIM), Universitat Pompeu Fabra, Barcelona, Spain



© The Author(s) 2021. This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>. The Creative Commons Public Domain Dedication waiver (<http://creativecommons.org/publicdomain/zero/1.0/>) applies to the data made available in this article, unless otherwise stated in a credit line to the data.

Pharmaceutical companies can also extract these series from their own internal repositories and use them isolated or combined with compounds from external sources. This fact, combined with recent developments in machine learning (ML) and deep learning (DL) methodologies [8] as well as with the implementation of many of these methods in open source libraries [9], create an ideal scenario for the development of predictive models with biomedical application.

Indeed, the use of ML and DL is becoming very popular in biomedical research. A few remarkable models developed recently have been listed in Table 1 as examples of applications of this methodology, illustrating their usefulness.

More and more, the models obtained by the application of ML are seen as valuable business assets. Accurate and appropriately shared models can bring a number of benefits if we are able to make effective use of existing expertise [17]. However, the true capability of a model for solving real-world problems critically depends on aspects related to model implementation, as the following.

Reproducibility

Models must produce the same results when used at different sites or times. This simple, basic requirement is difficult to meet if (i) the training data is not available and distinctively identified or (ii) the algorithms used are not documented with enough detail, or if it is not possible to use exactly the same software (same version and same platform). The fast evolution of computational tools (both hardware and software) makes it difficult to preserve a model for some time. This topic has been discussed by various authors, proposing diverse solutions for mitigating this problem like the use of appropriate standards for QSAR data interchanges [18] or a workflow for implementing published QSAR models and recommendations to modelers [19].

Accessibility

Models are digital assets to which the FAIR accessibility principle can also be applied [5, 20]. Ideally, access to existing models should be facilitated, particularly for models developed in academic environments. In practice, there are barriers related to the intellectual property of

the tools required to generate the predictions. This can apply to commercial applications used to generate 3D structures or molecular descriptors or even the modeling software itself. For this reason, the use of open source alternatives should be prioritized.

Not all accessibility barriers are related to intellectual property issues, and models should be implemented in a way that allows their use in different operative systems (e.g., Windows, Linux, iOS) and platforms (e.g., implemented as a desktop application or as REST [21] web services in centralized servers). This is particularly true in corporative environments, where company restrictive policies about OS or platforms could hinder access to useful models. Also, and not less important, is to facilitate the model use for non-experts by providing a friendly end-user interface.

Model management

A good model is a valuable asset for an organization, and as such, it should be managed using appropriate governance principles [17]. One of the first steps is to identify and store the model appropriately. This facilitates common tasks like knowing which model was used to generate some prediction or retrieving a certain model cited in a report. This task is hampered by the fact that models are not static entities. Models evolve as the software they use is updated or as the training series is enriched with new compounds for covering a broader chemical space. Consequently, models often have many versions that must be properly identified and stored as well, recording all the changes in the training series and the modeling software.

A separate task is to document the models. Models can be documented with different levels of detail for different purposes [22]. As a minimum, every model must be accompanied by documents allowing to reproduce the algorithm completely and to understand and interpret the prediction results. The documentation can be used for other purposes, like demonstrating to regulatory bodies the quality of the prediction for replacing experimental tests [23]. Therefore, we recommend a layered documentation structure, including basic mandatory information and more detailed optional layers.

Table 1 Examples of ML/DL applications in biomedical research

ML application	Brief explanation	References
Drug discovery	Identification of new bioactive compounds	[8–10]
Toxicity prediction	Identify hazardous substances	[11–13]
Precision medicine	Personalize medical treatment to patient idiosyncrasy	[14]
Imaging diagnostics	Identification of abnormalities from imaging	[15, 16]

Reporting

For the model developer, the meaning of a model prediction result is obvious; the model estimates the biological annotations present in the training series. However, users not involved in the process of model building lack this context. This often creates confusion and difficulties for users to interpret the model’s prediction, particularly when the model produces a numerical outcome. For this reason, as a minimum, model results must explicitly include the units in which they are expressed, a brief, concise explanation of how these results must be interpreted, and the level of confidence within which the prediction values must be clearly declared [22].

Every prediction has a certain uncertainty associated as a consequence of the errors present in the training series annotations, as well as the limitations of the model predictivity. For this reason, prediction results must be accompanied by a quantitative estimation of the individual prediction error. This estimation cannot be generic, based solely on the error observed for the compounds in the training series. It must also consider how far the query compound is away from the model applicability domain.

In the last decades, several solutions have been proposed for supporting the access to existing QSAR models or the development of new ones, overcoming the issues described above. One of the first was a Polynomial Neural Network published on-line in 1999 [24]. In a recent

review [25], these efforts were classified under four categories; research group-centric model collections, model collections from (Q)SAR oriented projects, (Q)SAR models in integrated modeling environments, and (Q)SAR model repositories. In the present article, we introduce Flame, a new modeling framework for facilitating the development, hosting, and use of predictive models in production environments. When comparing with existing resources, Flame belongs to the category of integrated modeling environments mentioned above. In the “Implementation” and “Results” sections, apart from describing its features, we compare Flame with other similar tools, highlighting the differential characteristics which make it highly valuable for certain QSAR modeling applications.

Flame was developed in the context of project eTRANSafe (IMI2 Joint Undertaking under Grant Agreement No. 777365), producing integrative data infrastructures and innovative computational methods to improve the feasibility and reliability of translational safety assessment during the drug development process. For this reason, Flame was originally designed to host predictive models for drug safety endpoints, even if it can be used with other applications in biomedical research.

Implementation

The Flame architecture is illustrated in Fig. 1. It consists of a Python library (the Flame backend), which can be used from a terminal with a command-line interface,

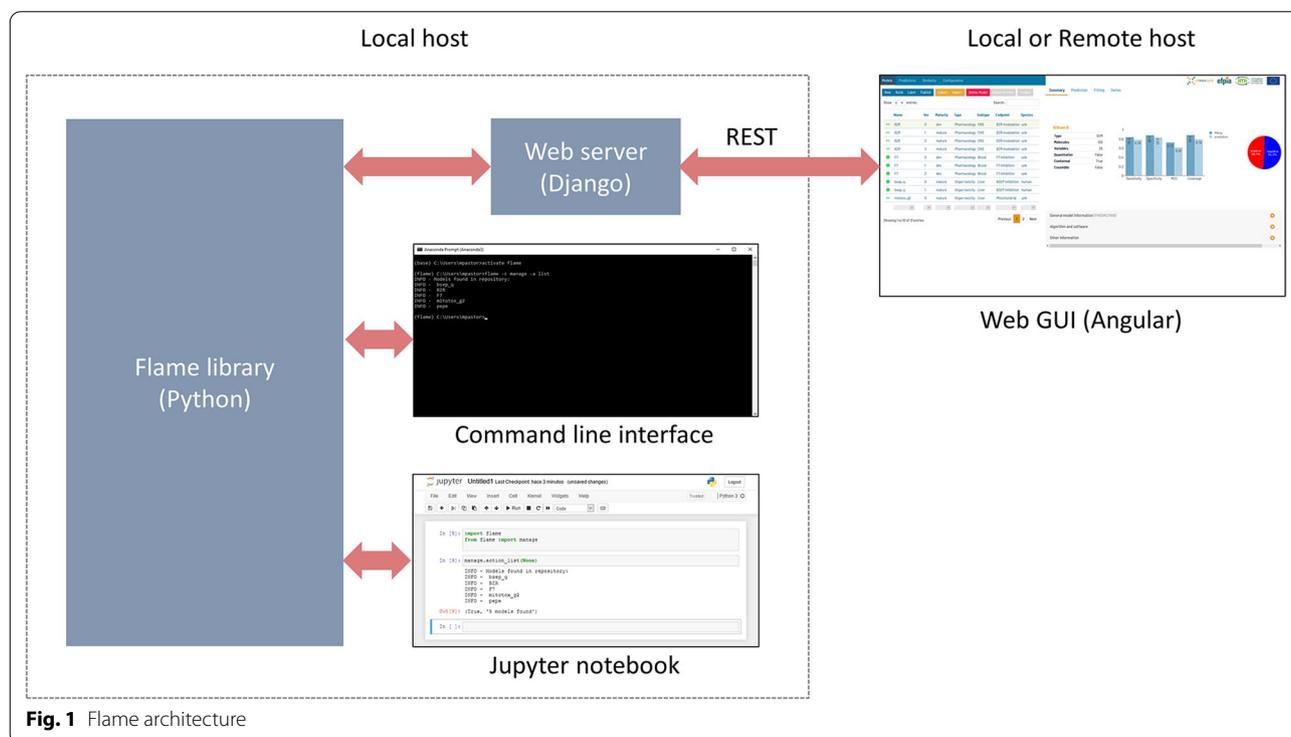


Fig. 1 Flame architecture

called from a Jupyter notebook [26], or scripts written shell languages (bash, bat, etc.). It also implements a web server (written in Django [27]) offering the library features as REST services [21] and a complete web interface (written in Angular [28]) providing a rich graphic user interface (GUI).

The GUI can be executed locally as a desktop application, starting the web server in the same computer running the Flame backend. It is also possible to run the Flame backend in a server and access the REST services from a remote client, thus allowing to run Flame as a departmental or global prediction service in corporate environments. This architecture differentiates Flame from other integrated modeling environments operating exclusively as web-services (e.g., CHEMBENCH [29], OCHEM [30]). The possibility of executing the software locally, either as a desktop app or in a local server, is a must when the data used for model training or prediction is confidential, and the company policies disallow to send it over the Internet.

The Flame backend and the optional flame web server make use of Conda [31] to define the libraries required, facilitate their automatic installation in a private environment. Conda also allows defining the acceptable library versions to avoid incompatibilities. Flame can be installed and used in Linux, Windows, and iOS operative systems.

The code was written using Object Oriented Programming (OOP) as a Python library. The main classes (see Table 2 and Fig. 2) can be classified as low-level or high-level. Low-level classes carry out simple tasks while the high-level classes execute model building and model prediction workflows using the low-level classes. For example, the default model building workflow implemented in high-level class *build* (Fig. 2) starts from a training series of annotated chemical structures. It uses class *idata* to import their chemical structures, normalize them, extract the biological annotations and generate molecular descriptors which are stored in a numerical matrix. The molecular descriptors and the annotations are sent

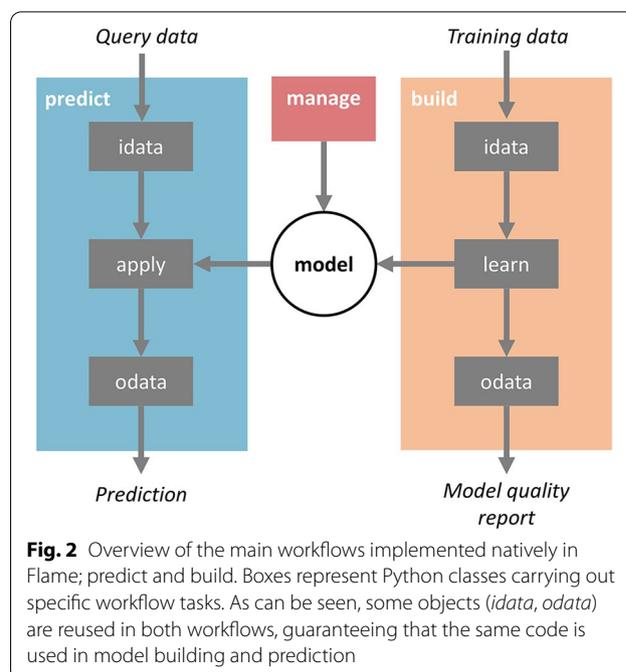


Fig. 2 Overview of the main workflows implemented natively in Flame; predict and build. Boxes represent Python classes carrying out specific workflow tasks. As can be seen, some objects (*idata*, *odata*) are reused in both workflows, guaranteeing that the same code is used in model building and prediction

to the class *learn*, which normalizes the numerical values and builds models using machine learning (ML) tools like Random Forest (RF). This model is stored in a machine-readable format (as a pickle serialized version of the scikit-learn *estimator object* [9, 32]) suitable to predict the biological properties of novel compounds. Finally, the class *odata* is used to format the results and produce suitable output. The default prediction workflow (Fig. 2) uses exactly the same low-level *idata* class to import and pre-process the structures. This workflow design has the advantage of guaranteeing that the predictions use exactly the same code used for model building, for equivalent tasks, thus producing consistent results. Then, the low-level class *apply* retrieves the *estimator* saved previously during the model building process for computing

Table 2 Main Python classes used in Flame

Type	Class	Functionality	Input	Output
High-level	Build	Generates a model	Training series	Model
	Predict	Uses an existing model to generate a prediction for a query compound	Query compound	Prediction
	Manage	Handles (create, delete, export and import) models in the repository	–	–
Low-level	Idata	Processes chemical structures to obtain molecular descriptors as an X matrix and annotations as a Y matrix (when provided)	Chemical structures	X (Y) numerical matrices
	Learn	Generates a model from the X and Y numerical matrices	X and Y numerical matrices	Model
	Apply	Uses an existing model to generate a prediction from an X matrix	X numerical matrix	Prediction
	odata	Formats results as human-readable output or computational formats suitable for the GUI	Results	Formatted results

the prediction results, which are formatted by the *odata* class to generate suitable output.

Most of the building and prediction workflow steps are configurable. For example, we can select the structure normalization algorithms or the molecular descriptors calculation method. Also, the methods themselves can be configured by adjusting their internal parameters. In Flame, the methods used to build a model and their configurable parameters are defined in a single parameter file (*parameters.yml*), which can be seen as the model blueprint. This file is stored in a folder, together with the original training series and the *estimator* generated by *build*. This folder contains a complete and comprehensive definition of how a model has been built. The model repository is a user-defined path in the computer filesystem where all these folders are stored. Flame can work simultaneously with diverse model repositories located in local or remote filesystems, a convenient feature to maintain separate model collections per project or user.

Flame models are used to predict the biological properties of new compounds using the *predict* workflow (see Fig. 2). Since models are folders, they can be saved, compressed, backed-up, or transmitted between Flame instances installed in different computers. In any of these cases, Flame guarantees that the predictions are reproducible. In this sense, Flame models can be seen as self-contained prediction engines. Flame provides commands to export and import models as a single binary file, consisting in the compressed version of the model folder. On import, the version of the libraries used to generate the model is checked to guarantee full compatibility and reproducibility.

The use of the parameter file described above offers limited customization since the user can select only among the algorithms and methods implemented natively in Flame. To overcome this limitation, the model workflows do not call the low-level classes directly but use a derived class stored locally within the model folder (see Fig. 3). These derived or child classes inherit all of the parent class properties, and in simple models this mechanism is the exact equivalent to calling the Flame classes directly. However, the child class methods can be overridden, allowing unlimited model customization. For example, advanced users can insert code calling external tools to generate molecular descriptors, include extra steps in the model building or prediction workflow or adapt the output to generate customized reports. Since these changes are written in the child class instance, stored locally within the model folder, they do not affect other models. Moreover, these changes are preserved when the model is saved or exported. This possibility to embed custom code in the building and prediction

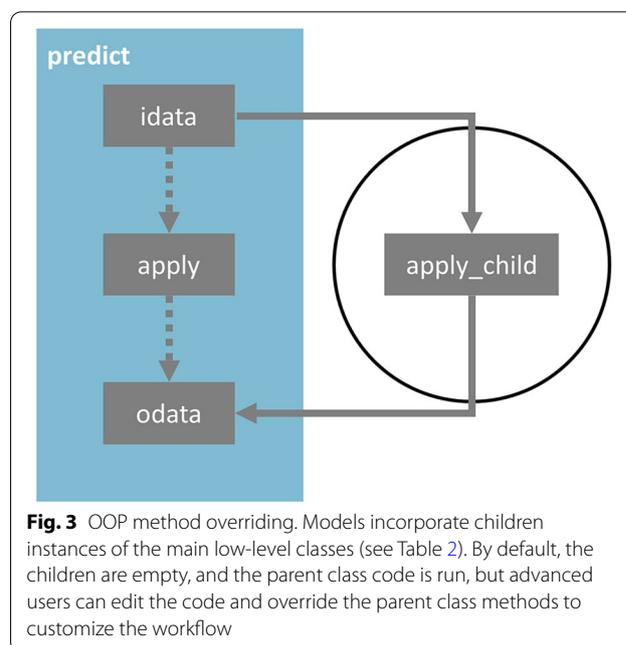


Fig. 3 OOP method overriding. Models incorporate children instances of the main low-level classes (see Table 2). By default, the children are empty, and the parent class code is run, but advanced users can edit the code and override the parent class methods to customize the workflow

workflows differentiates Flame from other integrated environments, either on-line or downloadable. To the best of our knowledge, it is only present in eTOXlab [33], a modeling framework developed in our group some time ago.

Results

Model building features

Flame can build predictive models starting from a single file in SDF file format containing the structures and the biological properties of a training series. The default model building workflow takes care of reading the structures, normalizing them, extracting the annotations, generating molecular descriptors, scaling their values and building a machine-learning model that is saved in a format suitable for predicting new compounds' properties.

Flame provides defaults for methods and parameters, but the user can customize them, either editing the parameter file *parameters.yml* when using Flame in command line mode or using the model building dialogue (Fig. 4) when using the Flame GUI.

Table 3 describes the methods implemented natively in Flame. All of them make use of open source libraries. The choice of models can be easily extended to include commercial products or external tools, using the code overriding technique described in "Implementation" section.

Typically, models are built starting from a collection of annotated chemical structures, but Flame can also use as input a tab-separated (TSV) table with pre-calculated molecular descriptors and annotations. Another option, rarely found in other modeling frameworks (but present

Input type
molecule

SDFFile name
GENERIC_NAME.name
Name of the compound name field in the SDF file

SDFFile activity
CLASS
Name of the activity field in the SDF file

SDFFile experimental
Experimental data field in the SDF file

SDFFile complementary
Complementary data field in the SDF file

quantitative
Should be true for quantitative endpoints and false for qualitative endpoints

normalize method
standardize
Selection of a standardization method

convert3D method
-none-
Selection of a 3D conversion method

ionize method
-none-
Selection of an ionization method

modelAutoscaling
StandardScaler
Scaling method. Null means that raw, unscaled data, will be used

computeMD method
 RDKit_properties
 RDKit_md
 morganFP
 custom
Selection of molecular descriptors to be used in model generation

Model
RF
List of available ML algorithms

Conformal
If true, use the conformal variant of the selected modeling method, when available

Conformal confidence
0.8
Conformal estimator confidence (from 0 to 1)

ModeValidationCV
kfold
Selection of cross-validation method

ModeValidationN
5
Number of folds

Feature selection
-none-
Whether to perform or not feature selection

imbalance
-none-
Whether to perform or not sub/over sampling strategies

RF parameters

class_weight
balanced
Weights associated with classes. If not given, all classes are supposed to have weight one

max_depth
max_depth
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples

max_features
sqrt
The maximum depth of the tree. If auto, then max_features=sqrt(n_features). If sqrt, idem as auto. If log2, then max_features=log2(n_features). If None, then max_features=n_features.

n_estimators
200
The number of trees in the forest

oob_score Whether to use out-of-bag samples to estimate the generalization accuracy

random_state
46
Random seed

Tune (optimization settings can be defined in the parameter file)

Fig. 4 Dialogue used to define the model building workflow methods and parameters (simplified)

in OCHEM [30]), is the possibility to use as input the prediction results of other models present in the repository. This option, called in Flame “model ensemble”, is interesting for integrating the results of multiple models. For qualitative models, multiple results can be summarized using majority voting. The prediction results of an ensemble of quantitative models can be summarized using their means or medians. Regressors and classifiers can also be trained with the model ensemble, using it as a sort of “molecular descriptors”, to generate a smarter result combination and obtain better predictions. When the ensemble models estimate the individual prediction

error, this information is considered by Flame, using appropriate probabilistic methods, to generate an estimation of the final prediction error. The description of these algorithms is beyond the present work scope and will be published in a separate article.

The last step of model building workflows is estimating the model quality using cross-validation. Flame presents information about the model goodness of fit, predictive quality, and some statistical information of the training series (e.g., value distribution). Since Flame can use diverse ML methods, we tried to generate comparable model quality indexes to facilitate the selection of the

Table 3 Overview of the main modeling methods and tools implemented natively in Flame

Modeling task	Method	Source
Structure normalization	Standardiser	[34]
	ChEMBL pipeline	[35, 36]
Molecular descriptors calculation	RDKit properties	[37]
	RDKit md	[37]
	RDKit Morgan fingerprints	[37]
Scaling	Raw	–
	Autoscaling	[9]
Machine learning	RF	[9, 38]
	SVM	[9, 39]
	PLS	[9, 40]
	XGBOOST	[41]
	Conformal regression	[42, 43]

best methods and parameters. The values shown are summarized in Table 4 for qualitative and quantitative endpoints.

The Flame GUI provides additional information oriented to diagnose the quality of the model and the training series, as shown in Fig. 5. For qualitative endpoints (left side of Fig. 5), the confusion matrix is shown as a 2×2 matrix. A radar plot is also used to represent, in the radius of its four sections, the relative size of the true positive, true negative, false positive, and false negative results. This information is shown separately for the model fitting and prediction, the latter being calculated using cross-validation methods selected by the user (default to five k-fold). Besides, Flame displays a

scatterplot of the training series using the two first Principal Components (PCs) obtained by running a Principal Component Analysis (PCA) with the calculated molecular descriptors. Objects (compounds) are colored red or blue according to their biological annotations (positive or negative, respectively). The positive and negative ratio of substances in the training series is depicted using a pie chart.

For quantitative endpoints (right side of Fig. 5), apart from the parameters mentioned in Table 4, the interface shows scatterplots of fitted/predicted values versus the experimental annotations. For conformal models, the confidence interval for the defined confidence level is also shown. Flame displays a scatterplot of the training series in a separate tab, like the one shown for qualitative endpoints. However, in this case, the substances are colored using the continuous scale included in the plot. The distribution of the annotation values is shown using a violin-type plot, which offers valuable information to diagnose a skewed value distribution or the presence of outliers. All the graphics representing the training series are interactive, and hovering the mouse cursor over the dots allows to display the 2D structure of the compounds they represent.

The model quality reports described above are persistent. All this information is stored within the model folder and can be retrieved and shown in subsequent work sessions.

Model predictions

Models stored in the repository can be used to predict the biological properties of a compound entering an SDFFile

Table 4 Model quality parameters shown in the Flame GUI

Endpoint type	Parameter	Definition
Qualitative	Sensitivity (fitting and prediction)	$\frac{TP}{TP+FN}$
	Specificity (fitting and prediction)	$\frac{TN}{TN+FP}$
	MCC (fitting and prediction)	$\frac{(TP*TN)-(FP*FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
Quantitative	SDEC (fitting)	$\sqrt{\frac{\sum (Y_{exp}-Y_{pred})^2}{n}}$
	SDEP (prediction)	$\sqrt{\frac{\sum (Y_{exp}-Y_{pred})^2}{n}}$
	r^2 (fitting)	$\frac{\sum (Y_{exp}-Y_{pred})^2}{\sum \sqrt{(Y_{mean}-Y_{pred})^2}}$
	q^2 (prediction)	$\frac{\sum (Y_{exp}-Y_{pred})^2}{\sum \sqrt{(Y_{mean}-Y_{pred})^2}}$
Conformal models	Conformal coverage	$\frac{\text{Samples inside confidence boundaries}}{\text{Total number of samples}}$
	Conformal accuracy	$\frac{\text{Samples predicted correctly}}{\text{Total number of samples}}$
	Mean interval (only quantitative)	$\frac{\sum Y_{max}-Y_{min} }{n}$

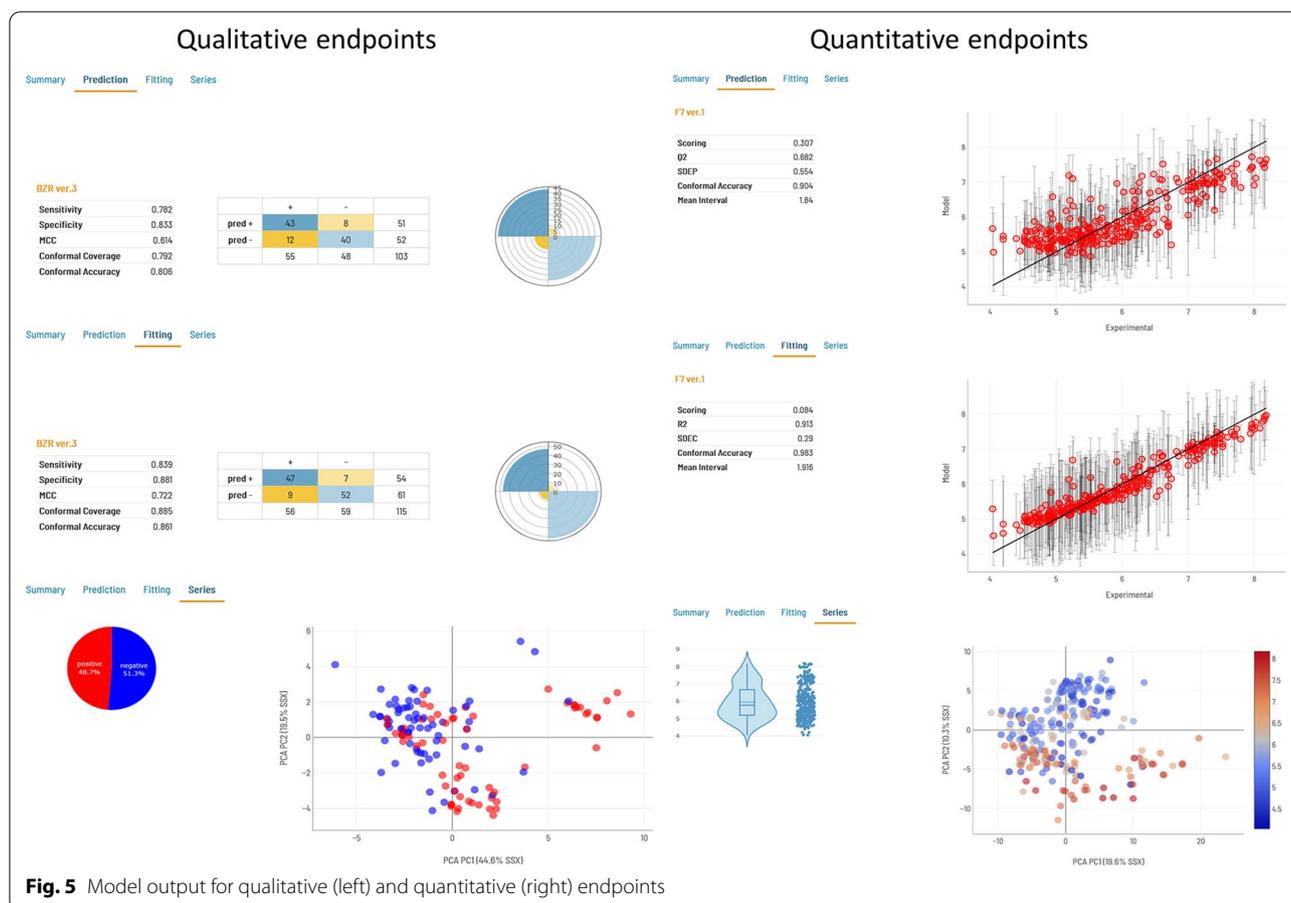


Fig. 5 Model output for qualitative (left) and quantitative (right) endpoints

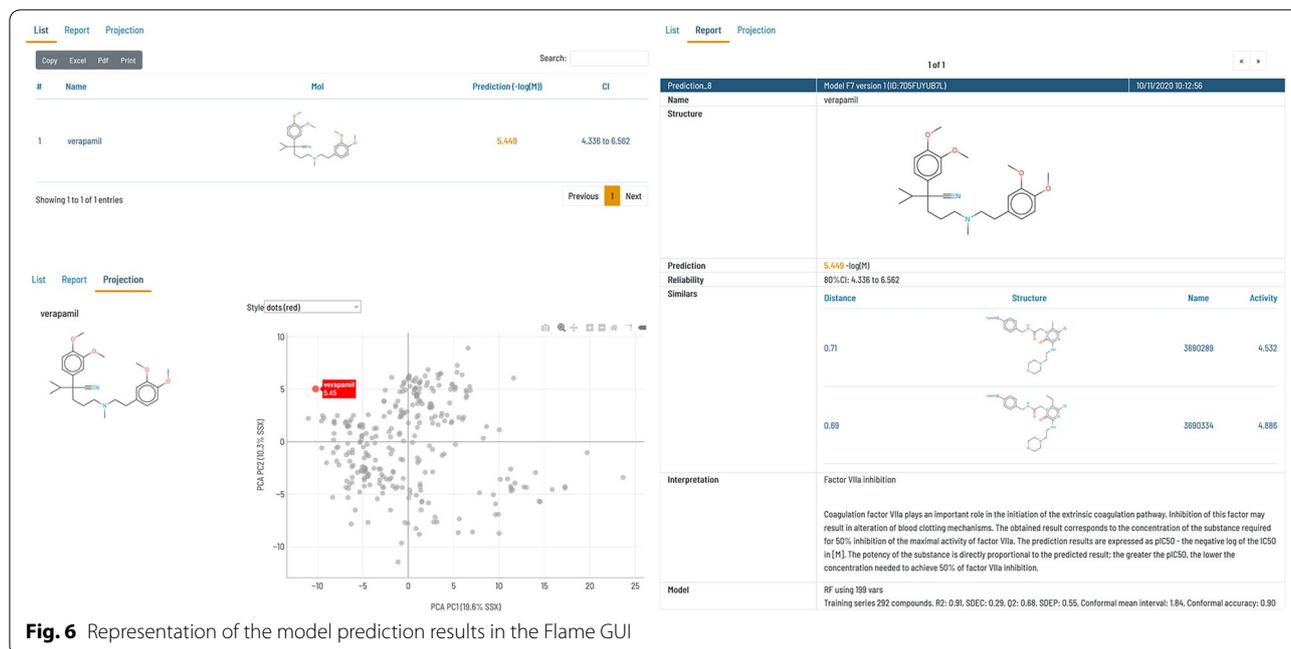
with its structure or sketching it in the included molecular editor. The prediction workflow will then apply to this structure the same pretreatment, molecular descriptors calculation, and x-matrix scaling used for the training series, using exactly the same source code, thus guaranteeing the maximum consistency. The molecular descriptors obtained are projected using the stored estimator to obtain the prediction results. Prediction results can be qualitative or quantitative, depending on the nature of the training series annotations. Models built using conformal regression [43] generate additional information about the prediction uncertainty. For quantitative endpoints, they provide a confidence interval, while for qualitative (binary) endpoints, the prediction result can be “uncertain”, meaning that the model cannot ascertain if the result is positive or negative. In either case, the model reports the prediction uncertainty at a probability (the CI confidence level or the probability that the result is correct, respectively) defined by the user.

Models are watermarked using a unique ID (a random string of ten ASCII uppercase chars) generated during the model building process. This ID is useful to guarantee the model identity, even when different models are

assigned the same name or exported to different Flame instances. Furthermore, when a model is used for prediction, its unique model ID is stored together with the prediction results. Predictions stored in the prediction repository keep record of the model version used to generate it, guaranteeing complete traceability.

As stated in the introduction, prediction results are often difficult to understand and interpret by users not involved in the model building. For this reason, the Flame GUI presents the prediction results in various formats, decorated with extra information aiming to facilitate the result interpretation and its use for decision making.

As shown in Fig. 6, results are displayed in three alternative views. First, they are presented as a list, including for every predicted compound its name, 2D structure, prediction result, and uncertainty information when available. This list is paged, searchable, and can be ordered by column values. It can also be exported to Excel or PDF formats, printed, or copied to the clipboard. Clicking in any of the list items displays a more detailed report for a single compound. Reports show the compound name, structure, and prediction result as well as complementary information about how to interpret

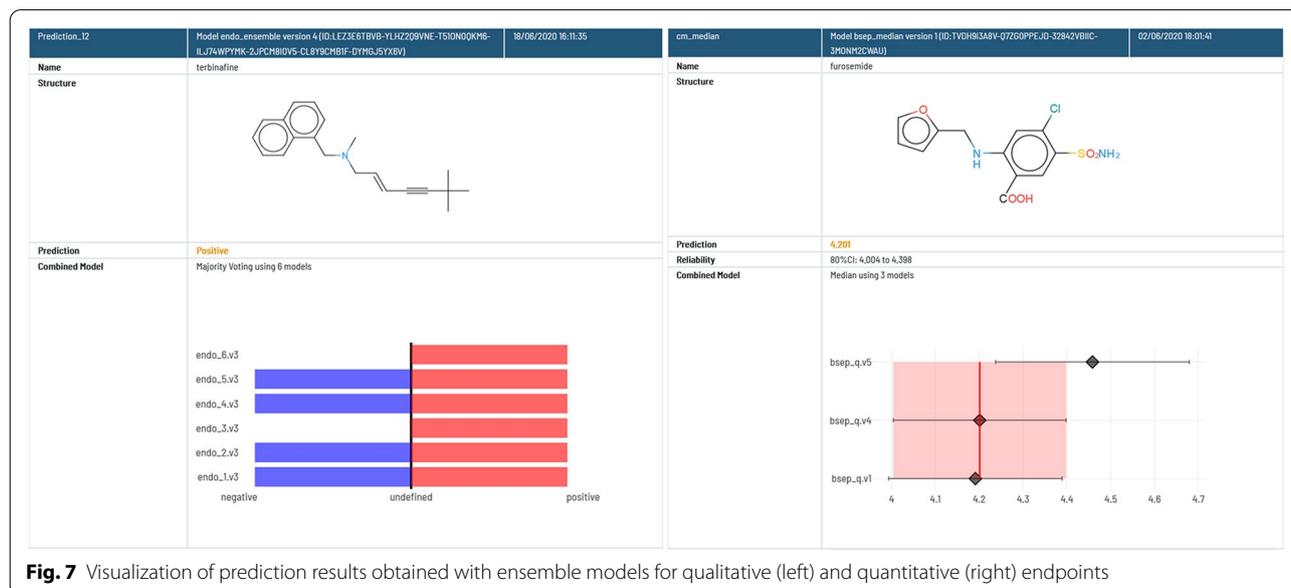


the result (extracted from the model documentation). Besides, a list of the closest compounds in the training series, labeled with their biological annotations, is also shown. For obtaining this list, the similarity is computed using the same molecular descriptors used for building the model.

When an ensemble of models is used for prediction, the prediction report shows the individual result of the low-level models and the combined result (Fig. 7). For

conformal binary classifiers (left of Fig. 7), the graphic shows the low-level model prediction results, indicating if the query compound belongs to class 0 (negative), class 1 (positive), both of them (inconclusive type I) or neither (inconclusive type II). For conformal quantitative models (right of Fig. 7), the predictions are shown with the corresponding confidence intervals.

Finally, the prediction results are also projected on the training series PCA scores scatterplot, generated as



explained in the previous section (Fig. 5). The aim of this representation is to show whether the predicted compound belongs to a region of the chemical space well represented by the training series or if it belongs to a less populated region. In this representation, the training series compounds can be displayed as grey dots or colored by the biological annotation. The predicted compound can be displayed as green circles with the compound names, as red dots, or as dots colored by the compound distance to model (DModX, see [44]). A high DModX value indicates that the predicted compound has original features not present in the training series, which can be detrimental to the prediction quality.

Finally, it should be mentioned that the predictions are stored in a persistent prediction repository, and therefore, it is possible to revisit previous predictions until they are actively removed by the user.

Model management

Once a model is built, it is stored in a separate folder of the model repository. This folder can contain multiple versions of the model. As a minimum, there is a *dev* version that is used for model development and is overwritten every time the model is re-built. Precisely for this reason, the *dev* version cannot be used for prediction. Model versions that the model developer considers worth storing should be published to generate version 1, 2, etc.

The main GUI window shows a list (Fig. 8) where models can be browsed and selected. Every model is identified with a name and version and labeled by Maturity, Type, Subtype, Endpoint, and Species. The labels are defined by the end-user and can be used to filter the models shown, making it easier to find models for a particular endpoint, species, or organ.

The command mode interface and the GUI provide model management commands for creating new models, publishing a model version, deleting a whole model tree with all the versions or any single model version.

Models can be exported using a command that produces a compressed version of the whole model folder. This file can be easily stored, backed-up, or sent in electronic formats (e.g., as an e-mail attachment). Once imported in any Flame instance, the model is copied to the model repository and becomes fully functional. During the importing step, the versions of the software libraries used for generating the models are checked, and in case of version mismatches, a warning message is shown.

Model documentation

Flame models are documented using a template based on the QMRF [45], taking advantage of our previous experience in model documentation [22]. When the model is built, Flame automatically completes in this template the fields describing the modeling methodology and quality. This half-completed document should be edited by the

	Name	Ver	Maturity	Type	Subtype	Endpoint	Species
==	BZR	0	dev	Pharmacology	CNS	BZR modulation	unk
==	BZR	1	mature	Pharmacology	CNS	BZR modulation	unk
==	BZR	2	mature	Pharmacology	CNS	BZR modulation	unk
==	BZR	3	mature	Pharmacology	CNS	BZR modulation	unk
●	F7	0	dev	Pharmacology	Blood	F7 inhibition	unk
●	F7	1	dev	Pharmacology	Blood	F7 inhibition	unk
●	F7	2	dev	Pharmacology	Blood	F7 inhibition	unk
●	bsep_q	0	dev	Organ toxicity	Liver	BSEP inhibition	human
●	bsep_q	1	dev	Organ toxicity	Liver	BSEP inhibition	human
==	mitotox_q2	0	mature	Organ toxicity	Liver	Mitochondrial	unk

Fig. 8 The models present in the model repository are shown in the GUI as a model list. Items can be sorted, browsed, and searched by text terms. Models include user-defined labels which can be used to filter the list content

modeler, using the GUI or editing a documentation file in *yaml* format using a text editor and importing it into the model. In either case, the model documentation is stored in the model folder and is included when the model is exported or published.

The model documentation has been split into three sections: General Model information, Algorithms, and Other information. The first and third sections should be completed by the modeler, while Flame automatically completes most of the second section. The Additional file 1 contains an example of a human-readable file in *yaml* format, suitable for being imported into a Flame model, with all the items included in these three sections. The Additional file 2 contains a PDF file showing how the model documentation is presented to the user in the Flame GUI.

Performance

In a typical modeling workflow, the same code (structure normalization, molecular descriptors calculation) is run for every compound in the input series, both for training series and prediction series. Therefore, the computation can be speed-up by splitting the series into n sub-series and assigning them to different computation threads, which are run in different CPUs. Flame can run in parallel the workflow tasks related to the calculation of the molecular descriptors, obtaining nearly linear speed-up. Another time-consuming step is model building and validation. By default, Flame applies the parallel processing implemented in the ML libraries (e.g., scikit-learn implements parallel processing in cross-validation and grid-search, while XGBoost uses it in the model building and validation). The use of GPUs is under development. A special Flame version supporting GPUs is planned to be released in the future, facilitating the efficient use of deep learning within the framework.

Additionally, during model development, it is common to rebuild the model repeatedly using diverse machine learning settings to optimize them. To speed up this process, Flame stores intermediate results of the calculation (e.g., the molecular descriptors matrix), thus saving the work of re-computing them in every cycle.

Flame has been used to develop models using series of very different size and characteristics. To give an idea of Flame performance and limitations we have included Table 5 with some benchmarking results.

Error handling

Any modeling software aiming to solve real-life problems should know how to deal with errors present in the input files. These errors can stop the modeling workflow for many reasons: input molecules can have a wrong structure, contain metals, counterions or water molecules. The

Table 5 Computation time for series of diverse size

Series	Original series size	Compounds removed ^a	Final series size	Time (s) ^b
A	2685	11	2674	20 s
B	5832	0	5832	32 s
C	126,368	114	126,254	600 s (10 min)
D	480,000	249	479,751	2160 s (36 min)

Computation tasks involving structure normalization, computation of RDKit descriptors, generation of a XGBoost model, and validation using fivefold validation (for series A and B) and twofold validation (for series C and D).

^a Compounds removed from the computation because RDKit was unable to compute molecular descriptors

^b Wall clock times, in a desktop PC with Windows 10 professional 64 bits, 32b Gb RAM and an AMD Ryzen 5 3600(6 cores) CPU

model building can also fail when the annotations are not correct. For this reason, a lot of effort was devoted to implementing appropriate error handling methods in Flame, able to identify and remove automatically molecules that cannot be processed and producing informative error reporting both in the GUI and the console. As an example of Flame robustness, the D series in Table 5 contains 480,000 structures extracted directly from ChEMBL, with no curation. Flame was able to process the series removing automatically 249 structures (0.05%), for which RDKit was not able to generate molecular descriptors. Modelers know that there are many potential sources of error, and Flame does not claim to be able to handle all error types. However, years of development and use by different modeling teams established Flame as a rather robust software.

Comparison with other integrated modeling environments

As mentioned in the introduction, many tools for supporting the development of QSAR models are available. A comparison of Flame with a representative sample of related software would be helpful to highlight its advantages. This comparison is focused on software that can be installed locally, discarding purely on-line tools like CHEMBENCH [29] or OCHEM [30]. In many cases, these are excellent options, but in situations where the models should be trained with confidential data or used to predict confidential data, they are not usable. Indeed, Flame was developed specifically for supporting modeling activities in the project eTRANSafe, aiming to develop predictive models for the pharmaceutical industry using their confidential data. Another selection criterion is software accessibility. For example, OpenMol-GRID [46] was one of the first integrated modeling environments, but it is not accessible anymore. Also related to the accessibility, commercial software, and software

Table 6 Locally installable software usable as an integrated modeling environment

Name	Version	License	Platform	Language	Model prediction	Model building
eTOXlab [33]	0.9.6	GNU GPL-3	Any	Python/VM	Yes	Yes
VEGA-QSAR [49]	1.1.5.47	GNU GPL-3	Any	Java	Yes	No
EPI-suite [50]	4.11	Copyright EPA, free of use	Windows		Yes	No
Kausar Automated-framework [51]	na	na	Any	KNIME	Yes	Yes
ToxTree [52]	3.1.0	GNU GPL-2	Any	Java	Yes	No
OECD QSAR ToolBox [53]	4.4.1	requires registration	Windows		Yes	No

requiring registration or special agreements (e.g., QSA-RIN-chems [47] and ChemProp [48]) is excluded from the comparison, focusing our attention on open source freely accessible tools. After applying these criteria, we have selected the tools listed in Table 6 as a representative sample of the state-of-the-art, which does not intend to be exhaustive.

The first tool listed, eTOXlab [33] was developed in our group, and part of its conceptual design was reused in Flame. It is an integrated modeling framework developed in Python and distributed as a source code or pre-installed in a virtual machine. It can develop new models starting from an annotated SDF file, store models in a repository, and use them for prediction. As in Flame, models can include children of the source coded classes for model customization. However, eTOXlab offers limited features, and, for example, it cannot be used as a web service, has a more limited panel of methods, and its GUI is primitive. VEGA-QSAR [49] and EPI-suite [50] are prediction-oriented tools containing a collection of very useful models, but they lack the Flame ability to develop new models. The Kausar Automated framework for QSAR model building [51] is a fully featured collection of KNIME workflows for model development and prediction. However, it is oriented mainly to model developers and lacks an interface that makes it suitable for end-users. Moreover, KNIME is not open source, hampering its installation in non-academic environments.

The two remaining tools in the table are special cases. ToxTree [52] is a tool for the estimation of Toxic Hazard using only the decision tree approach. The OECD QSAR ToolBox [53], in spite of its name, is not specifically aimed to develop or apply QSAR models. Its scope is broader, oriented to obtain chemical hazard assessments by retrieving experimental data from internal databases, simulating metabolism, and profiling the chemical properties of chemicals. This information is then used for read-across, finding structurally and mechanistically defined analogs and chemical categories.

Discussion

The development of Flame was justified by the need for an integrated modeling framework in the eTRANSafe project, meeting its specific needs as well as providing pragmatic solutions to the general requirements of any predictive model listed in the introduction. How Flame addresses these requirements?

Reproducibility

Models generated and stored in Flame are fully reproducible across Flame instances and can be easily exported and imported, always obtaining the same results. The use of controlled Conda environments and the tagging of the library versions used during the model generation provides reasonable control of the software libraries and versions used. However, Flame cannot avoid the obsolescence of the software and hardware. For medium to long-term model storage, saving images of the whole system using docker or virtual machines is recommended.

Accessibility

Flame is open source and uses only open source software. It is available in the most popular operative systems (Linux, Windows, and iOS). It can be used as a desktop application with a rich GUI, from the command line, integrated into scripts, in Jupyter notebooks, or as a web service. The GUI was designed for non-expert users, but experienced modelers can customize the models without limitations. Additionally, Windows and Linux installers are distributed on the GitHub page to facilitate its installation by non-expert users. These installers are self-contained, including all the libraries needed to install and run Flame without an Internet connection. This is an essential feature for its installation in corporate environments where security is critical and Internet connection is either blocked or filtered.

Governance

Flame incorporates advanced model management tools, supporting the whole model development cycle. Models can be developed, improved, and stored in a persistent model repository, where they can be labeled using up to four types of keywords. Models are also thoroughly documented using widely accepted standards and given a unique ID. The documentation is organized in sections using a structure close to the layered approach proposed in the introduction.

Reporting

Flame predictions are presented to the users in a variety of formats, some of them specifically designed to facilitate the interpretation by non-expert users, providing contextual information about the biological annotations and the result interpretation. Whenever the model allows, the prediction result is presented with information about its uncertainty, using rigorous formalisms (e.g., conformal regression) expressed in formats familiar to experimentalists (confidence intervals).

Conclusions

We presented Flame, an open source modeling framework that can be used for the easy development of QSAR-like models. The incorporated model building workflow only requires the input of a single annotated SDF file to generate a model, using default options. This workflow can be easily customized to use any of the natively supported methods and a wide variety of method parameters. Moreover, it incorporates mechanisms to implement unlimited customization by using model-linked source code overriding.

Many predictive modeling applications depend critically on addressing implementation issues that hinder the use of models in production environments. Our modeling framework provides reasonable solutions for most of these issues and facilitates a seamless transition from model development to model production with little effort. Models can be easily maintained, stored, exported, and imported, facilitating the collaboration between academic and private institutions.

Flame uses innovative methods to combine models by building models based on the results of other models. This adds unique flexibility for combining multiple models addressing the same endpoint or combining models representing multiple mechanisms contributing to the same endpoint in the toxicological field. Some interesting applications of this model combination tool have been obtained and will be published in due time.

Flame incorporates a rich web-based GUI, facilitating the model building, administration, and use in prediction.

Prediction results are presented to the user in various formats, including information like the substances in the training series closer to the predicted compound or projections of the query compounds on the training series chemical space.

Flame has been developed within eTRANSAFE, a large European project involving numerous pharmaceutical companies, some of which are testing Flame internally. The feedback obtained in this interaction has been a precious resource for designing a tool that can help drug developers and drug safety experts in their daily work.

The comparison with similar tools, installable locally, was favorable to Flame and highlighted some of its advantages. If the comparison is extended to include on-line services like CHEMBENCH and OCHEM, these platforms outperform Flame in some aspects, even if they lack the advanced model customization offered by locally installable tools.

For all these reasons, Flame can be considered a very useful tool with unique features. As yet, it does not incorporate all the modeling tools available, but we plan to keep enriching their features, incorporating other molecular descriptor generators and machine learning toolkits. In this respect, we plan to expand the Flame user's community beyond the eTRANSAFE consortium and interest developers that can contribute their code in future versions.

Supplementary Information

The online version contains supplementary material available at <https://doi.org/10.1186/s13321-021-00509-z>.

Additional file 1. Model documentation exported in human-readable yaml format that can be edited and imported.

Additional file 2. Large figure showing the aspect of the model documentation GUI in Flame.

Acknowledgements

We wish to thank Thomas Steger-Hartmann (Bayer AG, Pharmaceuticals, Berlin), Francois Pognan and Nils Oberhauser (Novartis, Basel) for manuscript review and valuable comments.

Authors' contributions

MP coded most of the Flame software and prepared the manuscript. JCGT wrote significant sections of the software and contributed to the manuscript. FS contributed to the software concept and design and reviewed the manuscript. All authors read and approved the final manuscript.

Authors' information

MP is the head of the Pharmacoinformatics laboratory at the GRIB (UPF), and leader of modeling related work packages at eTRANSAFE project. JCGT is a post-doctoral fellow at the MP's Pharmacoinformatics laboratory. FS is the head of the GRIB and eTRANSAFE academic coordinator.

Funding

This work has received funding from the eTRANSAFE project (Grant Agreement No. 777365), developed under the Innovative Medicines Initiative Joint Undertaking (IMI2), resources of which are composed of a financial contribution from the European Union's Seventh Framework Programme

(FP7/2007–2013) and EFPIA companies' in kind contributions. The authors of this article are also involved in other related IMI projects which contributed funding, such as TransQST (No. 116030) as well as the H2020 EU-ToxRisk project (No. 681002) and FAIRplus (No. 802750). The Research Programme on Biomedical Informatics (GRIB) is a member of the Spanish National Bioinformatics Institute (INB), funded by ISCIII and FEDER (PT17/0009/0014). The DCEXS is a 'Unidad de Excelencia María de Maeztu', funded by the AEI (CEX2018-000782-M). The GRIB is also supported by the Agència de Gestió d'Ajuts Universitaris i de Recerca (AGAUR), Generalitat de Catalunya (2017 SGR 00519).

Availability of data and materials

Flame source code is available at GitHub under GNU GPL-3.0 license at the following repositories: <https://github.com/phi-grib/flame> (backend), https://github.com/phi-grib/flame_API (web server), <https://github.com/phi-grib/flameWeb2> (frontend). No dataset was described nor required to support the conclusions of the manuscript.

Project name: Flame. Project home page: <https://github.com/phi-grib/flame> (backend), https://github.com/phi-grib/flame_API (web server), <https://github.com/phi-grib/flameWeb2> (frontend). Operating system(s): Platform independent. Tested in Windows, Linux, and iOS. Programming language: Python, Typescript (Angular). Other requirements: Flame uses a Conda environment defining dependencies to other Python libraries. License: GNU GPL-3.0

Declarations

Competing interests

The authors declare that they have no competing interests.

Received: 10 November 2020 Accepted: 8 April 2021

Published online: 19 April 2021

References

- Kim S, Thiessen PA, Bolton EE et al (2016) PubChem substance and compound databases. *Nucleic Acids Res* 44:D1202–D1213. <https://doi.org/10.1093/nar/gkv951>
- Gaulton A, Hersey A, Nowotka ML et al (2017) The ChEMBL database in 2017. *Nucleic Acids Res* 45:D945–D954. <https://doi.org/10.1093/nar/gkw1074>
- Sanz F, Pognan F, Steger-Hartmann T, Díaz C (2017) Legacy data sharing to improve drug safety assessment: the eTOX project. *Nat Rev Drug Discov* 16:811–812. <https://doi.org/10.1038/nrd.2017.177>
- eTOX dashboard. <https://etoxsys.eu/etoxsys.v3-demo-bk/dashboard>. Accessed 11 Feb 2021
- Wilkinson MD, Dumontier M, Ilij A et al (2016) The FAIR guiding principles for scientific data management and stewardship. *Sci Data* 3:160018. <https://doi.org/10.1038/sdata.2016.18>
- Cherkasov A, Muratov EN, Fourches D et al (2014) QSAR modeling: where have you been? Where are you going to? *J Med Chem* 57:4977–5010. <https://doi.org/10.1021/jm4004285>
- Muratov EN, Bajorath J, Sheridan RP et al (2020) QSAR without borders. *Chem Soc Rev* 49:3525–3564. <https://doi.org/10.1039/D0CS00098A>
- Rifaioğlu AS, Atas H, Martin MJ et al (2019) Recent applications of deep learning and machine intelligence on in silico drug discovery: methods, tools and databases. *Brief Bioinform* 20:1878–1912. <https://doi.org/10.1093/bib/bby061>
- Varoquaux G, Buitinck L, Louppe G et al (2015) Scikit-learn. *GetMobile Mob. Comput Commun* 19:29–33
- Vamathevan J, Clark D, Czodrowski P et al (2019) Applications of machine learning in drug discovery and development. *Nat Rev Drug Discov* 18:463–477. <https://doi.org/10.1038/s41573-019-0024-5>
- Luechtefeld T, Rowlands C, Hartung T (2018) Big-data and machine learning to revamp computational toxicology and its use in risk assessment. *Toxicol Res* 7:732–744. <https://doi.org/10.1039/c8tx00051d>
- Luechtefeld T, Marsh D, Rowlands C, Hartung T (2018) Machine learning of toxicological big data enables read-across structure activity relationships (RASAR) outperforming animal test reproducibility. *Toxicol Sci* 165:198–212. <https://doi.org/10.1093/toxsci/kfy152>
- Rabesandratana T (2016) A crystal ball for chemical safety. *Science* (80-) 351:651. <https://doi.org/10.1126/science.351.6274.651>
- Grapov D, Fahrman J, Wanichthanarak K, Khoomrung S (2018) Rise of deep learning for genomic, proteomic, and metabolomic data integration in precision medicine. *OMICS* 22:630–636. <https://doi.org/10.1089/omi.2018.0097>
- Abraham A, Pedregosa F, Eickenberg M et al (2014) Machine learning for neuroimaging with scikit-learn. *Front Neuroinform* 8:14. <https://doi.org/10.3389/fninf.2014.00014>
- Shen D, Wu G, Suk H-I (2017) Deep learning in medical image analysis. *Annu Rev Biomed Eng* 19:221–248. <https://doi.org/10.1146/annurev-bioeng-071516-044442>
- Palczewska A, Fu X, Trundle P et al (2013) Towards model governance in predictive toxicology. *Int J Inf Manag* 33:567–582. <https://doi.org/10.1016/j.jinfomgt.2013.02.005>
- Spijth O, Willighagen EL, Guha R et al (2010) Towards interoperable and reproducible QSAR analyses: exchange of datasets. *J Cheminform* 2:1–7. <https://doi.org/10.1186/1758-2946-2-5>
- Patel M, Chilton ML, Sartini A et al (2018) Assessment and reproducibility of quantitative structure–activity relationship models by the nonexpert. *J Chem Inf Model* 58:673–682. <https://doi.org/10.1021/acs.jcim.7b00523>
- Commission E (2018) Turning FAIR into reality. <https://op.europa.eu/s/oLAK>. Accessed 11 Feb 2021
- Roy Thomas Fielding (2000) Architectural styles and the design of network-based software architectures. University of California, Irvine
- Pastor M, Quintana J, Sanz F (2018) Development of an infrastructure for the prediction of biological endpoints in industrial environments. Lessons learned at the eTOX project. *Front Pharmacol* 9:1–8. <https://doi.org/10.3389/fphar.2018.01147>
- Rovida C (2020) Internationalization of read-across as a validated new approach method (NAM) for regulatory toxicology. *Altex*. <https://doi.org/10.14573/altex.1912181>
- Tetko IV, AksenoVA TI, Volkovich VV et al (2000) Polynomial neural network for linear and non-linear model selection in quantitative-structure activity relationship studies on the internet. *SAR QSAR Environ Res* 11:263–280. <https://doi.org/10.1080/1062936000803235>
- Tetko IV, Maran U, Tropsha A (2017) Public (Q)SAR services, integrated modeling environments, and model repositories on the web: state of the art and perspectives for future development. *Mol Inform* 36:1600082. <https://doi.org/10.1002/minf.201600082>
- Kluyver T, Ragan-Kelley B, Pérez F et al (2016) Jupyter notebooks—a publishing format for reproducible computational workflows. In: Loizides F, Schmidt B (eds) Positioning and power in academic publishing players, agents and agendas. IOS Press, Amsterdam, pp 87–90
- Django project. <https://www.djangoproject.com/>
- Angular. <https://angular.io/>
- Capuzzi SJ, Kim IS-J, Lam WI et al (2017) Chembench: a publicly accessible, integrated cheminformatics portal. *J Chem Inf Model* 57:105–108. <https://doi.org/10.1021/acs.jcim.6b00462>
- Sushko I, Novotarskyi S, Körner R et al (2011) Online chemical modeling environment (OCHEM): web platform for data storage, model development and publishing of chemical information. *J Comput Aided Mol Des* 25:533–554. <https://doi.org/10.1007/s10822-011-9440-2>
- Conda. <https://docs.conda.io/projects/conda/en/latest/>
- scikit-learn estimator term definition. <https://scikit-learn.org/stable/glossary.html#term-estimator>
- Carrío P, López O, Sanz F, Pastor M (2015) eTOXlab, an open source modeling framework for implementing predictive models in production environments. *J Cheminform* 7:8. <https://doi.org/10.1186/s13321-015-0058-6>
- Atkinson F standardizer. <https://github.com/flatkinson/standardiser>
- ChEMBL standardiser. https://github.com/chembl/ChEMBL_Structure_Pipeline
- Bento AP, Hersey A, Félix E et al (2020) An open source chemical structure curation pipeline using RDKit. *J Cheminform* 12:1–16. <https://doi.org/10.1186/s13321-020-00456-1>
- RDKit: open-source cheminformatics software. <https://www.rdkit.org/>
- Ho TK (1995) Random decision forests. *Proc Int Conf Doc Anal Recognit* 1:278–282. <https://doi.org/10.1109/ICDAR.1995.598994>
- Cortes C, Vapnik V (1995) Support-vector networks. *Mach Learn* 20:273–297. <https://doi.org/10.1023/A:1022627411411>

40. Wold S, Johansson ECM (1993) PLS—partial least squares projections to latent structures. In: Kubinyi H (ed) 3D-QSAR in drug design, theory, methods, and applications. ESCOM, Leiden, pp 523–550
41. Sharma N (2018) XGBoost. The extreme gradient boosting for mining applications. GRIN Verlag
42. Norinder U, Carlsson L, Boyer S, Eklund M (2015) Introducing conformal prediction in predictive modeling for regulatory purposes. A transparent and flexible alternative to applicability domain determination. *Regul Toxicol Pharmacol* 71:279–284. <https://doi.org/10.1016/j.yrtph.2014.12.021>
43. Non-conformist. <https://github.com/donlnz/nonconformist>
44. Eriksson L, Andersson PL, Johansson E, Tysklind M (2006) Megavariate analysis of environmental QSAR data. Part I—a basic framework founded on principal component analysis (PCA), partial least squares (PLS), and statistical molecular design (SMD). *Mol Divers* 10:169–186. <https://doi.org/10.1007/s11030-006-9024-6>
45. European Commission JRC-IHCP QSAR model reporting format. <https://publications.jrc.ec.europa.eu/repository/bitstream/JRC107491/kjna28713enn.pdf>. Accessed 11 Feb 2021
46. Sild S, Maran U, Lomaka A, Karelson M (2006) Open computing grid for molecular science and engineering. *J Chem Inf Model* 46:953–959. <https://doi.org/10.1021/ci050354f>
47. Gramatica P, Cassani S, Chirico N (2014) QSARINS-chem: insubria datasets and new QSAR/QSPR models for environmental pollutants in QSARINS. *J Comput Chem* 35:1036–1044. <https://doi.org/10.1002/jcc.23576>
48. UFZ Department of Ecological Chemistry (2019) ChemProp 6.7.1. <http://www.ufz.de/ecochem/chemprop>
49. Benfenati E, Manganaro A, Gini G (2013) VEGA-QSAR: AI inside a platform for predictive toxicology. In: Proceedings of the workshop “popularize artificial intelligence 2013”. Turin, Italy
50. EPA U (2021) Estimation Programs Interface Suite™ for Microsoft® Windows, v 4.11
51. Kausar S, Falcao AO (2018) An automated framework for QSAR model building. *J Cheminform* 10:1–23. <https://doi.org/10.1186/s13321-017-0256-5>
52. Patlewicz G, Jeliakova N, Safford RJ et al (2008) An evaluation of the implementation of the Cramer classification scheme in the Toxtree software. *SAR QSAR Environ Res* 19:495–524. <https://doi.org/10.1080/10629360802083871>
53. OECD QSAR ToolBox. <https://qsartoolbox.org/>

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Ready to submit your research? Choose BMC and benefit from:

- fast, convenient online submission
- thorough peer review by experienced researchers in your field
- rapid publication on acceptance
- support for research data, including large and complex data types
- gold Open Access which fosters wider collaboration and increased citations
- maximum visibility for your research: over 100M website views per year

At BMC, research is always in progress.

Learn more biomedcentral.com/submissions

