JOURNAL OF APPLIED CRYSTALLOGRAPHY

ISSN 1600-5767

Received 7 May 2024
Accepted 19 June 2024

Edited by S. Moggach, The University of Western Australia, Australia

Keywords: software; drug discovery; drug development; materials discovery; application programming interfaces; Cambridge Structural Database; CSD Python API.

Supporting information: this article has supporting information at journals.iucr.org/j

What has scripting ever done for us? The CSD Python application programming interface (API)

Richard A. Sykes,[a] Natalie T. Johnson,[a] Christopher J. Kingsbury,[a] Jürgen Harter,[a] Andrew G. P. Maloney,[a] Isaac J. Sugden,[a] Suzanna C. Ward,[a] Ian J. Bruno,[a] Stewart A. Adcock,[a] Peter A. Wood,[a] Patrick McCabe,[a] Alexandru A. Moldovan,[a] Francis Atkinson,[a,b] Ilenia Giangreco[a,b] and Jason C. Cole[a]*

[a]Cambridge Crystallographic Data Centre, 12 Union Road, Cambridge CB2 1EZ, United Kingdom, and [b]Exscientia plc, The Schrödinger Building, Oxford Science Park, Oxford OX4 4GE, United Kingdom. *Correspondence e-mail: cole@ccdc.cam.ac.uk

Since its first release in 2016, the Cambridge Structural Database Python application programming interface (CSD Python API) has seen steady uptake within the community that the Cambridge Crystallographic Data Centre serves. This article reviews the history of scripting interfaces, demonstrating the need, and then briefly outlines the technical structure of the API. It describes the reach of the CSD Python API, provides a selected review of its impact and gives some illustrative examples of what scientists can do with it. The article concludes with speculation as to how such endeavours will evolve over the next decade.

1. Introduction

1.1. A little bit of history

Python script development has become a must-have tool in scientific research since the language was first conceived and developed by Guido van Rossum and co-workers in 1991. The language quickly gained acceptance in the community. It has achieved its underlying aim of democratizing software development so that programming is no longer the realm of specialists, as had been intended from the outset of the language (*Computer Programming for Everybody*, https://www.python.org/doc/essays/cp4e/). Python leads coding popularity contests (see, for example, the *IEEE Spectrum Top Programming Languages 2023*, https://spectrum.ieee.org/the-top-programming-languages-2023) due to its rich array of modules and ease of use. Research training now tends to focus on Python when educating researchers in the skills deemed necessary in a modern research environment (https://software-carpentry.org/about/).

The Cambridge Crystallographic Data Centre (CCDC, https://www.ccdc.cam.ac.uk/) has long been a user of the Python language. Indeed, our desktop search software *ConQuest* (Bruno *et al.*, 2002) was built using Python 1.5 code (latterly migrated to Python 2.7) on top of a pre-existing Fortran application programming interface (API) for data access; we suspect that this made *ConQuest* one of the earliest examples of a fully production-ready piece of scientific software built with Python at its core.

The underlying access to the data in that work allowed developers internally at the CCDC to code simple interfaces to complex search tools, and to write a relatively intuitive interface that was considerably more user friendly than the tools that were currently available. Underneath, the provision

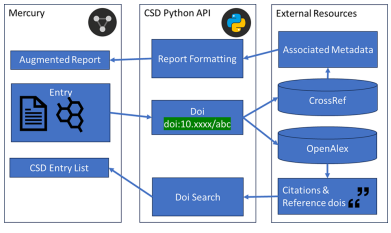


Table 1
Modules in the CSD Python API.

Core modules	
`ccdc.io`	File and database reading and writing
`ccdc.entry`	An entry in the CSD (or possibly an alternative data source)
`ccdc.crystal`	Crystal structure object representation
`ccdc.molecule`	Molecule object representation
`ccdc.diagram`	Chemical diagram representation and manipulation functionality
`ccdc.search`	Programmatic access to chemical and crystallographic searching in the CSD
`ccdc.interaction`	Analysis of interaction information in the CSD (Bruno *et al.*, 1997; Verdonk *et al.*, 1999; Wood *et al.*, 2013)
`ccdc.conformer`	Analysis and generation of conformations (Bruno *et al.*, 2004; Cole *et al.*, 2016, 2018)
`ccdc.descriptors`	Chemical descriptors and calculations
`ccdc.utilities`	General utilities, *e.g.* logging
Pharmaceutical discovery modules	
`ccdc.docking`	Objects to manipulate and control *GOLD* docking analyses (Jones *et al.*, 1997; Verdonk *et al.*, 2003; Cole *et al.*, 2005)
`ccdc.cavity`	Protein cavity searching and characterization
`ccdc.protein`	Protein molecular representation
`ccdc.pharmacophore`	Pharmacophore searching of the CSD and PDB through *CSD-CrossMiner* (Korb *et al.*, 2016)
`ccdc.screening`	Ligand-based and field-based virtual screening (Giangreco *et al.*, 2021)
Crystal form analysis modules	
`ccdc.particle`	Crystal particle analysis (Bryant *et al.*, 2018; Moldovan & Maloney, 2024)
`ccdc.morphology`	Crystal morphology analysis (Clydesdale *et al.*, 1991)
`ccdc.prediction`	Descriptors for predictive models relating to crystal structures

of a very simple direct data API was found to be extremely useful, and many internal research projects were made possible by research scientists without intricate knowledge of a specialized compiled language such as C or Fortran. We came to the realization that such access needed to be provided for all researchers wishing to use the information in the Cambridge Structural Database (CSD) (Groom *et al.*, 2016; https://www.ccdc.cam.ac.uk/solutions/software/csd/) and the advanced software methods we could provide that could make use of said data. Consequently, the idea of providing publicly accessible APIs was born and in 2016 the CCDC released Version 1.0.0 of the CSD Python API.

We were not alone in recognizing the benefits of simple programmatic access to software and services. In chemistry, software vendors and academic providers have long provided script-driven solutions which represent an evolution over conventional command line interfaces. Early incarnations of scripting languages included TRIPOS's bespoke SYBYL's Programming Language (SPL) and the Chemical Computing Group's Scientific Vector Language (SVL). Software visualizers have also provided scripting access (for example, the visualizer *RasMol* had a script interface in a bespoke language when it was initially released in 1992). Another example, *JMol*, and subsequently *JSMol*, used a scripting interface that could be embedded in Javascript controls; *AstexViewer* similarly provided its own bespoke scripting language, but could also be controlled directly through Java applets.

Other software vendors have released chemical software with Python APIs (in particular, OpenEye's *OEChem*, *ChemBioOffice* from PerkinElmer and *PyMol* from Schrodinger). Indeed, one could argue that having a simple API that can at least be accessed through Python is now almost a prerequisite for adoption in chemical and biological software. Many useful Python modules and toolkits also exist from the open-source community that facilitate mathematics, data sciences, biology and chemistry (for example chemistry- and biology-driven toolkits such as *RDKit*, *CDK*, *CACTVS*, *OpenBabel* and *BioPython*, and more general science- and mathematics-related tookits such as *NumPy*, *SciPy*, *matplotlib*, *scikit-learn*, *Pandas* and *BioPython*).

The CSD Python API also acts in the role of a data API. Data APIs are now common in the biological sciences. The Worldwide PDB (wwPDB, https://www.wwpdb.org/) provides data access through REST-based APIs. Similarly, APIs can be accessed for PubMed (https://pubmed.ncbi.nlm.nih.gov/), PubChem (https://pubchem.ncbi.nlm.nih.gov/) and other scientific data resources.

In this article, we present a short overview of what is exposed in the CSD Python API, a review of the impact of the CSD Python API in various chemical fields, some examples of its use, and finally speculation as to what the future may hold in data-driven sciences using crystallographic and chemical information.

2. The CSD Python API

2.1. Modules

The API contains modules that cover the broad range of functionality provided by the CCDC, including core search and analysis of the CSD, access to meta-databases such as *Mogul* (Bruno *et al.*, 2004) and *IsoStar* (Bruno *et al.*, 1997), and CCDC software tools for docking (Jones *et al.*, 1997; Verdonk *et al.*, 2003), conformer generation (Cole *et al.*, 2018) and solid-form assessment and particle analysis (Bryant *et al.*, 2018). Most of the software and services that the CCDC provides can be accessed through a CSD Python API module. The modules available, with a brief description of their purposes, are presented in Table 1. More information on the exact nature of the modules in the API can be found in the user documentation.

2.2. Underlying design

Each module is a wrapper around a larger volume of CCDC software written in C++. We use the *SWIG* interfacing library (Beazley *et al.*, 2022) to interact between C++ code and Python code. These libraries are privately imported and then convenience wrapper classes are written that expose the underlying C++ code in a more Pythonic interface than is achievable directly from a *SWIG* layer.

The choice of *SWIG* would in principle allow other scripting implementations in other languages, such as Rust, Julia, Perl or PHP, should the need arise, as *SWIG* is designed to be agnostic to the scripting language used.

2.3. Extensions for integration

The CSD Python API further includes additional packages and classes to enable integration with CCDC desktop applications and third-party applications. For example, the CCDC provides an application interface class that allows integrations into our graphical user interfaces.

The *ccdc_knime* package provides a separate installable set of tools that facilitate implementations of *KNIME* chemical workflow nodes (Berthold *et al.*, 2008), showing an example of how the CSD Python API has been used to provide a convenient integration. The CCDC also supports similar resources to integrate services into the BIOVIA package *Pipeline Pilot*.

2.4. Open-source examples repository

Alongside the CSD Python API, the CCDC provides an open-source repository (https://github.com/ccdc-opensource) for deposition of scripts (A. Moldovan and E. Myers, https://github.com/ccdc-opensource/csd-python-api-scripts). The examples cover a broad range of functionalities and demonstrate how to use the API in project work, or are projects that have been published. We have included the examples in this paper in that repository (see https://github.com/ccdc-opensource/csd-python-api-scripts/tree/main/api_paper_2024).

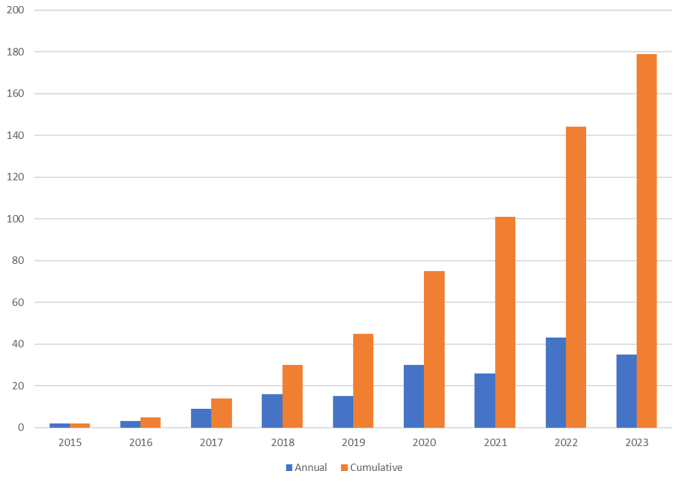


Figure 1
The growth of Google Scholar references to the text string 'CSD Python API' by year.

3. Examples of using CCDC data through the API

The CSD Python API has been used extensively in publications since its first official release in 2016. A Google Scholar search for the exact phrase 'CSD Python API' yielded 180 results at the time of writing. This is probably a significant underestimate of the number of uses of the API as not all use cases will be published and not all will directly refer to the API textually in the title or abstract, instead referring to the complete CSD system. The growth in use, however, can still be gleaned from the results in Fig. 1

Here we present a brief review of some example cases where access to such an API has facilitated new science and methods that would have been challenging previously.

3.1. Generation of useful subsets

Two larger-scale subsets have been developed for general use using the CSD Python API. Researchers at the University of Cambridge have collaborated with the CCDC to develop a CSD metal–organic framework (MOF) collection of programmatically ready representations of MOFs along with subsets of 1D, 2D and 3D MOF structures in the CSD (Moghadam *et al.*, 2017; Li *et al.*, 2020, 2022, 2023); these MOF subsets have then been widely searched and analysed in subsequent papers (with over 600 citations since 2017) for the purposes of materials screening and materials prediction (Sarkisov *et al.*, 2020; Glasby *et al.*, 2023).

Another useful subset is the CSD Drug Subset (Bryant *et al.*, 2019), which contains structures of approved drugs within the CSD [derived using the DrugBank (https://www.drugbank.com/) approved list]. Both these efforts were enabled by programmatic access to the data; they are now kept up to date by automated scripted processes that use the CSD Python API.

A computational API to data has less restriction than a conventional graphical user interface (GUI) in that the end user can craft their own search criteria to mine the underlying databases and intertwine other computational methods for further assessment of the available data. There are several cases where the CSD Python API has been used to mine the CSD and reduce the collection to a smaller subset of entries with specific features and properties that can then be subjected to more detailed analysis. For example, researchers were able to hunt for kryptoracemates (Clevers & Coquerel, 2020), to survey chiral compounds and their associated molecules (Rekis, 2020), and to survey high-pressure crystal structures (Kaźmierczak & Patyk-Kaźmierczak, 2021).

3.2. Large-scale surveys, subset property annotation and screening

Easy Pythonic access to the CSD allows end users to undertake larger-scale surveys of structural data. Several examples have appeared in the literature in recent years, including a validation of the temperature data in the CSD (Johnston *et al.*, 2018), an analysis of displacement parameters in crystal structures (Bond, 2021), a review of solvates in the CSD (Werner & Swift, 2021) and an analysis of thermal

expansion properties of organic molecules (Lee & Dumitrescu, 2021).

Access to data via the CSD Python API has enabled annotation of CSD entries with derivative properties from calculations. Several efforts have specifically annotated larger data sets with calculated properties. The *OCELOT* database (Ai *et al.*, 2021) contains descriptors for theoretical and experimental structures that characterize electronic and optical properties. Such data are useful for building predictive models for crystalline organic semiconductors. The experimental structures used for the calculations were a subset of structures from the CSD. Another group calculated quantum mechanical geometries for more than 86 000 transition metal structures in the CSD (Balcells & Skjelstad, 2020). In another effort, high carrier mobility descriptors have been calculated for more than 96 000 structures in the CSD (Schober *et al.*, 2016); the data were subsequently used for model creation and prediction and the process can be seen as a method of virtual screening for materials. Workers have also screened the CSD to find singlet fission molecules (Padula *et al.*, 2019).

High numbers of MOF structures from the CSD have been characterized using quantum mechanical methods for training faster machine learning models. These models can, in turn, be used for virtual screening of potential new MOFs (Rosen *et al.*, 2020). Similarly, MOF structures have been annotated with property predictions for understanding the influence of topology type on absorption properties (Moghadam *et al.*, 2020). These public-domain initiatives enhance the value of structural data by providing useful derivative information for further experimentation in the academic community.

Seeking novel catalyst designs is another potential use of the CSD which programmatic access makes more tractable. Short *et al.* (2023) have developed a high-throughput screening methodology for potential catalyst ligand detection using a catalophore approach.

Finally, we note that access to database searching through a scripting interface allows large-scale searching for many queries. In work by Giangreco *et al.* (2017), the CSD was searched for many matched molecular pairs of crystal structures. These pairs were used to populate a meta-database for lookup of related pairs of crystal structures, which are in turn interesting for informing on crystal structure design. Such a study would be nigh impossible via just a GUI-based tool due to the number of searches required.

3.3. Focused predictive model building

Smaller subsets of structures identified using the CSD Python API can be used for building predictive models. For example, co-crystals have been analysed and predicted using machine learning models integrated to the CSD via the API (Vriza *et al.*, 2021, 2022). Similar work has been undertaken for solvate prediction (Xin *et al.*, 2019) and crystallizability prediction (Wicker & Cooper, 2015; Frade *et al.*, 2020). Oxidation-state prediction has also been facilitated by data access (Reeves *et al.*, 2019). The Kulik group are working towards the classification and discovery of hemilabile ligands in the CSD using the CSD Python API (Kevlishvili *et al.*, 2023). Another recent endeavour has been the building of models via transfer learning for the prediction of common properties of interest (toxicity, odour and synthetic reaction yield) from 2D starting points, but incorporating inferred 3D information; this effort used organic molecules taken from the CSD as a useful training base for the model (King-Smith, 2024). It is intriguing that using 3D crystallographic information in this way can enhance the predictivity of models in seemingly far-removed areas of chemistry, perhaps demonstrating the importance of 3D structure in determining these properties and the need for such information to be represented in quantitative structure–property relationship (QSPR) models more generally.

3.4. Natural language processing

Another interesting path facilitated by API access is the availability of the associated publication digital object identifier (DOI). This means that, with other tools such as *ChemDataExtractor* (Swain & Cole, 2016), it is possible to cross-mine the primary literature associated with a given crystal structure.

For example, Nandy and co-workers took advantage of this link from CSD structures to their primary articles. They used paper scraping methods and natural language processing methods to extract stability data (Nandy *et al.*, 2022). This has then helped to facilitate computational discovery of stable MOFs for methane to methanol catalysis in a later publication (Adamji *et al.*, 2023).

Glasby *et al.* (2023) used similar methods to text mine synthesis method, solvent, organic linker, metal precursor and topology information for associated MOF structures. This has allowed the building of a model to estimate various properties of a new MOF, including its potential cost.

Methods for exploring functional materials have also proved interesting. Seyedraoufi and co-workers (Seyedraoufi *et al.*, 2022, 2023; Dypvik Sødahl *et al.*, 2023) are developing methods to generate their own subset of entries in the CSD that might be organic proton-transfer ferroelectrics. The structures found were further analysed using density functional theory to explore potential ferroelectric properties in the resultant data set.

The CSD Python API's link to the primary literature is also helping to facilitate research into conglomerate crystallization. Walsh *et al.* (2022) have text mined (and then manually inspected) a data set of crystal structures that arose due to crystallization of conglomerates from a racemic mixture. Such structures are of significant interest in synthetic routes, as they can provide a pathway that introduces necessary chiral centres spontaneously without the need for specific chiral catalysts.

Finally, in a rather lighter example, Willett *et al.* (2020) have used the access to the DOIs in entries via the API to facilitate access to publication titles through CrossRef (Crossref Metadata Search, https://search.crossref.org/), which in turn allowed analysis of crystal structure related science trends in subjects broken down by year.

4. Examples of using CCDC software through the API

Another benefit of providing the API is that it allows end users to employ the software to generate more customized outputs specific to their area of research and to integrate the scientific software that the CCDC provides into automated workflows.

4.1. Direct use of software

The interaction module has facilitated the creation of a tool for exploring fragment hotspots within proteins (Radoux *et al.*, 2016; Curran *et al.*, 2020). This has in turn been used, via the provided integration, to create software that incorporates target-specific pharmacophoric information into deep generative models for fragment elaboration (Hadfield *et al.*, 2022); such software can aid with fragment-based drug design. Hadfield and co-workers have developed the *STRIFE* program, which out-performs other methods for fragment enumeration by accounting for the local model information that the fragment hotspots can provide.

Analysis of molecular interactions has been facilitated via the API: Kuhn and co-workers have developed powerful models for understanding the likelihood of a given interaction geometry in a protein–ligand binding site (Kuhn *et al.*, 2019; Tosstorff *et al.*, 2020). This work has led to quantitative structure–activity relationship (QSAR) models that can be used to aid structure-based design (Tosstorff *et al.*, 2022).

Crystallographic endeavours have also been aided by access to the CSD Python API. Wright *et al.* (2020) have used the API to understand more deeply the nature of conformational change between different polymorphic structures. Creation of models of polymorphic solid solutions has been aided by access to editable structures and structure editing tools provided in the CSD Python API (Hill *et al.*, 2023). High-pressure studies, too, have been aided by access to the CSD Python API. A tool built on the API has been developed to understand void volumes and packing coefficients as a function of pressure (Wilson *et al.*, 2022); this in turn has facilitated other research to understand the impact of pressure on hydrogen bonds in crystal co-formers (Ward *et al.*, 2023).

In the materials space, by providing access to morphology calculations, the CSD Python API has facilitated high-throughput screening for thin-film organic heteroepitaxy in entries within the CSD (Dull *et al.*, 2023). In principle, such an approach could be extended beyond CSD structures to hypothetical structures in future work. Other focused studies of morphology, including a study of how polymorph growth can be directed using Au(111) surfaces, has been aided by descriptor calculations available through the CSD Python API (Ma *et al.*, 2023).

4.2. Integrations and workflows

One common use case is to allow both academic and industrial software providers to integrate use of CCDC software and services in specific workflows. For example, the RCSB (a member organization of the wwPDB) provides a suite of scripts that they use to aid in their curation efforts (`py-rcsb_utils_ccdc`, https://github.com/rcsb/py-rcsb_utils_ccdc). They use these scripts to integrate the CSD Python API into workflows that make use of the CSD in the global deposition processes used by the wwPDB. The scripts use the CCDC's *Mogul* software through the API to enhance structural assessment of bound ligands.

Providers of protein refinement software Global Phasing also take advantage of the CSD Python API to integrate the CCDC's conformational analysis tool *Mogul* into their workflows in the *Grade2* component of their *BUSTER* workflow (Smart *et al.*, 2021). *Mogul* is used to generate ligand dictionaries for use in structural refinement.

The docking program *GOLD* (Jones *et al.*, 1997) has also been integrated into workflows using the API. For example, a 'reverse docking' workflow has been developed (Ruiz-Moreno *et al.*, 2021), and the availability of *KNIME* nodes has facilitated the creation of a workflow for covalent docking (David *et al.*, 2022). The API is in addition used in the *DockStream* workflow (Guo *et al.*, 2021).

In another example, a workflow for writing a docking protocol to target GABA-A receptors has been developed (Fabjan *et al.*, 2021). One benefit of using a script for such endeavours is that it naturally leads to a degree of repeatability if the workflow code is published along with the input configuration files.

Integrations are not limited to macromolecular chemistry and discovery sciences. In the synchrotron X-ray data management platform *DawnScience* (Filik *et al.*, 2017), some integrations have been created that allow searching for structures with specific unit cells in the CSD from within the *DawnScience* platform.

The field of crystal structure prediction (CSP) also benefits from API access. Iuzzolino *et al.* (2017) have used the conformer generator built into the API programmatically to enhance sampling in crystal structure prediction. In another CSP-related report, access to rugosity calculations has led to suggestions of paths for identifying accessible but so far unobserved crystal structures (Montis *et al.*, 2022).

The `ApplicationInterface` class in the CSD Python API allows users to develop scripts that integrate directly with the CCDC's desktop GUI programs *Mercury* and *Hermes*. The class supports both receiving structural information sent from the desktop software and the ingesting of information from external environments to send for visualization. This has facilitated the extension of the CCDC's *Mercury* interface (Macrae *et al.*, 2020) to create bespoke analyses by third parties. For example, in the *MrPIXEL* software, the program *PIXEL* has been integrated into *Mercury* for easy use (Reeves *et al.*, 2020). The CCDC also makes extensive use of this approach to provide prototype analysis scripts to collaborative partners.

5. Illustrative examples of using the CSD Python API

To showcase the CSD Python API further, here we provide some short descriptions of new scripts that could be useful in research. We have tried to pick examples from across the

portfolio, namely a script to show how searching can be more powerful in the API, a script that shows how one can cross-mine third-party resources to garner insights into a specific CSD entry and report said insights within the *Mercury* software using HTML reports, an example from the discovery field that allows a user to dock molecules selected on the basis of similarity to other previously high-scoring molecules, an example that shows how one can generate powerful visualizations of metal–ligand complexes using Voronoi surfaces, and finally an example that demonstrates a consistent approach for describing predicted particle shapes, a method that is of potential use in particle formulation studies.

5.1. Extending substructure searching using the CSD Python API

5.1.1. Motivation. The CSD Python API provides an excellent means for extending searches to allow more sophisticated queries that are not supported within the standard CCDC desktop and online software. One common problem that can occur is that an end user will wish to find a specific class of compound with a given scaffold and a subset of specific substitutions. The standard substructure searching can find the scaffold easily enough, but filtering the data down to a specific set of substitutions is very challenging, and repeating the search each time a new data release is created can be a laborious process.

For example, recently we were asked by a user how to find all *tri-substituted* isoflavone molecules in the CSD where the substituent was one of a small set of common functionalities. The user wished to carry out a gap analysis of common biologically relevant tri-substituted isoflavones to see if there were any compounds which would be worth studying via crystallographic methods. We can develop a script to answer this question, at least for structures where 3D coordinates are available.

5.1.2. The extended query. Isoflavones can have oxygen substitutions at many locations around the isoflavone ring system (Fig. 2 outlines isoflavone nomenclature) and different substitutions are relevant in different biological molecules. To find all possibilities where exactly three groups have an oxygen substituent is a challenge; *ConQuest* can support 'variable points of attachment' in searching, which takes a user towards the answer, but then limiting the search to only those hit molecules that are exactly tri-substituted is not currently possible without visual inspection, and furthermore, it is not possible to search for substituents that are one of a subset of possibilities (namely O, OH, OMe or OPh but not, say, OSi).

In the past, this type of query would have provided gainful employment for a PhD student for many an hour, since realistically said student would be forced to perform several different queries, find many potential hits containing the ring scaffold and at least three hydroxyl groups, and manually filter out false positives, a process that could be time consuming, very error prone and hard to repeat on future database versions.

The problem can now be resolved by searching the CSD using a SMARTS query through the CSD Python API and then writing post-processing code to inspect molecules further as required. The CSD Python API supports a large proportion of the SMARTS query language (Weininger, 1997). Using this language, it is possible to express a complex isoflavone query which is recursive in nature.

5.1.3. The script. The script for this example is available at https://github.com/ccdc-opensource/csd-python-api-scripts/tree/main/api_paper_2024/example_1.

The script for performing the search is very simple. More details of the query used can be found in the supporting information (Section S1), but we highlight a few key elements here. Substructure searching using a SMARTS string in the CSD can be performed in a few lines of code, as illustrated in Fig. 3, which shows a simple search of the CSD for a pyridine molecule.

The CSD Python API also provides a simple means to extend searching using the `HitProcessor` class. This extension allows a user to write a customized search object that post-processes hits with additional conditions. This mechanism is used to post-filter hits in the search for tri-substituted isoflavones. A simple example of a `HitProcessor` class is given in the CCDC's CSD Python API documentation.

5.1.4. Results. The search yields 101 hits. These hits represent all isoflavones in the CSD substituted with one of O, OH, OCH_3 or OPh at one of the ten possible substitution points. Using simple post-processing within the script to remove entries with more than three non-hydrogen substituents in total, the number of hits can be reduced to 24 examples in the CSD. The predominant form of tri-substituted isoflavone in the CSD is 4′,5,7. This corresponds to the naturally occurring isoflavone genistein. Many of the hit structures are genistein, genistein co-formers or genistein salts.

One additional advantage of using a script is that we can also re-direct it to other data sources. PubChem (Kim *et al.*, 2023) was searched for entries containing the isoflavone core using the PubChem online search service (https://pubchem.ncbi.nlm.nih.gov/). This yielded an SD file containing 30 170

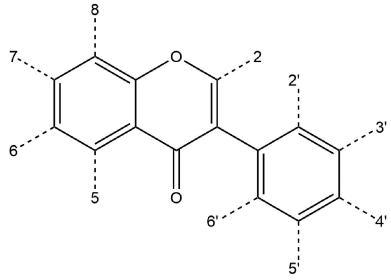


Figure 2
Isoflavone nomenclature.

```
searcher = SubstructureSearch()
searcher.add_substructure("c1ncccc1")
hits = searcher.search()
```

Figure 3
Simple substructure searching for pyridine.

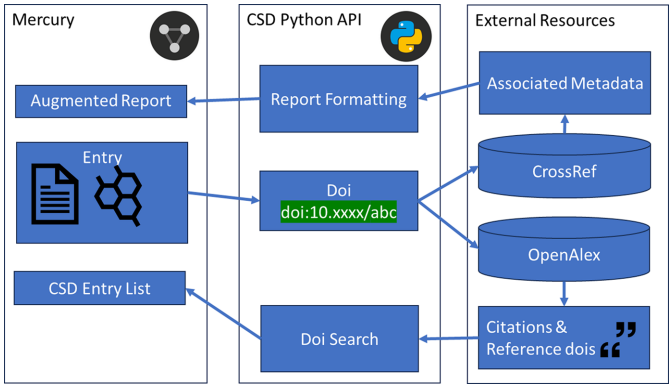


Figure 4
A visual summary of a DOI-driven workflow to find related entries by publication citation.

hits. After converting the bond types of the entries in the file to CCDC conventions, this subset of PubChem entries could be searched using the same CSD Python API script. In total, 7833 matched the SMARTS query, reducing to 428 hits that, after post-processing, were shown to be tri-substituted isoflavones.

The distributions of hits from the two respective databases are included in the supporting information (Table S1). Far more isoflavones occur in PubChem, with 29 different substitution patterns represented (of the 70 theoretically possible) in contrast to just five in the CSD. The most common missing pattern in the CSD is the 3′,4′,7 substitution pattern which occurs 22 times in PubChem. There are noticeable absences from both databases: there seems to be no example where the isoflavone has been substituted with one of the

```python
def get_entries_for_doi(doi):
    """
    Search the Cambridge Structural Database for any entries associated with a single DOI
    :param doi: The document object identifier to search for
    """
    start = time.time()
    if doi is None:
        return []

    searcher = TextNumericSearch()
    searcher.add_doi(doi)
    x = searcher.search()
    logger.debug(f"Search time for doi {doi}: {time.time() - start}")
    return x


def get_entries_for_dois(dois):
    """
    Search the Cambridge Structural Database for any entries associated with a list of DOIs
    :param dois: The document object identifiers to search for
    """
    start = time.time()
    if len(dois) == 0:
        return []

    if len(dois) == 1:
        return get_entries_for_doi(dois[0])

    # As we are searching for more than one DOI so let's build a combination search 'ORing' each
    # case
    combo = TextNumericSearch()
    combo.add_doi(dois[0])

    for d in dois[1:]:
        ts = TextNumericSearch()
        ts.add_doi(d)
        combo |= ts

    # We can then run the search using the combined search method
    searcher = CombinedSearch(combo)
    x = searcher.search()
    logger.debug(f"Search time for multiple dois : {time.time() - start}")
    return x
```

Figure 5
A code snippet illustrating the use of combination searching.

desired substituents at the 2-position, and there are no examples of a 2′,6′,*X*-substituted pattern; one can speculate that these species may be chemically unstable or synthetically challenging.

5.2. Mining third-party resources for structural context

5.2.1. Motivation. Conventional searching of the CSD uses various methods such as substructure or similarity searching, text and compound name searching, author searching, and crystallographic feature searching. These methods are highly useful but do not always find all relationships between structures. One obvious link between one structure and another is field of practice: we would expect citations from papers containing one structure to contain other structures that are linked by a common theme.

The CSD Python API, combined with other Python and REST APIs, can be very powerful for mining for such relationships. Each CCDC refcode has associated publication information and many have an associated publication DOI.

5.2.2. The script. The script for this example is available at https://github.com/ccdc-opensource/csd-python-api-scripts/tree/main/api_paper_2024/example_2.

In this script, we developed a simple example and the workflow is summarized in Fig. 4.

First, a DOI is retrieved from a CSD entry. This is then used to search CrossRef through the Python module *habanero* (https://habanero.readthedocs.io/en/latest/) and OpenAlex (Priem *et al.*, 2022) through Python code that calls the OpenAlex web API. The CrossRef record allows tabulation of title, first author and other information (abstract, funding information, first author institution and Scopus journal subject areas if available). We can then look up referenced and cited work in OpenAlex. The DOIs of these other reports can then be searched for in the CSD using text searching to retrieve entries published in these associated papers. Such a cyclical through-reference approach leads to links between entries that would not always be obvious from conventional searching.

The `ApplicationInterface` class is then used to tabulate the outputs so they can be viewed in *Mercury* as an HTML report and a spreadsheet of structures in the CSD.

Most of the script and functions within it are not related to the CSD Python API, as they tackle searching and retrieval of information from CrossRef and OpenAlex, but two elements are worth highlighting.

Firstly, we look at the use of the search classes for text and numeric searching and for combination searching (Fig. 5). The code creates an arbitrary set of sub-queries using the `TextNumericQuery` class. These can be combined with Boolean logic (in this case 'or') and then be searched using a `CombinedSearch` object.

The second highlight (Fig. 6) is the use of the class `ApplicationInterface`. This class can be used for formatting and creation of tabulated HTML reports from analysed data which are then presented in *Mercury*. In addition, a

```
with interface.html_report(
     title='Citation and Reference Information for Paper DOI associated with %s' % doi) as report:

     # Write the section header for Entry Details
     report.write_section_header('Crossref Record')
     # Assemble a dictionary of information and labels to go into the "Entry Details" table
     # Generate a HTML table from the entry details and write it to the report

     data = tabulate_crossref_record(doi, logger, email_address=email_address)

     identifiers = linked_csd_entries(doi, identifier)
     if len(identifiers) > 0:
         data.append(["<b>Other entries from the same paper</b>", '\n'.join(identifiers)])
     else:
         data.append(["<b>No other entries from the same paper</b>"])
     try:
         citation_data = search_citation_data(doi, identifier, \
                                              email_address=email_address, datasource=source)

     except ReferencesAndCitationsError as e:
         citation_data = None
         data.append(["<b>Citation lookup failed</b>", str(e)])

     if citation_data:
          data.append(["<b>Number of cited references found</b>", str(len(citation_data[1]))])
          data.append(["<b>Number of papers that cite this paper</b>", str(len(citation_data[0]))])

     report.write(html_table(data=data, table_id='crossref_details'))
```

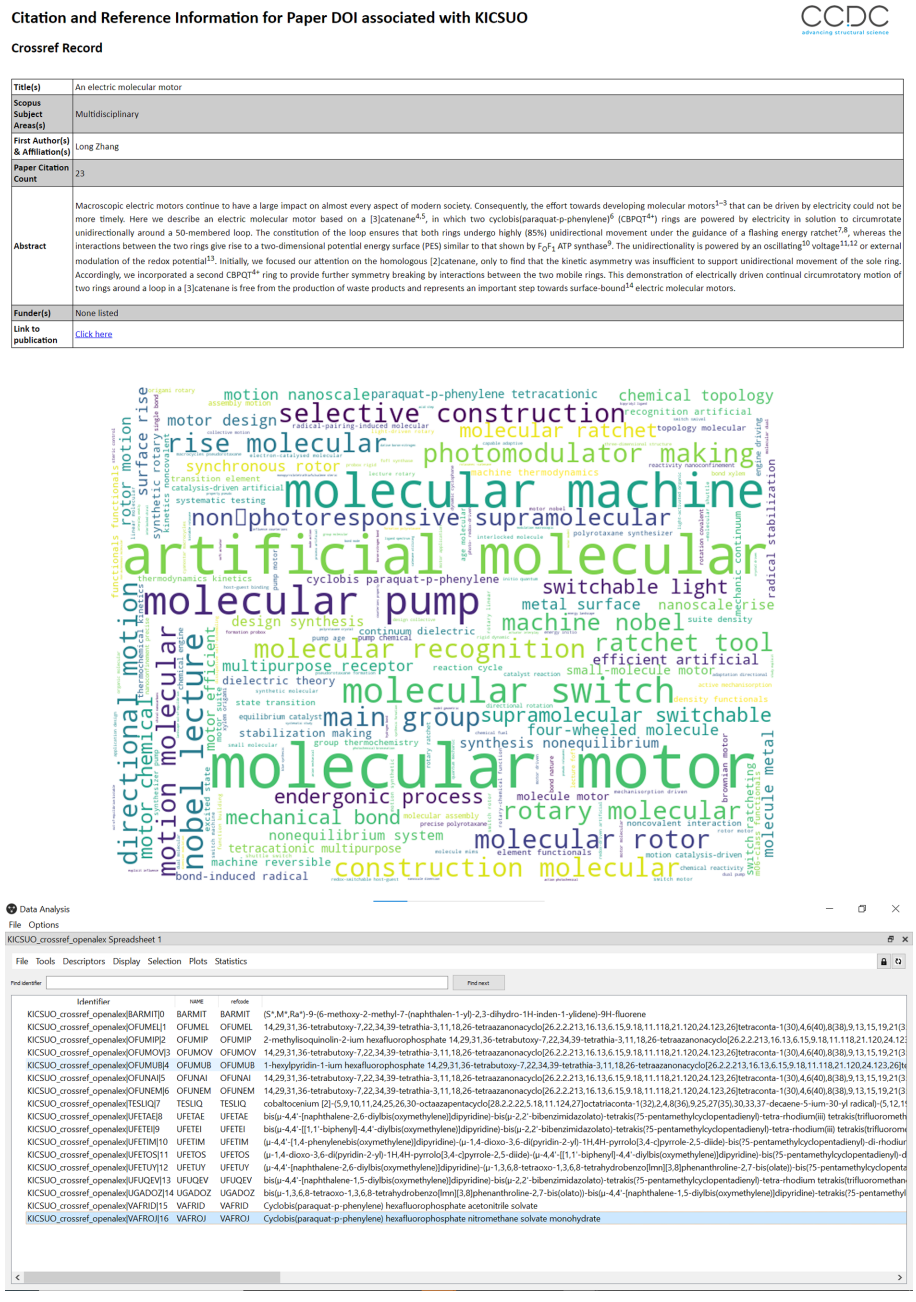
Figure 6
A code snippet illustrating the use of the application interface class and associated HTML table formatting.

tabulation in a TAB-separated value file can be sent back directly to the *Mercury* GUI, thus allowing a user to have a browsable set of CSD entries with spreadsheet values.

5.2.3. Results: an example report. The report was run for the CSD entry KICSUO (Zhang *et al.*, 2023). The structure is taken from a paper in which the authors are developing an electric molecular motor.

Running the code from within *Mercury* yields an HTML report containing a list of entry data presented in a spreadsheet, tabulated information relating to the entry that is extracted from CrossRef featuring the paper title, abstract, Scopus keywords, first author and funder information, and a word cloud of bigrams taken from the text of all the titles of literature available in OpenAlex that either is referenced within the parent publication or cites the parent publication (Fig. 7).

The word cloud is a useful simple visualization of the themes associated with the crystal structure; in this case, the word cloud highlights many bigrams relating to types of molecular machinery development. While such a link is apparent from the original article's title, such relationships to other CSD entries would be hard to find using conventional database searching. The listed entries returned from the search also yield interesting cross-relationships.



Citation and Reference Information for Paper DOI associated with KICSUO

CCDC advancing structural science

Crossref Record

Title(s)	An electric molecular motor
Scopus Subject Areas(s)	Multidisciplinary
First Author(s) & Affiliation(s)	Long Zhang
Paper Citation Count	23
Abstract	Macroscopic electric motors continue to have a large impact on almost every aspect of modern society. Consequently, the effort towards developing molecular motors[1–3] that can be driven by electricity could not be more timely. Here we describe an electric molecular motor based on a [3]catenane[4,5], in which two cyclobis(paraquat-p-phenylene)[6] ($CBPQT^{4+}$) rings are powered by electricity in solution to circumrotate unidirectionally around a 50-membered loop. The constitution of the loop ensures that both rings undergo highly (85%) unidirectional movement under the guidance of a flashing energy ratchet[7,8], whereas the interactions between the two rings give rise to a two-dimensional potential energy surface (PES) similar to that shown by F_oF_1 ATP synthase[9]. The unidirectionality is powered by an oscillating[10] voltage[11,12] or external modulation of the redox potential[13]. Initially, we focused our attention on the homologous [2]catenane, only to find that the kinetic asymmetry was insufficient to support unidirectional movement of the sole ring. Accordingly, we incorporated a second $CBPQT^{4+}$ ring to provide further symmetry breaking by interactions between the two mobile rings. This demonstration of electrically driven continual circumrotatory motion of two rings around a loop in a [3]catenane is free from the production of waste products and represents an important step towards surface-bound[14] electric molecular motors.
Funder(s)	None listed
Link to publication	Click here

Figure 7
Script outputs associated with the original structural publication. (*a*) The CrossRef report. (*b*) A word cloud of bigrams generated from citing and referenced articles. (*c*) The browsable spreadsheet of associated CSD entries in *Mercury*.

In principle, the same script can be used for finding entries and citation information relating to any article DOI, even if the article is not associated with crystal structures in the CSD; this can be useful for identifying relevant associated crystallographic information from related papers very easily.

5.3. Similarity-driven docking

5.3.1. Motivation. A key benefit of the CSD Python API is that it allows users to write scripts that can run multiple methods in a single workflow. For example, it is possible to run the CCDC's docking program in a more embedded fashion, using the interactive docking mode provided in the CSD Python API. In this mode, the API creates an instance of *GOLD* in memory that can receive ligands directly from the script as a stream of data, rather than having the docking software read them from a file.

One example that is provided in the CSD suite is similarity-driven docking. In this workflow, we first run a similarity search in the CSD; any hits are then docked into a pre-prepared protein target, and the best scoring docking poses are retained. If any docking pose has a score within 10% of the best scores observed so far, it is used as a starting point for a subsequent cycle of similarity searching and docking. Here, we have added a derived version of this script that, alternatively, can search in the ChEMBL database (Mendez *et al.*, 2019) as a source of information. This gives an exemplar of how the *GOLD* docking program can be embedded into a more complex workflow. The workflow is summarized in Fig. 8.

5.3.2. The script. The script for this example is available at https://github.com/ccdc-opensource/csd-python-api-scripts/tree/main/api_paper_2024/example_3.

A key feature used in this script is *GOLD* in interactive docking mode; this allows a user to set up a *GOLD* daemon object on a socket and send individual molecules to the socket for docking. This avoids repeated initialization when used to receive ligands from external data sources. The ChEMBL Python API (Davies *et al.*, 2015) is used to facilitate similarity searching in the ChEMBL database.

5.3.3. Results. The script is meant as an example which could be used to guide development of more sophisticated approaches, but to showcase it, the PDB entry 1fm9 (Gampe *et al.*, 2000) was configured for using default settings in *GOLD*. The full configuration is included as an example in the open-source repository. Entry 1fm9 is an example of a PPAR-γ/RXR-α crystal structure, a heterodimeric nuclear receptor that is a known target for anti-diabetic drugs. There are many known inhibitors that have indications against this target in ChEMBL. We were interested to see if the similarity script could simulate a drug optimization programme and so created a starting structure based on compound 1 (troglitazone, CHEMBL408) in a known PPAR-γ hit-to-lead optimization programme (Collins *et al.*, 1998). A key question is whether starting from compound 1 could automatically lead us to compound 2 in the same paper. After 57 cycles, the similarity and docking script had significantly optimized away from the initial molecule and retrieved several high-scoring molecules, including the substrate of 1fm9 (farglitazar, CHEMBL107367). This structure was the 531st structure docked. The exact ordering of retrieval will, however, vary from run to run of the script, as the *GOLD* score [divided by $\sqrt{(N \text{ heavy atoms})}$] is used to prioritize each cycle. *GOLD* is a stochastic docking program, and so variance in the scores will change the priority order for starting points from one run to the next.

The script mostly finds solutions containing the carboxylate group, including the substrate bound in PDB entry 1fm9 in the earlier cycles. The code effects an exploration around the related chemical space to the start point and generates logical suggestions for alternative ligands. Some examples are tabulated in Table S3. The earlier cycles gradually move the structure away from the thiazolidinedione fragment found in troglitazone towards the benzophenone fragment found in the

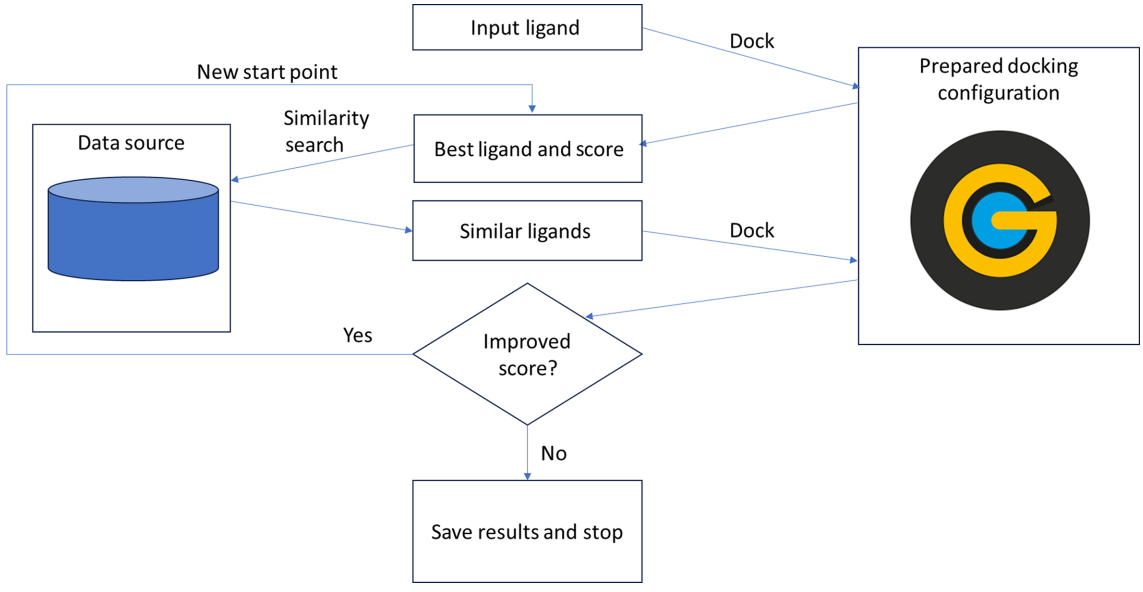


Figure 8
The workflow for similarity-driven docking.

later series, and in the bound substrate of 1fm6. Later cycles explore modifications to the benzophenone fragment, though fail to find significant further improvements. The trajectory of optimization can be visualized using the sequence of best performing structures retrieved as the script progresses (Table S4).

This script is meant merely as a demonstration. One weakness of this approach is that it explores only clearly chemically very similar space to the substrate. Another possible weakness of this method is that it relies on docking scores and the docking protocol used. It is far easier to find the substrate bound to the original docked system, as the protein is optimized perfectly for this type of substrate (re-docking a substrate to its original protein is known to be significantly easier than cross-docking). Finally, we note that docking scores tend to correlate with molecular size; while we have a maximum molecular weight limit in the run, one can still tend to optimize towards larger structures with high numbers of saturated carbons, as these atoms score well but in a non-specific way. By using a normalized score, this effect is somewhat ameliorated.

In principle, however, integration of other scoring methodologies, or post-docking relaxation which would reduce self-docking bias, is very tractable as part of the workflow. One could adapt the approach further using alternative search methods. For example, rather than conventional similarity, one could use 3D pharmacophoric patterns. One could also imagine using a synthesis-driven approach, creating novel possible molecules, instead of similarity searching to drive towards modified new ligands from a known starting point. Thus, we demonstrate the value of the integration of these various cheminformatic methods in a scripted environment.

5.4. Visualizing metal–metal interactions using Voronoi tessellation

5.4.1. Motivation. Metal-to-metal interactions are often an important aspect of understanding the properties of metal-containing structures, in both through-space and bonding terms. This example is intended to demonstrate how complex geometric models can be constructed using data from crystal structures and external libraries.

The unpaired electrons associated with an isolated metal atom can, as two of these metals are brought together, result in the formation of an exchange interaction and the correlation of spins – ferromagnetic ordering of multiple types (Atkins *et al.*, 2010). When close, individual metals can form direct bonds of multiple types, with subtle differences as a function of the distances between partners. Describing the observed bonding within a crystal structure is an important aspect of relaying and contextualizing the unique aspects and expected behaviour of these compounds. Communicating these interactions can be difficult in large clusters or among magnetic materials without appropriate visualizations.

With ferromagnetism as an example, interactions can be understood as an Ising model network of magnetic domains, each comprising a single metal atom (Coronado, 2020). When represented by Voronoi–Dirichlet polyhedra, these domains become tessellating blocks with faces equidistant to individual atoms and are therefore able to stand in for the interaction between them (Blatov, 2004). This can, in ideal terms, show directly from the geometry which simplified Ising models are appropriate for this crystal system.

Voronoi–Dirichlet polyhedra have numerous applications in crystallography under various names, such as the domain of influence or Wigner–Seitz cell, or the Brillouin zone in reciprocal space. Blatov (2004) has written an excellent summary of the theory and applications of this mathematical transform to crystallography, and recently molecular crystals have been investigated more thoroughly [see Savchenkov *et al.* (2023) and references therein].

5.4.2. The script. The script for this example is available at https://github.com/ccdc-opensource/csd-python-api-scripts/tree/main/api_paper_2024/example_4.

This script initializes a `CrystalVoronoi` class, which transforms a crystal structure (here `ccdc_crystal`) into a Voronoi representation. The key code is shown in Fig. 9. The molecular unit (or limited polymer) is defined as '`molec`' and a collection of atoms (points) within a given radius (`distance_range`, default 15 Å) of these central atoms is produced as '`shell`'. This ensures the polyhedral representations will terminate at symmetry-related atoms. The unique coordinates and labels are extracted with `numpy.unique` to remove duplicates, and finally the Voronoi division of this space can be calculated.

The remaining parts of the script deal with plotting these calculated polyhedra and overlaid atoms.

Voronoi polyhedra are calculated with

`scipy.spatial.Voronoi`

(Barber *et al.*, 1996; Virtanen *et al.*, 2020), and the closed Voronoi polyhedra are visualized using

```
molec = ccdc_crystal.molecule
shell = [x for x in ccdc_crystal.molecular_shell(distance_range=(0., float(spherelimit))).atoms
        + molec.atoms if x.is_metal]
uq_coords, uq_indices = unique(atleast_2d(array([array(x.coordinates) for x in metal_shell])),
                               axis = 0, return_index = True)
all_labels = [x.label for x in metal_shell]
uq_labels = [all_labels[y] for y in uq_indices]
tv = scipy.spatial.Voronoi(uq_coords)
```

Figure 9
A code snippet from the `MetalVoronoi` visualization script.

`scipy.spatial.ConvexHull`

and

`plotly.graph_objects.Mesh3D`

functions (Plotly Technologies Inc., https://plot.ly). Molecular representations with weighted diagrams additionally require *pyvoro* (https://pypi.org/project/pyvoro/).

A copy of this script and an example notebook for interacting directly with the figures and polyhedra are provided at the above URL.

5.4.3. Results. The visual results of `CrystalVoronoi.Figs` show stark differences in metal-containing materials by dimensionality. As an example, pyrazolyl-containing ligands are often used in coordination chemistry (Halcrow, 2009) and can bridge two metal centres, usually with a distance between 3.0 and 4.5 Å for first-row transition metals (8648 instances across 3371 structures from the CSD). With diverse metal centres, ligand steric demand, and additional binding sites, co-ligands and crystal growth conditions, a plethora of different clusters and polymeric forms can occur in different crystals. When these centres are magnetic, this distance is conducive to magnetic ordering between these centres.

Four examples are shown of different metals and bridging ligands in Fig. 10, namely ABEVEM (Pickl & Pöthig, 2021), a dinuclear 'Siamese twin' porphyrinoid bis-chelate; CIPGEQ (Tong *et al.*, 2017), a cluster of four iron centres; AHUCEN (Veronelli *et al.*, 2015), a plate-like construction with a 1D stack of close contacts; and DIDNAJ (Al Isawi *et al.*, 2023), a metal-based 'nano-jar' of Cu^{II} centres. Inspection of these plots indicates, respectively, dimeric, tetrameric, polymeric and metallocyclic networks derived from the spatial arrangement of the metal centres.

5.5. Accessing and visualizing particle information

5.5.1. Motivation. The CSD Python API enables researchers to build on the functionality of the CSD portfolio and create prototypes that help them gain a better understanding of the data or analyse them in a new context. A clear benefit of the CSD Python API is the ability to access object information that would typically only be used when rendering to a desktop interface. Here, we illustrate how object data can be used for classification and for 3D visualization using third-party libraries.

With an interest in particles, a general challenge is how to describe their shape consistently, *i.e.* the particle morphology. Whilst we can generate morphologies by predicting their facet representations using methods such as BFDH (Bravais–Friedel–Donnay–Harker; Donnay & Harker, 1937), assigning these resultant shapes to distinct classes on the basis of their aspect ratios is not a trivial task. This example automates the classification and displays the results in an interactive HTML file.

5.5.2. The script. The script for this example is available at https://github.com/ccdc-opensource/csd-python-api-scripts/tree/main/api_paper_2024/example_5.

Fig. 11 shows a simplified view of the data flow. The script simply uses the crystal object to calculate a morphology, in this case using the BFDH method, and then accesses the related 3D object data via facets and oriented bounding boxes to compute the shape classification and visualize the morphology. The shape classification is as described by Angelidakis *et al.* (2022). Both classification and morphology are plotted using *Plotly*, yielding interactive graphs

The morphology object contains an explicit depiction of the convex hull calculated in C++, including the vertex for each

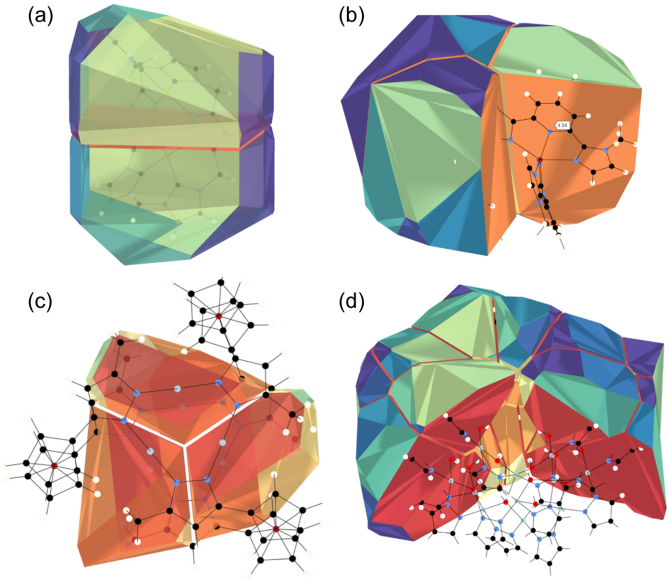


Figure 10
Voronoi tessellations of four metal complexes, (*a*) ABEVEM, (*b*) CIPGEQ, (*c*) AHUCEN and (*d*) DIDNAJ.

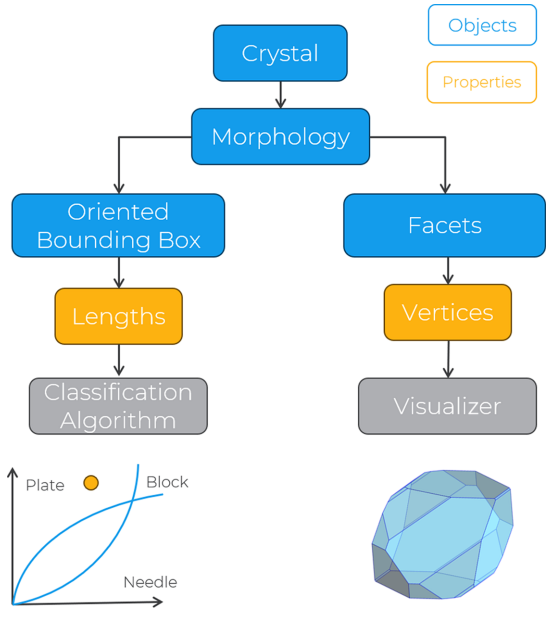


Figure 11
A flowchart showing a simplified view of the script's data flow, resulting in two interactive graphs.

facet and the oriented bounding box. Briefly, the oriented bounding box is computed by minimizing the volume of an orthonormal box that can rotate to include all facet vertices. The oriented bounding box describes the aspect ratio of the morphology and thus can be used for classification. Facet information is accessed via

```
morphology.facets[i].corners
```

for each facet. The bounding box dimensions are also easily accessible via

```
morphology.oriented_bounding_box.lengths
```

As these properties are computed in the C++ layer, they are relatively quick to compute (5 ms to compute the bounding box three times in the example structure on a single core).

A significant part of the script has been written to parse the data into the interactive graphs created with *Plotly*. Instructions on how to run the script are included in the `ReadMe.md` file.

5.5.3. Results. Running the script generates an HTML file, the results of which can be seen in Fig. 12. Due to the higher aspect ratio morphology, CBMZPN03 was adopted as the example structure. Using the oriented bounding box and subsequent classification standardizes the shape description that is assigned to a given particle, allowing for direct comparisons between morphologies. An example of the morphology generated for HXACAN28 is shown in Fig. S2 where it is classified as a block and evidently appears to have a smaller aspect ratio.

It should be noted that, whilst the script uses BFDH as the morphology generation method, other methods are available in the CSD Python API via the `ccdc.morphology` module. Furthermore, one could input experimentally measured morphologies or those computed with other methods using the `morphology.from_growth_rates` function.

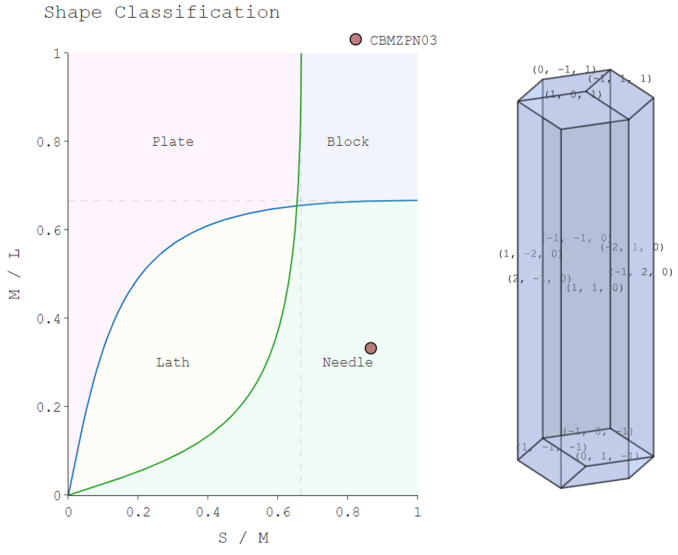


Figure 12
The HTML output of particle shape calculations for CBMZPN03. The shape classification is shown on the left, describing the particle as a needle. The corresponding BFDH morphology is shown on the right.

The ability to access the data required to construct virtual particles extends beyond visualization and classification. These data would allow further analysis to calculate the particle characteristics linked to the individual facets and those representations as a function of the overall shape.

6. Summary and outlook

In this short review and showcase of examples, we have illustrated the benefit of providing an application programming interface to the CSD and associated software tools. As can be seen from the growth in use, the CSD Python API is beginning to have impact within the community, facilitating research paradigms that would have been challenging prior to its release. The examples demonstrate not only the value of having an API to the CCDC's data, software and services but also the value of having multiple different APIs and toolkits accessible within a single programming language. This empowers users to integrate seamlessly the information stored within one software system with analytical methods available elsewhere, enabling more comprehensive and efficient workflows. Furthermore, the ability to write scripts and workflows that can be recorded in software repositories and then re-used greatly aids aims to make research data and outputs FAIR (findable, accessible, interoperable and reproducible); it is far easier to reproduce and adapt research if a workflow for such work is readily available.

It is important to acknowledge that Python APIs have their limitations. Python's interpreted nature inherently makes it slower than compiled languages, leading many scientific applications, including those developed by the CCDC, to utilize faster compiled languages for computationally intensive tasks while leveraging Python for its user-friendly interface and scientific ecosystem. This approach leads to challenges downstream, as it means different packages can sometimes conflict with each other. For example, there is an interesting exposition of how such issues affected the popular *SciPy* package with the release of Python 3.12 (Obermeier, 2023).

These challenges have led some developers to have a rethink. New languages are coming into the ecosystem which try to tackle some of the challenges faced; for example, the language Julia is designed to provide scripting-like behaviour but with compiled language speed. We have noted (Glasby *et al.*, 2024) that this speed can be very advantageous in some endeavours. Future developments in the cheminformatics community will probably see increased adoption of such languages, in particular to give better support to high-performance computing tasks.

Many data APIs are provided through RESTful services offering streamlined access to information via URLs. Indeed, the CrossRef and OpenAlex APIs used in one of the examples here provide access in this way. This has the advantage that the API access is somewhat decoupled from the underlying implementation language; a developer can write a relatively data API agnostic layer for making requests and then parse the information locally. We recognize a similar need for

scientists wishing to develop scientific software that accesses the CSD.

For now, though, providing access to the CSD data and software through a Python-based interface is, as we hope we have demonstrated here, a highly useful and growing *modus operandi* for taking advantage of the huge corpus of structural information in the CSD, as well as the advanced algorithms and methods that underly CCDC software. The large array of useful scientific tools available in the Python ecosystem alongside CCDC offerings further enriches the experience. Training materials for CCDC tools including the CSD Python API are readily available through the CCDC's web site (Gimondi, Ward and Bryant, *On-demand CSDU Modules*, https://www.ccdc.cam.ac.uk/community/training-and-learning/csdu-modules/). The pedagogical benefits of using Python for educational materials and workshops are significant as it offers an intuitive platform for learning and experimenting with structural chemistry algorithms and CSD data, using a programming language that is increasingly familiar to data scientists and structural chemists.

Acknowledgements

The authors of this paper are a select group who have had oversight of aspects of the CSD Python API or who have been directly involved in the writing of the review or examples outlined. That said, the CSD Python API is a huge collaborative endeavour over many years at the CCDC. We acknowledge all the scientists and developers at the CCDC and beyond who have contributed to its development, initially as an internal tool and latterly available to external users. Thanks in particular to Elna Pidcock.

References

Adamji, H., Nandy, A., Kevlishvili, I., Román-Leshkov, Y. & Kulik, H. J. (2023). *J. Am. Chem. Soc.* **145**, 14365–14378.
Ai, Q., Bhat, V., Ryno, S. M., Jarolimek, K., Sornberger, P., Smith, A., Haley, M. M., Anthony, J. E. & Risko, C. (2021). *J. Chem. Phys.* **154**, 174705.
Al Isawi, W. A., Hartman, C. K., Singh, P., Zeller, M. & Mezei, G. (2023). *Inorg. Chem.* **62**, 5716–5728.
Angelidakis, V., Nadimi, S. & Utili, S. (2022). *Powder Technol.* **396**, 689–695.
Atkins, P. W., Overton, T. L., Rourke, J. P., Weller, M. T. & Armstrong, F. A. (2010). *Shriver and Atkins' Inorganic Chemistry*, 5th ed. Oxford University Press.
Balcells, D. & Skjelstad, B. B. (2020). *J. Chem. Inf. Model.* **60**, 6135–6146.
Barber, C. B., Dobkin, D. P. & Huhdanpaa, H. (1996). *ACM Trans. Math. Softw.* **22**, 469–483.
Beazley, D., Fulton, W., Matus, M. & Ballabio, L. (2022). *SWIG: Simplified Wrapper and Interface Generator*, https://www.swig.org/.
Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K. & Wiswedel, B. (2008). *Data Analysis, Machine Learning and Applications*, edited by C. Preisach, H. Burkhardt, L. Schmidt-Thieme & R. Decker, pp. 319–326. Berlin, Heidelberg: Springer.
Blatov, V. A. (2004). *Crystallogr. Rev.* **10**, 249–318.
Bond, A. D. (2021). *Acta Cryst.* B**77**, 357–364.
Bruno, I. J., Cole, J. C., Edgington, P. R., Kessler, M., Macrae, C. F., McCabe, P., Pearson, J. & Taylor, R. (2002). *Acta Cryst.* B**58**, 389–397.
Bruno, I. J., Cole, J. C., Kessler, M., Luo, J., Motherwell, W. D. S., Purkis, L. H., Smith, B. R., Taylor, R., Cooper, R. I., Harris, S. E. & Orpen, A. G. (2004). *J. Chem. Inf. Comput. Sci.* **44**, 2133–2144.
Bruno, I. J., Cole, J. C., Lommerse, J. P. M., Rowland, R. S., Taylor, R. & Verdonk, M. L. (1997). *J. Comput. Aided Mol. Des.* **11**, 525–537.
Bryant, M. J., Black, S. N., Blade, H., Docherty, R., Maloney, A. G. P. & Taylor, S. C. (2019). *J. Pharm. Sci.* **108**, 1655–1662.
Bryant, M. J., Maloney, A. G. P. & Sykes, R. A. (2018). *CrystEngComm*, **20**, 2698–2704.
Clevers, S. & Coquerel, G. (2020). *CrystEngComm*, **22**, 7407–7419.
Clydesdale, G., Docherty, R. & Roberts, K. J. (1991). *Comput. Phys. Commun.* **64**, 311–328.
Cole, J. C., Groom, C. R., Korb, O., McCabe, P. & Shields, G. P. (2016). *J. Chem. Inf. Model.* **56**, 652–661.
Cole, J. C., Korb, O., McCabe, P., Read, M. G. & Taylor, R. (2018). *J. Chem. Inf. Model.* **58**, 615–629.
Cole, J. C., Murray, C. W., Nissink, J. W. M., Taylor, R. D. & Taylor, R. (2005). *Proteins*, **60**, 325–332.
Collins, J. L., Blanchard, S. G., Boswell, G. E., Charifson, P. S., Cobb, J. E., Henke, B. R., Hull-Ryde, E. A., Kazmierski, W. M., Lake, D. H., Leesnitzer, L. M., Lehmann, J., Lenhard, J. M., Orband-Miller, L. A., Gray-Nunez, Y., Parks, D. J., Plunkett, K. D. & Tong, W.-Q. (1998). *J. Med. Chem.* **41**, 5037–5054.
Coronado, E. (2020). *Nat. Rev. Mater.* **5**, 87–104.
Curran, P. R., Radoux, C. J., Smilova, M. D., Sykes, R. A., Higueruelo, A. P., Bradley, A. R., Marsden, B. D., Spring, D. R., Blundell, T. L., Leach, A. R., Pitt, W. R. & Cole, J. C. (2020). *J. Chem. Inf. Model.* **60**, 1911–1916.
David, L., Mdahoma, A., Singh, N., Buchoux, S., Pihan, E., Diaz, C. & Rabal, O. (2022). *Bioinform. Adv.* **2**, vbac090.
Davies, M., Nowotka, M., Papadatos, G., Dedman, N., Gaulton, A., Atkinson, F., Bellis, L. & Overington, J. P. (2015). *Nucleic Acids Res.* **43**, W612–W620.
Donnay, J. D. H. & Harker, D. (1937). *Am. Mineral.* **22**, 446–467.
Dull, J. T., He, X., Viereck, J., Ai, Q., Ramprasad, R., Otani, M. C., Sorli, J., Brandt, J. W., Carrow, B. P., Tinoco, A. D., Loo, Y.-L., Risko, C., Rangan, S., Kahn, A. & Rand, B. P. (2023). *Adv. Mater.* **35**, 2302871.
Dypvik Sødahl, E., Seyedraoufi, S., Görbitz, C. H. & Berland, K. (2023). *Cryst. Growth Des.* **23**, 8607–8619.
Fabjan, J., Koniuszewski, F., Schaar, B. & Ernst, M. (2021). *Front. Neurosci.* **14**, 611953.
Filik, J., Ashton, A. W., Chang, P. C. Y., Chater, P. A., Day, S. J., Drakopoulos, M., Gerring, M. W., Hart, M. L., Magdysyuk, O. V., Michalik, S., Smith, A., Tang, C. C., Terrill, N. J., Wharmby, M. T. & Wilhelm, H. (2017). *J. Appl. Cryst.* **50**, 959–966.
Frade, A. P., McCabe, P. & Cooper, R. I. (2020). *CrystEngComm*, **22**, 7186–7192.
Gampe, R. T. Jr, Montana, V. G., Lambert, M. H., Miller, A. B., Bledsoe, R. K., Milburn, M. V., Kliewer, S. A., Willson, T. M. & Xu, H. E. (2000). *Mol. Cell*, **5**, 545–555.
Giangreco, I., Cole, J. C. & Thomas, E. (2017). *Cryst. Growth Des.* **17**, 3192–3203.
Giangreco, I., Mukhopadhyay, A. & Cole, J. C. (2021). *J. Chem. Inf. Model.* **61**, 5841–5852.
Glasby, L. T., Cordiner, J. L., Cole, J. C. & Moghadam, P. Z. (2024). *Chem. Mater.* https://doi.org/10.1021/acs.chemmater.4c00762.
Glasby, L. T., Gubsch, K., Bence, R., Oktavian, R., Isoko, K., Moosavi, S. M., Cordiner, J. L., Cole, J. C. & Moghadam, P. Z. (2023). *Chem. Mater.* **35**, 4510–4524.
Groom, C. R., Bruno, I. J., Lightfoot, M. P. & Ward, S. C. (2016). *Acta Cryst.* B**72**, 171–179.
Guo, J., Janet, J. P., Bauer, M. R., Nittinger, E., Giblin, K. A., Papadopoulos, K., Voronov, A., Patronov, A., Engkvist, O. & Margreitter, C. (2021). *J. Cheminform.* **13**, 89.

Hadfield, T. E., Imrie, F., Merritt, A., Birchall, K. & Deane, C. M. (2022). *J. Chem. Inf. Model.* **62**, 2280–2292.
Halcrow, M. A. (2009). *Dalton Trans.* pp. 2059–2073.
Hill, A., Kras, W., Theodosiou, F., Wanat, M., Lee, D. & Cruz-Cabeza, A. J. (2023). *J. Am. Chem. Soc.* **145**, 20562–20577.
Iuzzolino, L., Reilly, A. M., McCabe, P. & Price, S. L. (2017). *J. Chem. Theory Comput.* **13**, 5163–5171.
Johnston, D., Sarjeant, A. & Wiggin, S. (2018). *Acta Cryst.* A**74**, a398.
Jones, G., Willett, P., Glen, R. C., Leach, A. R. & Taylor, R. (1997). *J. Mol. Biol.* **267**, 727–748.
Kaźmierczak, M. & Patyk-Kaźmierczak, E. (2021). *Acta Cryst.* B**77**, 1012–1020.
Kevlishvili, I., Duan, C. & Kulik, H. (2023). *J. Phys. Chem. Lett.* **14**, 11100–11109.
Kim, S., Chen, J., Cheng, T., Gindulyte, A., He, J., He, S., Li, Q., Shoemaker, B. A., Thiessen, P. A., Yu, B., Zaslavsky, L., Zhang, J. & Bolton, E. E. (2023). *Nucleic Acids Res.* **51**, D1373–D1380.
King-Smith, E. (2024). *Chem. Sci.* **15**, 5143–5151.
Korb, O., Kuhn, B., Hert, J., Taylor, N., Cole, J., Groom, C. & Stahl, M. (2016). *J. Med. Chem.* **59**, 4257–4266.
Kuhn, B., Gilberg, E., Taylor, R., Cole, J. & Korb, O. (2019). *J. Med. Chem.* **62**, 10441–10455.
Lee, A. van der & Dumitrescu, D. G. (2021). *Chem. Sci.* **12**, 8537–8547.
Li, A., Bueno-Perez, R. & Fairen-Jimenez, D. (2022). *Chem. Sci.* **13**, 13507–13523.
Li, A., Bueno-Perez, R. & Fairen-Jimenez, D. (2023). *AI-Guided Design and Property Prediction for Zeolites and Nanoporous Materials*, edited by G. Sastre & F. Daeyaert. pp. 201–232. Chichester: Wiley.
Li, A., Bueno-Perez, R., Wiggin, S. & Fairen-Jimenez, D. (2020). *CrystEngComm*, **22**, 7152–7161.
Ma, C. Y., Geatches, D., Hsiao, Y.-W., Kwokal, A. & Roberts, K. J. (2023). *Cryst. Growth Des.* **23**, 4522–4537.
Macrae, C. F., Sovago, I., Cottrell, S. J., Galek, P. T. A., McCabe, P., Pidcock, E., Platings, M., Shields, G. P., Stevens, J. S., Towler, M. & Wood, P. A. (2020). *J. Appl. Cryst.* **53**, 226–235.
Mendez, D., Gaulton, A., Bento, A. P., Chambers, J., De Veij, M., Félix, E., Magariños, M. P., Mosquera, J. F., Mutowo, P., Nowotka, M., Gordillo-Marañón, M., Hunter, F., Junco, L., Mugumbate, G., Rodriguez-Lopez, M., Atkinson, F., Bosc, N., Radoux, C. J., Segura-Cabrera, A., Hersey, A. & Leach, A. R. (2019). *Nucleic Acids Res.* **47**, D930–D940.
Moghadam, P. Z., Li, A., Wiggin, S. B., Tao, A., Maloney, A. G. P., Wood, P. A., Ward, S. C. & Fairen-Jimenez, D. (2017). *Chem. Mater.* **29**, 2618–2625.
Moghadam, Z., Li, P., Liu, A., Bueno-Perez, R., Wang, R., Wiggin, S. B., Wood, S. A. & Fairen-Jimenez, P. (2020). *Chem. Sci.* **11**, 8373–8387.
Moldovan, A. A. & Maloney, A. G. P. (2024). *Cryst. Growth Des.* **24**, 4160–4169.
Montis, R., Hursthouse, M. B., Kendrick, J., Howe, J. & Whitby, R. J. (2022). *Cryst. Growth Des.* **22**, 559–569.
Nandy, A., Terrones, G., Arunachalam, N., Duan, C., Kastner, D. W. & Kulik, H. J. (2022). *Sci. Data*, **9**, 74.
Obermeier, A. (2023). *The 'Eu' in Eucatastrophe – Why SciPy Builds for Python 3.12 on Windows are a Minor Miracle*, https://labs-g49mcsy9c-quansight.vercel.app/blog/building-scipy-with-flang.
Padula, D., Omar, H., Nematiaram, T. & Troisi, A. (2019). *Energy Environ. Sci.* **12**, 2412–2416.
Pickl, T. & Pöthig, A. (2021). *Organometallics*, **40**, 3056–3065.
Priem, J., Piwowar, H. & Orr, R. (2022). *arXiv*:2205.01833.
Radoux, C. J., Olsson, T. S. G., Pitt, W. R., Groom, C. R. & Blundell, T. L. (2016). *J. Med. Chem.* **59**, 4314–4325.
Reeves, M. G., Wood, P. A. & Parsons, S. (2019). *Acta Cryst.* B**75**, 1096–1105.
Reeves, M. G., Wood, P. A. & Parsons, S. (2020). *J. Appl. Cryst.* **53**, 1154–1162.
Rekis, T. (2020). *Acta Cryst.* B**76**, 307–315.
Rosen, A., Iyer, S., Ray, D., Yao, Z., Aspuru-Guzik, A., Gagliardi, L., Notestein, J. & Snurr, R. Q. (2020). *ChemRxiv*, https://doi.org/10.26434/chemrxiv.13147616.v1.
Ruiz-Moreno, A. J., Dömling, A. & Velasco-Velázquez, M. A. (2021). *Cancer Cell Signaling: Methods and Protocols*, edited by M. Robles-Flores, pp. 31–43. New York: Springer US.
Sarkisov, L., Bueno-Perez, R., Sutharson, M. & Fairen-jimenez, D. (2020). *ChemRxiv*, https://doi.org/10.26434/chemrxiv.12923558.v1.
Savchenkov, A. V., Ahmed, E., Karothu, D. P. & Naumov, P. (2023). *Cryst. Growth Des.* **23**, 6484–6490.
Schober, C., Reuter, K. & Oberhofer, H. (2016). *J. Phys. Chem. Lett.* **7**, 3973–3977.
Seyedraoufi, S., Sødahl, E. D., Görbitz, C. H. & Berland, K. (2023). *arXiv*:2306.00363.
Seyedraoufi, S., Sødahl, E. D., Nilsen, O., Gørbitz, C. H. & Berland, K. (2022). *Acta Cryst.* A**78**, e499.
Short, M. A. S., Tovee, C. A., Willans, C. E. & Nguyen, B. N. (2023). *Catal. Sci. Technol.* **13**, 2407–2420.
Smart, O. S., Sharff, A., Holstein, J., Womack, T. O., Flensburg, C., Keller, P., Paciorek, W., Vonrhein, C. & Bricogne, G. (2021). *Grade2*. Version 1.6.0. Global Phasing Ltd, Cambridge, United Kingdom. https://gphl.gitlab.io/grade2_docs/grade2.pdf.
Swain, M. C. & Cole, J. M. (2016). *J. Chem. Inf. Model.* **56**, 1894–1904.
Tong, J., Demeshko, S., Dechert, S. & Meyer, F. (2017). *Eur. J. Inorg. Chem.* **2017**, 4333–4343.
Tosstorff, A., Cole, J. C., Taylor, R., Harris, S. F. & Kuhn, B. (2020). *J. Chem. Inf. Model.* **60**, 6595–6611.
Tosstorff, A., Rudolph, M. G., Cole, J. C., Reutlinger, M., Kramer, C., Schaffhauser, H., Nilly, A., Flohr, A. & Kuhn, B. (2022). *J. Comput. Aided Mol. Des.* **36**, 753–765.
Verdonk, M. L., Cole, J. C., Hartshorn, M. J., Murray, C. W. & Taylor, R. D. (2003). *Proteins*, **52**, 609–623.
Verdonk, M. L., Cole, J. C. & Taylor, R. (1999). *J. Mol. Biol.* **289**, 1093–1108.
Veronelli, M., Dechert, S., Demeshko, S. & Meyer, F. (2015). *Inorg. Chem.* **54**, 6917–6927.
Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., Vijaykumar, A., Bardelli, A. P., Rothberg, A., Hilboll, A., Kloeckner, A., Scopatz, A., Lee, A., Rokem, A., Woods, C. N., Fulton, C., Masson, C., Häggström, C., Fitzgerald, C., Nicholson, D. A., Hagen, D. R., Pasechnik, D. V., Olivetti, E., Martin, E., Wieser, E., Silva, F., Lenders, F., Wilhelm, F., Young, G., Price, G. A., Ingold, G., Allen, G. E., Lee, G. R., Audren, H., Probst, I., Dietrich, J. P., Silterra, J., Webber, J. T., Slavič, J., Nothman, J., Buchner, J., Kulick, J., Schönberger, J. L., de Miranda Cardoso, J. V., Reimer, J., Harrington, J., Rodríguez, J. L. C., Nunez-Iglesias, J., Kuczynski, J., Tritz, K., Thoma, M., Newville, M., Kümmerer, M., Bolingbroke, M., Tartre, M., Pak, M., Smith, N. J., Nowaczyk, N., Shebanov, N., Pavlyk, O., Brodtkorb, P. A., Lee, P., McGibbon, R. T., Feldbauer, R., Lewis, S., Tygier, S., Sievert, S., Vigna, S., Peterson, S., More, S., Pudlik, T., Oshima, T., Pingel, T. J., Robitaille, T. P., Spura, T., Jones, T. R., Cera, T., Leslie, T., Zito, T., Krauss, T., Upadhyay, U., Halchenko, Y. O. & Vázquez-Baeza, Y. (2020). *Nat. Methods*, **17**, 261–272.
Vriza, A., Canaj, A. B., Vismara, R., Kershaw Cook, L. J., Manning, T. D., Gaultois, M. W., Wood, P. A., Kurlin, V., Berry, N., Dyer, M. S. & Rosseinsky, M. J. (2021). *Chem. Sci.* **12**, 1702–1719.
Vriza, A., Sovago, I., Widdowson, D., Kurlin, V., Wood, P. A. & Dyer, M. S. (2022). *Digit. Discov.* **1**, 834–850.
Walsh, M. P., Barclay, J. A., Begg, C. S., Xuan, J., Johnson, N. T., Cole, J. C. & Kitching, M. O. (2022). *JACS Au*, **2**, 2235–2250.

Ward, M. R., Bull, C. L., Funnell, N. P., Warren, M. R. & Oswald, I. D. H. (2023). *Int. J. Pharm.* **647**, 123514.
Weininger, D. (1997). *SMARTS – A Language for Describing Molecular Patterns*, https://www.daylight.com/dayhtml/doc/theory/theory.smarts.html.
Werner, J. E. & Swift, J. A. (2021). *CrystEngComm*, **23**, 1555–1565.
Wicker, J. G. P. & Cooper, R. I. (2015). *CrystEngComm*, **17**, 1927–1934.
Willett, P., Cole, J. C. & Bruno, I. J. (2020). *CrystEngComm*, **22**, 7233–7241.
Wilson, C. J. G., Cervenka, T., Wood, P. A. & Parsons, S. (2022). *Cryst. Growth Des.* **22**, 2328–2341.
Wood, P. A., Olsson, T. S. G., Cole, J. C., Cottrell, S. J., Feeder, N., Galek, P. T. A., Groom, C. R. & Pidcock, E. (2013). *CrystEngComm*, **15**, 65–72.
Wright, S. E., Bryant, M. J. & Cruz-Cabeza, A. J. (2020). *CrystEngComm*, **22**, 7217–7228.
Xin, D., Gonnella, N. C., He, X. & Horspool, K. (2019). *Cryst. Growth Des.* **19**, 1903–1911.
Zhang, L., Qiu, Y., Liu, W.-G., Chen, H., Shen, D., Song, B., Cai, K., Wu, H., Jiao, Y., Feng, Y., Seale, J. S. W., Pezzato, C., Tian, J., Tan, Y., Chen, X.-Y., Guo, Q.-H., Stern, C. L., Philp, D., Astumian, R. D., Goddard, W. A. & Stoddart, J. F. (2023). *Nature*, **613**, 280–286.