


SOFTWARE

Open Access



pyKVFinder: an efficient and integrable Python package for biomolecular cavity detection and characterization in data science

João Victor da Silva Guerra^{1,2*}, Helder Veras Ribeiro-Filho¹, Gabriel Ernesto Jara¹, Leandro Oliveira Bortot¹, José Geraldo de Carvalho Pereira¹ and Paulo Sérgio Lopes-de-Oliveira^{1,2*} 

*Correspondence:

joao.guerra@lnbio.cnpem.br; paulo.oliveira@lnbio.cnpem.br

¹ Brazilian Center for Research in Energy and Materials (CNPEM), Brazilian Biosciences National Laboratory (LNBio), R. Giuseppe Máximo Scolfaro, 10000 - Bosque das Palmeiras, Campinas, SP 13083-100, Brazil

Full list of author information is available at the end of the article

Abstract

Background: Biomolecular interactions that modulate biological processes occur mainly in cavities throughout the surface of biomolecular structures. In the data science era, structural biology has benefited from the increasing availability of biostructural data due to advances in structural determination and computational methods. In this scenario, data-intensive cavity analysis demands efficient scripting routines built on easily manipulated data structures. To fulfill this need, we developed pyKVFinder, a Python package to detect and characterize cavities in biomolecular structures for data science and automated pipelines.

Results: pyKVFinder efficiently detects cavities in biomolecular structures and computes their volume, area, depth and hydrophobicity, storing these cavity properties in NumPy arrays. Benefited from Python ecosystem interoperability and data structures, pyKVFinder can be integrated with third-party scientific packages and libraries for mathematical calculations, machine learning and 3D visualization in automated workflows. As proof of pyKVFinder's capabilities, we successfully identified and compared ADRP substrate-binding site of SARS-CoV-2 and a set of homologous proteins with pyKVFinder, showing its integrability with data science packages such as matplotlib, NGL Viewer, SciPy and Jupyter notebook.

Conclusions: We introduce an efficient, highly versatile and easily integrable software for detecting and characterizing biomolecular cavities in data science applications and automated protocols. pyKVFinder facilitates biostructural data analysis with scripting routines in the Python ecosystem and can be building blocks for data science and drug design applications.

Keywords: Cavity detection, Cavity characterization, NumPy, Python, Data structure, Data science, Automated pipelines, Molecular dynamics

Background

At present, we are going through the era of data science and the computational and structural biology fields have hugely benefited from the growing availability of biostructural data, as pointed out by [1]: “Structural biology meets data science”. Advances in



© The Author(s), 2021. **Open Access** 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.

X-ray crystallography and electron microscopy techniques have expanded the determination of novel structures [2]. In the meantime, advances in computational resources have driven the use of *in silico* methods to simulate the dynamics of biomolecules and carried out the implementation of artificial intelligence to model biomolecular structures [3]. Together, all of this structural data provides fertile ground for data interpretation through data science or automated analysis frameworks, but data-intensive analysis asks for efficient and integrable scripting routines with an easily manipulated data structure.

In this scenario, we developed pyKVFinder, an open-source python package for cavity detection and characterization abstracted into multidimensional arrays. In proteins or other macromolecules, solvent-exposed clefts or buried cavities play a key role in ligand binding, which can ultimately regulate biological function of the macromolecule [4]. For this reason, the identification and characterization of ligand-binding sites are the basis of rational structure-based drug discovery and design [5, 6]. pyKVFinder adopts the original geometrical grid-and-sphere-based detection method as implemented in KVFinder [7], which has been improved in the latest parallel version, parKVFinder [8]. Detected cavities in parKVFinder, as in many other well-known programs, such as CavVis [9], fpocket [10], GHECOM [11], ConCavity [12], MSpocket [13] and POVME 3.0 [14], are human-readable and easily displayed in molecular visualization programs, but are not properly structured to be directly scripted into automated pipelines or data science frameworks. Programming languages are constantly evolving in the data science field, with Python being one of the most popular in the community [15]. However, only a few initiatives have been launched to make cavity detection programs easier to integrate into data science protocols. For instance, Cambridge Crystallographic Data Centre has licensed commercial software suites that use a Python API to integrate its structural database with workflows and third-party applications [16], and one of its API modules is aimed at detecting ligand-binding cavities using the LIGSITE algorithm [17]. As an open-source initiative, Biobb_vs is a Python package designed to detect and analyze cavities in automated workflows. Biobb_vs is part of BioExcel Building Blocks [18] and uses fpocket to detect cavities in virtual screening pipelines. In this case, despite the interoperability achieved, the workflows depend on handling data files generated by fpocket.

To fulfill this need, pyKVFinder is wrapped into Python and, using Python's well-established data structure (e. g., NumPy array), benefits from the language ecosystem interoperability. pyKVFinder can be integrated with third-party scientific packages and libraries for mathematical calculations, statistical analysis, and tridimensional visualization. Moreover, users can explore the functionality of pyKVFinder step-by-step using interactive interfaces, such as IPython/Jupyter notebooks. As mentioned above, one of pyKVFinder's main contributions in data science workflows is to translate the detected cavities from tridimensional coordinates of cavity points to NumPy arrays, a data structure that allows for a wide diversity of scientific computation and efficient storage and access to N-dimensional arrays (ndarrays), also called tensors [19]. Ndarrays also provide efficient ways of handling data for mathematical operations and are the popular choice of input data type for machine learning Python libraries such as scikit-learn [20].

Besides conventional cavity properties such as volume and area, which are stored as Python dictionaries, pyKVFinder computes cavity depth and hydrophathy. Similar to

cavity points, these spatial and physicochemical properties are stored as Python ndarrays and can be visualized using Python molecular visualization widgets. Thus, pyKVFinder provides a versatile way to detect and characterize biomolecular cavities and integrate this information into data science or automated workflows.

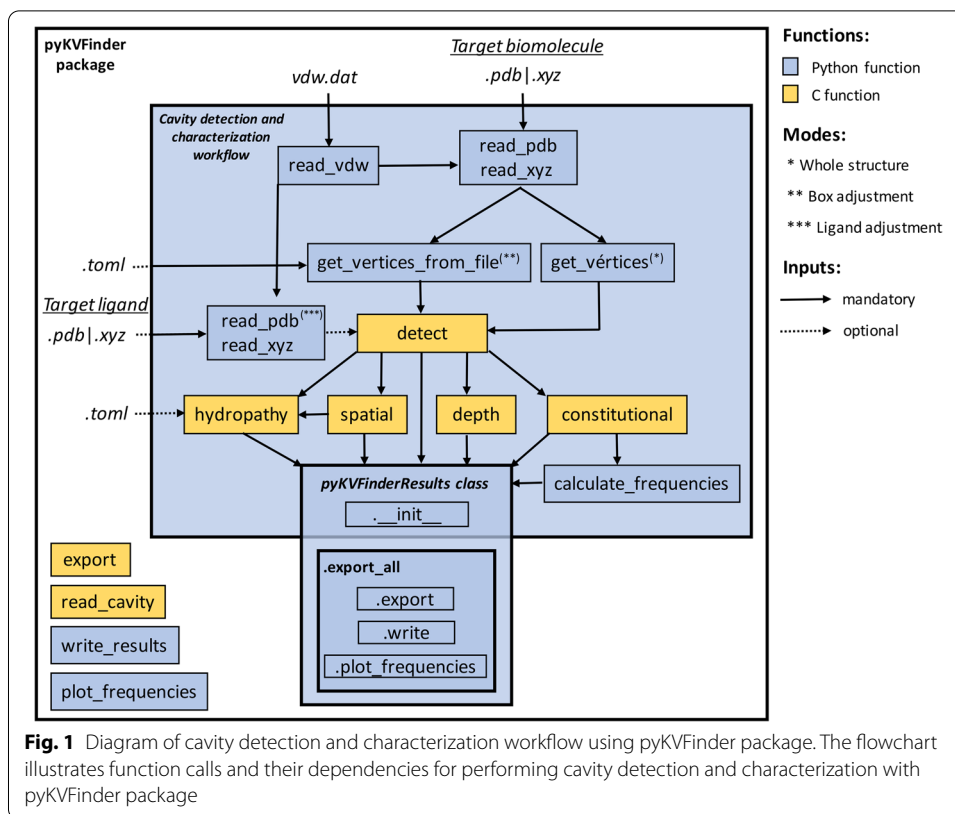
Implementation

Python-C parallel KVFinder (pyKVFinder) applies a Simplified Wrapper and Interface Generator (SWIG; <http://www.swig.org/>) to extend grid operations written in C to Python, a high-level programming language. pyKVFinder can be easily installed with the pip package management system. In pyKVFinder, the target biomolecule is inserted into a regular 3D grid, which is stored as an ndarray, considering the van der Waals radii of the atoms. To detect cavities, pyKVFinder uses a dual-probe algorithm that scans the biomolecular structure, as described in [7, 8]. In summary, a small probe, dubbed Probe In, and a larger probe, dubbed Probe Out, translate over the grid points, defining two molecular surfaces with different accessibility. Biomolecular cavities are defined as the non-overlapped regions between the molecular surfaces. On each detected cavity, pyKVFinder may perform spatial, depth, hydrophathy and constitutional characterizations. In this workflow, C routines undergo multithreaded parallelization to speed up cavity detection and characterization, with the OpenMP API creating a user-defined number of parallel threads, where grid chunks are distributed among these threads to perform independent operations.

Python package

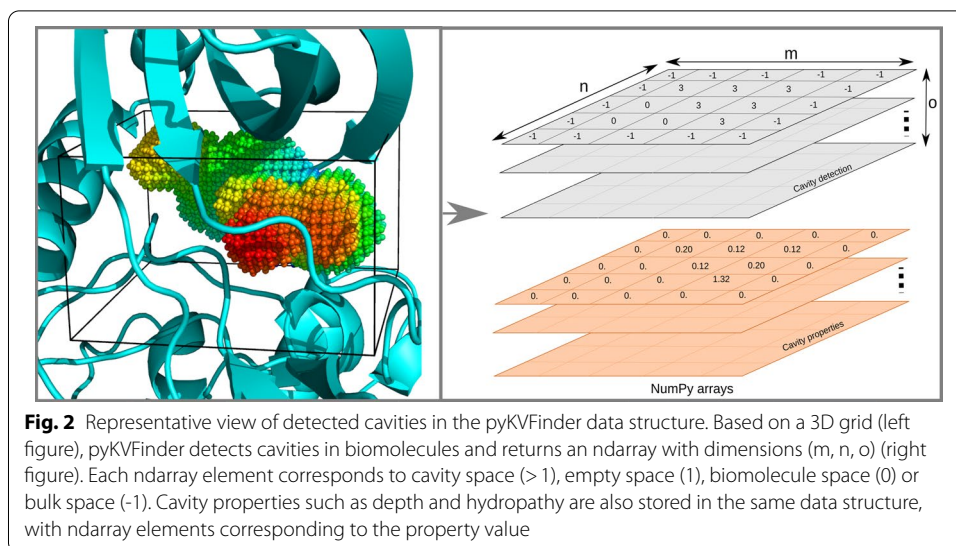
pyKVFinder can be imported as a package in the Python environment and users can decide to run the full cavity detection and characterization workflow through the *run_workflow* function or run pyKVFinder functions in a stepwise fashion. At the latter, users can integrate pyKVFinder functions into third-party Python packages and benefit from interactive IPython/Jupyter notebooks. By running pyKVFinder in Python environment, users can visualize the detected cavities through the Python NGL Viewer widget [21]. Still, the full workflow was also coded as a command-line interface. Either way, users have access to a full set of customizable parameters to detect and characterize biomolecular cavities. A schematic diagram of cavity detection and characterization workflow is described in Fig. 1.

In the cavity detection and characterization workflow, *read_vdw* function reads a customizable van der Waals (vdW) radii file (*.dat* extension; default *vdw.dat*) into a nested Python dictionary, which is first indexed by the three-letter residue code (e. g., ALA, ARG, ASH, etc.), and then indexed by the atom name (e. g., C, N, O, CA, etc.) to access its respective radius value. The vdW radii file defines the radius values for each atom by residue and when not defined, uses a generic value based on the atom type. The built-in file (*vdw.dat*) applies the vdW radii of the Amber ff99 force field [22]. Afterwards, *read_pdb* or *read_xyz* function gets the vdW dictionary and reads a target PDB or XYZ file (*.pdb* or *.xyz* extension), respectively. The atomic data is stored in an ndarray with residue number, chain identifier, residue name, atom name, xyz coordinates and radius per atom.



To perform the cavity detection on the whole structure, the 3D grid is defined based on the target biomolecule coordinates and the user-defined parameters: grid spacing and Probe Out diameter. Thus, the grid coordinates are extracted from the atomic data ndarray, using *get_vertices* function that defines the grid origin and XYZ axis. The first vertex (p1) is the minimum xyz coordinates of the atomic data. The second (p2), third (p3) and fourth (p4) vertices represent the maximum coordinate along the X, Y and Z axes, respectively. Given these four vertices, the original coordinate system is transformed in the *detect* function, using translation and rotation, into a new coordinate system with P1 as its origin. Also, for internal calculations, the sum of the grid spacing and the Probe Out size is padded in each direction of the 3D grid. With user-defined parameters and atomic coordinates, the *detect* function creates and fills the 3D grids with Probe In and Probe Out probes, and compares these grids to define the biomolecular cavities, which are returned in an ndarray. In this ndarray (Fig. 2), each element corresponds to cavity space (> 1), empty space (1), biomolecule space (0) or bulk space (- 1).

Instead of performing the cavity detection on the whole structure, the search space can be set to a custom search box, called box adjustment mode. This mode is defined based on a TOML-formatted configuration file (.toml extension; Additional file 1: Fig. A1), which can explicitly define the vertices coordinates or pass a list of residues of the target biomolecule to be used as reference instead of the whole structure. The box can also be drawn using parKVFinder PyMOL plugin and its coordinates can be extracted from parKVFinder parameters file. Thus, the *get_vertices_from_file* function loads a box configuration file and atomic data from *read_pdb* or *read_xyz* function, which returns



the grid coordinates (p1, p2, p3 and p4) and selects the atoms inside the custom 3D grid with their respective atomic data. Hence, these parameters are passed to *detect* function, together with *box_adjustment* flag, to perform cavity detection on a custom 3D grid.

Either way, whole structure mode or box adjustment mode, another space segmentation method, called ligand adjustment mode, can be applied to constrain the search space around a target ligand, defined by a PDB or XYZ file (*.pdb* or *.xyz* extension). For this mode, *read_pdb* or *read_xyz* function gets the vdW dictionary and reads a ligand PDB or XYZ file and returns the ligand atomic coordinates with their respective radius that must be passed to *detect* function to further restrict the search space within a radius of the target ligand.

Cavity characterization

With the ndarray of detected cavities, pyKVFinder package may perform four characterization procedures, i.e., spatial, constitutional, depth and hydrophathy characterizations. The spatial characterization, using *spatial* function, defines the surface points in an ndarray, and estimates the volume and area of the detected cavities, following the methodology proposed in [8]. The constitutional characterization, using *constitutional* function, defines the interface residues that surround the detected cavities, checking if the atoms of the residues are within a radius, which is the sum of the Probe In size and the atom radius. Alternatively, the *constitutional* function can ignore backbone contacts by enabling *ignore_backbone* flag. This function stores the interface residues in a Python dictionary. Using the interface residues, the *calculate_frequencies* function calculates the occurrence of each residue and classes of residues per cavity. The classes of amino acid residues [23] are aliphatic apolar (R1), aromatic (R2), polar uncharged (R3), negatively charged (R4), positively charged (R5) and non-standard (RX) (Additional file 1: Table A1).

The depth characterization identifies the degree of burial of the binding site. This characterization applies *depth* function to identify the cavity-bulk boundary by applying a spatial filter, which defines a boundary point as cavity points where at least one direct

neighbor is a bulk point, marking it with the negative of the cavity integer (Additional file 1: Fig. A2). Subsequently, the depth of each cavity point is heuristically estimated by the shortest Euclidean distance between the cavity point and its respective boundary points (Eq. 1). Each cavity point is distributed among the user-defined number of threads to perform this chunk of distance calculations (Additional file 1: Fig. A2). With the depth of all cavity points calculated and stored as an ndarray (Fig. 2), the maximum and average depths are calculated for each detected cavity and returned as Python dictionaries.

$$\widehat{D}_i = \min_j [d(p_i, p_j)] = \min_j \left[\sqrt{(p_{i_x} - p_{j_x})^2 + (p_{i_y} - p_{j_y})^2 + (p_{i_z} - p_{j_z})^2} \right], \quad (1)$$

where \widehat{D}_i is the depth of the cavity point i , $d(p_i, p_j)$ is the Euclidean distance between points i and j , p_i is the x, y, z coordinates of a cavity point i , p_j is the x, y, z coordinates of a cavity-bulk boundary point j .

The hydrophathy characterization extracts the water attractiveness of the interface residues surrounding the binding site. This characterization uses *hydropathy* function at surface points detected in spatial characterization to map a target hydrophobicity scale on them. Firstly, a customizable TOML-formatted hydrophobicity scale file (*.toml* extension) is loaded into a Python dictionary, which is indexed by the three-letter residue code (e. g., ALA, ARG, ASH, etc.) to access the respective hydrophobicity value. With the dictionary loaded, the function identifies the nearest interface residues for each surface point and projects the residue's hydrophobicity value onto them (Fig. 2). Alternatively, backbone contacts may be ignored by enabling the *ignore_backbone* flag. Finally, with the hydrophobicity mapped on the surface and returned as an ndarray, the average hydrophathy is calculated for each detected cavity and stored in a Python dictionary. The hydrophobicity scale file defines the scale name and hydrophobicity value for each residue and, when not defined, assigns zero to the missing residues (Additional file 1: Fig. A3). The package contains six built-in hydrophobicity scales: Eisenberg and Weiss [24], Hessa and Heijne [25], Kyte and Doolittle [26], Moon and Fleming [27], Wimley and White [28] and Zhao and London [29]. Benefiting from the Python environment, users can test different scales without having to perform the cavity detection step every time. A pre-released version of hydrophathy characterization has been successfully applied to compare cavities of alphavirus-related proteins [30].

The cavity detection and characterization objects, from *detect*, *spatial*, *constitutional*, *calculate_frequencies*, *depth*, *hydropathy* functions, can be stored into a *pyKVFinderResults* class that accumulates them in its attributes. As this data structure is filled, these attributes can be exported to files through *export*, *write* and *plot_frequencies* methods. The *export* method writes cavities together with their surface points to a PDB-formatted file with depth values in the B-factor column, while surface points with hydrophathy values in the B-factor column are written to another PDB-formatted file. The *write* method saves file paths, volume, area, interface residues and their frequencies, maximum and average depth, and average hydrophathy in a TOML-formatted file. The *plot_frequencies* method plots bar charts of frequencies per cavity in a PDF file. These three methods are also wrapped in *export_all* method. Although we presented the full workflow, all functions explained in this section can be applied independently in a step-by-step manner. In this scenario, the *export*, *write* and *plot_frequencies* methods have their counterparts in

the *export*, *write_results* and *plot_frequencies* functions of pyKVFinder package, respectively. Additionally, the *read_cavity* function reads a cavity file (*.pdb* extension), written by pyKVFinder, parKVFinder or KVFinder, and a target PDB or XYZ file (*.pdb* or *.xyz* extension), and returns an ndarray with each element corresponding to the cavity space (> 1), biomolecule space (0), or bulk or empty space (-1), similar to the output of the *detect* function. In this way, it allows to recharacterize a previously detected cavity or characterize a cavity with manually trimmed points.

Results and discussion

Usage example

To demonstrate the use of pyKVFinder and how it benefits from the Python ecosystem, we identified the substrate-binding pocket of the ADP-ribose phosphatase (ADRP) domain of SARS-CoV-2 nsp3 protein in the apo form (PDB ID: 6WEN). Still under investigation to determine its exact functions in coronavirus life cycle, the ADRP domain recognizes ADP-ribose 1'' phosphate [31, 32] and seems to have an important role in virulence and innate immunity regulation to infection [33–35]. In this regard, recent efforts have been made to characterize ADP-ribose substrate-binding pocket and evaluate this site as a putative antiviral drug target [36, 37].

Visualizing detected cavities with NGL Viewer in Jupyter notebook

pyKVFinder successfully detected the ADRP substrate-binding cavity and determined traditional cavity properties such as volume, area and residues surrounding the ADP-ribose cavity (Fig. 3a, upper panel). For instance, we used pyKVFinder *calculate_frequencies* and *plot_frequencies* functions to determine the composition of the type of residues surrounding the cavity and plotted this composition as a bar chart (Additional file 1: Fig. A4). In pyKVFinder, this step is performed using matplotlib library [38], but users are free to analyze data and present results on their favorite graphing library. As observed in Fig. 3a, the ADP-ribose site forms a cleft sandwiched between ADRP α -helices and the main contacts involve residues from coil regions, which could possibly explain the pocket plasticity upon substrate binding [31, 32]. These results were visualized using NGL Viewer on a Jupyter notebook; alternatively, users can use another molecular visualization tool for notebooks or load results into the parKVFinder PyMOL Plugin [8].

Characterizing hydrophathy and depth of cavities

We also inspected the ADRP substrate-binding cavity through other two points of view: depth and hydrophathy. Those physicochemical descriptions are usually essential for drug development [36]. In apo form, despite being solvent-exposed, the cavity has some internal components (red color) that can reach a more central portion of the ADRP β -sheet (Fig. 3a, middle panel). The hydrophathy analysis shows that the cavity core is most hydrophobic (yellow color), with some polar residues on the edges (blue color) that may contribute to the design of more specific ligands. Since pyKVFinder stores the properties to be colored in cavities in the B-factor column of a PDB-formatted file, users can easily change the style and color scheme in most of molecular visualization programs.

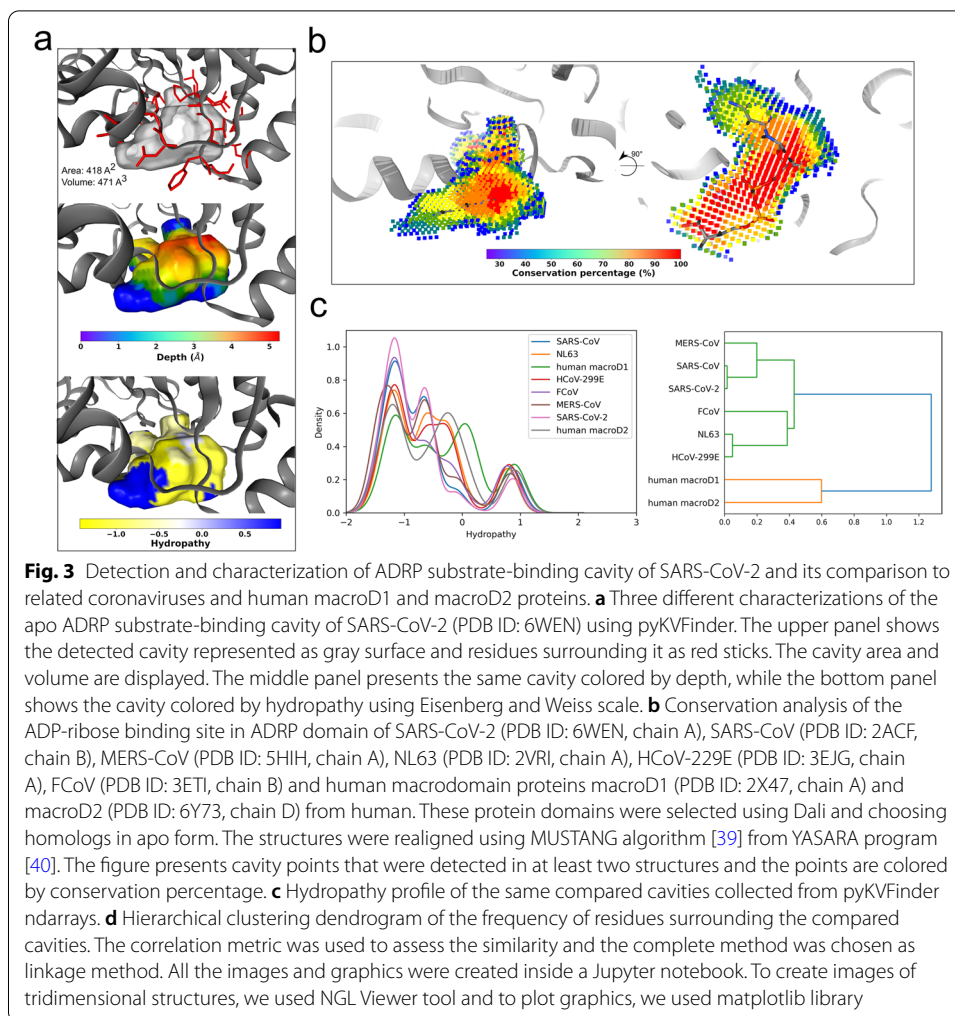


Fig. 3 Detection and characterization of ADRP substrate-binding cavity of SARS-CoV-2 and its comparison to related coronaviruses and human macroD1 and macroD2 proteins. **a** Three different characterizations of the apo ADRP substrate-binding cavity of SARS-CoV-2 (PDB ID: 6WEN) using pyKVFinder. The upper panel shows the detected cavity represented as gray surface and residues surrounding it as red sticks. The cavity area and volume are displayed. The middle panel presents the same cavity colored by depth, while the bottom panel shows the cavity colored by hydrophathy using Eisenberg and Weiss scale. **b** Conservation analysis of the ADP-ribose binding site in ADRP domain of SARS-CoV-2 (PDB ID: 6WEN, chain A), SARS-CoV (PDB ID: 2ACF, chain B), MERS-CoV (PDB ID: 5HIH, chain A), NL63 (PDB ID: 2VRI, chain A), HCoV-229E (PDB ID: 3EJG, chain A), FCoV (PDB ID: 3ETI, chain B) and human macrodomain proteins macroD1 (PDB ID: 2X47, chain A) and macroD2 (PDB ID: 6Y73, chain D) from human. These protein domains were selected using Dali and choosing homologs in apo form. The structures were realigned using MUSTANG algorithm [39] from YASARA program [40]. The figure presents cavity points that were detected in at least two structures and the points are colored by conservation percentage. **c** Hydrophathy profile of the same compared cavities collected from pyKVFinder ndarrays. **d** Hierarchical clustering dendrogram of the frequency of residues surrounding the compared cavities. The correlation metric was used to assess the similarity and the complete method was chosen as linkage method. All the images and graphics were created inside a Jupyter notebook. To create images of tridimensional structures, we used NGL Viewer tool and to plot graphics, we used matplotlib library

Using NumPy operations to present conservation and matplotlib library to plot hydrophathy distribution of cavities

In addition to SARS-CoV-2, the ADRP domain is also present in other related coronaviruses and has the macroD1 and macroD2 as homologous in humans. For this reason, we used pyKVFinder to detect the ADRP substrate-binding site in aligned ADRP domains from different species and compare their properties. Firstly, we applied arithmetic operations on the ndarrays of the detected cavities to determine the cavity conservation among the species. As observed in Fig. 3b, the ADRP cavity has a core (red points) very conserved in the analyzed species which is occupied by the diphosphate and ribose of ADP and the second ribose bound to ADP in the ADRP substrate-bound form. In turn, adenosine occupies a less conserved cavity region, which may indicate that the structure of this site in some species changes to accommodate ADP-ribose substrate. To compare the cavity hydrophathy across species, we plotted a hydrophathy distribution from the hydrophathy ndarray using the matplotlib library [38] (Fig. 3c, left graph). The distribution clearly shows the hydrophobic characteristic of the pocket that is mostly shared between ADRP substrate-binding pockets of coronaviruses. Interestingly, the human macroD1 and macroD2 seem to shift the distribution to a less hydrophobic profile.

This finding should be better evaluated, as the differences between these homologous domains that share the same substrate can contribute to the design of specific ligands for viral ADRP domains.

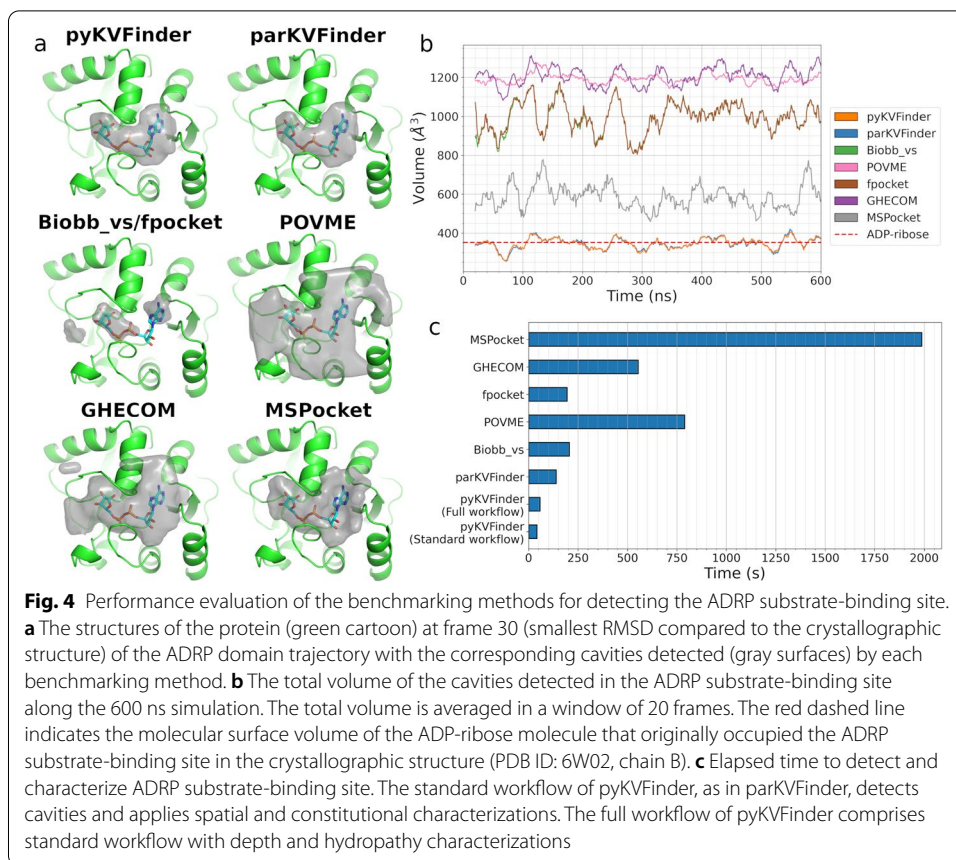
Hierarchical clustering of cavity residues using SciPy package

Finally, since pyKVFinder uses native Python dictionaries to store the residues surrounding the detected cavity, we can easily tabulate the residue frequency. With this information, we performed a hierarchical clustering, an unsupervised machine learning algorithm, using the SciPy package [41], and represented clusters arrangement as a dendrogram (Fig. 3c, right graph). The ADRP cavity of SARS-CoV-2 grouped with that of SARS-CoV, demonstrating the high identity between these betacoronaviruses. Close to them, we can observe another betacoronavirus, MERS-CoV. On the other hand, the alphacoronaviruses NL63 and HCoV-229E and the feline FCoV are grouped together. Further away from the coronaviruses' domains are the two human macrodomain proteins, macroD1 and D2. Despite the cavity of ADRP or macro D1/D2 sharing the same substrate, ADP-ribose, these results show that the profile of the residues surrounding these cavities follows evolutionary traces.

Benchmarking

In addition to identifying and characterizing the ADRP substrate-binding site of SARS-CoV-2 and a set of homologous proteins, we simulated ADRP domain of SARS-CoV-2 (PDB ID: 6W02, chain B) without its ligand, ADP-ribose, for 600 ns, extracting a frame at regular intervals of 1 ns (Additional file 1). Thus, we used pyKVFinder with its box adjustment mode to detect and estimate the volume of the ADP-ribose binding site throughout 600 frames of the ADRP domain's trajectory. This analysis was repeated with other well-known software: POVME [14], Biobb_vs [18], MSPocket [13], GHECOM [11], fpocket [10] and parKVFinder [8]. Biobb_vs, as mentioned in the Background section, is a Python package that allow scripting, while POVME, MSPocket, GHECOM, fpocket, GHECOM and parKVFinder are command-line interfaces. A detailed description of software parameters and versions is in Additional file 1.

All these methods successfully detected the pocket of the ADRP substrate-binding site, in which the shape and volume vary slightly during the molecular dynamics simulation (Fig. 4). The shape of the detected cavities defined by pyKVFinder and parKVFinder finely adjust to the original ligand in the binding site, as well as MSPocket (Fig. 4a) Besides that, the volume calculated by pyKVFinder ($346.8 \pm 78.7 \text{ \AA}^3$) and parKVFinder ($346.5 \pm 79.3 \text{ \AA}^3$) is closely related to the volume of ADP-ribose (351.1 \AA^3 ; molecular surface volume estimated by YASARA program [40]), the ligand that originally occupied the binding site in the crystallographic structure used in the molecular dynamics simulations (Fig. 4b). Nevertheless, the differences in the shape and volume of detected cavities derive from the methodology employed (e.g., Voronoi tessellation, alpha spheres, and grid-and-sphere), the cavity-bulk boundary definition, and the ability to segment the space. For instance, pyKVFinder, parKVFinder and POVME can segment the search space, which trims points outside this custom space, while the other methods only explore the whole structure, which includes neighboring regions at the binding site.



Besides being able to accurately detect biomolecular cavities, current software must also perform fast detection and characterizations. Thus, we also evaluated the elapsed time to execute these benchmarking methods (Fig. 4c). pyKVFinder outperformed all analyzed methods. Even when applying the newly available characterization, depth and hydrophathy, pyKVFinder’s elapsed time only increased 36%, still outperforming other benchmarking methods. Further, compared to its counterpart, parKVFinder, pyKVFinder was 3.3 times faster in detecting ADRP binding site. The main reason for the performance gain is the additional possibility to parallelize routines, i. e., the insertion of atoms in the 3D grid in *detect* function, based on ndarrays. Hence, experienced users requiring scripting routines are encouraged to use pyKVFinder due to its improved performance, while newcomers should prioritize parKVFinder due to its simplicity of installation and execution. Further, the scalability of pyKVFinder, upon increasing number of threads, follows the same behavior presented by parKVFinder [8].

Despite all methods characterizing volume, each method has its own set of characterizations to be performed on the detected cavities. However, the cavities data structure is only accessible inside the Python ecosystem in pyKVFinder, which provides ndarrays and Python dictionaries. The ndarrays stores cavity points, surface points, hydrophathy for each surface point and depth for each cavity point, while Python dictionaries stores volume, area, average hydrophathy, maximum depth and average depth,

and interface residues and their frequencies per detected cavity. Thus, users may develop new characterizations and/or analysis pipelines with these data structures.

Future development

pyKVFinder will undergo continuous improvements and updates, according to its applications by the scientific community. In the future, pipelines will be implemented in molecular dynamics and machine learning, along with new features that are valuable to ligand-binding site characterization. Additionally, pyKVFinder aims to offload its routines to the GPU for performance enhancement in data-intensive applications.

Conclusion

pyKVFinder provides an efficient and integrable Python package for cavity detection and characterization in biomolecular structures for data science and automated pipelines. In addition to fast, accurate and efficient cavity detection and characterization, pyKVFinder stores spatial and physicochemical properties in Python ndarrays, that ease scripting and data analysis. Further, pyKVFinder performance was benchmarked against well-known geometrical methods for cavity detection and characterization. Finally, we have successfully shown an application of pyKVFinder integration with matplotlib, NGL Viewer, SciPy and Jupyter notebook, that compared the ADRP substrate-binding site of SARS-CoV-2 in homologous proteins.

Availability and requirements

Project name: pyKVFinder

Project home page: <https://github.com/LBC-LNBio/pyKVFinder>

Operating system(s): any supporting Python > = 3.7 (tested on Linux and macOS)

Programming language: Python, C

Other requirements: swig > = 4.0.1, toml > = 0.10.2, numpy > = 1.20.3, matplotlib > = 3.3.3

License: GNU General Public License v3.0

Any restrictions to use by non-academics: None.

Abbreviations

ADRP: ADP-ribose phosphatase; CoV: Coronavirus; MERS: Middle East Respiratory Syndrome; ndarrays: N-dimensional arrays; PDB: Protein Data Bank; R1: Aliphatic apolar; R2: Aromatic; R3: Polar uncharged; R4: Negatively charged; R5: Positively charged; RX: Non-standard; SARS: Severe Acute Respiratory Syndrome; SWIG: Simplified Wrapper and Interface Generator; TOML: Tom's Obvious, Minimal Language; vdW: Van der Waals.

Supplementary Information

The online version contains supplementary material available at <https://doi.org/10.1186/s12859-021-04519-4>.

Additional file 1. The Additional file 1 contains **Table A1**, **Figures A1**, **A2**, **A3** and **A4**, and a detailed description of the molecular dynamics simulation of ADRP domain of SARS-CoV-2 and the benchmarking procedure. **Table A1** reports the classes of amino acid residues. **Figure A1** shows examples of box configuration files. **Figure A2** shows the methodology of depth characterization. **Figure A3** shows the methodology of hydrophathy characterization. **Figure A4** shows a bar chart of residues frequencies.

Acknowledgements

We thank the Brazilian Biosciences National Laboratory (LNBio), part of the Brazilian Center for Research in Energy and Materials (CNPEM) for accessibility to the Computational Biology Laboratory (LBC).

Authors' contributions

JVSG, JGCP and PSLO conceptualize and design the code package structure. JVSG implemented the code. HVRF, LOB, GEJ, and JGCP tested and performed validation checks. JVSG and HVRF wrote the manuscript. LOB, GEJ, JGCP and PSLO reviewed the manuscript and provided critical revision. PSLO supervised coding steps and decisions. All authors read and approved the final manuscript.

Funding

This work was supported by the Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) [Grant Number 2018/00629-0], Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) [Grant Number 350244/2020-0], and Brazilian Center for Research in Energy and Materials (CNPEM). None of these funding bodies provided any direct role nor influence in the design of the study and collection, analysis, and interpretation of data and in writing the manuscript.

Availability of data and materials

pyKVFinder source code, documentation and tutorials are available in the Python Package Index (PyPI) repository, <https://pypi.org/project/pyKVFinder>, and the GitHub repository, <https://github.com/LBC-LNBio/pyKVFinder>. Documentation and tutorials are available at pyKVFinder webpage, <https://lbc-lnbio.github.io/pyKVFinder>.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Author details

¹Brazilian Center for Research in Energy and Materials (CNPEM), Brazilian Biosciences National Laboratory (LNBio), R. Giuseppe Máximo Scolfaro, 10000 - Bosque das Palmeiras, Campinas, SP 13083-100, Brazil. ²Graduate Program in Pharmaceutical Sciences, Faculty of Pharmaceutical Sciences, University of Campinas, Campinas, SP, Brazil.

Received: 25 October 2021 Accepted: 7 December 2021

Published online: 20 December 2021

References

1. Mura C, Draizen EJ, Bourne PE. Structural biology meets data science: does anything change? *Curr Opin Struct Biol*. 2018;52:102.
2. Burley SK, Berman HM, Bhikadiya C, Bi C, Chen L, Di Constanzo L, et al. Protein Data Bank: the single global archive for 3D macromolecular structure data. *Nucleic Acids Res*. 2019;47:D520–8.
3. Tunyasuvunakool K, Adler J, Wu Z, Green T, Zielinski M, Židek A, et al. Highly accurate protein structure prediction for the human proteome. *Nature*. 2021;596:590.
4. Liang J, Woodward C, Edelsbrunner H. Anatomy of protein pockets and cavities: measurement of binding site geometry and implications for ligand design. *Protein Sci*. 1998;7:1884–97. <https://doi.org/10.1002/pro.5560070905>.
5. Sotriffer C, Klebe G. Identification and mapping of small-molecule binding sites in proteins: computational tools for structure-based drug design. *Farm*. 2002;57:243–51. [https://doi.org/10.1016/S0014-827X\(02\)01211-9](https://doi.org/10.1016/S0014-827X(02)01211-9).
6. Henrich S, Salo-Ahen OMH, Huang B, Rippmann FF, Cruciani G, Wade RC. Computational approaches to identifying and characterizing protein binding sites for ligand design. *J Mol Recognit*. 2009;23:209–19. <https://doi.org/10.1002/jmr.984>.
7. Oliveira SH, Ferraz FA, Honorato RV, Xavier-Neto J, Sobreira TJ, de Oliveira PS. KVFinder: steered identification of protein cavities as a PyMOL plugin. *BMC Bioinform*. 2014;15:197. <https://doi.org/10.1186/1471-2105-15-197>.
8. da Silva Guerra JV, Ribeiro Filho HV, Bortot LO, Honorato RV, de Carvalho Pereira JG, Lopes-de-Oliveira PS. ParkVFinder: a thread-level parallel approach in biomolecular cavity detection. *SoftwareX*. 2020;12:100606.
9. Simões TMC, Gomes AJP. CavVis—a field-of-view geometric algorithm for protein cavity detection. *J Chem Inf Model*. 2019;59:786–96. <https://doi.org/10.1021/acs.jcim.8b00572>.
10. Le Guilloux V, Schmidtke P, Tuffery P. Fpocket: an open source platform for ligand pocket detection. *BMC Bioinform*. 2009;10:168. <https://doi.org/10.1186/1471-2105-10-168>.
11. Kawabata T. Detection of multiscale pockets on protein surfaces using mathematical morphology. *Proteins*. 2010;78:1195–211. <https://doi.org/10.1002/prot.22639>.
12. Capra JA, Laskowski RA, Thornton JM, Singh M, Funkhouser TA. Predicting protein ligand binding sites by combining evolutionary sequence conservation and 3D structure. *PLoS Comput Biol*. 2009. <https://doi.org/10.1371/journal.pcbi.1000585>.

13. Zhu H, Pisabarro MT. MSPocket: an orientation-independent algorithm for the detection of ligand binding pockets. *Bioinformatics*. 2011;27:351–8. <https://doi.org/10.1093/bioinformatics/btq672>.
14. Wagner JR, Sørensen J, Hensley N, Wong C, Zhu C, Perison T, et al. POVME 3.0: software for mapping binding pocket flexibility. *J Chem Theory Comput*. 2017;13:4584–92.
15. Raschka S, Patterson J, Nole C. Machine learning in Python: main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*. 2020;11:193.
16. Groom CR, Bruno IJ, Lightfoot MP, Ward SC. The Cambridge structural database. *Acta Crystallogr Sect B Struct Sci Cryst Eng Mater*. 2016;72:171–9.
17. Hendlich M, Rippmann F, Barnickel G. LIGSITE: Automatic and efficient detection of potential small molecule-binding sites in proteins. *J Mol Graph Model*. 1997;15:359–63.
18. Andrio P, Hospital A, Conejero J, Jordá L, Del Pino M, Codo L, et al. BioExcel Building Blocks, a software library for interoperable biomolecular simulation workflows. *Sci Data*. 2019. <https://doi.org/10.1038/s41597-019-0177-4>.
19. Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. *Nature*. 2020;585:357–62.
20. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: machine learning in Python. *J Mach Learn Res*. 2011;12:2825–30.
21. Nguyen H, Case DA, Rose AS. NGLview—interactive molecular graphics for Jupyter notebooks. *Bioinformatics*. 2018;34:1241.
22. Wang J, Cieplak P, Kollman PA. How well does a restrained electrostatic potential (RESP) model perform in calculating conformational energies of organic and biological molecules? *J Comput Chem*. 2000;21:1049–74.
23. Nelson DL, Cox MM. *Lehninger principles of biochemistry*, 4th edition. 2004.
24. Eisenberg D, Weiss RM, Terwilliger TC. The hydrophobic moment detects periodicity in protein hydrophobicity. *Proc Natl Acad Sci*. 1984;81:140–4.
25. Hessa T, Kim H, Bihlmaier K, Lundin C, Boekel J, Andersson H, et al. Recognition of transmembrane helices by the endoplasmic reticulum translocon. *Nature*. 2005;433:377–81.
26. Kyte J, Doolittle RF. A simple method for displaying the hydropathic character of a protein. *J Mol Biol*. 1982;157:105–32.
27. Moon CP, Fleming KG. Side-chain hydrophobicity scale derived from transmembrane protein folding into lipid bilayers. *Proc Natl Acad Sci*. 2011;108:10174–7.
28. Wimley WC, White SH. Experimentally determined hydrophobicity scale for proteins at membrane interfaces. *Nat Struct Mol Biol*. 1996;3:842–8.
29. Zhao G, London E. An amino acid “transmembrane tendency” scale that approaches the theoretical limit to accuracy for prediction of transmembrane helices: relationship to biological hydrophobicity. *Protein Sci*. 2006;15:1987–2001.
30. Ribeiro-Filho HV, Coimbra LD, Cassago A, Rocha RPF, da Silva Guerra JV, de Felicio R, et al. Cryo-EM structure of the mature and infective Mayaro virus at 4.4 Å resolution reveals features of arthritogenic alphaviruses. *Nat Commun*. 2021. <https://doi.org/10.1038/s41467-021-23400-9>.
31. Michalska K, Kim Y, Jedzejczak R, Maltseva NI, Stols L, Endres M, et al. Crystal structures of SARS-CoV-2 ADP-ribose phosphatase: from the apo form to ligand complexes. *IUCrJ*. 2020;5:536.
32. Frick DN, Virdi RS, Vuksanovic N, Dahal N, Silvaggi NR. Molecular basis for ADP-ribose binding to the Mac1 domain of SARS-CoV-2 nsp3. *Biochemistry*. 2020;178:104793.
33. Claverie J-M. A putative role of de-mono-ADP-Ribosylation of STAT1 by the SARS-CoV-2 Nsp3 protein in the cytokine storm syndrome of COVID-19. *Viruses*. 2020;12:646.
34. Fehr AR, Channappanavar R, Jankevicius G, Fett C, Zhao J, Athmer J, et al. The conserved coronavirus macrodomain promotes virulence and suppresses the innate immune response during severe acute respiratory syndrome coronavirus infection. *MBio*. 2016. <https://doi.org/10.1128/mBio.01721-16>.
35. Eriksson KK, Cervantes-Barragán L, Ludewig B, Thiel V. Mouse hepatitis virus liver pathology is dependent on ADP-Ribose-1st-Phosphatase, a viral function conserved in the alpha-like supergroup. *J Virol*. 2008;82:12325–34.
36. Brosey CA, Houli JH, Katsonis P, Balapiti-Modarage LPF, Bommagani S, Arvai A, et al. Targeting SARS-CoV-2 Nsp3 macrodomain structure with insights from human poly(ADP-ribose) glycohydrolase (PARG) structures with inhibitors. *Prog Biophys Mol Biol*. 2021. <https://doi.org/10.1016/j.pbiomolbio.2021.02.002>.
37. Robson B. The use of knowledge management tools in viroinformatics. Example study of a highly conserved sequence motif in Nsp3 of SARS-CoV-2 as a therapeutic target. *Comput Biol Med*. 2020;125:103963.
38. Hunter JD. Matplotlib: a 2D graphics environment. *Comput Sci Eng*. 2007;9:90–5.
39. Konagurthu AS, Whisstock JC, Stuckey PJ, Lesk AM. MUSTANG: a multiple structural alignment algorithm. *Proteins Struct Funct Bioinform*. 2006;64:559–74.
40. Krieger E, Vriend G. YASARA View—molecular graphics for all devices—from smartphones to workstations. *Bioinformatics*. 2014;30:2981–2.
41. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 10: fundamental algorithms for Scientific Computing in Python. *Nat Methods*. 2020;17:261–72.

Publisher's Note

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