Short version

Step 1: Create two directories named ‘core’ and ‘ligand’.

These two folders should (if applicable) contain all input cores and ligands, respectively.

Step 2: ﻿In qd\_input.py alter the path variable to the folder containing the previously made ‘core’ and ’ligand’ folders.

Set path to os.getcwd() to use the current directory.

Step 3: Enter all input cores and ligands as a list in the ﻿input\_cores and ﻿input\_ligands variables.

Optional arguments can be attached to each molecule (see Long version).

Step 4: Alter any optional argument in the argument\_dict variable.

Step 5: Start the job by running qd\_input.py (*e.g.* ‘python qd\_input.py’).

Long version

Step 1 & 2: Nothing to add; see Short version

Step 3: Supported filetypes

The script currently supports a wide range of different input formats for molecules.

The accepted formats can roughly be divided into three categories:

* Files (.xyz, .pdb & .mol): Common filetypes containing a single molecule.

If only the filename is provided (*e.g.* Ligand1.xyz) the script will search for this file in path/ligand/. Alternatively, the absolute path to the file can be provided (*e.g.* /some/random/folder/Ligand1.xyz).

* Python objects (﻿<str>, <plams.Molecule> & <rdkit.Chem.Mol>): Provide a single input molecule directly via a pythonic object.

Strings are assumed to be SMILES strings.

* Molecule lists (.txt, .xlsx & folders): Files or folders containing one or more molecules.

These input files/folders will be searched (recursively) for any valid input files (*e.g.* .xyz, .pdb or SMILES strings). If only the file-/folder-name is provided (*e.g.* ligand\_list.txt) the script will search for this file/folder in path/ligand/. Alternatively, the absolute path to the file can be provided (*e.g.* /some/random/folder/ ligand\_list.txt). If an empty string, ‘None’ or <None> is provided, all files and folders in path/core/ or path/ligand/ will be searched.

Step 3: Optional arguments

﻿guess\_bonds <bool> = False

﻿ Try to guess bonds in the molecule based on types and positions of atoms.

Is set to False by default, with the exception of .xyz files.

﻿column <int> = 0

﻿ The column containing the to be imported molecules.

Relevant for .txt and .xlsx files if they contain multiple columns.

﻿row <int> = 0

﻿ The first row containing a to be imported molecule.

Relevant for .txt and .xlsx files.

﻿sheet\_name <str> = Sheet1

﻿ The name of the sheet containing the to be imported molecules.

Relevant for .xlsx files multiple sheets or sheets with non-default names.

﻿core\_indices <list>[<int>, …] = []

﻿Manually specify the atomic index of one ore more core atom(s) that will be attached to ligands. Alternatively, all atoms of a given element can be marked and replaced instead (see argument\_dict).

﻿ligand\_indices <list>[<int>] or <list>[<int>, <int>] = []

Manually specify the atomic index <int> of the ligand atom that will be attached to core. If two atomic indices are provided, the bond between list[0] and list[1] will be broken and the molecule containing list[0] is attached to the core.

﻿Serves as an alternative to the functional group based substructure\_search().

Example:

Input\_cores = yaml.load(﻿"""

- Mol1.xyz

- - Mol2.xyz

- guess\_bonds: True

- - Mol3.xyz

- guess\_bonds: False

- core\_indices: [1, 7, 83, 95]

﻿""")

Step 4: Optional arguments

﻿dir\_name\_list <list>[<str>, <str>, <str>] = [core, ligand, QD]

A list containing the names of the be created folders.

List[0] is the core input folder, list[1] the ligand input folder and list[2] the quantum dot output folder.

dummy <int> or <str> = Cl

The atomic number or atomic symbol of the atoms in the core that should be replaced with ligands. Alternatively, dummy atoms can be manually specified with the core\_indices variable.

use\_database <bool> = True

Enables or disables the use of database\_name.

database\_name <list>[<str>, <str>] = [ligand\_database.xlsx, QD\_database.xlsx]

Name plus extension of the (to be) created databases where all results will be stored. When possible, ligands and quantum dots will be pulled from these databases instead of reoptimizing their geometry.

ligand\_opt <bool> = True

Split the ligand into linear fragments and then recombine these fragments, searching for the optimal dihedral angle of the newly (re-)formed bond in the process. Involved an optimization with RDKit UFF.

split <bool> = True

False: The ligand is to be attached to the core in its entirety.

False examples:

–NR4+ 🡪 –NR4+

–O2CR 🡪 –O2CR

–HO2CR 🡪 –HO2CR

–H3CO2CR 🡪 –H3CO2CR

True: A proton, counterion or functional group first is to be removed first from the ligand.

True examples:

X–.NR4+ 🡪 –NR4+

HO2CR 🡪 –O2CR

Na+.–O2CR 🡪 –O2CR

H3CO2CR 🡪 –O2CR

ligand\_crs <bool> = False

Calculate the ligand volume, surface area and octanol/water partition coefficient

with ADF MOPAC + COSMO-RS.

qd\_opt <bool> = False

Optimize the quantum dot (*i.e*. core + all ligands) with ADF UFF.

The geometry of the core and ligand atoms directly attached to the core are frozen during this optimization.

maxiter <int> = 500

The maximum number of geometry iterations during qd\_opt.

qd\_int <bool> = False

Perform an activation strain analyses on the ligands attached to the quantum dot surface with RDKit UFF. The core is removed during this process; the analyses is thus exclusively focused on ligand deformation and inter-ligand interaction.

Yields three terms:

*E*strain: The energy required to deform the ligands from their equilibrium geometry to the geometry they adopt on the quantum dot surface. This term is, by definition, destabilizing.

*E*int: The mutual interaction between all deformed ligands. This term is characterized by the non-covalent interaction between ligands (UFF Lennard-Jones potential) and, depending on the inter-ligand distances, can be either stabilizing or destabilizing.

*E*: The sum of *E*strain and *E*int: accounts for both the destabilizing ligand deformation and (de-)stabilizing interaction between all ligands in the absence of the core.