

SOFTWARE

Open Access



PIKACHU: a Python-based informatics kit for analysing chemical units

Barbara R. Terlouw^{*}, Sophie P. J. M. Vromans and Marnix H. Medema^{*} 

Abstract

As efforts to computationally describe and simulate the biochemical world become more commonplace, computer programs that are capable of *in silico* chemistry play an increasingly important role in biochemical research. While such programs exist, they are often dependency-heavy, difficult to navigate, or not written in Python, the programming language of choice for bioinformaticians. Here, we introduce PIKACHU (Python-based Informatics Kit for Analysing CHemical Units): a cheminformatics toolbox with few dependencies implemented in Python. PIKACHU builds comprehensive molecular graphs from SMILES strings, which allow for easy downstream analysis and visualisation of molecules. While the molecular graphs PIKACHU generates are extensive, storing and inferring information on aromaticity, chirality, charge, hybridisation and electron orbitals, PIKACHU limits itself to applications that will be sufficient for most casual users and downstream Python-based tools and databases, such as Morgan fingerprinting, similarity scoring, substructure matching and customisable visualisation. In addition, it comes with a set of functions that assists in the easy implementation of reaction mechanisms. Its minimalistic design makes PIKACHU straightforward to use and install, in stark contrast to many existing toolkits, which are more difficult to navigate and come with a plethora of dependencies that may cause compatibility issues with downstream tools. As such, PIKACHU provides an alternative for researchers for whom basic cheminformatic processing suffices, and can be easily integrated into downstream bioinformatics and cheminformatics tools. PIKACHU is available at <https://github.com/BTheDragonMaster/pikachu>.

Keywords: Cheminformatics kit, Python, Structure visualisation, *In silico* chemistry, Molecular fingerprinting

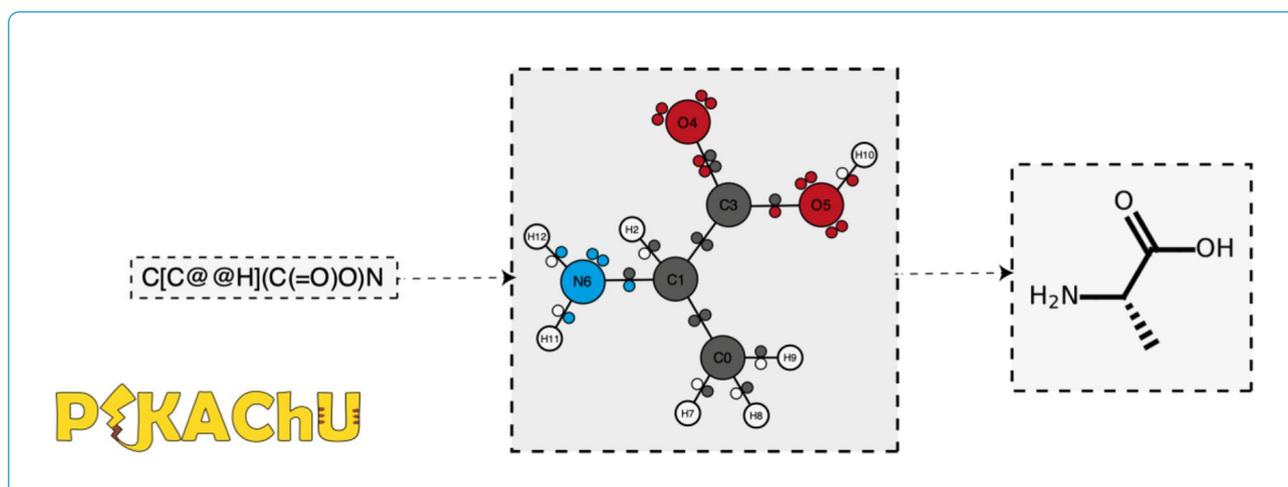
Graphical Abstract

*Correspondence: barbara.terlouw@wur.nl; marnix.medema@wur.nl

Bioinformatics Group, Wageningen University, Droevendaalsesteeg 1, 6708 PB Wageningen, The Netherlands



© The Author(s) 2022. **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.



Introduction

In a data-driven world where the discovery of novel natural and synthetic molecules is increasingly necessary, *in silico* chemical processing has become an essential part of biological and chemical research. Novel metabolites are compared or added to searchable chemical databases such as ChEBI [6], PubChem [10], NP Atlas [20], and COCONUT [17]; molecular structures are predicted from biological pathways [3, 16]; and bioactivities and pharmaceutical properties are predicted from chemical structure [1, 18, 21]. Such analyses rely on robust cheminformatics kits that can perform basic chemical processing, such as fingerprint-based similarity searches, substructure matching, molecule visualisation and chemical featurisation for machine learning purposes.

Typically, molecular processing by cheminformatics kits begins with the reading in of molecular data from chemical data formats, ranging from one-dimensional to three-dimensional molecular representations. One such format is the SMILES (Simplified Molecular-Input Line Entry System) format, which represents a molecule as a one-dimensional string, describing atom composition, connectivity, stereochemistry, and charge. More elaborate formats such as PDB and MOL use text files to store not just the abovementioned properties but also atom coordinates in three-dimensional space.

Depending on the application, different formats and subsequent processing are appropriate. Due to the vast number of possible chemical analyses, exhaustive cheminformatics kits have accumulated into software libraries that are so large that they can be hard to navigate, and rely on so many dependencies that they can be difficult to implement in software packages. As a result, the trade-off between time spent accessing and integrating these cheminformatics kits into a codebase and time spent on actual analyses is disproportionate for users

that need to perform simple *in silico* analyses such as reading in SMILES, drawing a molecule, or visualising a substructure. One popular open-source cheminformatics kit that suffers from this problem is RDKit [11]. While RDKit is an incredibly fast and powerful library that supports an immense variety of possible chemical operations, its use of both Python and C++ as programming languages as well as the sheer number of dependencies it relies on frequently causes compatibility issues when integrating RDKit into other programs, and disproportionately increases the number of libraries that need to be installed. Therefore, while RDKit is great for heavy-duty *in silico* analyses such as computing 3D conformers for a compound or constructing electron density maps, it is a bit heavyweight for the basic operations that most researchers in bioinformatics and cheminformatics require.

A second widely-used cheminformatics kit is CDK [22]. Written in Java, it is well-suited for implementation in web applications, and has successfully been used for molecular processing in the COCONUT database [17], the Cytoscape application chemViz2 [13], and the scientific workflow platform KNIME (Konstanz Information Miner) [2]. However, with Python becoming the programming language of choice for many scientists [4], especially those working in the growing field of (deep) neural networks, CDK is not always an ideal fit.

To make basic cheminformatics processing more accessible for Python programmers, we therefore introduce PIKACHU: a Python-based Informatics Kit for Analysing Chemical Units. PIKACHU is a flexible cheminformatics tool with few dependencies. It can parse molecules from SMILES, visualise chemical structures and substructures in matplotlib, perform Extended Connectivity Fingerprinting (ECFP) [15] and Tanimoto similarity searches, and execute basic reactions with a focus on natural

product chemistry. Therefore, we hope that PIKACHU can provide a convenient alternative for many Python-based bio- and cheminformatics tools and databases that only demand basic chemical processing.

Methods and implementation

Software description

PIKACHU is implemented in Python (v3.9.7). Its only dependency is the common Python package matplotlib (v3.4.3). PIKACHU can be run on Windows, MacOS, and Linux systems.

Parsing molecules from SMILES

PIKACHU takes a SMILES string as input and from it builds a graph object, in which nodes represent atoms and edges represent bonds (Fig. 1). For each atom, PIKACHU initially stores information on chirality, aromaticity, charge, and connectivity. For each bond, it stores bond type (single, double, triple, quadruple, or aromatic), neighbouring atoms, and cis–trans stereochemistry for double bonds. Once all atoms, bonds, and their connectivities have been stored, electron shells and orbitals are constructed for each non-hydrogen atom. Next, we determine the valency for each non-hydrogen atom, taking into account atom charge. For atoms of variable valency such as sulphur (2, 4 or 6) and phosphorus (3 or 5), we select a valency that is equal to or higher than the sum of non-hydrogen bonds and explicit hydrogen bonds, prioritising smaller valencies. Double, triple, quadruple and aromatic bonds contribute proportionally to this sum. If insufficient bonding orbitals are available to achieve the desired valency, the electrons in the valence shell are excited to higher-energy orbitals, such that each orbital contains at most one electron. Implicit hydrogens are then added to the structure such that the pre-determined valencies are obeyed.

An exception is made for nitrogens of valency 5, which are not chemically possible due to insufficient bonding orbitals but can sometimes be encountered in SMILES strings, especially in those describing compounds containing nitro groups. If such a valency 5 nitrogen is attached to at least one oxygen through a double bond, this double bond is interpreted as a single bond instead, the oxygen's charge is set to -1 and the nitrogen's charge is set to 1 , such that the nitrogen's valency becomes 4 and bonding laws are obeyed.

Subsequently, electrons are allocated to the p-orbitals of π bonds in double, triple and quadruple bonds, and atom hybridisation is determined from steric number. Then, all cycles in the graph are detected using an open-source Python implementation [12] of the simple cycle detection algorithm described by D. Johnson in 1975 [8]. PIKACHU removes all cycles smaller than

three atoms and identifies the smallest set of unique smallest rings (SSSR).

Next, the SSSR is used for aromaticity detection. This is done recursively: in each round, each cycle that has not yet been added to the set of aromatic cycles is evaluated with Hückel's $4n + 2$ rule on planar rings. We chose to assess aromatic cycles rather than systems as Hückel's rule is not always reliable for cyclic systems [7]. First, the hybridisation of all atoms in the cycle is examined. All atoms must be sp^2 -hybridised, or sp^3 -hybridised with a delocalisable lone pair that can be promoted to a p-orbital. If the cycle is planar and the sum of double bonds and lone pairs is odd, the cycle is considered aromatic. Aromatic bond stretches are locally kekulised, and double bonds are subsequently counted. When a cycle is considered aromatic, bonds and atoms in the cycle are set to aromatic, and lone pairs of sp^3 -hybridised atoms are promoted to p-orbitals such that the new hybridisation is sp^2 . Recursion is needed in case double bonds in cyclic systems are defined in such a way that not all sub-cycles contain the required number of bonds to obey Hückel's rule: when adjacent bonds are updated to aromatic, they will be counted in the next round of aromaticity detection (Additional file 2: Fig. S1). When, after an iteration, the number of aromatic cycles no longer changes, all aromatic cycles have been detected. From these cycles, PIKACHU defines aromatic systems, where aromatic cycles are considered part of an aromatic system if they share a bond with at least one other aromatic cycle in the system.

Electrons involved in σ bonds and aromatic bonds are only allocated after aromaticity detection. As electrons involved in aromatic systems are not localised to specific atoms or bonds, the p-orbitals of atoms in aromatic systems are emptied and their electrons stored in an `AromaticSystem` object.

Finally, any unpaired electrons are dropped back to lower-energy orbitals. A structure object is returned which can be visualised, kekulised, analysed through substructure matching and molecular fingerprinting, and altered through an assortment of built-in and custom chemical reactions.

If a SMILES string yields a structure object that is chemically incorrect due to too many or too few bonds being attached to an atom or valence shells not being filled appropriately in the case of organic atoms, PIKACHU gives a `StructureError`, informing the user that the parsed structure is chemically incorrect and gives a rough indication of why. Two examples of such `StructureError` messages are '*Error parsing "F/C(\Cl)=C(F)/Cl": Conflicting double bond stereochemistry*' and '*Error parsing "CN(=O)=O": Basic bonding laws have been violated*'.

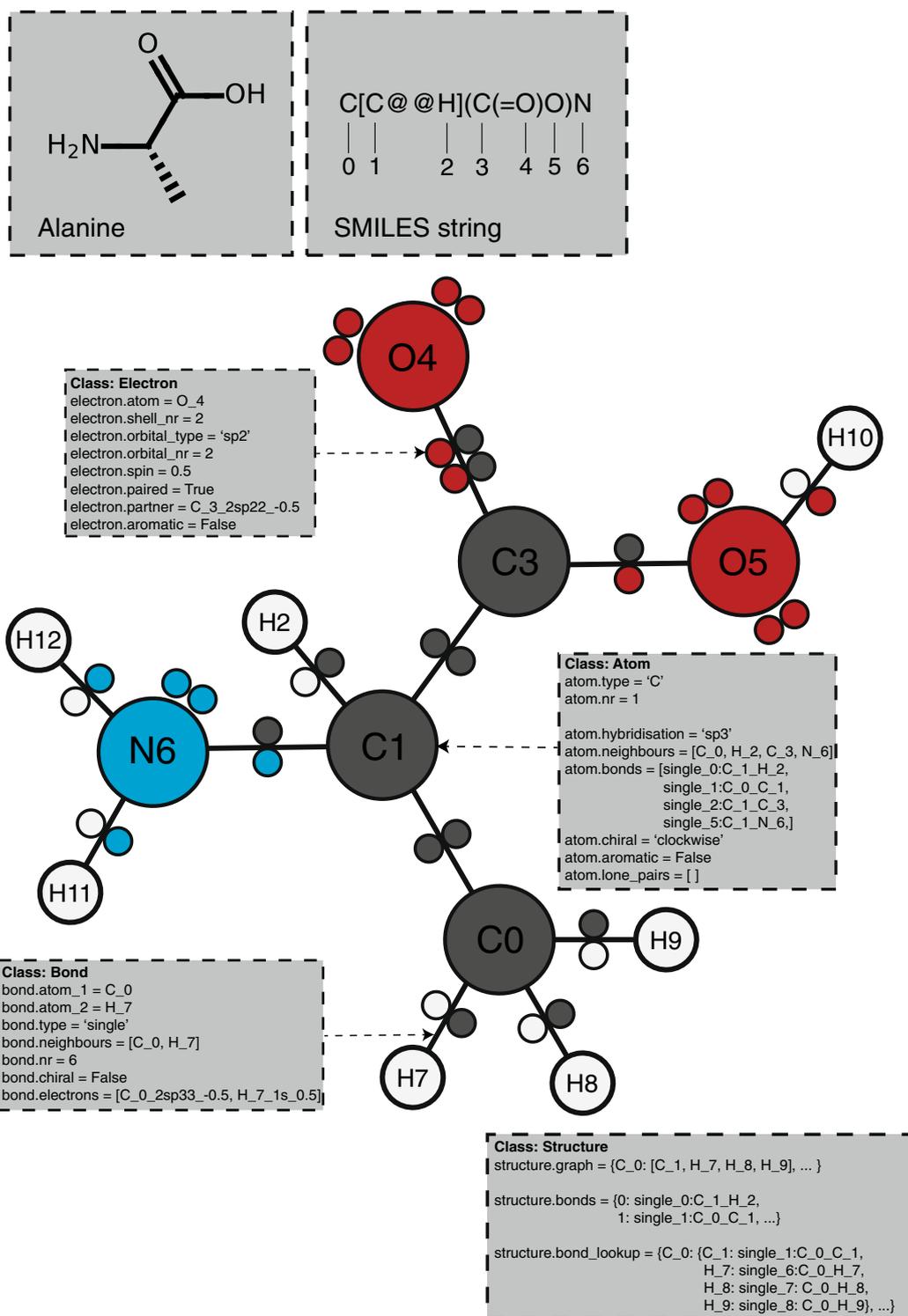


Fig. 1 Overview of the internal structure of PIKACHU's structure graphs. This example uses L-alanine, a small amino acid. The four bottom boxes in grey indicate attributes for each of PIKACHU's major classes: Structure, Atom, Bond and Electron.

Visualisation and kekulisation

Prior to visualisation, aromatic systems within a structure are kekulised so that aromatic systems can be represented by alternating single and double bonds. PIKACHU kekulises aromatic systems using a Python implementation [23] of Edmonds' Blossom Algorithm for maximum matching [5]. Next, atoms are positioned using PIKACHU's drawing software. PIKACHU's python-based drawing algorithm was adapted and improved from SmilesDrawer [14], an open-source JavaScript library for molecular visualisation. While written in different programming languages, the algorithms underlying the drawing software of PIKACHU and SmilesDrawer are largely identical. We will briefly recap this algorithm below; more detailed descriptions of the algorithm's elements can be found in the SmilesDrawer paper [14].

First, if indicated, PIKACHU's drawing algorithm removes hydrogens from the graph. Next, it finds the smallest set of smallest rings in the structure graph. As SmilesDrawer's SSSR implementation sometimes failed to detect some rings, leading to unreadable structure renderings (Additional file 2: Fig. S2), we implemented the SSSR algorithm ourselves. Next, like SmilesDrawer, PIKACHU classifies all rings into one of three groups: simple rings, overlapping rings, and bridged rings. Simple rings are standalone rings that do not have any overlapping atoms with any other rings. Overlapping rings are rings that overlap with one or more other rings, where the overlap between any two rings can comprise at most two atoms, any atom in the overlap is part of at most two rings, and no atoms in the ring overlap with bridged rings. Finally, bridged rings are rings that share more than two atoms with another ring, contain atoms that are part of three or more rings, or share atoms with another bridged ring (Fig. 2A).

After ring systems have been identified, atoms are placed onto a 2D coordinate system. If the molecule contains rings, positioning starts with the placement of an atom in a ring, prioritising bridged rings over simple and overlapping rings. Then, the graph is traversed one atom at a time in depth-first fashion. If an atom is part of a ring, the entire ring or ring system get placed at once. In the case of simple and overlapping rings, ring placement can be done using simple polygon geometry. For bridged rings, atoms are positioned using the force-spring model described by Kamada and Kawai [9], where all atoms of the bridged system are initially placed in a circle, and then pulled towards their optimal positions by minimising the difference between the desired bond length and the distance between neighbouring atoms, and maximising distances between non-neighbouring atoms. Non-ring atoms are positioned a bond length away from the previous atom, where the angle with respect to the previous

atom is determined by the number of neighbours the atom has (Fig. 2C), and the size of the molecular subtree behind each neighbouring atom (Fig. 2D). Stereochemically restricted double bonds are always forced into the appropriate cis- or trans conformation. Unlike SmilesDrawer, which directly infers bond stereochemistry from the SMILES string, PIKACHU draws this information from bond objects stored in the molecular graph. As an improvement on SmilesDrawer, PIKACHU attempts to resolve wrongly depicted stereobonds in rings by mirroring one of the neighbouring atoms into the ring. PIKACHU always selects the atom with the smallest protruding side chain for this purpose (Additional file 2: Fig. S3). When multiple consecutive stereobonds are found in a ring, PIKACHU adjusts them in order, never rotating a neighbour of the same bond twice.

Once all atoms have been assigned initial coordinates, atoms adjacent to rings are flipped outside of their ring where possible. Then, the drawing is checked for overlaps between atoms, and these overlaps are resolved by rotating branches of the molecule around single bonds. In PIKACHU, we have included an extra 'finetuning' option that is not present in SmilesDrawer. When the finetuning flag is set to True, all pairs of clashing atoms are detected. Then, the shortest path is calculated between all clashing atoms. First, PIKACHU determines which bonds are rotatable: bonds are considered unrotatable when they are a chiral bond, are adjacent to a chiral bond, or are in a cycle. As rotations around bonds located equally far away from two clashing atoms likely have the greatest impact on clash resolution, PIKACHU selects the rotatable bond that is positioned as close to the centre of the shortest path as possible. Next, PIKACHU takes the resulting set of bonds found for all clashes, and rotates each at 30° intervals, assessing and storing the number of clashes in the drawing after each iteration. The angle for which the number of steric clashes is minimised is chosen (Fig. 2B).

Finally, some bonds adjacent to chiral centres are replaced with backward and forward wedges. They are placed such that they do not neighbour more than one chiral centre where possible, they are not part of a ring, and point in the direction of the shortest branch leading from a chiral centre, in that order of priority. The resulting image is subsequently written to a.svg or.png file or displayed directly in matplotlib.

Structure annotation

PIKACHU provides the option to add custom annotations to structures. Each Atom instance contains an 'annotations' attribute, which points to an AtomAnnotation instance. An AtomAnnotations instance can contain as many annotations as the user requires. Annotations can be added to all atoms in a structure at once by defining

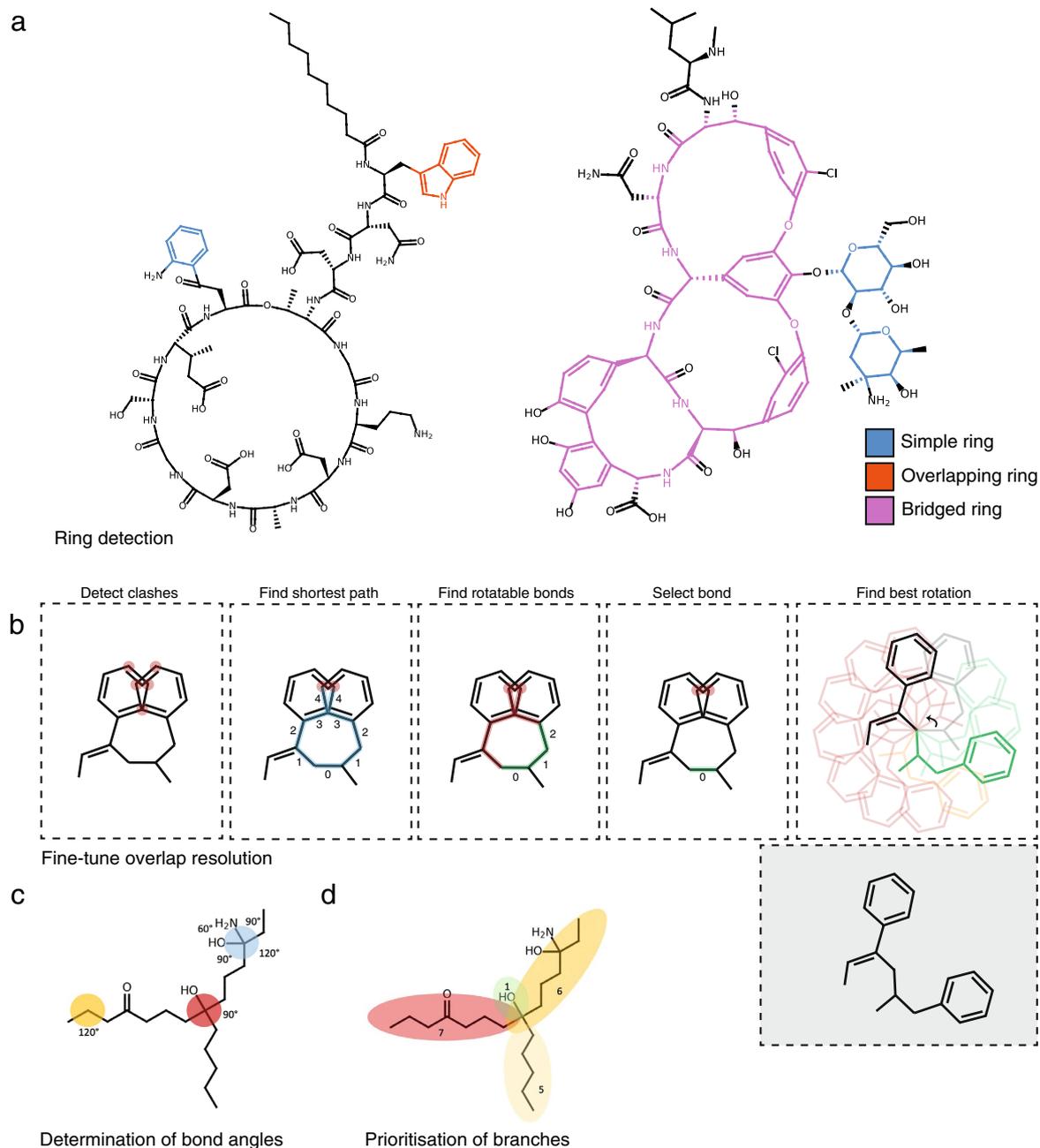


Fig. 2 PIKACHU's drawing algorithm. **A** Examples of simple (blue), overlapping (red) and bridged (pink) rings. Note that the aromatic rings in pink become part of the bridged ring system because they overlap with bridged rings. **B** PIKACHU's 'finetuning' algorithm. First, clashes are detected and the shortest path between them is found. The rotatable bond with the shortest distance to the centre of the shortest path is chosen (indicated with numbers). 12 rotations at incremental angles of 30° are evaluated for clashes. The best rotation is chosen. **C** Determination of bond angles based on neighbouring atoms. If an atom has 3 or fewer non-hydrogen neighbours, the angles default to 120° (yellow). If an atom has 4 non-hydrogen neighbours, angles default to 90° if three or more of the branches have a depth more than 1, or three or four branches have a depth of exactly 1 (red). If however two of the branches have a depth of exactly 1 (blue), the angle is set to 120° between the two longest branches, 90° between any short branch and any long branch, and 60° between the shortest branches. **D** Positioning of neighbouring branches depends on the depth of each branch: the two longest branches (red and dark yellow, depths 7 and 6 respectively) are always placed opposite one another.

the name of the attribute with a string, and optionally providing a default value for the attribute. Subsequently, specific values can be added and retrieved for specific atoms or atom sets. A manual providing an example can be found on the PIKACHU wiki.

Substructure matching

PIKACHU detects occurrences of a substructure in a superstructure in five steps. In all steps, hydrogens are ignored. First, PIKACHU checks for each atom type in the substructure if enough atoms of these types are accounted for in the superstructure. Second, it assesses for each atom in the substructure whether an atom exists in the superstructure with the same connectivity, looking at directly neighbouring bonds and atoms. Third, using the atom with the most diverse connectivity as a seed, it finds matches of the substructure in the superstructure using a depth-first search algorithm, ignoring stereochemistry. By first looking at atom type and atom connectivity, and by using atoms of diverse connectivity as seeds for substructure matching, the number of calls to the computationally expensive depth-first search function is minimised. Fourth, for each match, it determines if all chiral centres in the substructure have the same orientation as corresponding chiral centres in the superstructure. Fifth, PIKACHU checks if cis–trans orientation of double bonds in the substructure matches that of double bonds in the superstructure. Chiral centre and double bond stereochemistry checks can be toggled by the user independently of one another. If chirality of bonds and atoms are considered, substructures with undefined stereochemistry will still match to parent structures with defined stereochemistry. This does not apply in reverse: if a stereocentre or stereobond is defined for a substructure, it will not match to parent structures with undefined stereochemistry.

The algorithm described is somewhat similar to the Ullmann algorithm [19], which first assesses if a candidate subgraph contains enough nodes of the correct degree prior to substructure matching and selects nodes of the most unique degree as seeds. A key difference is that PIKACHU's substructure matching algorithm also takes into account the identity of a node's neighbours, not just a node's degree.

Substructures can be easily visualised through a range of functions in PIKACHU's 'general' library.

Fingerprinting

PIKACHU uses ECFP [15], which is an improved version of the classical Morgan fingerprinting also taking into account cycle membership, to perform similarity searches and convert molecules to bit vectors for machine learning featurisation. Using Python's inbuilt hashlib library,

PIKACHU initialises each atom to a 32-bit hash, derived from a tuple containing information on heavy neighbours, valence, atomic number, atomic weight, charge, hydrogen neighbours, and ring membership. Then, each atom hash is iteratively updated with hashes from its neighbours, as well as the distance from the neighbour to the atom and stereochemical information if the atom is a chiral centre. The number of iterations depends on a radius which can be set to any number (default=2 for ECFP-4 fingerprinting). The ECFP algorithm was described in detail by Rogers and Hahn in 2010 [15]. Finally, duplicate hashes are removed, as well as different hashes representing the same substructure, yielding a set of 32-bit hashes that constitutes a molecule's fingerprint.

Using ECFP fingerprinting, PIKACHU can calculate Jaccard/Tanimoto distance and/or similarity between any two molecules. Furthermore, PIKACHU can convert molecule libraries into bit vectors of varying lengths (default=1024) and an accompanying list of substructures represented by those bit vectors that can be used in downstream machine learning algorithms.

Defining reaction targets

In order to facilitate implementation of reactions and reaction pathways, PIKACHU lets users define target bonds or atoms within substructures with a set of dedicated functions. These functions take a SMILES string representing a substructure, and either one or two integers that define an atom or a bond between two atoms respectively. For example, the SMILES string 'C(=O)NC', accompanied by the integers 0 (pointing to the first C atom) and 2 (pointing to the N atom), represents a peptide bond. The occurrences of these bonds/atoms are then detected within a superstructure through a substructure search and are returned as a list of bonds/atoms. Subsequently, the returned bonds/atoms can be used as reaction targets, for instance for bond hydrolysis or atom methylation, using functions in PIKACHU for breaking or creating bonds and adding or removing atoms. Reactions currently have to be encoded manually using a library of functions included in PIKACHU, which include functions for creating bonds, breaking bonds, adding and removing atoms, and splitting disconnected graphs into separate structures. We provided in-built condensation and hydrolysis functions, as well as a more elaborate ketoreductase function, as examples on our GitHub page.

Characterisation and visualisation of the polyketide ketoreduction reaction

We demonstrated the implementation of reactions using PIKACHU by characterising and visualising a polyketide ketoreduction reaction. We built the ketoreduction reaction by first defining a reaction target as described above, in this case a β -keto bond, and detecting its position in a polyketide chain. Next, we wrote a function that reduces the double carbonyl bond to a single bond, which identifies and removes the π -electrons in the double bond, sets the bond type to single, adjusts the hybridisation of the atoms involved and finally updates the structure object through PIKACHU's refresh functions. To finalise the reaction, two hydrogen atoms were added to the carbon and oxygen atoms of the former carbonyl bond using PIKACHU's `add_atom` function. Finally, to visualise the reaction, we highlighted the atoms and bonds of the newly formed hydroxyl group in red and drew the molecule.

Detailed instructions on how to make full use of PIKACHU's range of functionalities, as well as the script used to implement the ketoreduction reaction, can be found in the online documentation.

Validation

To assess the correctness of PIKACHU's SMILES reading and writing software, we converted all SMILES strings from the NP Atlas database into PIKACHU Structure instances. Subsequently, we converted these structure instances back to SMILES strings. Next, we canonicalized the PIKACHU-generated SMILES and the original SMILES using RDKit (v2020.09.1.0), setting the 'isomericSmiles' flag to 'True' such that correct interpretation of cis–trans bond configuration and the stereochemistry of chiral centres could also be assessed. If the two canonicalized SMILES were identical, a SMILES to structure to SMILES conversion was considered correct.

To measure PIKACHU's drawing readability, atom coordinates were computed with PIKACHU and RDKit's `rdCoordGen` module (v2020.09.1.0) for the 32,552 molecules in the NPAtlas database (v2021_08) and the 100,000 smallest molecules from the ChEMBL database (release 30). Next, all drawings were assessed for clashes. A clash was defined as two non-neighbouring atoms sitting at less than half an average bond length distance from each other in Euclidean space. Total number of clashes, number of structures containing clashes, and the number of structures that gave drawing errors were recorded.

To assess PIKACHU's drawing accuracy, we included a MOL file writer into PIKACHU, which stores PIKACHU-computed atom coordinates and connectivities as a MOL file. We generated such MOL files for the entire NP Atlas database and the 100,000 smallest molecules

from the ChEMBL database, read the resulting MOL files with RDKit's `rdMolFiles` module (v2020.09.1.0), stored the resulting molecules as SMILES strings, and using RDKit canonicalized both the original input SMILES and the SMILES produced from the PIKACHU-generated MOL files setting the 'isomericSmiles' flag to 'True'. If the SMILES were identical, a PIKACHU-generated drawing was considered 'correct'.

Speed assessment

PIKACHU's speed was assessed with Python's 'time' module. As we particularly designed PIKACHU with natural product chemistry in mind, which typically involves larger and more heavily cyclised compounds than most molecules stored in small-molecule databases, we decided to test drawing speed on two different databases: the NP Atlas database and the ChEMBL database. For each database, we randomly selected 10,000 molecules and timed drawing speed at 10, 20, 50, 100, 200, 500, 1000, 2000, 5000 and 10,000 drawn structures using Python's 'time' module.

Results and discussion

PIKACHU is a dependency-light cheminformatics kit implemented entirely in Python. With only `matplotlib` as dependency and an extensive readme, wiki, tutorials, and example scripts on its GitHub page, PIKACHU is easy to run and install, and suitable for integration into bioinformatics and cheminformatics pipelines. Below, we will first assess PIKACHU's ability to correctly interpret SMILES, draw structures accurately and readably, detect and visualize substructures, and perform ECFP fingerprinting. We also measured PIKACHU's SMILES reading and structure drawing speed. Next, we demonstrate how PIKACHU can be used to implement and visualise reactions. Finally, we compare PIKACHU to the state-of-the-art cheminformatics kits/chemical drawing libraries RDKit, ChemDraw and SmilesDrawer.

SMILES reading and writing

We assessed PIKACHU's ability to parse and generate correct SMILES syntax by comparing the SMILES it converts to SMILES generated by RDKit, a well-established cheminformatics package. As PIKACHU was created with natural product chemistry in mind, which typically involves large and heavily cyclised molecules the function of which depends heavily on stereochemistry, we performed our validation with the NP Atlas database. This database contains 32,552 manually curated natural product structures and their corresponding isomeric SMILES strings. PIKACHU failed to convert 1 SMILES from NP Atlas (~0.003%) to structure graphs, which was

an erroneous SMILES describing a nitrogen atom with a valency of 5, which is impossible considering that nitrogen only has four electron orbitals available for bonding in its valence shell. This SMILES attempted to describe a nitro group, which PIKACHU tolerates even for valency 5 nitrogens. However, this representation of a nitro group was unconventional and incorrect, with two superfluous hydrogen atoms attached to the nitrogen. This demonstrates how the detailed graph-based, object-oriented encoding of chemical structures down to the electron level in PIKACHU intrinsically ensures that all structures that are loaded are chemically valid. Of the remaining 32,551 SMILES-to-structure-graph-to-SMILES conversions, only one yielded a SMILES string that described different chemistry than the original: three carbon-13 atoms were interpreted as carbon-12 (Additional file 2: Table S1, third row). As PIKACHU does not yet support isotopic differentiation, this is not unexpected.

Additionally, we manually assessed the correctness of 22 SMILES-to-graph conversions by reading in and subsequently drawing the SMILES in PIKACHU. We chose the SMILES such that a variety of syntax representations and chemistries were represented, including rings, aromatic systems, charge, stereocentres and bond stereochemistry. Some SMILES describe the same structures but use a different syntax. PIKACHU handled all SMILES correctly, accurately detecting and visualising all aforementioned chemical properties (Fig. 3).

PIKACHU is not suitable for reading in molecules with a high number of recursive cycles, such as buckminsterfullerene. As PIKACHU detects all possible cycles within a molecule to determine aromaticity of cyclic systems, this step takes so long to compute that the program appears to get 'stuck'. However, there exist only a handful of examples of such molecules, none of which have any real practical biological or chemical relevance.

Structure visualisation

Another key feature of PIKACHU is molecular visualisation from SMILES. PIKACHU's drawing software relies on similar logic to that of SmilesDrawer, a JavaScript SMILES drawing library. In our software, we added a few improvements: we fixed cis–trans stereochemistry detection (Additional file 2: Fig. S3), included an extra overlap resolution step (Fig. 2B), and implemented an improved version for finding the smallest subset of smallest rings, a key step in correctly depicting cycles. There is always a bit of debate regarding the visualisation of molecular macrocycles. Many organic chemists opt for a 'honeycomb' architecture, as employed by ChemDraw and CDK, to better represent the 3D architecture of a molecule, hinting at long-distance interactions that may take place within the compound (Additional file 2: Fig. S4B).

However, this representation does not instantly draw the eye to sites of cyclisation, a drawback for natural product biologists and bioinformaticians who are often interested in the biosynthetic steps involved in a compound's assembly. As PIKACHU was created with natural product chemistry in mind, we chose to use a polygon representation for macrocycles, which clearly shows cyclisation sites (Additional file 2: Fig. S4A).

While PIKACHU always detects and interprets aromaticity internally, it currently only supports drawing structures in a kekulised format.

The most important aspect of automated molecular visualisation is accuracy: users need to be able to rely on the correctness of drawing software, especially when processing a large number of structures at once making it impossible to inspect each image independently. To this purpose, we visualised a chemically diverse set of structures from the ChEMBL and NP Atlas databases, and tested if RDKit could interpret correct chemistry from PIKACHU-generated atom coordinates. Out of the 32,552 structures in the NP Atlas database, only 40 (~0.12%) were drawn incorrectly. Of these, 33 were drawn wrongly due to incorrect depiction of cis–trans chemistry of double bonds adjacent to nested rings (Additional file 2: Table S1). Additionally, PIKACHU failed to convert 8 structures to drawings (~0.02%). One of these was the same incorrectly defined SMILES describing the valency 5 nitrogen that was found previously. The remaining failures largely resulted from ChiralityErrors: errors raised by PIKACHU when it cannot correctly depict cis/trans chemistry of a double bond. With 32,504 correctly drawn structures, PIKACHU achieves a drawing accuracy of 99.85%. PIKACHU performed comparably on the 100,000 smallest molecules from the ChEMBL database (99.21% accuracy, 0.16% incorrect drawings, 0.63% unsuccessful conversions; Fig. 4A). Comprehensive lists and example depictions of SMILES leading to incorrect interpretations can be found in Additional file 2: Table S1 and Additional file 1.

We additionally assessed the readability of these PIKACHU-rendered drawings by automatically detecting steric clashes from PIKACHU-generated atom coordinate sets. Only 4.95% of NP Atlas SMILES renderings contained steric clashes, with ~1.62 clashes per clashing structure. PIKACHU was better at processing the ChEMBL database, with only ~0.30% of drawings containing clashing atoms. This makes sense, as NP Atlas contains a higher proportion of highly cyclised systems and large molecules, properties which make it more difficult to readably depict a molecule in a plane. As the honeycomb approach of depicting molecules solves some of these issues, we hope to implement the option to visualise molecules

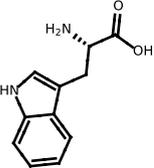
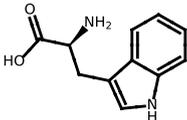
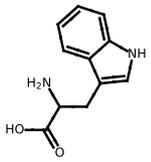
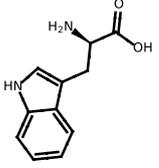
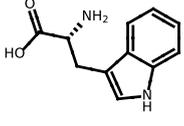
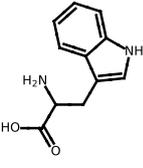
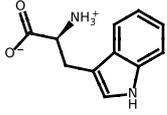
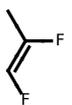
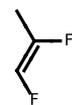
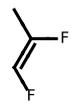
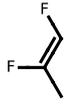
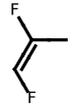
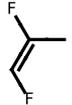
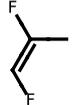
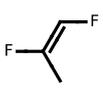
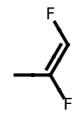
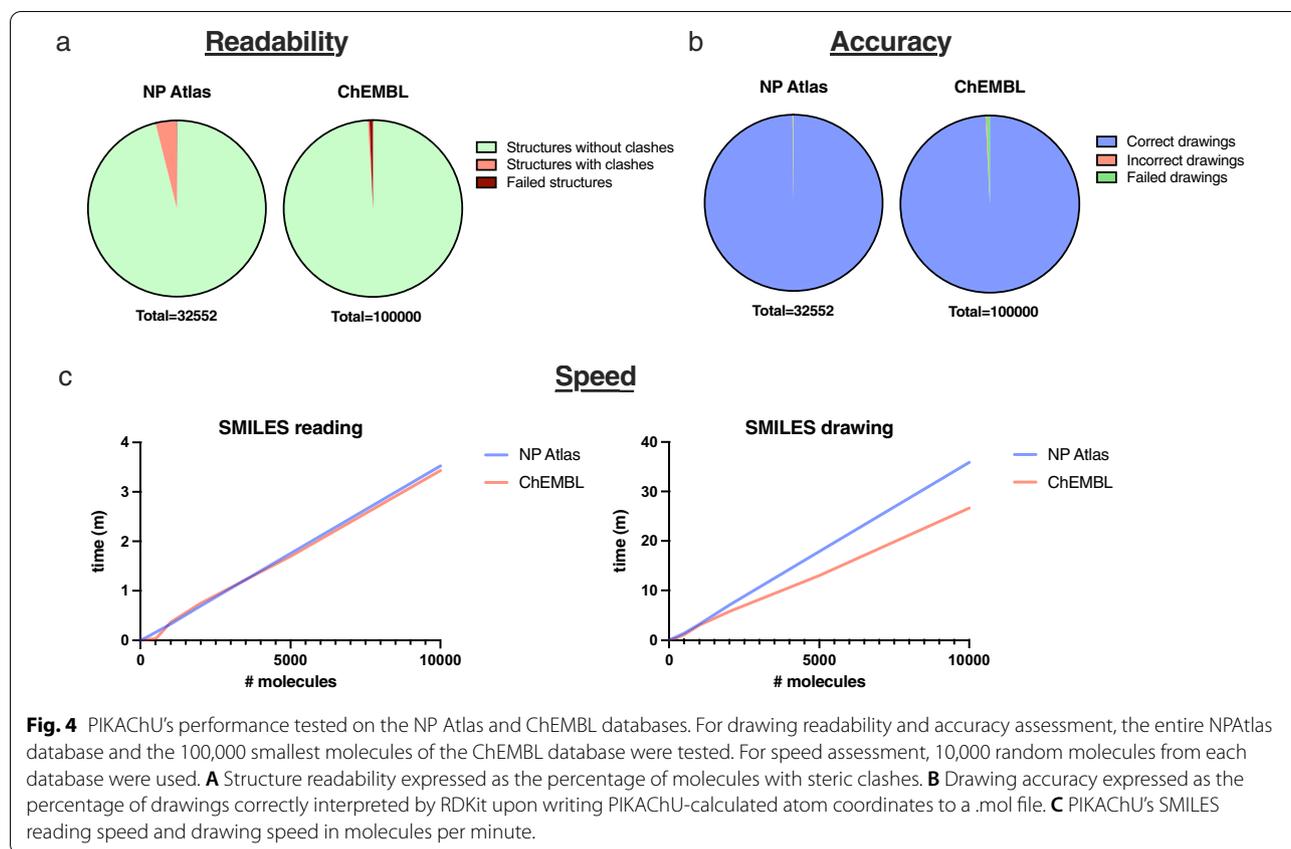
L-Tryptophan	 <chem>C1=CC=C2C(=C1)C(=CN2)C[C@@H](C(=O)O)N</chem>	 <chem>c1[nH]c2ccccc2c1C[C@H](N)C(=O)O</chem>	 <chem>NC(Cc1[nH]c2ccccc12)C(=O)O</chem>		
D-Tryptophan	 <chem>C1=CC=C2C(=C1)C(=CN2)C[C@H](C(=O)O)N</chem>	 <chem>c1[nH]c2ccccc2c1C[C@@H](N)C(=O)O</chem>	 <chem>NC(Cc1[nH]c2ccccc12)C(=O)O</chem>		
L-Tryptophan zwitterion	 <chem>c1[nH]c2ccccc2c1C[C@H]([NH3+])C(=O)[O-]</chem>				
cis-difluoromethylethylene	 <chem>F/C=C(F)/C</chem>	 <chem>F/C=C(F)C</chem>	 <chem>F/C=C(F)/C</chem>	 <chem>C/C(F)=C/F</chem>	 <chem>F/C(C)=C/F</chem>
trans-difluoromethylethylene	 <chem>F/C=C(F)/C</chem>	 <chem>F/C=C(F)C</chem>	 <chem>F/C=C(F)/C</chem>	 <chem>C/C(F)=C/F</chem>	 <chem>F/C(C)=C/F</chem>
benzene	 <chem>c1ccccc1</chem>	 <chem>C1C=CC=C=C1</chem>	 <chem>C1=CC=CC=C1</chem>	 <chem>C=1C=CC=CC1</chem>	 <chem>C=1C=CC=CC=1</chem>

Fig. 3 Assessment of PIKACHU's SMILES reader. Structures were drawn from the SMILES written beneath the molecule depictions. All 22 structures were correctly drawn.



using either the honeycomb or the polygon strategy in the future.

In Fig. 5, we show nine examples of structures rendered by PIKACHU. Due to the drawing algorithm that PIKACHU employs for complex ring systems, 5-membered and 6-membered rings often appear distorted, as observed for vancomycin and aplasmomycin B. Additionally, PIKACHU's overlap resolution step, while resolving a lot of steric clashes, sometimes results in carbon-carbon bonds being placed at an 180° angle which makes the structure less interpretable, as seen for PIKACHU's depiction of the molecule nanokid.

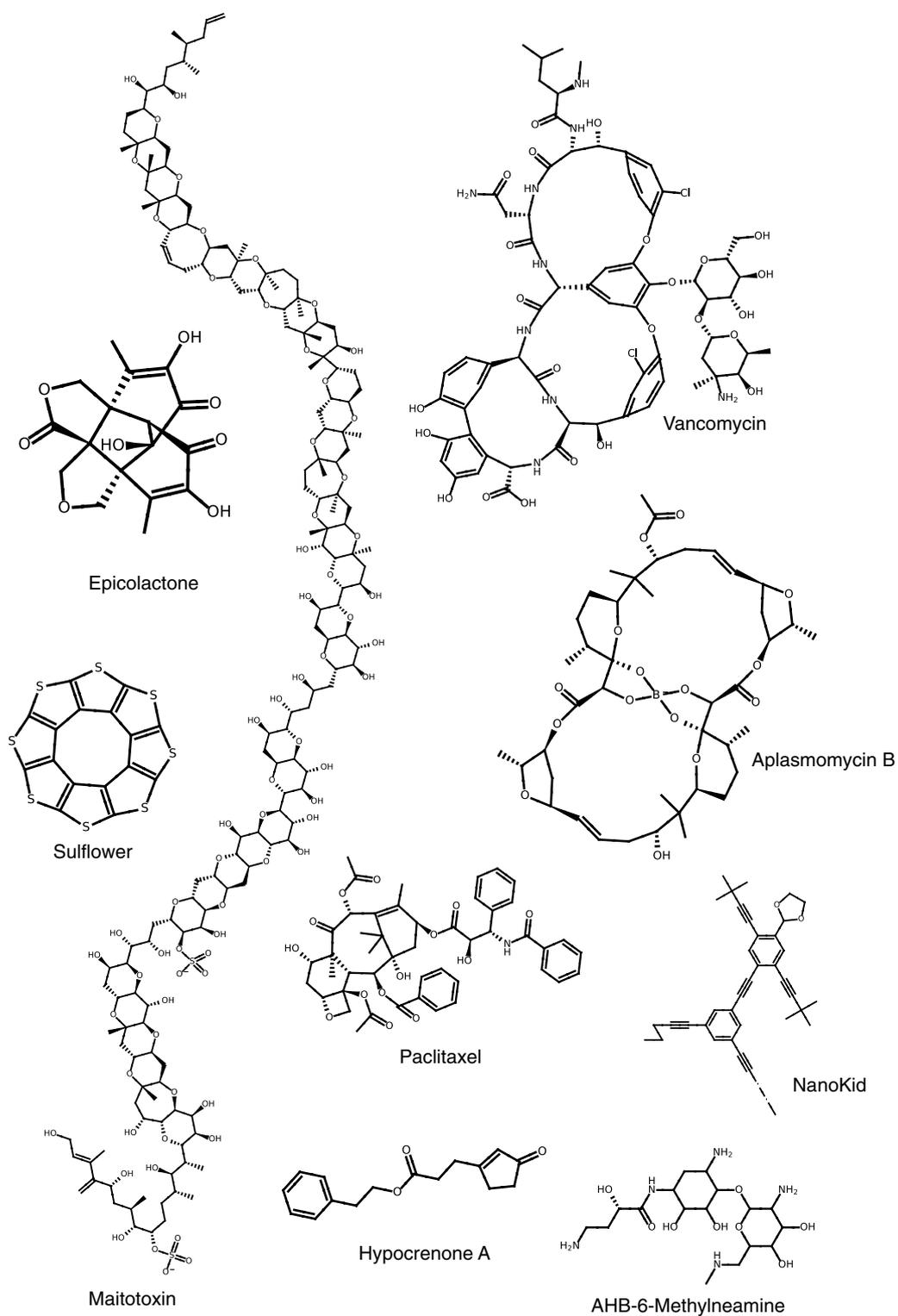
Speed assessment

We assessed PIKACHU's SMILES reading and structure drawing speed by drawing 10,000 random molecules from the NP Atlas database and the ChEMBL database (Fig. 4C). With an average reading speed of ~2,874 SMILES per minute and an average drawing speed of ~279 molecules per minute for the NP Atlas database and ~375 molecules per minute for the ChEMBL database on a single laptop core, it is clear that PIKACHU spends the bulk of its time rendering an image on atom positioning, not on SMILES reading. The discrepancy

between the two databases can be explained by the nature of the molecules contained within them: typically, natural products are larger and more cyclised than the average small molecule. This makes PIKACHU's drawing speed one order of magnitude slower than RDKit's (Additional file 2: Table S2), which is expected considering that PIKACHU is a pure Python package while RDKit generates drawings with pre-compiled C++ code. Also, PIKACHU's finetuning step is computationally expensive, likely leading to an increase in computational time. Still, PIKACHU is fast enough for integration into pure-Python bioinformatics and cheminformatics pipelines.

Substructure detection

A dedicated set of functions ensures that performing substructure searches using PIKACHU is straightforward. With a single line of code, users can visualise a single occurrence of a substructure, all occurrences of a substructure, or all occurrences of a range of substructures in a chemical compound (Fig. 6A). Substructure searches are fast due to several pre-processing steps, ensuring that the expensive graph matching algorithm is only executed

**Fig. 5** Various molecules rendered by PIKACHU

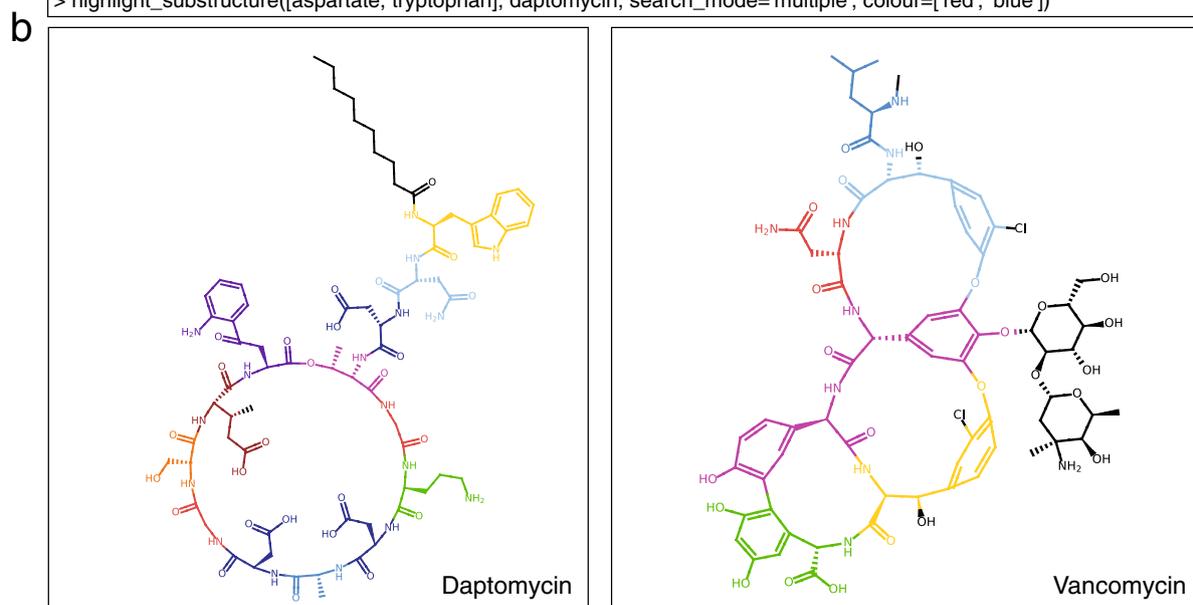
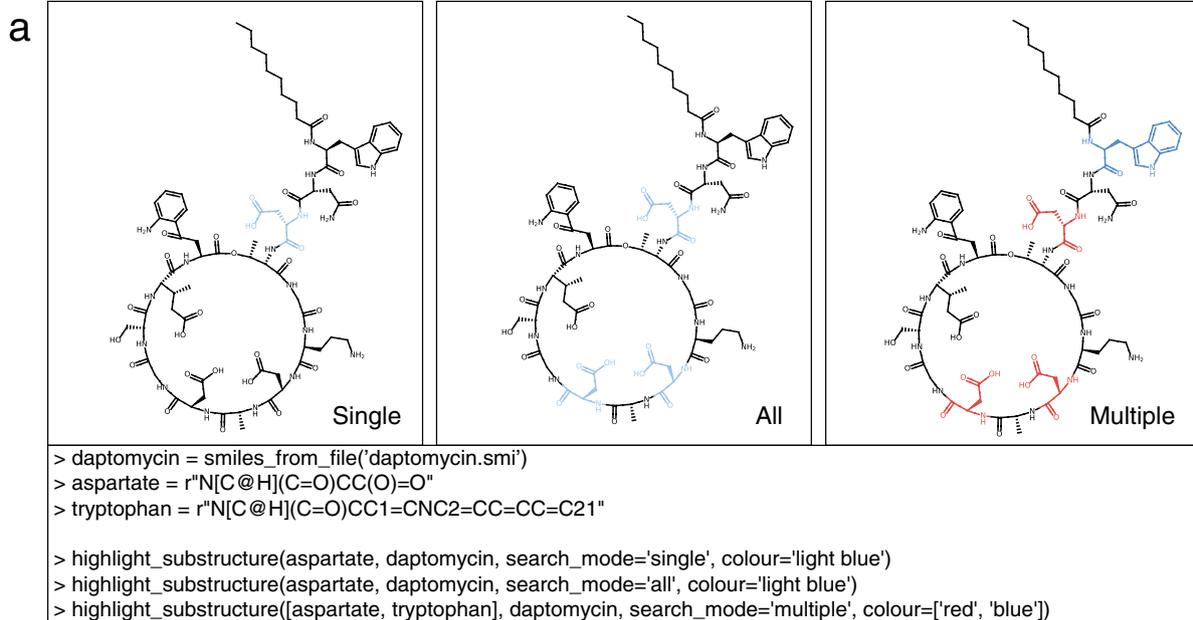


Fig. 6 Substructure matching with PIKACHU. **A** From left to right: examples of highlighting a single instance of a substructure, all instances of a substructure, or all instances of multiple substructures. In the example, occurrences of aspartic acid and tryptophan were searched in the superstructure daptomycin. The code used to generate the images is displayed underneath the panels. **B** PIKACHU's substructure matching algorithm used to visualise all amino acid components of the antibiotics daptomycin (left) and vancomycin (right). Code can be found at https://github.com/BTheDragonMaster/pikachu/blob/main/example_scripts/amino_acid_composition.py

when a match is likely. Stereochemistry matching, activated by default, can be toggled on and off.

With PIKACHU's substructure matching algorithm, we visualised the amino acid composition of the cyclic peptides daptomycin and vancomycin, using only a single line of code for each (Fig. 6B). Colours are fully and easily

customisable, and can be provided as hex codes or as colour names.

ECFP fingerprinting

To quickly determine the approximate similarity between two molecules, PIKACHU employs ECFP fingerprinting

[15]. PIKACHU hashes each molecule into a set of unique identifiers, each of which represents a substructure. Collectively, these identifiers make up a molecule's fingerprint. Then, PIKACHU calculates the Jaccard/Tanimoto similarity between two molecules by comparing their fingerprints, giving a measure of molecular similarity and/or distance.

Here, we showcase PIKACHU's ECFP fingerprinting by calculating and subsequently constructing a tSNE plot of the molecular distances between 36 calcium-dependent lipopeptides. Lipopeptides of the same family grouped together (Additional file 2: Fig. S5), confirming that PIKACHU's ECFP fingerprinting yields reliable measures of chemical similarity.

Additionally, PIKACHU's ECFP fingerprinting makes it possible to generate bit vectors from molecule sets, where each element in the vector represents the presence/absence of a specific substructure. These can subsequently be used as interpretable molecular featurisations for machine learning.

Building in silico reactions using PIKACHU

PIKACHU provides a platform for the creation and visualisation of reaction mechanisms by providing a range of reaction functions that can be used to make or break molecular bonds, add or remove atoms and alter the chirality of stereocentres. In addition to these built-in reaction building blocks, PIKACHU allows users to easily define more complex reactions through the manipulation of atom and bond object attributes. Additionally, PIKACHU supports fully customisable structure annotation, which is useful for keeping track of reaction steps, reaction targets, or atom origin. As a proof of principle, we used PIKACHU to define and visualise a polyketide ketoreduction reaction, catalysed by a ketoreductase polyketide synthase domain during polyketide synthesis, employing both built-in and custom reaction functions (Additional file 2: Fig. S6). This example, as well as a comprehensive guide containing instructions on how to build reaction mechanisms using PIKACHU, can be found in the online documentation.

While creating reaction pathways with PIKACHU enforces chemically correct conversions by checking at each step if a structure is chemically correct, it is more laborious than similar functionalities in other cheminformatics kits such as RDKit, which use reaction SMILES and

atom mapping to perform chemical reactions. We intend to implement reaction SMILES and atom mapping into PIKACHU in the future.

PIKACHU compared to state-of-the-art chemical drawing software

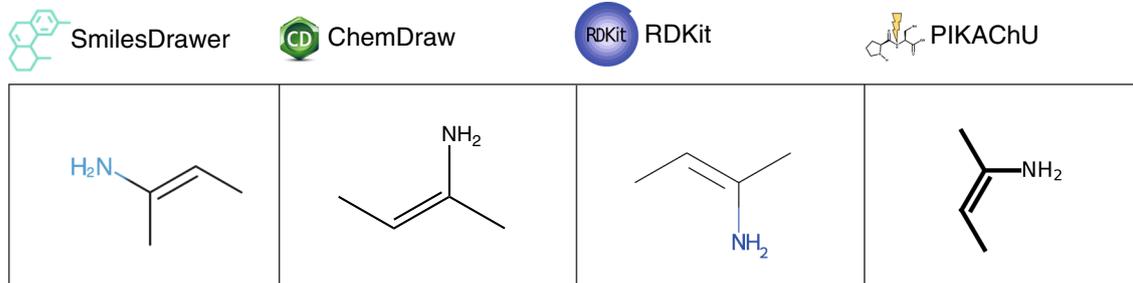
Finally, we assessed how PIKACHU performs compared to existing chemical drawing software. To this purpose, we visualised various structures in PIKACHU (v1.0.5), RDKit (v2020.09.1.0), ChemDraw (v20.1.0.112) and SmilesDrawer (v1,2.0), and manually assessed drawing quality and correctness (Fig. 7). Only SmilesDrawer occasionally produced an incorrect structure, confusing cis–trans stereochemistry when stereochemistry is defined in or after a branch (Fig. 7A). For heavily cyclised molecules (Fig. 7B–E), we see a difference between the 'honeycomb' (RDKit and ChemDraw) and the 'polygon' (PIKACHU and SmilesDrawer) approaches of cycle positioning. The honeycomb approach ensures minimal distortion of microcycles, even when they are part of larger systems; as such, RDKit and ChemDraw render molecules such as vancomycin with fewer distortions than SmilesDrawer and PIKACHU (Fig. 7B). However, when the honeycomb approach does not work because of steric constraints, forcing microcycles into regular polygons can distort the macrocyclic structure to the extent that the drawing becomes unreadable. This is the case for aplasmomycin B (Fig. 7C), which is drawn with fewer bond overlaps by SmilesDrawer and PIKACHU. In structures where microcycles and macrocycles are separate, there is little difference in structure rendering between the two approaches (Fig. 7D, E). It must be said that while readable, these depictions are still far from optimal, and chemists could produce better diagrams by manually tweaking the drawings of these molecules in ChemDraw. This demonstrates that even with state-of-the-art software, 2-dimensional automatic visualisation remains a challenge, particularly for constrained ring systems.

PIKACHU has a slight advantage over RDKit in drawing molecules of varying sizes, automatically adjusting the canvas size based on the size of the molecule to be drawn. This also means that PIKACHU's font size, bond length and bond thickness maintain a constant ratio across different drawings, which is not the case for RDKit (Fig. 7D, E). While it is possible to manually adjust canvas size in RDKit, some extra coding steps are required to achieve this.

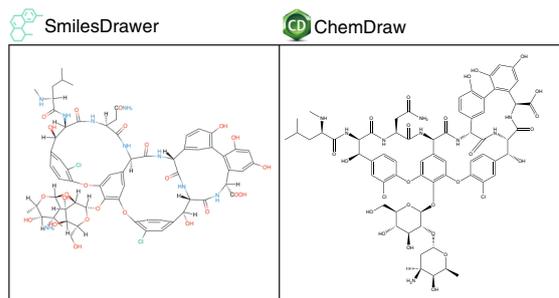
(See figure on next page.)

Fig. 7 Comparison of PIKACHU to various other chemical drawing software packages. **A** SmilesDrawer, ChemDraw, RDKit and PIKACHU drawings given the SMILES string C/C=C(N)/C. While ChemDraw, RDKit and PIKACHU all draw the cis–trans stereochemistry of the double bond correctly, with the amino group cis of the methyl group, SmilesDrawer draws the stereobond in the wrong orientation. **B** SmilesDrawer, ChemDraw, RDKit and PIKACHU drawings of the heavily cyclised molecule vancomycin. **C** SmilesDrawer, ChemDraw, RDKit and PIKACHU drawings of the molecule aplasmomycin B, **D** epicolactone, and **E** paclitaxel.

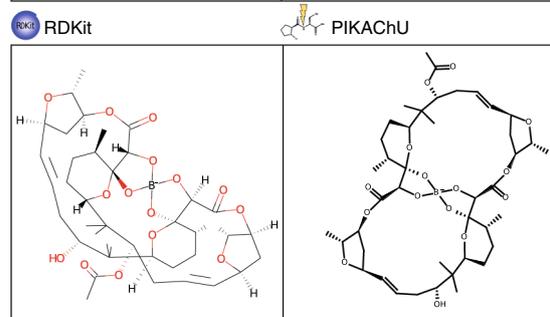
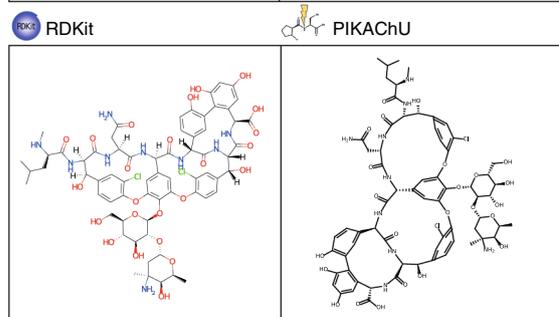
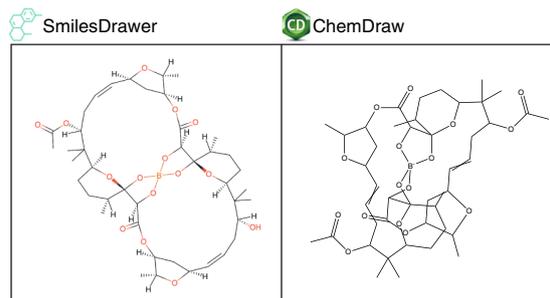
a



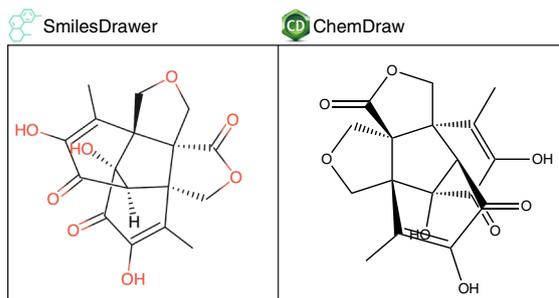
b



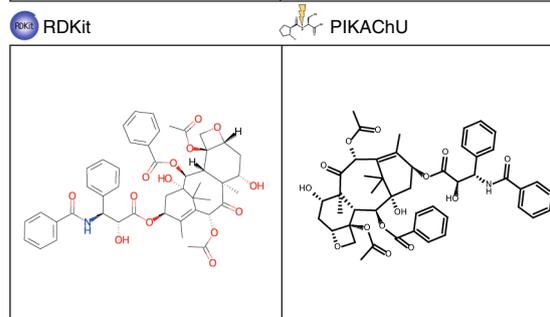
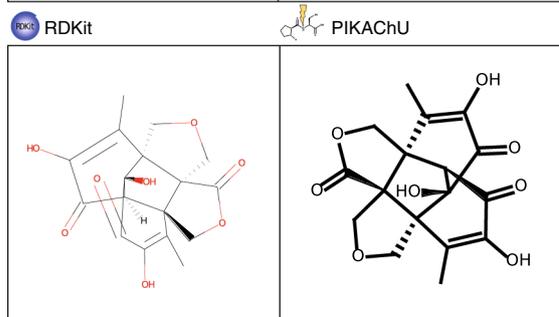
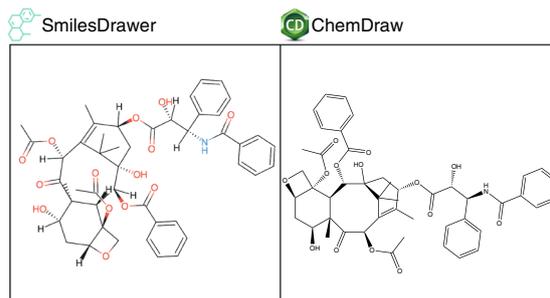
c



d



e

**Fig. 7** (See legend on previous page.)

PIKACHU's visual output is far more customizable than that of SmilesDrawer, allowing for molecule rotation, drawing multiple molecules on a single canvas, and custom colouring of each individual bond and atom, supporting hex-codes as well as a range of descriptive strings.

While ChemDraw accommodates high-quality and highly customisable visualisation, it is not open-source. This makes PIKACHU more suitable for integration into automated open-source pipelines required by many projects.

Conclusions

We developed PIKACHU, a dependency-light cheminformatics library implemented entirely in Python. Having extensively validated our software, we conclude that, while RDKit heavily outperforms PIKACHU in terms of speed, PIKACHU performs sufficiently fast and reliably to be suitable for cheminformatics and bioinformatics pipelines. Backed by extensive online documentation, easy and straightforward installation, and state-of-the-art automated visualisation software, we hope that PIKACHU can provide a convenient alternative for chem- and bioinformaticians programming in Python.

Abbreviations

PIKACHU: Python-based Informatics Kit for Analysing Chemical Units; SMILES: Simplified Molecular-Input Line Entry System; ECFP: Extended Connectivity FingerPrinting.

Supplementary Information

The online version contains supplementary material available at <https://doi.org/10.1186/s13321-022-00616-5>.

Additional file 1: True failed SMILES. SMILES failed because of incorrect chemistry.

Additional file 2: Figure S1. PIKACHU's recursive aromaticity detection. Aromatic cycles are individually and recursively detected and later joined into cyclic systems. **Figure S2.** Faulty ring detection by SmilesDrawer leads to unreadable structure renderings. SmilesDrawer's SSSR implementation fails to detect one of the macrocycles, leading to the unreadable structure shown on the right. PIKACHU's SSSR implementation (left) does recognise this ring, and therefore renders the structure correctly. **Figure S3.** PIKACHU resolves incorrectly drawn chiral bonds in rings. PIKACHU (left) correctly depicts cis-trans chemistry of stereobonds in rings that SmilesDrawer (right) cannot visualise. **Figure S4.** Two approaches for visualising macrocycles. A. Daptomycin visualised using the 'polygon' approach in PIKACHU. B. Daptomycin visualised using the 'honeycomb' approach in ChemDraw. **Figure S5.** tSNE plot of 36 calcium-dependent lipopeptides drawn in a 2D plane based on the Tanimoto distances between their structures as computed by PIKACHU. The script and structures used to draw this figure can be found in the `example_scripts` and `example_structures` folders in GitHub, respectively. **Figure S6.** Visualisation of the polyketide ketoreduction reaction, built and visualised by PIKACHU. The reduced group is highlighted in red. The script performing this reaction can be found on GitHub. **Supplementary Table S1.** Examples of SMILES that PIKACHU does not draw correctly, compared to ChemDraw drawings drawn from the same SMILES. **Table S2.** Drawing times of molecules rendered by RDKit and PIKACHU. Time is indicated in minutes.

Acknowledgements

Daniel Probst for providing an in-depth explanation of the SmilesDrawer software; Rutger Bosch for reporting software bugs; Zach Reitz, Joris Louwen, Lotte Pronk, David Meijer, Hannah Augustijn, Kumar Singh, Huali Xie, Jiayi Jing, Mohammad Alanjary, Chrats Melkonian, Justin van der Hooft, and Catarina Sales E Santos Loureiro for testing the PIKACHU software.

Author contributions

BRT: Development and testing of the PIKACHU software, wrote the initial version of the manuscript and further edited it based on the suggestions of the other authors. SPJMV: Significant contributions to writing the manuscript, extensive testing and debugging of the PIKACHU software, writing and executing example scripts. MHM: Significant contributions to writing the manuscript, suggestions for software features, research supervision. All the authors read and approved the final manuscript.

Funding

This work was supported by the Novel Antibacterial Compounds and Therapies Antagonising Resistance program (NACTAR) from the Dutch Research Council (NWO) [project number 16440].

Availability of data and materials

The PIKACHU software is made available under an open-source (MIT) license and can be found at <https://github.com/BTheDragonMaster/pikachu>. A wiki can be found at <https://github.com/BTheDragonMaster/pikachu/wiki>. Scripts used for the results section of this paper are made available at https://github.com/BTheDragonMaster/pikachu/tree/main/example_scripts. The NPAtlas database used in our analyses can be downloaded at <https://www.npatlas.org/download>.

Declarations

Competing interests

M.H.M. is a member of the Scientific Advisory Board of Hexagon Bio and co-founder of Design Pharmaceuticals.

Received: 8 January 2022 Accepted: 21 May 2022

Published online: 07 June 2022

References

- Alvarsson J, Lampa S, Schaal W, Andersson C, Wikberg JES, Spjuth O (2016) Large-scale ligand-based predictive modelling using support vector machines. *J Cheminform* 8(1):1–9. <https://doi.org/10.1186/s13321-016-0151-5>
- Beisken S, Meinel T, Wiswedel B, De FLF, Berthold M (2013) KNIME-CDK: Workflow-driven cheminformatics. *BMC Bioinform* 14(257):2–5
- Blin K, Shaw S, Kloosterman AM, Charlop-Powers Z, Van Wezel GP, Medema MH, Weber T (2021) AntiSMASH 6.0: improving cluster detection and comparison capabilities. *Nucleic Acids Res* 49(W1):W29–W35. <https://doi.org/10.1093/nar/gkab335>
- Cass S (2021) Top Programming Languages 2021. *IEEE Spectrum*. <https://spectrum.ieee.org/top-programming-language>. Accessed 7 Nov 2021.
- Edmonds J (1965) Paths, trees, and flowers. *Can J Math* 17:449–467
- Hastings J, Owen G, Dekker A, Ennis M, Kale N, Muthukrishnan V, Turner S, Swainston N, Mendes P, Steinbeck C (2016) ChEBI in 2016: improved services and an expanding collection of metabolites. *Nucleic Acids Res* 44(D1):D1214–D1219. <https://doi.org/10.1093/nar/gkv1031>
- Hückel E (1931) Quantentheoretische Beiträge zum Benzolproblem—I. Die Elektronenkonfiguration des Benzols und verwandter Verbindungen. *Zeitschrift für Phys* 70(3–4):204–286. <https://doi.org/10.1007/BF01339530>
- Johnson D (1975) Finding all the elementary cycles of a digraph. *SIAM J Comput* 4(1):77–84
- Kamada T, Kawai S (1989) An algorithm for drawing general undirected graphs. *Inf Process Lett* 31(1):7–15. [https://doi.org/10.1016/0020-0190\(89\)90102-6](https://doi.org/10.1016/0020-0190(89)90102-6)
- Kim S, Chen J, Cheng T, Gindulyte A, He J, He S, Li Q, Shoemaker BA, Thiessen PA, Yu B, Zaslavsky L, Zhang J, Bolton EE (2021) PubChem in

- 2021: New data content and improved web interfaces. *Nucleic Acids Res* 49(D1):D1388–D1395. <https://doi.org/10.1093/nar/gkaa971>
11. Landrum G RDKit: open-source cheminformatics. <http://www.rdkit.org>. Accessed 7 Nov 2021.
 12. Miles LH (2019) Cycle detection. <https://github.com/qpwo/python-simple-cycles>. Accessed 21 Aug 2021.
 13. Morris J, Jiao D (2016) ChemViz2: cheminformatics App for Cytoscape <http://www.rbvi.ucsf.edu/cytoscape/chemViz2/>. Accessed 7 Nov 2021.
 14. Probst D, Reymond JL (2018) SmilesDrawer: parsing and drawing SMILES-encoded molecular structures using client-side javascript. *J Chem Inf Model* 58(1):1–7. <https://doi.org/10.1021/acs.jcim.7b00425>
 15. Rogers D, Hahn M (2010) Extended-connectivity fingerprints. *J Chem Inf Model* 50:742–754
 16. Skinnider MA, Johnston CW, Gunabalasingam M, Merwin NJ, Kieliszek AM, MacLellan RJ, Li H, Ranieri MRM, Webster ALH, Cao MPT, Pfeifle A, Spencer N, To QH, Wallace DP, Dejong CA, Magarvey NA (2020) Comprehensive prediction of secondary metabolite structure and biological activity from microbial genome sequences. *Nat Commun* 11(1):1–9. <https://doi.org/10.1038/s41467-020-19986-1>
 17. Sorokina M, Merseburger P, Rajan K, Yirik MA, Steinbeck C (2021) COCO-NUT online: collection of open natural products database. *J Cheminform* 13(1):1–13. <https://doi.org/10.1186/s13321-020-00478-9>
 18. Stokes JM, Yang K, Swanson K, Jin W, Cubillos-Ruiz A, Donghia NM, MacNair CR, French S, Carfrae LA, Bloom-Ackerman Z, Tran VM, Chiappino-Pepe A, Badran AH, Andrews IW, Chory EJ, Church GM, Brown ED, Jaakkola TS, Barzilay R, Collins JJ (2020) A deep learning approach to antibiotic discovery. *Cell* 180(4):688–702.e13. <https://doi.org/10.1016/j.cell.2020.01.021>
 19. Ullmann JR (1976) An algorithm for subgraph isomorphism. *J ACM* 23(1):31–42. <https://doi.org/10.1145/321921.321925>
 20. Van Santen JA, Jacob G, Singh AL, Aniebo V, Balunas MJ, Bunsko D, Neto FC, Castaño-Espriu L, Chang C, Clark TN, Cleary Little JL, Delgadillo DA, Dorrestein PC, Duncan KR, Egan JM, Galey MM, Haeckl FPJ, Hua A, Hughes AH, Isakova D, Khadiilkar A, Lee JH, Lee S, Legrow N, Liu DY, Macho JM, McCaughey CS, Medema MH, Neupane RP, O'Donnell TJ, Paula JS, Sanchez LM, Shaikh AF, Soldatou S, Terlouw BR, Tran TA, Valentine M, Van Der Hoof JJJ, Vo DA, Wang M, Wilson D, Zink KE, Linington RG (2019) The natural products atlas: an open access knowledge base for microbial natural products discovery. *ACS Cent Sci* 5(11):1824–1833. <https://doi.org/10.1021/acscentsci.9b00806>
 21. Volkamer A, Kuhn D, Rippmann F, Rarey M (2012) Dogsitescorer: a web server for automatic binding site prediction, analysis and druggability assessment. *Bioinformatics* 28(15):2074–2075. <https://doi.org/10.1093/bioinformatics/bts310>
 22. Willighagen EL, Mayfield JW, Alvarsson J, Berg A, Carlsson L, Jeliazkova N, Kuhn S, Pluskal T, Rojas-Chertó M, Spjuth O, Torrance G, Evelo CT, Guha R, Steinbeck C (2017) The Chemistry Development Kit (CDK) v2.0: atom typing, depiction, molecular formulas, and substructure searching. *J Cheminform* 9(1):1–19. <https://doi.org/10.1186/s13321-017-0220-4>
 23. Yorkyer (2020) Python implementation of Edmonds' Blossom Algorithm. <https://github.com/yorkyer/edmonds-blossom>. Accessed 24 Aug 2021.

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

