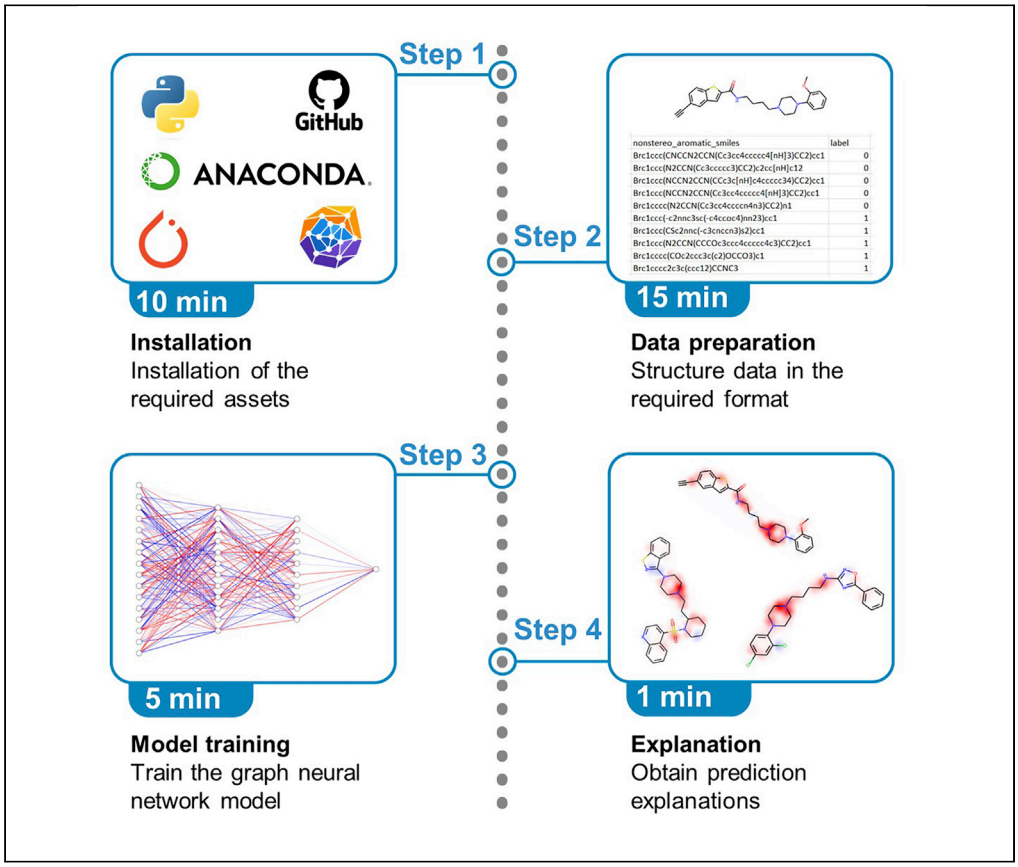
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol to explain graph neural network predictions using an edge-centric Shapley value-based approach



Here we present EdgeSHAPer, a workflow for explaining graph neural networks by approximating Shapley values using Monte Carlo sampling. In this protocol, we describe steps to execute Python scripts for a chemical dataset from the original publication; however, this approach is also applicable to any user-provided dataset. We also detail steps encompassing neural network training, an explanation phase, and analysis via feature mapping.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Andrea Mastropietro, Giuseppe Pasculli, Jürgen Bajorath

mastropietro@diag.uniroma1.it (A.M.)
bajorath@bit.uni-bonn.de (J.B.)

Highlights

Explain graph neural network models with the EdgeSHAPer approach

Install the custom code and prepare the input data

Train a graph neural network

Execute the explainer module and analyze the results

Mastropietro et al., STAR Protocols 3, 101887 December 16, 2022
https://doi.org/10.1016/j.xpro.2022.101887

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol to explain graph neural network predictions using an edge-centric Shapley value-based approach

Andrea Mastropietro,[1,2,3,*] Giuseppe Pasculli,[1] and Jürgen Bajorath[2,4,*]

[1]Department of Computer, Control and Management Engineering Antonio Ruberti (DIAG) Sapienza University of Rome, 00185 Rome, Italy

[2]Department of Life Science Informatics and Data Science, B-IT, LIMES Program Unit Chemical Biology and Medicinal Chemistry, Rheinische Friedrich-Wilhelms-Universität, Friedrich-Hirzebruch-Allee 5/6, 53115 Bonn, Germany

[3]Technical contact

[4]Lead contact

*Correspondence: mastropietro@diag.uniroma1.it (A.M.), bajorath@bit.uni-bonn.de (J.B.)
https://doi.org/10.1016/j.xpro.2022.101887

SUMMARY

Here we present EdgeSHAPer, a workflow for explaining graph neural networks by approximating Shapley values using Monte Carlo sampling. In this protocol, we describe steps to execute Python scripts for a chemical dataset from the original publication; however, this approach is also applicable to any user-provided dataset. We also detail steps encompassing neural network training, an explanation phase, and analysis via feature mapping.

For complete details on the use and execution of this protocol, please refer to Mastropietro et al. (2022).[1]

BEFORE YOU BEGIN

This protocol details the use of EdgeSHAPer, an explanation method for any graph neural network (GNN) that relies on a Monte Carlo sampling procedure for the approximation of Shapley values, which are used to quantify edge importance. The software was developed in the Windows environment but is also usable under Linux and macOS. The method was implemented using Python code with the aid of several deep learning libraries such as PyTorch[4] and PyTorch Geometric.[5] Below we illustrate the installation and workflow with compound data from the original EdgeSHAPer publication[1] as well as with custom data. The installation and execution times reported are based on a machine with the capabilities listed in the materials and equipment section. Using a different system will likely alter the performances and execution times.

Installation

Timing: 10 min

1. Install Python 3 (version 3.8 tested and recommended) and the required packages creating a conda virtual environment (suggested):
   a. Install Anaconda from https://www.anaconda.com/products/distribution.
   b. Download or clone the original GitHub repository from https://github.com/AndMastro/EdgeSHAPer:

```
>git clone https://github.com/AndMastro/EdgeSHAPer.git
```

Note: Git should be installed to run the former command.

c. Create a brand-new conda environment using the chosen *.yml* file provided in the repository containing all the necessary libraries:
   i. Open the file *.yml* corresponding to the desired PyTorch and CUDA version and edit the parameter *prefix* with your conda environments folder.
   ii. Open a terminal of your choice in the repository folder (the Anaconda Prompt is suitable).
   iii. Run the following command:

```
>conda env create -f edgeshaper_*.yml
```

Note: We provide *.yml* files with different versions of PyTorch and CUDA. Further versions may be compatible but not tested yet. Choose the proper versions according to your machine and GPU capabilities.

Alternatives: Instead of using Anaconda, one can use a local Python installation and add the required packages using *pip*. The list of the required packages can be found in the key resources table. The libraries can be installed via:

```
>pip install name_of_package
```

Troubleshooting 1.

2. Install the additional module required for the visualization by running the command:

```
>pip install git+git://github.com/c-feldmann/rdkit_heatmaps
```

Troubleshooting 2.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Compound activity data	ChEMBL 30	ChEMBL: https://doi.org/10.6019/CHEMBL.database.30
Putative aggregators	Aggregator Advisor	http://advisor.docking.org/faq/#Data
Data sets	Mastropietro et al.[1] Feldmann et al.[3]	https://github.com/AndMastro/EdgeSHAPer/tree/main/experiments/data Mendeley Data: https://doi.org/10.17632/bs6myg75tr.2
Software and algorithms		
RDKit 2021.09.4	Zenodo	Zenodo: https://doi.org/10.5281/zenodo.6605135
Lilly-Medchem-Rules	GitHub	https://github.com/IanAWatson/Lilly-Medchem-Rules
EdgeSHAPer	This paper Mastropietro et al.[1] Zenodo[2]	https://github.com/AndMastro/EdgeSHAPer Zenodo: https://doi.org/10.5281/zenodo.7267068
scikit-learn 1.0.2	GitHub	https://github.com/scikit-learn/scikit-learn
PyTorch 1.10.1 or PyTorch 1.10.1_cuda10.2 or PyTorch 1.12.1 or PyTorch 1.12.1_cuda11.6 (use CUDA versions if a GPU is available)	GitHub	https://github.com/pytorch/pytorch
rdkit-heatmap 0.1	GitHub	https://github.com/c-feldmann/rdkit_heatmaps

(*Continued on next page*)

Continued		
REAGENT or RESOURCE	SOURCE	IDENTIFIER
matplotlib 3.5.1	GitHub	https://github.com/matplotlib/matplotlib
networkx 2.6.3	GitHub	https://github.com/networkx/networkx
numpy 1.22.0	GitHub	https://github.com/numpy/numpy
tqdm 4.62.3	GitHub	https://github.com/tqdm/tqdm
pyg 2.0.3	GitHub	https://github.com/pyg-team/pytorch_geometric
torchdrug 0.1.2	GitHub	https://github.com/DeepGraphLearning/torchdrug
pandas 1.3.5	GitHub	https://github.com/pandas-dev/pandas
Other		
Intel Core i7-12700H @ max 4.70 GHz CPU	N/A	N/A
NVIDIA GeForce RTX 3060 Laptop GPU	N/A	N/A
16 GB RAM	N/A	N/A
Windows/Linux/macOS Operating System	N/A	N/A

MATERIALS AND EQUIPMENT

Computational resources		
Component	Brand	Model/capabilities/version
CPU	Intel	Core i7-12700H @ max 4.70 GHz
GPU	NVIDIA	GeForce RTX 3060 Laptop (6 GB)
RAM	N/A	16 GB
Operating System	Windows/Linux/macOS	11/Ubuntu 22.04/Catalina

Alternatives: The protocol was also tested using a machine with Windows 10, Intel Core i7700HQ @ max 3.8 GHz, NVIDIA GeForce GTX 1060 Max-Q with 6 GB of dedicated memory, and 16 GB of RAM. The execution times are longer, as expected, but the software is readily usable. Different configurations are expected to be suitable as well, with likely performance differences.

STEP-BY-STEP METHOD DETAILS

EdgeSHAPer is applicable to a trained GNN model. We show how to derive a four-layer graph convolutional network (GCN)[6] and then use EdgeSHAPer to explain predictions for an input graph of choice. Initially, we present a step-by-step guide on how to use EdgeSHAPer on chemical compounds encoded as SMILES strings[7] and the resulting molecular graphs to which the method was originally applied. Then, we show how to import the module into any Python program for custom data and tasks.

Data preparation

Timing: 15 min

The first step consists of data preparation. A specific format is required and must be generated (manual step).

1. The data need to be formatted as a comma separated value (*.csv*) file, in which the SMILES string and label of each compound must be present (any other attribute will be ignored by the program) (Figure 1 provides an example):
   a. Create a *.csv* file with the required format containing all the molecules.
   b. Place the file in a folder of interest.

nonstereo_aromatic_smiles	label
Brc1ccc(CNCCN2CCN(Cc3cc4ccccc4[nH]3)CC2)cc1	0
Brc1ccc(N2CCN(Cc3ccccc3)CC2)c2cc[nH]c12	0
Brc1ccc(NCCN2CCN(CCc3c[nH]c4ccccc34)CC2)cc1	0
Brc1ccc(NCCN2CCN(Cc3cc4ccccc4[nH]3)CC2)cc1	0
Brc1cccc(N2CCN(Cc3cc4ccccn4n3)CC2)n1	0
Brc1ccc(-c2nnc3sc(-c4ccoc4)nn23)cc1	1
Brc1ccc(CSc2nnc(-c3cnccn3)s2)cc1	1
Brc1ccc(N2CCN(CCCOc3ccc4ccccc4c3)CC2)cc1	1
Brc1cccc(COc2ccc3c(c2)OCCO3)c1	1
Brc1cccc2c3c(ccc12)CCNC3	1

Figure 1. Excerpt of a suitable input file
The columns in this file simply indicate the SMILES field and the label field. Custom names instead must be specified in the scripts. In our case study, we have binary labels stating the compound to be active (0) or inactive (1) (consistent with standard programming indexing) against a target of interest (in this case, the dopamine D2 receptor). If additional columns are present, they are ignored (not used in the code).

2. Open the *parameters.yml* file which contains configurable parameters for the trainer and explainer scripts. Edit this file based on needs.

   Optional: It is also possible to create *.txt* files listing compounds used as training, validation, and test sets. Here, the molecules are identified by their SMILES strings, separated by a new line. An exemplary file is shown in Figure 2 (having same format for training, validation, and test sets).

   Note: The repository contains data from the original publication,[1] which can be used to test the algorithm.

Graph neural network training

Timing: 5 min

This step is required for training a GNN. We provide a script to derive a four-layer GCN. However, this step can be omitted if a GNN model is already available. In this case, please, refer to the use of EdgeSHAPer via custom code as an alternative execution (see below).

3. Run the script for model training:
   a. Open a terminal in the repository root folder.
   b. If not already active, activate the conda environment:

```
>conda activate edgeshaper_env
```

   Troubleshooting 3.
   c. Run the *trainer_script.py*:

```
>python trainer_script.py
```

   The script will load the arguments contained in the configuration file *parameters.yml*. This file may be subject to change with future developments, so please, refer to the GitHub repository for an up-to-date version. The current editable and main parameters include:

```
O=C(NC1CCC(CCN2CCC(c3cccc4c3CCO4)CC2)CC1)c1ccc(N2CCOCC2)nc1
COc1ccccc1COc1ccc(S(=O)(=O)N2CCCC(C)(O)C2C(=O)NO)cc1
CC[n+]1ccc(CCC(=O)C2Cc3cc(OC)c(OC)cc3S2)cc1
O=C(Nc1ccc(CCN2CCN(c3ccccc3)CC2)cc1)Nc1cccnc1
CC#CCOc1cnc(C2=Nc3cc(C4(C)CC(C(F)(F)F)OC(N)=N4)c(F)cc3CN2)cn1
CCCN1CCN(c2ccc(O)c3[nH]c(=O)ccc23)CC1
CCCC=C1Oc2ccc(F)cc2-c2ccc3c(c21)C(C)=CC(C)(C)N3
Oc1ccc2c(c1)CCCNCCc1ccccc1C2
CN1CCN(C2=Nc3cc(Cl)ccc3N(NC(=O)c3ncccc4ccccc34)c3ccccc32)CC1
```

Figure 2. Excerpt of a file for the specification of training, validation, and test sets
All three files have the same format (SMILES strings separated by a new line).

i. DATA_FILE: your dataset in *.csv* format.
ii. TRAIN_DATA_FILE (optional): *.txt* file with training samples.
iii. VALIDATION_DATA_FILE (optional): *.txt* file with validation samples.
iv. TEST_DATA_FILE (optional): *.txt* file with test samples.
v. SAVE_FOLDER_DATA_SPLIT (optional): the folder path where to save the data split into training, validation, and test sets. Three *.txt* files will be generated if this option is used.
vi. SMILES_FIELD_NAME: column name for the SMILES field in DATA_FILE.
vii. LABEL_FIELD_NAME: column name for the label field in DATA_FILE.
viii. MODEL_SAVE_FOLDER: location in which the trained model will be saved.
ix. HIDDEN_CHANNELS: number of hidden channels used for the neural network.
x. BATCH_SIZE: batch size used for neural network training.
xi. EPOCHS: number of epochs for which the network will be trained.
xii. SEED (optional): seed for the random number generator.

Note: At the time of writing, the default values found in the *parameters.yml* file are the ones used in Mastropietro et al.[1]

Note: If training, validation, and test data are not provided, the complete dataset will be divided into training, validation, and test sets according to an 80%:10%:10% ratio. In this case, it might be useful to specify the parameter SAVE_FOLDER_DATA_SPLIT in order to save the resulting data split. The generated files have the same format as the file in Figure 2.

EdgeSHAPer explanation execution

Timing: 1–1.5 min per molecule

The last step facilitates the execution of the EdgeSHAPer explainability module. The user is required to submit a *.txt* file containing the names of the compounds whose predictions should be explained, separated by a new line.

⚠ CRITICAL: The molecules to be explained should not be contained in the training or validation sets according to standard machine learning practice.

4. Run the explainer script:
   a. With a terminal opened in the repository root folder, run:

```
>python explainer_script.py
```

The parameters read from the *parameters.yml* file include:
i. MODEL: model file to load.
ii. DATA_FILE: your dataset in .csv format.
iii. MOLECULES_TO_EXPLAIN: *.txt* file with the SMILES strings of the molecules to explain.
iv. TARGET_CLASS: class label for which the explanation should be computed.
v. SMILES_FIELD_NAME: column name for the SMILES field in DATA_FILE.
vi. LABEL_FIELD_NAME: column name for the label field in DATA_FILE.
vii. MINIMAL_SETS: Boolean flag stating whether to compute minimal informative sets (full details are provided in the original publication).
viii. SAVE_FOLDER_PATH: folder path where to save the explanations, along with additional information.
ix. HIDDEN_CHANNELS: number of hidden channels in the network to be loaded.
x. SAMPLING_STEPS: number of Monte Carlo sampling steps.

xi. VISUALIZATION: Boolean flag specifying whether to generate visualizations for the generated explanations (which will be saved in SAVE_FOLDER_PATH).
xii. TOLERANCE (optional): permitted deviation between the predicted probability and sum of Shapley values approximation.
xiii. SEED (optional): seed for the random number generator.

Alternatives: The previous step is used to run the EdgeSHAPer algorithm from a script. However, EdgeSHAPer can also be imported into a project, enabling customizable applications. To use EdgeSHAPer as a component of custom Python code, proceed as follows:

```
from edgeshaper import edgeshaper

model = YOUR_GNN_MODEL

edge_index = YOUR_GRAPH_EDGE_INDEX

x = YOUR_GRAPH_NODES_FEATURES

device = "cuda" # or "cpu"

target_class = YOUR_TARGET_CLASS

edges_explanations = edgeshaper(model, x, edge_index, M = 100, target_class = TARGET_CLASS,
device = device)
```

The *edgeshaper* function returns a Python list containing Shapley values for the edges in the same order as in the provided *edge_index*. This allows a user to freely execute EdgeSHAPer in custom code, providing high flexibility and the possibility of explaining predictions in applicability domains other than cheminformatics. The parameter *model* denotes the pre-trained GNN model used for the prediction, *x* and *edge_index* are the respective features of the graph nodes and the edge index indicating the links among nodes. *M* is the number of Monte Carlo sampling steps to perform and *target_class* is the class label for which the explanation is performed. Finally, *device* indicates whether the model should run on GPU for hardware acceleration or on CPU. Further details concerning additional accepted parameters are provided in the GitHub repository.

A second alternative is the use of the provided *Edgeshaper* class, which offers additional functionalities. First, instantiate an *Edgeshaper* object, then call its methods:

```
from edgeshaper import Edgeshaper

edgeshaper_explainer = Edgeshaper(model, x, edge_index, device = device)

edges_explanations = edgeshaper_explainer.explain(M = 100, target_class = TARGET_CLASS, P =
None, deviation = TOLERANCE, log_odds = False, seed = SEED)
```

The method *explain* applies the EdgeSHAPer algorithm and returns a list of the Shapley values calculated for each edge (the order is consistent with the one in *edge_index*). *P* is a parameter used for the generation of the random graphs (indicating the edge existence probability) for Monte Carlo sampling; passing *None* will use the graph density of the explained graph as the default probability (more details are reported in the original publication[1]). The parameter *deviation* can be used to set a permitted deviation of the sum of the Shapley values from the predicted probability. The default setting is *None*, corresponding to no predefined deviation and performing the requested Monte Carlo sampling steps *M*. Further parameters include *log_odds* and *seed*. The former is a Boolean flag set to use log odds instead of probability as the target for the Shapley value approximation. The latter represents the optional seed for the random number generator.

Additional methods provided by the *Edgeshaper* class are *compute_pertinent_positivite_set* and *compute_minimal_top_k_set*, which return the minimal informative sets from the explanations along with Infidelity and Fidelity scores, respectively. Let us consider the following example:

```
pert_positive_set, infidelity_score = edgeshaper_explainer.compute_pertinent_positivite_
set()

minimal_top_k_set, fidelity_score = edgeshaper_explainer.compute_minimal_top_k_set()
```

Then, after computing explanations for the compound and optionally minimal sets, they can be visualized with the method *visualize_molecule_explanations*:

```
edgeshaper_explainer.visualize_molecule_explanations(smiles, save_path = SAVE_PATH, per-
tinent_positive = True, minimal_top_k = True)
```

This method relies on several parameters. The first parameter is *smiles*, which contains the SMILES strings of the molecule to be explained and the second one is *save_path*, which indicates the folder where to save the generated images. Finally, the parameters *pertinent_positive* and *minimal_top_k_sets* determine whether visualizations are also created for the minimal informative sets.

Note: The GitHub repository is continuously maintained and updated. Hence, settings and scripts are modified periodically to improve the algorithm and enhance the number of features provided. However, to ensure reproducibility, we keep an active branch in the repository named *protocol*, representing a snapshot of the protocol presented here. A README file states if the main branch is up-to-date or if it has recently been edited. Apart from such updates, general use instructions will remain valid including the main up-to-date branch.

EXPECTED OUTCOMES

For each input compound, the output of EdgeSHAPer includes several files. The first output file is a *.txt* file with Shapley values importance scores for each edge in the graph (e.g., bond in a molecular graph) along with additional information in accordance with specified parameter settings. An excerpt of this file is presented in Figure 3.

This file contains information about the explanations. It reports the class for which the explanation was carried out and the SMILES string of the explained molecule. Importantly, the Shapley value for each edge (indicated by its index) and the sum of the Shapley values themselves are given. Finally, the indices comprising the minimal informative sets are reported together with Fidelity (FID+) and Infidelity (FID-) scores. The latter scores are used to evaluate the quality of an explanation. Ideally, a model should achieve high Fidelity and low Infidelity values. Additional details concerning these evaluation metrics can be found in the original paper.[1]

Furthermore, a series of high-resolution *.png* images highlighting the bond with color gradients according to their importance magnitude is obtained, together with minimal informative sets, if requested (for more details, see the original work[1] and the corresponding GitHub repository). Exemplary output images are shown in Figure 4.

```
Explaining class 0 for compound: O=S1(=O)c2cccc3cccc(c23)N1CCCCCCN1CCN(c2ccccn2)CC1

Shapley values for edges:
(0,1): 0.004804589437952713
(1,0): 0.037527296611804414
(1,2): 0.021103782884184873
(1,3): -0.052546348327925124
(1,13): -0.040318356688768466
(2,1): -0.022028606948458616
(3,1): .....................
.............................
.............................
(31,20): 0.0486218152587206
(31,30): 0.0022247345312195093

Sum of Shapley values: 0.9590877693201229

Minimal top k set edge index:
[[20, 20, 21], [31, 21, 20]]

FID+: 0.9269812181591988

Pertinent positive set edge index:
[[20, 20, 21, 23, 3, 20, 23], [31, 21, 20, 22, 1, 19, 30]]

FID-: 0.3659490942955017
```

Figure 3. Excerpt of a file generated using *explainer_script.py*

This file contains explanation results such as the Shapley values assigned to each edge and minimal informative sets. Additional information might be added in future updates.

QUANTIFICATION AND STATISTICAL ANALYSIS

A variance and convergence analysis of EdgeSHAPer was reported in the original work to study the evolution of the Shapley value approximation for increasing numbers of sampling steps. In this analysis, 100 steps were found to be sufficient for obtaining high-quality approximations. Figure 5 shows the (A) variance and (B) error for the sum of the Shapley values for all edges of a test compound.

LIMITATIONS

EdgeSHAPer relies on a Monte Carlo sampling approach for the approximation of Shapley values. Thus, the magnitude of importance values might slightly vary across multiple runs employing different seeds for random number generators, given the intrinsic stochastic nature of the method.

TROUBLESHOOTING

Problem 1

Related to "installation". The environment files were generated and tested under a Windows system. The installation via the *.yml* file may fail while using a different operating system.

Potential solution

Install the required packages using the *pip* alternative.

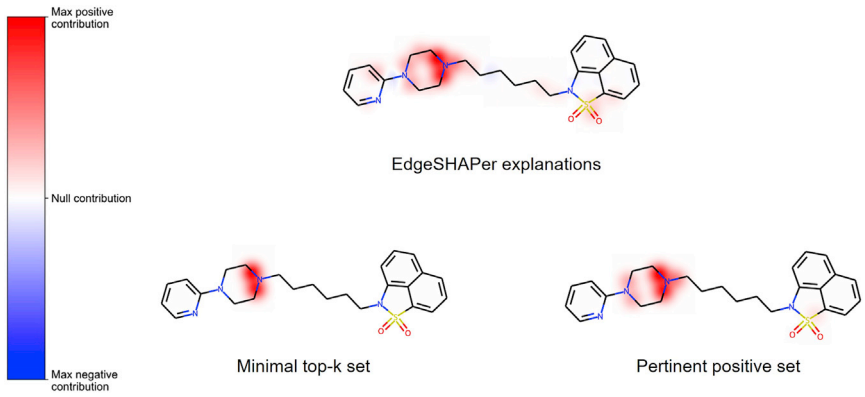


Figure 4. Output heatmap generated by the *explainer_script.py*

If requested via the *parameters.yml* file, minimal informative sets are also generated. Red gradient bonds make positive contributions to the output probability, while blue gradient bonds indicate negative contributions.

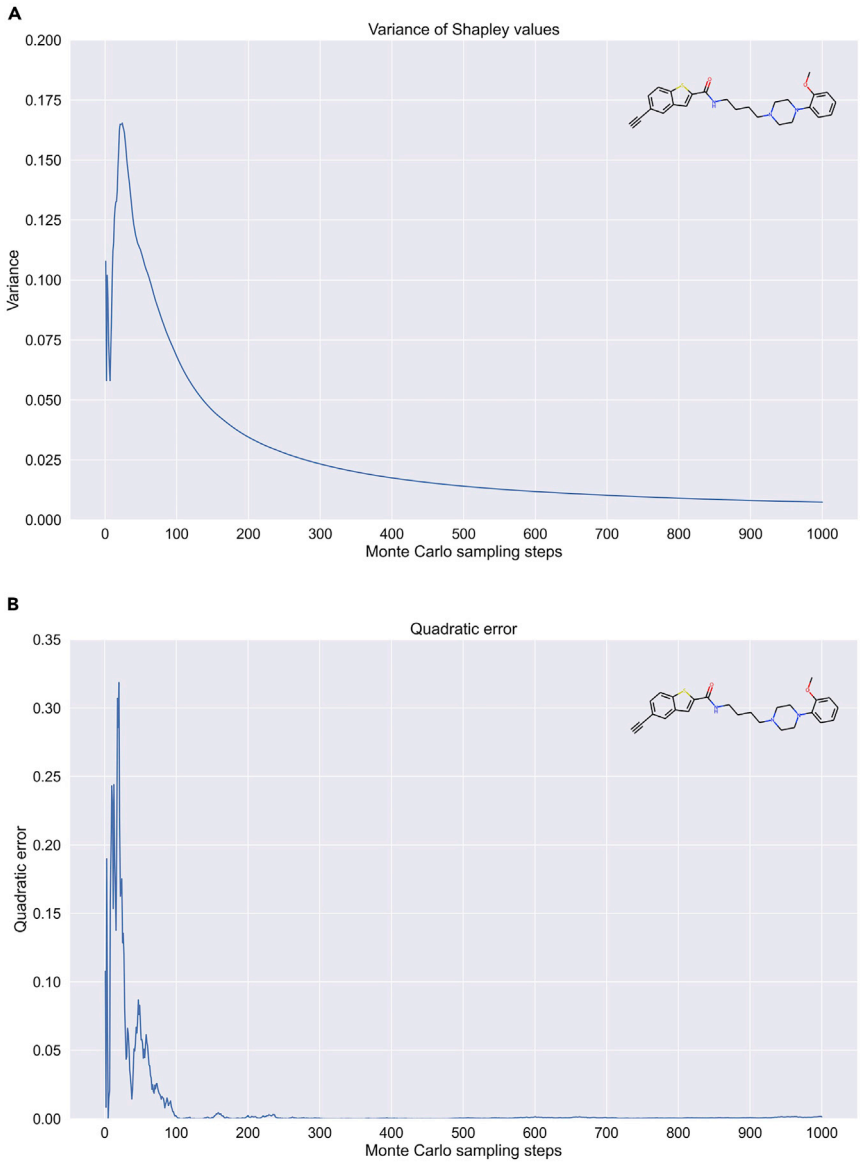


Figure 5. Variance and convergence analysis of EdgeSHAPer

(A and B) (A) Variance and (B) quadratic error for the sum of Shapley values of a test compound. The figure was taken from the original publication.[1]

Problem 2

Related to "installation". The command.

```
>pip install git+git://github.com/c-feldmann/rdkit_heatmaps
```

could fail producing an error.

Potential solution

- run the following command instead:

```
>pip install git+https://github.com/c-feldmann/rdkit_heatmaps
```

Problem 3

Related to "graph neural network training". If operating in a Windows PowerShell one might encounter the error message "conda is not recognized as an internal or external command". This means that conda was not initialized in the PowerShell.

Potential solution

- run the following command:

```
>conda init
```

- restart your terminal.

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and software should be directed to and will be addressed by the lead contact, J.B. (bajorath@bit.uni-bonn.de).

Materials availability

Not applicable.

Data and code availability

The source code and compound data used by this protocol can be accessed at https://github.com/AndMastro/EdgeSHAPer and are also provided in an open access desposition at Zenodo: https://doi.org/10.5281/zenodo.7267068.[2] The compound data, training, validation, and test sets are also available as Mendeley Data: https://doi.org/10.17632/bs6myg75tr.2.[3]

ACKNOWLEDGMENTS

We are grateful to Christian Feldmann and Raquel Rodríguez-Pérez for their contributions to the original study. This work has been partially supported (A.M.) by the EC H2020RIA project "SoBigData++" (871042).

AUTHOR CONTRIBUTIONS

Conceptualization, A.M., G.P., J.B.; software, A.M., G.P.; investigation, A.M., G.P.; writing – original draft, A.M., G.P., J.B.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. Mastropietro, A., Pasculli, G., Feldmann, C., Rodríguez-Pérez, R., and Bajorath, J. (2022). EdgeSHAPer: bond-centric Shapley value-based explanation method for graph neural networks. iScience *25*, 105043. https://doi.org/10.1016/j.isci.2022.105043.

2. Mastropietro, A., Feldmann, C., and Pasculli, G. (2022). AndMastro/EdgeSHAPer: v1.0.0 (Zenodo). https://doi.org/10.5281/zenodo.7267068.

3. Feldmann, C., Mastropietro, A., and Pasculli, G. (2022). Compounds With Activity Against the Dopamine D2 Receptor (Mendeley Data). V2, https://doi.org/10.17632/bs6myg75tr.2.

4. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). PyTorch: an imperative style, high-performance deep learning library. Preprint at arXiv. https://doi.org/10.48550/arXiv.1912.01703.

5. Fey, M., and Lenssen, J.E. (2019). Fast graph representation learning with PyTorch geometric. Preprint at arXiv. https://doi.org/10.48550/arXiv.1903.02428.

6. Kipf, T.N., and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. Preprint at arXiv. https://doi.org/10.48550/arXiv.1609.02907.

7. Weininger, D. (1988). SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. J. Chem. Inf. Model. *28*, 31–36. https://doi.org/10.1021/ci00057a005.