Electronic Supplementary Information

ICHOR: A Modern Pipeline for Producing Gaussian Process Regression Models for Atomistic Simulations

Matthew Burn and Paul Popelier *

Department of Chemistry, The University of Manchester, Manchester, M13 9PL, Britain

*To whom correspondence should be addressed:

Phone: +44 161 3064511. E-mail: pla@manchester.ac.uk

Table of Contents

1 ICHOR Interface

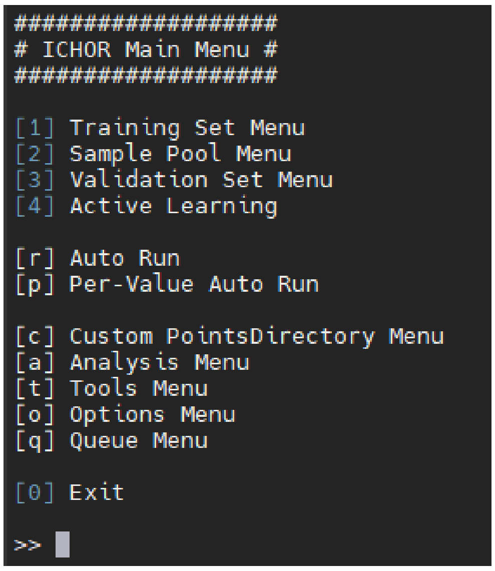
ICHOR provides three interfaces:

- `Terminal User Interface (TUI)`
- `Command Line Interface (CLI)`
- `Python Package (Library Interface)`

The TUI is designed for the average user and provides a series of menus in order to provide the most used ICHOR functionality in a manner that is an easy to use and to understand. The CLI provides access to a select list of ICHOR functions to provide access to ICHOR's functionality (when an interactive terminal may not be available, i.e. in a bash script for use on a compute node in a HPC cluster. The library interface is designed to provide all of ICHOR's tools, which includes a variety of common tasks in computational chemistry.

1.1 Terminal User Interface (TUI)

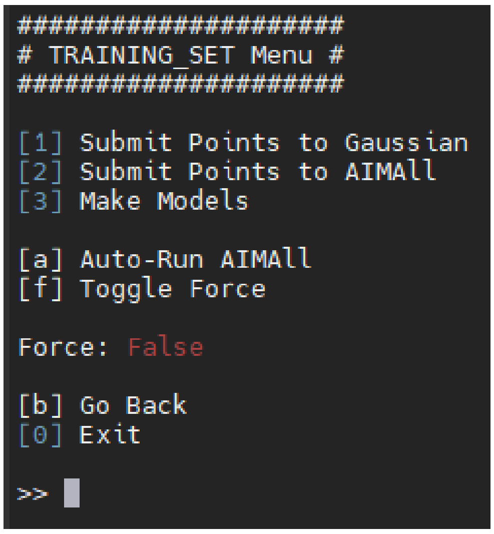
ICHOR's TUI is based upon a simple text menu designed for high portability and minimal resource usage. ICHOR's main menu is invoked when running the `ichor` command if installed or when calling `python` on the location of the ICHOR directory.



```
###################
# ICHOR Main Menu #
###################

[1] Training Set Menu
[2] Sample Pool Menu
[3] Validation Set Menu
[4] Active Learning

[r] Auto Run
[p] Per-Value Auto Run

[c] Custom PointsDirectory Menu
[a] Analysis Menu
[t] Tools Menu
[o] Options Menu
[q] Queue Menu

[0] Exit

>>
```

Figure S1. Screenshot of ICHOR's default main menu.

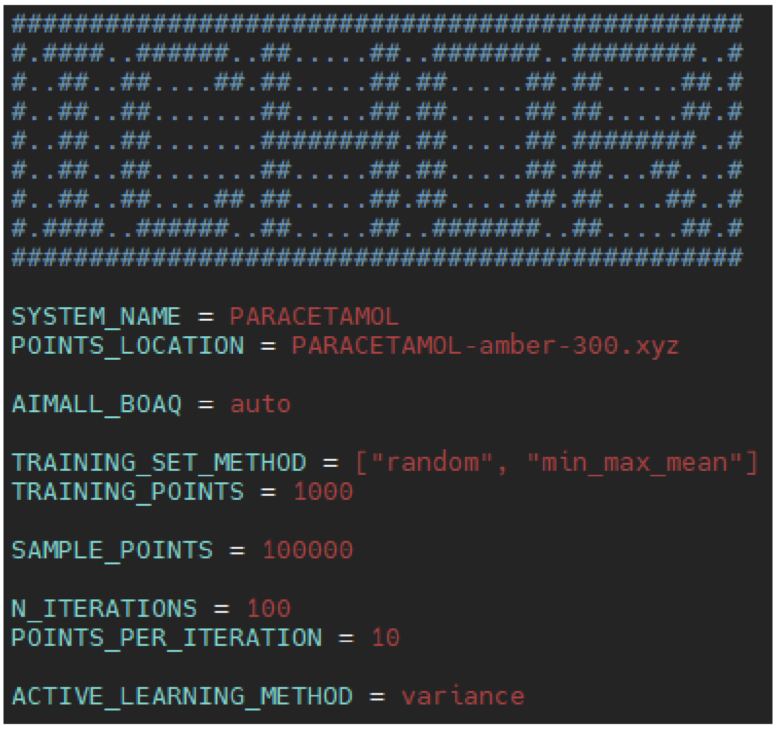
ICHOR's menu consists of a menu title followed by a series of options and messages, and is completed with a prompt. The ICHOR prompt contains all the functionality one expects of a modern prompt including history search, auto-completion and auto-complete suggestions based on the current input using the "tab" character. The prompt also allows for shortcuts not observed in the current menu, which is useful for bypassing submenus when performing common operations. Finding all options of a menu can be accessed in the usual prompt manner using tab completion (i.e. double pressing the "tab" button). The majority of main menu functions provides access to submenus that control specific ICHOR tasks, such as job submission or model analysis.



```
#####################
# TRAINING_SET Menu #
#####################

[1] Submit Points to Gaussian
[2] Submit Points to AIMAll
[3] Make Models

[a] Auto-Run AIMAll
[f] Toggle Force

Force: False

[b] Go Back
[0] Exit

>>
```

Figure S2. An example of a submenu for controlling the computation of the training set.

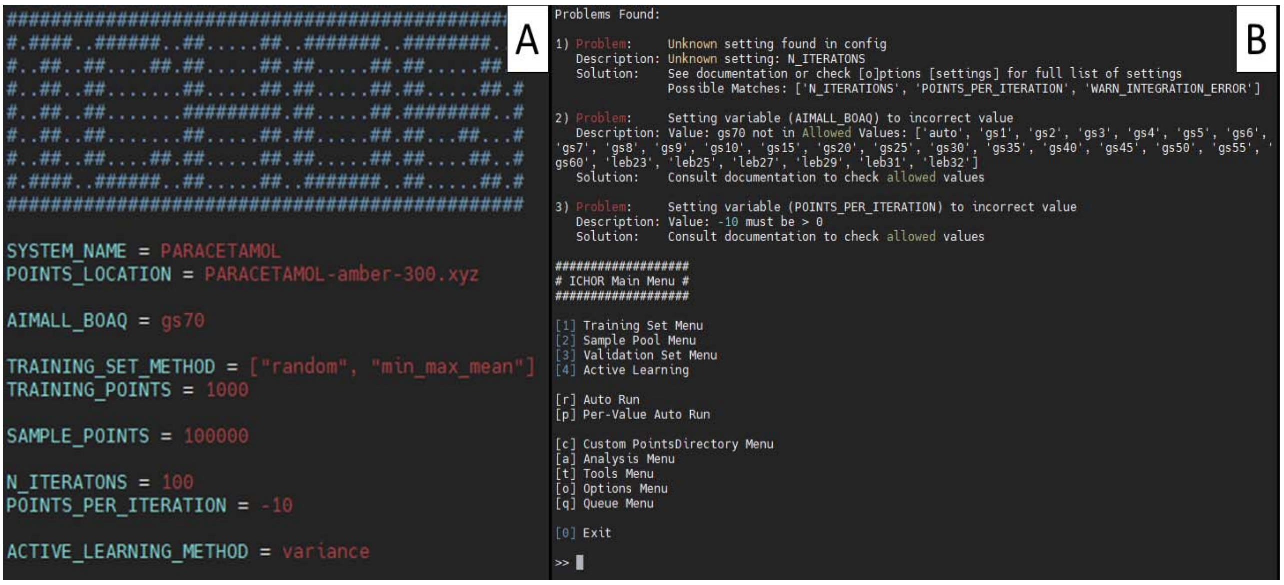
The user controls the global parameters of the ICHOR TUI via a *config* file. The currently supported config file formats are .properties and YAML. The .properties is the preferred choice due to its simplicity. ICHOR treats global parameters as simple key value stores where each value has a type and an optional range of values.



```
###############################################
#.####..######..##.....##..#######..########..#
#..##..##....##.##.....##.##.....##.##.....##.#
#..##..##.......##.....##.##.....##.##.....##.#
#..##..##.......#########.##.....##.########..#
#..##..##.......##.....##.##.....##.##...##...#
#..##..##....##.##.....##.##.....##.##....##..#
#.####..######..##.....##..#######..##.....##.#
###############################################

SYSTEM_NAME = PARACETAMOL
POINTS_LOCATION = PARACETAMOL-amber-300.xyz

AIMALL_BOAQ = auto

TRAINING_SET_METHOD = ["random", "min_max_mean"]
TRAINING_POINTS = 1000

SAMPLE_POINTS = 100000

N_ITERATIONS = 100
POINTS_PER_ITERATION = 10

ACTIVE_LEARNING_METHOD = variance
```

Figure S3. Example config.properties file for a paracetamol active learning run.

Internally, ICHOR converts all control parameters to the speficied type and checks if they are within the bounds set by the developer. If the user makes an error such as misspelling a parameter or entering an out-of-bounds value, an error message is displayed to the user when initialising the TUI application.



```
#################################################
#.####..######..##.....##..#######..########.
#..##..##....##.##.....##.##.....##.##.....##
#..##..##.......##.....##.##.....##.##.....##.#
#..##..##.......#########.##.....##.########..#
#..##..##.......##.....##.##.....##.##...##...#
#..##..##....##.##.....##.##.....##.##....##..#
#.####..######..##.....##..#######..##.....##.#
##################################################

SYSTEM_NAME = PARACETAMOL
POINTS_LOCATION = PARACETAMOL-amber-300.xyz

AIMALL_BOAQ = gs70

TRAINING_SET_METHOD = ["random", "min_max_mean"]
TRAINING_POINTS = 1000

SAMPLE_POINTS = 100000

N_ITERATONS = 100
POINTS_PER_ITERATION = -10

ACTIVE_LEARNING_METHOD = variance
```

```
Problems Found:

1) Problem:     Unknown setting found in config
   Description: Unknown setting: N_ITERATONS
   Solution:    See documentation or check [o]ptions [settings] for full list of settings
                Possible Matches: ['N_ITERATIONS', 'POINTS_PER_ITERATION', 'WARN_INTEGRATION_ERROR']

2) Problem:     Setting variable (AIMALL_BOAQ) to incorrect value
   Description: Value: gs70 not in Allowed Values: ['auto', 'gs1', 'gs2', 'gs3', 'gs4', 'gs5', 'gs6',
'gs7', 'gs8', 'gs9', 'gs10', 'gs15', 'gs20', 'gs25', 'gs30', 'gs35', 'gs40', 'gs45', 'gs50', 'gs55', '
gs60', 'leb23', 'leb25', 'leb27', 'leb29', 'leb31', 'leb32']
   Solution:    Consult documentation to check allowed values

3) Problem:     Setting variable (POINTS_PER_ITERATION) to incorrect value
   Description: Value: -10 must be > 0
   Solution:    Consult documentation to check allowed values

###################
# ICHOR Main Menu #
###################

[1] Training Set Menu
[2] Sample Pool Menu
[3] Validation Set Menu
[4] Active Learning

[r] Auto Run
[p] Per-Value Auto Run

[c] Custom PointsDirectory Menu
[a] Analysis Menu
[t] Tools Menu
[o] Options Menu
[q] Queue Menu

[0] Exit

>>
```

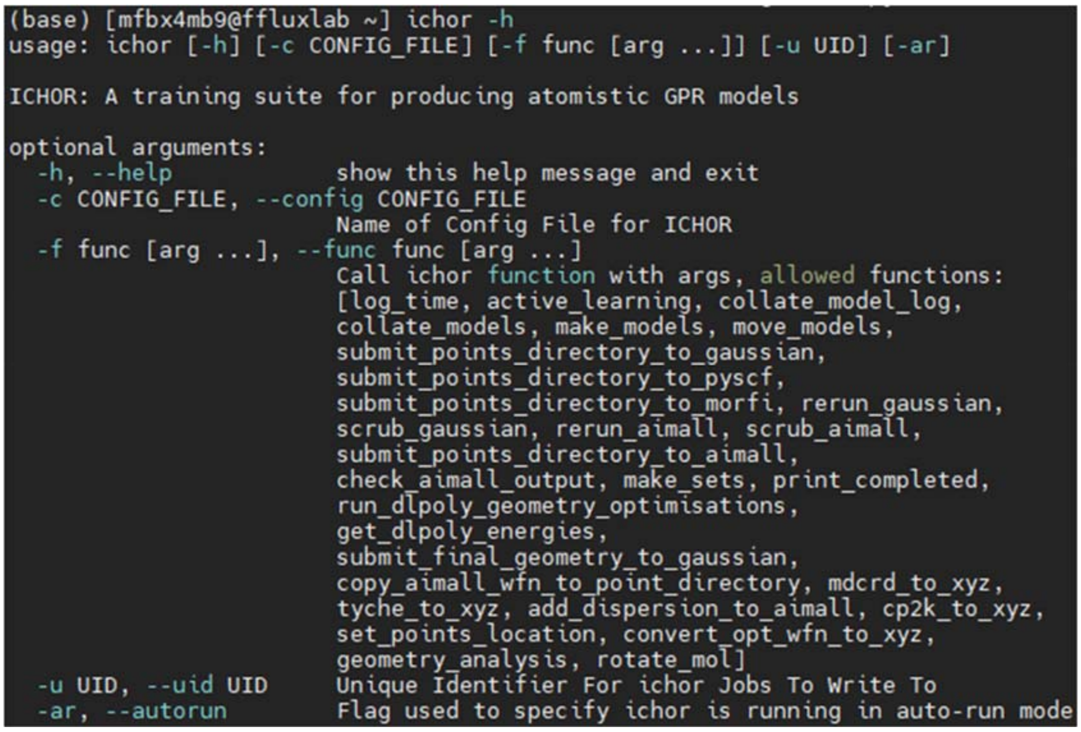
Figure S4. Screenshots showing (a) an example config file with errors in the spelling of N_ITERATIONS, and incorrect values for the AIMALL_BOAQ and POINTS_PER_ITERATION settings; (b) the example output from running ICHOR with the example config file.

Figure **S4** shows that, if ICHOR cannot find the specified parameter name, then the parameter in the config file is assumed to be misspelled and several suggestions are given via the *gestalt* pattern matching algorithm. Another common error is inputting an invalid value. Two cases are demonstrated in Figure **S4** whereby a parameter must be set to specified values only. For example, AIMALL_BOAQ cannot be set to the non-existing "gs70" of AIMAll, nor can the POINTS_PER_ITERATION parameter be negative. Other checks include that specified files exist, that they are formatted correctly, and that the shape of defined arrays are correct.

ICHOR's TUI is aimed at the general ICHOR user and is therefore designed to provide access to most of ICHOR's utilities in such a way that will reduce user error.

1.2 Command Line Interface (CLI)

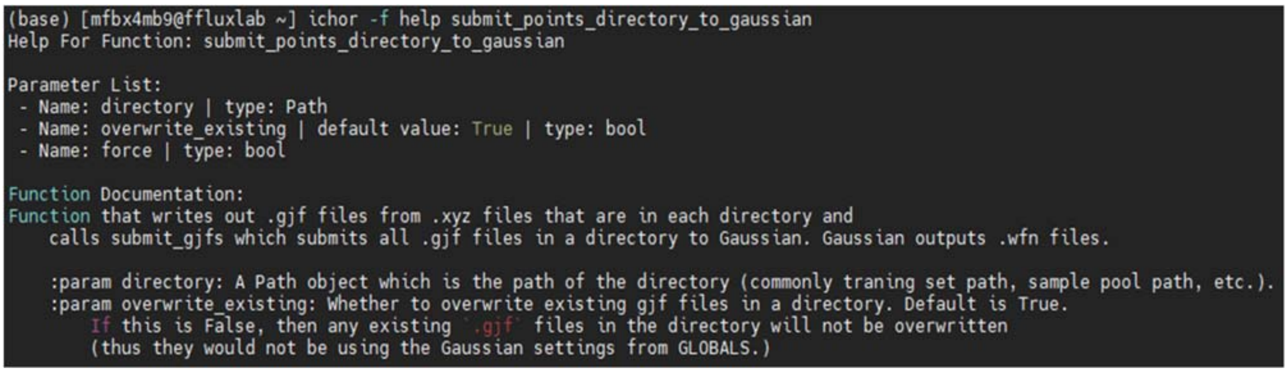
ICHOR's CLI is designed for a more advanced user who may require the use of a specific ICHOR function in a non-interactive shell. The main reason for using the ICHOR CLI is when running on a HPC cluster compute node where an interactive terminal session is often unavailable. It is often the case that such a node will be a non-interactive session and therefore the user is unable to navigate the various options in a TUI menu. All ICHOR CLI functions may be viewed via the help menu.



```
(base) [mfbx4mb9@ffluxlab ~] ichor -h
usage: ichor [-h] [-c CONFIG_FILE] [-f func [arg ...]] [-u UID] [-ar]

ICHOR: A training suite for producing atomistic GPR models

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG_FILE, --config CONFIG_FILE
                        Name of Config File for ICHOR
  -f func [arg ...], --func func [arg ...]
                        Call ichor function with args, allowed functions:
                        [log_time, active_learning, collate_model_log,
                        collate_models, make_models, move_models,
                        submit_points_directory_to_gaussian,
                        submit_points_directory_to_pyscf,
                        submit_points_directory_to_morfi, rerun_gaussian,
                        scrub_gaussian, rerun_aimall, scrub_aimall,
                        submit_points_directory_to_aimall,
                        check_aimall_output, make_sets, print_completed,
                        run_dlpoly_geometry_optimisations,
                        get_dlpoly_energies,
                        submit_final_geometry_to_gaussian,
                        copy_aimall_wfn_to_point_directory, mdcrd_to_xyz,
                        tyche_to_xyz, add_dispersion_to_aimall, cp2k_to_xyz,
                        set_points_location, convert_opt_wfn_to_xyz,
                        geometry_analysis, rotate_mol]
  -u UID, --uid UID     Unique Identifier For ichor Jobs To Write To
  -ar, --autorun        Flag used to specify ichor is running in auto-run mode
```

Figure S5. ICHOR help dialogue.

Most ICHOR options are instance-specific flags such as specifying a config file via the `-c` flag and specifying a unique instance id via the `-u` flag. Internally, ICHOR developers define external functions allowing the function to be called via the `-f` flag in the CLI. ICHOR provides a help utility for each external function providing a list of function parameters with the function's documentation, if available.



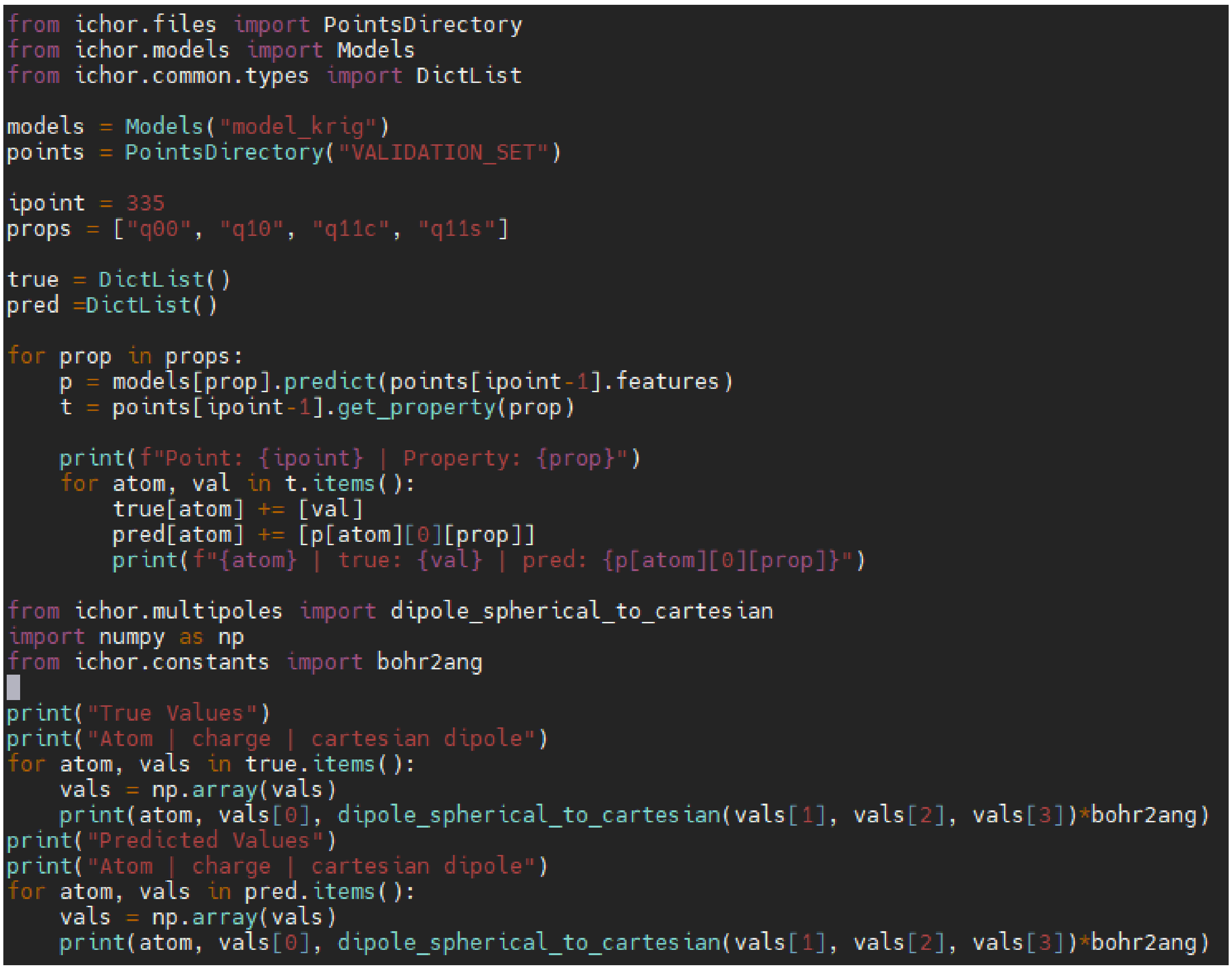
```
(base) [mfbx4mb9@ffluxlab ~] ichor -f help submit_points_directory_to_gaussian
Help For Function: submit_points_directory_to_gaussian

Parameter List:
 - Name: directory | type: Path
 - Name: overwrite_existing | default value: True | type: bool
 - Name: force | type: bool

Function Documentation:
Function that writes out .gjf files from .xyz files that are in each directory and
    calls submit_gjfs which submits all .gjf files in a directory to Gaussian. Gaussian outputs .wfn files.

    :param directory: A Path object which is the path of the directory (commonly traning set path, sample pool path, etc.).
    :param overwrite_existing: Whether to overwrite existing gjf files in a directory. Default is True.
        If this is False, then any existing `.gjf` files in the directory will not be overwritten
        (thus they would not be using the Gaussian settings from GLOBALS.)
```

Figure S6. Example ICHOR help dialogue for running the submit_points_directory_to_gaussian external function.

As mentioned previously, the CLI is designed for when an interactive TUI approach would not be appropriate. Users may request extra functionality to be made available via the CLI if the functionality is not yet accessible from the CLI and is not capable of interfacing ICHOR via the library interface.

1.3 Library Interface

ICHOR has been designed in such a way that it is accessible both as an application and as a library. Using `setuptools`, ICHOR is accessible via a `pip` installable python package. The library interface opens up all ICHOR functions and classes for developers to use freely allowing for easy automation of computational chemistry tasks if the specific functionality cannot be found by neither the TUI nor the CLI. The library interface is designed for the most advanced ICHOR users providing access to program interfaces, analysis and HPC automation tools allowing for high level control over common computational chemistry tasks. Figure **S7** shows an example of a script using the ICHOR library interface.



```python
from ichor.files import PointsDirectory
from ichor.models import Models
from ichor.common.types import DictList

models = Models("model_krig")
points = PointsDirectory("VALIDATION_SET")

ipoint = 335
props = ["q00", "q10", "q11c", "q11s"]

true = DictList()
pred =DictList()

for prop in props:
    p = models[prop].predict(points[ipoint-1].features)
    t = points[ipoint-1].get_property(prop)

    print(f"Point: {ipoint} | Property: {prop}")
    for atom, val in t.items():
        true[atom] += [val]
        pred[atom] += [p[atom][0][prop]]
        print(f"{atom} | true: {val} | pred: {p[atom][0][prop]}")

from ichor.multipoles import dipole_spherical_to_cartesian
import numpy as np
from ichor.constants import bohr2ang

print("True Values")
print("Atom | charge | cartesian dipole")
for atom, vals in true.items():
    vals = np.array(vals)
    print(atom, vals[0], dipole_spherical_to_cartesian(vals[1], vals[2], vals[3])*bohr2ang)
print("Predicted Values")
print("Atom | charge | cartesian dipole")
for atom, vals in pred.items():
    vals = np.array(vals)
    print(atom, vals[0], dipole_spherical_to_cartesian(vals[1], vals[2], vals[3])*bohr2ang)
```

Figure S7. Example script utilising the ICHOR library interface to check the true versus predicted values for the atomic charges and dipole moments of a specific point in a validation set before converting the dipole moments from the standard spherical representation to the Cartesian form, and converting the units of the dipole moments from eÅ to eBohr.

It is worth noting here that ICHOR has a sophisticated implementation of file and directory handling designed to spend as little time in I/O as possible. ICHOR is designed to efficiently work with a large volume of files, for example, when working with training or validation sets. When creating the variable `points`(Figure S7) it would be wasteful to read the entire validation set because the intent of the code is only to access the true values of a single point out of the validation set. This intent cannot be known ahead of time, which is why ICHOR implements lazy file reading whereby only the files that are necessary will be read, in this case the files containing the multipole moment data of point `335` (`ipoint` in Figure S7) and the geometry of point `335`. It is also true that accessing the multipole data is a computationally intensive task because multipoles are required to be converted and rotated as is described in a later section. For such files, ICHOR implements a caching function such that each time the property is requested, it need not be recalculated but instead read straight from the cache.

2 External Program Interfaces

2.1 Molecular Dynamics Software

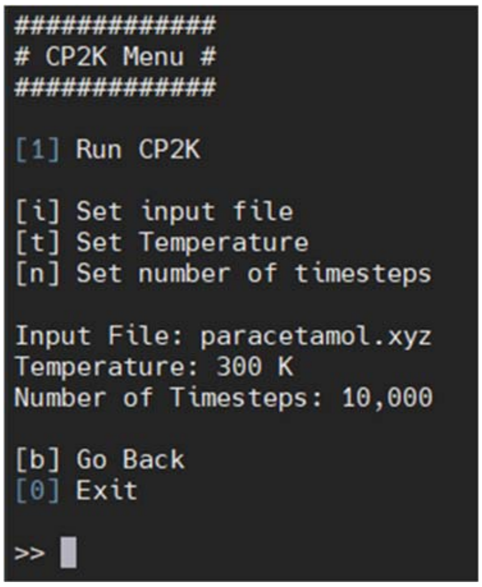
2.1.1 CP2K

ICHOR implements an interface to the *ab initio* molecular dynamics (AIMD) simulation package CP2K providing the user with simple tools to control the running of the package. CP2K is an open-source AIMD package implementing a wide variety of quantum chemistry (QC) methods.

Table S1. Parameters associated with the execution of a CP2K job.

Parameter	Description	Default Value
CP2K_NCORES	Number of cores to use to run CP2K	8
CP2K_INPUT	Input file to use for initial geometry	
CP2K_TEMPERATURE	Temperature to run CP2K at	300 K
CP2K_STEPS	How many timesteps to run CP2K for	10,000
CP2K_TIMESTEP	Timestep length in fs	1.0 fs
CP2K_METHOD	QC Method to use for AIMD simulation	BLYP
CP2K_BASIS_SET	Basis set to use for AIMD simulation	6-31G
CP2K_DATA_DIR	The location of the CP2K data directory containing data files such as the basis sets. ICHOR automatically finds this location if the machine is known.	

All parameters may be set via the config file, then edited and run using the CP2K submenu shown below in Figure S8. ICHOR then generates the CP2K input file and submission script before submitting to the HPC cluster to execute on a compute node, shown below in Code S1.



```
#############
# CP2K Menu #
#############

[1] Run CP2K

[i] Set input file
[t] Set Temperature
[n] Set number of timesteps

Input File: paracetamol.xyz
Temperature: 300 K
Number of Timesteps: 10,000

[b] Go Back
[0] Exit

>>
```

Figure S8. Example of a CP2K submenu.

Code S1. Example CP2K input file generated for a paracetamol 300 K AIMD simulation

```
&GLOBAL
  PROJECT PARACETAMOL
  RUN_TYPE MD

  IOLEVEL LOW
&END GLOBAL

&FORCE_EVAL
  METHOD Quickstep
  &DFT
    BASIS_SET_FILE_NAME /opt/cp2k/6.1.0/data/BASIS_SET
    POTENTIAL_FILE_NAME /opt/cp2k/6.1.0/data/GTH_POTENTIALS

    CHARGE 0
    MULTIPLICITY 1

    &MGRID
      CUTOFF [Ry] 400
    &END MGRID

    &QS
```

```
      METHOD GPW
      EPS_DEFAULT 1.0E-10
      EXTRAPOLATION ASPC
    &END QS

    &POISSON
      PERIODIC XYZ
      POISSON_SOLVER PERIODIC
    &END POISSON

    &SCF
      SCF_GUESS ATOMIC
      MAX_SCF 30
      EPS_SCF 1.0E-6
      &OT
        PRECONDITIONER FULL_SINGLE_INVERSE
        MINIMIZER DIIS
      &END OT
      &OUTER_SCF ! repeat the inner SCF cycle 10 times
        MAX_SCF 10
        EPS_SCF 1.0E-6 ! must match the above
      &END

      &PRINT
        &RESTART OFF
        &END
      &END PRINT
    &END SCF

    ! specify the exchange and correlation treatment
    &XC
      &XC_FUNCTIONAL BLYP
      &END XC_FUNCTIONAL
      ! adding Grimme's D3 correction (by default without C9 terms)
      &VDW_POTENTIAL OFF
        POTENTIAL_TYPE PAIR_POTENTIAL
        &PAIR_POTENTIAL
          PARAMETER_FILE_NAME /opt/cp2k/6.1.0/data/dftd3.dat
          TYPE DFTD3
          REFERENCE_FUNCTIONAL BLYP
          R_CUTOFF [angstrom] 16
        &END PAIR_POTENTIAL
      &END VDW_POTENTIAL
    &END XC
  &END DFT

  ! description of the system
  &SUBSYS
    &CELL
      ABC [angstrom] 25.0 25.0 25.0
    &END CELL

    ! atom coordinates can be in the &COORD section,
    ! or provided as an external file.
    &COORD
      C      -1.38560834       0.72661988      -0.00292676
      C      -2.52666419      -0.04742133      -0.23694580
      C      -2.41236928      -1.43117557      -0.29038093
      C      -1.18720485      -2.04836114      -0.07946452
      C      -0.04862960      -1.27341805       0.15841538
      C      -0.13407250       0.12044156       0.17620361
      H      -3.48252883       0.44418211      -0.38604681
      O      -3.48567490      -2.22483703      -0.56333636
      H      -1.10644072      -3.13096919      -0.12055897
      H       0.89627539      -1.79073203       0.29208601
      H      -1.48804768       1.80826856       0.02171883
      N       0.98753667       0.96801054       0.27825986
      C       2.30318451       0.58452382       0.46843260
      H       0.81371650       1.95621785       0.14798346
      O       2.70539466      -0.50846074       0.84759059
      C       3.26747056       1.67956983       0.09012606
      H       3.25502871       1.80133223      -0.99748830
      H       4.27853519       1.40340843       0.40187161
      H       2.99545989       2.62216457       0.57130942
      H      -4.24536121      -1.65936431      -0.77684901
    &END COORD

    ! keep atoms away from box borders,
    ! a requirement for the wavelet Poisson solver
    &TOPOLOGY
      &CENTER_COORDINATES
      &END
    &END TOPOLOGY

    &KIND C
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q4
    &END KIND
    &KIND C
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q4
    &END KIND
    &KIND C
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q4
    &END KIND
    &KIND C
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q4
    &END KIND
    &KIND C
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q4
    &END KIND
    &KIND C
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q4
    &END KIND
    &KIND H
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q1
    &END KIND
    &KIND O
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q6
    &END KIND
    &KIND H
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q1
    &END KIND
    &KIND H
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q1
    &END KIND
    &KIND H
      BASIS_SET 6-31G*
      POTENTIAL GTH-BLYP-q1
    &END KIND
    &KIND N
```

```
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q5
      &END KIND
      &KIND C
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q4
      &END KIND
      &KIND H
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q1
      &END KIND
      &KIND O
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q6
      &END KIND
      &KIND C
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q4
      &END KIND
      &KIND H
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q1
      &END KIND
      &KIND H
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q1
      &END KIND
      &KIND H
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q1
      &END KIND
      &KIND H
        BASIS_SET 6-31G*
        POTENTIAL GTH-BLYP-q1
      &END KIND
    &END SUBSYS
  &END FORCE_EVAL

  ! how to propagate the system, selection via RUN_TYPE in the &GLOBAL
  section
  &MOTION
    &GEO_OPT

      OPTIMIZER LBFGS ! Good choice for 'small' systems (use LBFGS for
  large systems)
      MAX_ITER  100
      MAX_DR    [bohr] 0.003 ! adjust target as needed
      &BFGS
      &END BFGS
    &END GEO_OPT
    &MD
      ENSEMBLE NVT  ! sampling the canonical ensemble, accurate
  properties might need NVE
      TEMPERATURE [K] 300
      TIMESTEP [fs] 1.0
      STEPS 10000
      &THERMOSTAT
        TYPE NOSE
        REGION GLOBAL
        &NOSE
          TIMECON 50.
        &END NOSE
      &END THERMOSTAT
    &END MD

    &PRINT
      &TRAJECTORY
        &EACH
          MD 1
        &END EACH
      &END TRAJECTORY
      &VELOCITIES OFF
      &END VELOCITIES
      &FORCES OFF
      &END FORCES
      &RESTART OFF
      &END RESTART
      &RESTART_HISTORY OFF
      &END RESTART_HISTORY
    &END PRINT
  &END MOTION
```

Providing an interface to generate the CP2K input enables inexperienced CP2K users to easily set up a simulation and be productive without much background knowledge necessary. ICHOR automatically defines each atom type for the geometry from the atomic numbers, QC method and basis set defined by the user.

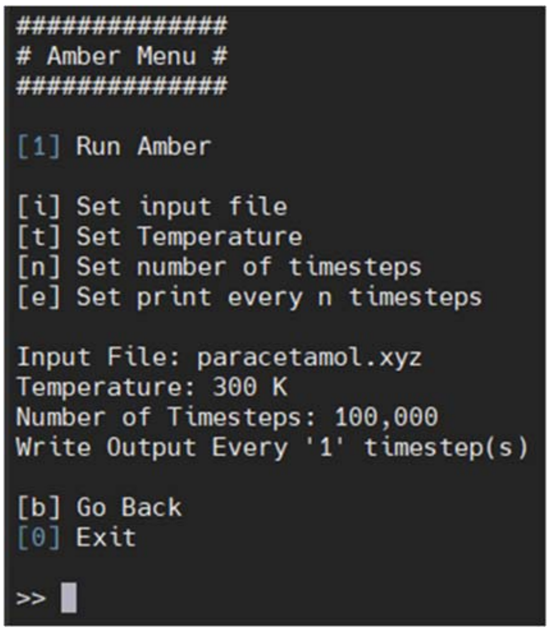
2.1.2 AMBER

For larger simulations AIMD techniques become unfeasible thus cajoling one to use classical MD techniques. ICHOR implements an AMBER interface, similar to the CP2K interface shown in the previous section.

Table S2. Parameters used to control AMBER simulations in ICHOR.

Parameter	**Description**	**Default Value**
AMBER_NCORES	Number of cores to use to run AMBER	1
AMBER_TEMPERATURE	Temperature to run AMBER at	300 K
AMBER_STEPS	How many timesteps to run AMBER for	100,000
AMBER_TIMESTEP	Timestep length in ps	0.001 ps
AMBER_LN_GAMMA	Log-Gamma parameter used for the Langevin thermostat	0.7

Much like CP2K, AMBER may be accessed and executed from the MD tools AMBER submenu.



```
##############
# Amber Menu #
##############

[1] Run Amber

[i] Set input file
[t] Set Temperature
[n] Set number of timesteps
[e] Set print every n timesteps

Input File: paracetamol.xyz
Temperature: 300 K
Number of Timesteps: 100,000
Write Output Every '1' timestep(s)

[b] Go Back
[0] Exit

>>
```

Figure S9. Example AMBER submenu.

Running a classical MD simulation is a little more involved than an AIMD simulation because the atom type selection is defined not only by atomic number but also by atomic environment. ICHOR implements a method of generating a .mol2 file for any given geometry, which determines automatically atom and bond types defined by the SYBYL MOL2 format specification.

Code S2. Example MOL2 paracetamol file generated by ICHOR from initial geometry

```
@<TRIPOS>MOLECULE
paracetamol
   20    20     1     0     0
SMALL
none

@<TRIPOS>ATOM
      1 C   -2.1990     3.2770     0.6990 ca        1 para    0.000000
      2 C1  -3.3400     2.5030     0.4650 ca        1 para    0.000000
      3 C2  -3.2260     1.1190     0.4120 ca        1 para    0.000000
      4 C3  -2.0010     0.5020     0.6230 ca        1 para    0.000000
      5 C4  -0.8620     1.2770     0.8600 ca        1 para    0.000000
      6 C5  -0.9480     2.6710     0.8780 ca        1 para    0.000000
      7 H   -4.2960     2.9950     0.3160 ha        1 para    0.000000
      8 O   -4.2990     0.3250     0.1390 oh        1 para    0.000000
      9 H1  -1.9200    -0.5810     0.5810 ha        1 para    0.000000
     10 H2   0.0830     0.7600     0.9940 ha        1 para    0.000000
     11 H3  -2.3020     4.3590     0.7240 ha        1 para    0.000000
     12 N    0.1740     3.5180     0.9800 ns        1 para    0.000000
     13 C6   1.4900     3.1350     1.1700 c         1 para    0.000000
     14 H4   0.0000     4.5070     0.8500 hn        1 para    0.000000
     15 O1   1.8920     2.0420     1.5500 o         1 para    0.000000
     16 C7   2.4540     4.2300     0.7920 c3        1 para    0.000000
     17 H5   2.4410     4.3520    -0.2960 hc        1 para    0.000000
     18 H6   3.4650     3.9540     1.1040 hc        1 para    0.000000
     19 H7   2.1820     5.1720     1.2730 hc        1 para    0.000000
     20 H8  -5.0590     0.8910    -0.0750 ho        1 para    0.000000
@<TRIPOS>BOND
     1     1     2 ar
     2     1     6 ar
     3     1    11 1
     4     2     3 ar
     5     2     7 1
     6     3     4 ar
     7     3     8 1
     8     4     5 ar
     9     4     9 1
    10     5     6 ar
    11     5    10 1
    12     6    12 1
    13     8    20 1
    14    12    13 1
    15    12    14 1
```

```
     16     13     15 2
     17     13     16 1
     18     16     17 1
     19     16     18 1
     20     16     19 1
```

With the MOL2 file, ICHOR generates a tleap and md.in file in order to generate the necessary files to run AMBER.

Code S3. Example tleap file for paracetamol input

```
source leaprc.gaff2
mol = loadmol2 PARACETAMOL.mol2
loadamberparams PARACETAMOL.frcmod
saveamberparm mol PARACETAMOL.prmtop PARACETAMOL.inpcrd
quit
```

Code S4. Example MD input file for production AMBER run.

```
Production
 &cntrl
  imin=0,
  ntx=1,
  irest=0,
  nstlim=100000,
  dt=0.001,
  ntf=1,
  ntc=1,
  temp0=300,
  ntpr=1,
  ntwx=1,
  ntwv=0,
  ntwf=0,
  ioutfm=0,
  cut=999.0,
  ntb=0,
  ntp=0,
  ntt=3,
  gamma_ln=0.7,
  tempi=0.0,
  ig=-1
 /
```

ICHOR defaults to running AMBER as a single molecule in vacuum using GAFF parameters resulting in the absenceof periodic boundary conditions and a large cutoff value. ICHOR's AMBER interface is aimed at novice AMBER users to provide the tools to run a simple MD simulations. The outputs from more advanced tailored AMBER simulations may also be used by ICHOR through the `mdcrd_to_trajectory` utility function in the CLI.

2.1.3 TYCHE

Although not MD simulation software, TYCHE is the final external program used by ICHOR to generate input coordinates. TYCHE samples the normal modes of a system through the derivative of the molecular wavefunction and then performs Boltzmann sampling from each normal mode to produce molecular configurations. Similarly to the MD methods, the molecular distortions of the TYCHE output may be controlled via a temperature parameter.

Table S3. Parameters used to run Tyche.

Parameter	Description	Default Value
TYCHE_NCORES	Number of cores to use to run TYCHE	1
TYCHE_TEMPERATURE	Temperature to run TYCHE at	300 K
TYCHE_STEPS	How many timesteps to run TYCHE for	10,000
TYCHE_LOCATION	The location of the TYCHE executable, this parameter is assigned automatically if ICHOR is able to identify the machine	

In order to run TYCHE, three files are required: a wavefunction file, a wavefunction (second) derivatives file and a TYCHE parameter file called `freq.param`. The wavefunction and derivative files are generated by running the geometry through a GAUSSIAN frequency calculation with the `output=wfn` and `punch=derivatives` keywords. The third file (`freq.param`) specifies the control parameters to run TYCHE.

Code S5. Example freq.param input file for a TYCHE paracetamol run

```
Sampling                  : Vibrate
Name                      : PARACETAMOL
Temperature               : 450
Nseed                     : 1
Nsample                   : 9999
Natoms                    : 20
Nsteps                    : 100
GJF                       : nosymm
Interactions              : 1-4
WhichModes                : All
MaxModes                  : 100
MCTemp                    : 300
```

2.2 Quantum Chemistry Program Interfaces

Because ICHOR produces atomistic GPR models using quantum chemical topology (QCT) data, a wavefunction is required to pass onto the QCT program that performs the atomic partitioning (AIMALL or MORFI). A quantum chemistry (QC) program calculates the wavefunction of a geometry. At the time of writing, ICHOR interfaces with two such QC programs: GAUSSIAN and PySCF (via the PANDORA interface).

2.2.1 GAUSSIAN

GAUSSIAN is the primary QC program used by ICHOR to produce wavefunctions. To run a calculation through GAUSSIAN, a GAUSSIAN input file (`.gjf`) is required. For consistent models, it is necessary that all QC calculations are identical in method. ICHOR ensures this requirement by rewriting each input file with the specified GAUSSIAN parameters.

Table S4. Parameters used to generate Gaussian input files.

Parameter	Description	Default Value
GAUSSIAN_NCORES	Number of cores to use to run GAUSSIAN	2
GAUSSIAN_MEMORY_LIMIT	Memory limit for Gaussian jobs	1GB
METHOD	QC method used to run Gaussian jobs	B3LYP
BASIS_SET	Basis set used to run Gaussian jobs	6-31+G(d,p)
KEYWORDS	The keywords used to run Gaussian jobs	nosymm

GAUSSIAN input files are then written in a standard format ensuring a wavefunction will be outputted in the correct location.

Code S6. Example Gaussian input file for a paracetamol geometry

```
%nproc=2
%mem=1GB
#p B3LYP/6-31+g(d,p) nosymm output=wfn

PARACETAMOL0001

0 1
C       0.11335000     -1.23865000      0.87480000
C       0.85535000     -1.22565000      2.05680000
C       1.17835000      0.03635000      2.63680000
```

```
C        0.93835000      1.27235000      1.95980000
C        0.28435000      1.20635000      0.69580000
C       -0.09765000     -0.04565000      0.13580000
H        1.13535000     -2.09865000      2.60580000
O        1.77135000      0.02835000      3.89480000
H        1.26735000      2.21335000      2.30380000
H        0.07535000      2.12335000      0.12080000
H       -0.29565000     -2.15465000      0.51080000
N       -0.66565000     -0.15765000     -1.17120000
C       -1.00165000      0.69735000     -2.19820000
H       -0.76565000     -1.05365000     -1.39520000
O       -0.94765000      1.91235000     -2.12820000
C       -1.42965000     -0.02765000     -3.52420000
H       -0.67265000      0.31235000     -4.27520000
H       -2.47265000      0.20035000     -3.71420000
H       -1.39965000     -1.12465000     -3.42020000
H        2.12935000     -0.87565000      4.03080000

TRAINING_SET/PARACETAMOL0001/PARACETAMOL0001.wfn
```

2.2.2 PySCF

It is not practical to retrieve the two-particle-density matrix (2PDM) from GAUSSIAN. However, the 2PDM is retrieved from PySCF, which then enables the calculation of atomic properties related to dispersion. ICHOR interfaces with PySCF via an in-house software package named PANDORA, which controls the execution of PySCF and that of the in-house QCT program MORFI. The interface to PANDORA is similar to the interface of GAUSSIAN whereby PANDORA is run via an input file and controlled via parameters.

Table S5. Parameters used to control PySCF via PANDORA.

Parameter	Description	Default Value
PYSCF_NCORES	Number of cores to use to run PySCF	2
METHOD	PySCF QC Method	CCSD
KEYWORDS	PySCF Basis Set	6-31G

Code S7. Example of a PANDORA input file for paracetamol

```
{
    "system": {
        "name": "paracetamol",
        "geometry": [
            ["C",  0.11335000, -1.23865000,  0.87480000],
            ["C",  0.85535000, -1.22565000,  2.05680000],
            ["C",  1.17835000,  0.03635000,  2.63680000],
            ["C",  0.93835000,  1.27235000,  1.95980000],
            ["C",  0.28435000,  1.20635000,  0.69580000],
            ["C", -0.09765000, -0.04565000,  0.13580000],
            ["H",  1.13535000, -2.09865000,  2.60580000],
            ["O",  1.77135000,  0.02835000,  3.89480000],
            ["H",  1.26735000,  2.21335000,  2.30380000],
            ["H",  0.07535000,  2.12335000,  0.12080000],
            ["H", -0.29565000, -2.15465000,  0.51080000],
            ["N", -0.66565000, -0.15765000, -1.17120000],
            ["C", -1.00165000,  0.69735000, -2.19820000],
            ["H", -0.76565000, -1.05365000, -1.39520000],
            ["O", -0.94765000,  1.91235000, -2.12820000],
            ["C", -1.42965000, -0.02765000, -3.52420000],
            ["H", -0.67265000,  0.31235000, -4.27520000],
            ["H", -2.47265000,  0.20035000, -3.71420000],
            ["H", -1.39965000, -1.12465000, -3.42020000],
            ["H",  2.12935000, -0.87565000,  4.03080000]
        ]
    },
    "pandora": {
        "ccsdmod": "ccsdM"
    },
    "pyscf": {
        "method": "ccsd",
        "basis": "unc-6-31g"
    },
    "morfi": {
        "grid": {
            "radial": 10.0,
            "angular": 5,
            "radial_h": 8.0,
            "angular_h": 5
        }
    }
}
```

2.3 Quantum Chemical Topology Interfaces

To calculate the atomic properties to input to the GPR model, the wavefunction must be partitioned into quantum topological atomic and their properties computed by a quantum chemical topology (QCT) program. To accompany the two QC programs shown in the previous section, ICHOR interfaces with two QCT programs: AIMAll and the in-house QCT program MORFI (via the PANDORA Interface).

2.3.1 AIMAll

AIMAll is the primary route for partitioning wavefunctions into quantum topological atoms. At the time of writing, it calculates all atomic properties except for the correlation (dispersion) energies (which is calculated using MORFI). AIMAll is a powerful program that takes a wavefunction outputted from GAUSSIAN or PySCF in the PROAIM format and partitions the wavefunction into atomic properties. Similarly to the other external programs ICHOR that interfaces with, the execution of AIMAll is controlled using global parameters. AIMAll is a mature, well developed QCT software package, which requires no extra input files from ICHOR other than the specification of certain control parameters in the wavefunction file. The execution of AIMAll is performed through command line parameters controlled by the parameters shown in Table S6. ICHOR then reads the atomic properties from the AIMAll integration output files (`.int`) providing the atomic properties for use within ICHOR, to employ for the creation of models and for model analysis.

Table S6. Parameters used to control the execution of AIMAll.

Parameter	Description	Default Value
AIMALL_NCORES	Number of cores to use to run AIMAll	2
AIMALL_ENCOMP	Determines which energy components to calculate	3
AIMALL_BOAQ	Sets basin quadrature to use during integration	gs20
AIMALL_IASMESH	Inter atomic surface mesh used during integration	fine
AIMALL_BIM	Basin integration method	auto
AIMALL_CAPTURE	Determines Promega capture method	auto
AIMALL_EHREN	Determines how to calculate Ehrenfest Forces	0
AIMALL_FEYNMAN	Determines whether to calculate Feynman Forces	False

AIMALL_IASPROPS	Specifies whether to calculate inter atomic surface properties	True
AIMALL_MAGPROPS	Determines method to use to calculate magnetic properties	none
AIMALL_SOURCE	Determines whether to calculate atomic source contributions	False
AIMALL_IASWRITE	Specifies whether to write inter atomic surface files	False
AIMALL_ATIDSPROPS	Controls calculation of atomic isodensity surface properties	0.001
AIMALL_WARN	Output AIMAll warning messages	True
AIMALL_SCP	Determines how much progress is outputted to the console	some
AIMALL_DELMOG	Specifies whether to delete the aomtic mog files	True
AIMALL_SKIPINT	Specifies whether to skip[atomic integrations	False
AIMALL_F2W	Controls output format when converting formatted check file to wavefunction	wfx
AIMALL_F2WONLY	Controls whether to continue calculation after converting formatted check file	False
AIMALL_MIR	Sets maximum atomic integration radius (negative value indicates auto determination)	-1.0
AIMALL_CPCONN	Controls the intensity of the critical point connectivity search	moderate
AIMALL_INTVEEAA	Controls which algorithm to use for Vee(A,A) calculations	new
AIMALL_ATLAPRHOCPS	Controls whether to locate and characterise critical points of the laplacian of Rho	False

AIMALL_WSP	Determines whether to write molecular graph	True
AIMALL_SHM	Controls which spherical harmonic moments to output	5
AIMALL_MAXMEM	Maximum memory (MB) that may be used by AIMAll	2400
AIMALL_VERIFYW	Controls whether to perform verification on inputted wavefunction	yes
AIMALL_SAW	Show atomic windows	False
AIMALL_AUTONNACPS	Controls automatic critical point search	True

2.3.2 MORFI

The dispersion energy (i.e. correlation energy from CCSD) requires the use of the in-house QCT program MORFI. Much like PySCF, MORFI is interfaced via the in-house software package PANDORA due to the complexity of setting up a MORFI calculation. MORFI is a parallellised FORTRAN code requiring recompilation for each specific MORFI run. This setup and compilation is handled by PANDORA and controlled by ICHOR through the use of the PANDORA input file shown in Code S7 and a set of control parameters.

Table S7. Parameters used for running MORFI via the PANDORA interface.

Parameter	Description	Default Value
MORFI_NCORES	Number of cores to use to run MORFI	4
PANDORA_CCSDMOD	Specifies the CCSD dispersion correction modification function	ccsdM
MORFI_RADIAL	Sets the number of radial points to use in the atomic integration	10.0
MORFI_ANGULAR	Sets the number of angular points to use in the atomic integration	5
MORFI_RADIAL_H	Sets the number of radial points to use for hydrogen atoms in the atomic integration	5
MORFI_ANGULAR_H	Sets the number of angular points to use for hydrogen atoms in the atomic integration	8.0

ICHOR then reads the MORFI output file (`.mout`), allowing ICHOR to access the atomic integration results and dispersion energy terms for combination with the AIMAll atomic properties. When computing the IQA energy from a CCSD wavefunction using AIMAll, the Müller dispersion correction is applied, which needs to be accounted for when computing the dispersion energy with MORFI. Removing the Hartree Fock energy contribution of the CCSD energy results in a smaller energy. The integration of this smaller energy is accurate using only a (very) small integration grid. This advantage drastically decreases the amount of time it takes to run MORFI. Both options (ccsdHF and ccsdM) are provided to ICHOR through the PANDORA_CCSDMOD parameter. Table S8 summarises .

Table S8. PANDORA_CCSDMOD parameter options.

Parameter	Description
ccsd	No modification to the CCSD energy
ccsdHF	CCSD energy with the Hartree Fock energy contribution removed
ccsdM	CCSD energy with the Hartree Fock and Müller correction contributions removed

2.4 FEREBUS

ICHOR uses an external program in order to perform Gaussian process regression (GPR). ICHOR allows for easy replacement of the GPR implementation without having to modify the ICHOR source. The GPR engine of choice is currently the in-house program FEREBUS. FEREBUS defines a standard input and output file format to ensure that, regardless of the GPR implementation, ICHOR will be able to both produce models with the GPR engine and interpret the models produced by it.

FEREBUS requires two input files: a training set and an input configuration file. The training set file is a comma-separated value (`.csv`) file consisting of input features and output properties with a header to label each value type. The FEREBUS configuration file uses the TOML file format to define the parameters used to optimise the GPR model.

Code S8. Example of a FEREBUS configuration file

```
[system]
name = "METHANOL"
natoms = 6
atoms = [
  {name="C1", alf=[1, 2, 3]}
]
properties = ["iqa"]

```

```
[model]
mean = "constant"
optimiser = "pso"
kernel = "k1*k2"
likelihood = "marginal"

[optimiser]
search_min = 0.0
search_max = 3.0

[optimiser.pso]
swarm_size = 50
iterations = 1000
inertia_weight = 0.729
cognitive_learning_rate = 1.494
social_learning_rate = 1.494
stopping_criteria="relative_change"

[optimiser.pso.relative_change]
tolerance=1e-08
stall_iterations=50

[kernels.k1]
type = "rbf"
active_dimensions = [1, 2, 3, 4, 5, 7, 8, 10, 11]

[kernels.k2]
type = "periodic"
active_dimensions = [6, 9, 12]
```

The TOML file format provides an unambiguous, simple format for specifying execution parameters. Although there exists a Python implementation for serializing and deserialising the TOML file format, one of ICHOR's design requirements is to be highly portable and therefore have the fewest dependencies as possible. At the time of writing, the usage of the TOML format in ICHOR only extends to the serialisation of the FEREBUS config parameters as seen above in Code S8. Therefore the use of the standard implementation for serialising and deserialising TOML is not necessary. In other words, ICHOR does not need to implement the entire TOML specification and therefore does not need an additional dependency to do so.

Much like other external programs, the user is able to specify the FEREBUS optimisation settings via a set of control parameters.

Table S9. Parameters used to control the execution of FEREBUS.

Parameter	Description	Default Value
FEREBUS_NCORES	Number of cores to use to run FEREBUS	4
FEREBUS_SWARM_SIZE	Number of particles in PSO swarm	50
FEREBUS_NUGGET	Sets initial nugget parameter	1.0×10^{-10}
FEREBUS_THETA_MIN	Minimum value for theta initialisation	0.0
FEREBUS_THETA_MAX	Maximum value for theta initialisation	5
FEREBUS_INERTIA_WEIGHT	Specifies inertia weight for PSO	0.729
FEREBUS_COGNITIVE_LEARNING_RATE	Specifies cognitive learning rate for PSO	1.494
FEREBUS_SOCIAL_LEARNING_RATE	Specifies social learning rate for PSO	1.494
FEREBUS_MEAN	Specifies mean function to use	constant
FEREBUS_OPTIMISATION	Specifies optimisation method to use	PSO
FEREBUS_TOLERANCE	Specifies tolerance parameter used in PSO relative change stopping criterion	1.0×10^{-8}
FEREBUS_STALL_ITERATIONS	Specifies the number of stall itertations used in PSO relative change stopping criterion	50
FEREBUS_MAX_ITERATION	Specifies the maximum number of PSO iterations	1000

FEREBUS outputs a model file detailing the exact parameters and data used to generate the model as well as the optimised hyperparameters to be used for predictions using the model.

Code S9. Example of a FEREBUS model file.

```
# [metadata]
# program ferebus
# version 7.2.0
# nugget .1000000000000000E-09
# likelihood 348.9145663746718
# method = B3LYP
# basis-set = 6-31+g(d,p)

[system]
name METHANOL
atom C1
property iqa
ALF 1 3 2

[dimensions]
number_of_atoms 6
number_of_features 12
number_of_training_points 10

[mean]
type constant
value -38.04872095788889

[kernels]
number_of_kernels 1
composition k1*k2

[kernel.k1]
type rbf
number_of_dimensions 9
active_dimensions 1 2 3 4 5 7 8 10 11
lengthscale 0.32096237 0.18393264 0.99397568 0.5876017 0.48725134
0.90065895 0.46908177 0.4205679 0.75621913

[kernel.k2]
type rbf
number_of_dimensions 3
active_dimensions 6 9 12
lengthscale 0.14463587 0.02498353 0.43328024


[training_data]
```

```
units.x bohr bohr radians bohr radians radians bohr radians radians
bohr radians radians
units.y Ha
scaling.x none
scaling.y none

[training_data.x]
2.85064 2.04588 2.03424 2.09898 2.55388 -1.82680 2.12218 .64684 -
2.11206 3.85470 1.09976 -.00321
2.70755 2.02730 1.94060 2.14570 2.52306 -2.08213 2.15295 .55709 -
2.25723 3.82563 1.09486 -.04654
2.79929 2.11869 1.92634 2.13477 2.67195 -2.19195 2.07375 .71118 -
2.16350 3.75194 1.09568 .125862
2.73129 2.09673 1.98250 2.06029 2.59495 -2.13244 2.06885 .64671 -
2.29471 3.68825 1.12487 .277844
2.73095 2.09482 2.06995 2.06514 2.51628 -2.17765 1.94543 .70285 -
2.12713 3.62280 1.05668 .057452
2.69062 2.12511 1.97101 2.07515 2.61698 -2.08392 2.29233 .64708 -
2.39057 3.78873 1.15716 .280104
2.70905 2.08496 2.17988 2.08824 2.53619 -2.00574 2.02610 .67640 -
2.21704 3.93721 1.23008 .330925
2.73706 2.08279 1.85823 2.07297 2.74731 -2.13072 2.15101 .76038 -
2.24995 3.73374 1.05744 .047668
2.77127 2.17498 1.97612 2.10017 2.65806 -2.18309 2.05554 .79270 -
2.20301 3.62161 1.02148 .004557
2.72609 2.16825 1.88322 2.09059 2.54924 -2.28730 2.08369 .65725 -
2.23212 3.58900 1.45655 .541989

[training_data.y]
-38.05544
-38.04779
-38.05431
-38.04663
-38.03974
-38.04674
-38.04144
-38.05264
-38.05041
-38.04711

[weights]
-2594.2461
-7982.0474
```

```
-127.94520
-776.37281
57.2102186
-11.711176
169.110913
11088.8747
178.083364
```

ICHOR is required to interpret this model file in full in order to perform model analysis and active learning. Full details of model implementation details are found in Section 6.1.

3 High Performance Computing Cluster Interface

ICHOR is often required to run thousands of tasks simultaneously when producing models. For example, when calculating the atomic properties for each training point in a training set, there may be thousands of tasks, each taking several hours to complete. Running such computational tasks on a single machine would take an extraordinarily long time. Add in the requirement of per-atom active learning where the same computation is repeated for each atom in a system and it is obvious that it is necessary to parallelise the process. There are two forms of scaling the amount of computation ICHOR can perform: vertical and horizontal. Vertical scaling is used by ICHOR on a per-task basis; for example, multiple cores may be used by FEREBUS when training a model as a method of speeding up the production of a single model. Horizontal scaling involves running many tasks concurrently. Using the same FEREBUS example, ICHOR is able to execute many FEREBUS instances simultaneously to produce many models at the same time. The combination of horizontal and vertical scaling strategies is key to ICHOR's success.

Vertical scaling is a handled on a task-by-task basis and controlled by ICHOR using parameters set by the user. Each task is in control of how well the number of cores assigned to the task are used. For instance, AIMAll utilises multiple cores much less effectively than FEREBUS. Hence the lower default value of the NCORES parameter for AIMAll compared to FEREBUS as can be seen from Tables S6 and S9. Therefore, instead of relying on vertical scaling for less parallelisable tasks such as AIMAll, horizontal scaling strategies are employed as an alternative.

ICHOR is designed as a software package to make the best use of HPC clustering systems. HPC clustering systems provide a method of running tasks across many compute nodes simultaneously. ICHOR makes use of the ability to submit jobs to many compute nodes to provide horizontal scaling whilst making GPR models. When calculating the atomic properties for each point in the training set, many thousands of GAUSSIAN and AIMAll tasks may be required. Such tasks are well suited for horizontal scaling because all tasks may be submitted to the HPC cluster at once, and executed in a highly parallel environment.

To make full use of HPC clusters, ICHOR implements several tools and interfaces to perform such high-throughput computing efficiently and reliably.

3.1 Batch System Interface

ICHOR currently implements an interface to two cluster job schedulers: Sun Grid Engine (SGE) and Simple Linux Utility for Resource Management (SLURM). ICHOR is scheduler-agnostic whereby the scheduler used does not affect how ICHOR handles jobs internally but instead provides an interface layer that handles the execution of certain common tasks:

- `Submission script syntax`
- `Submitting jobs to the queueing system`
- `Monitoring currently running jobs`
- `Modifying/Deleting currently running jobs`

3.2 Submission Script

One of the more complex parts of the ICHOR codebase is the implementation of the creating of submission scripts. ICHOR's submission scripts are batch-system-agnostic, machine-agnostic and are able to efficiently chain multiple commands to execute in a highly parallel manner.

Code S10. Example of a Gaussian SGE submission script

```bash
#!/bin/bash -l
#$ -wd /work/gaussian
#$ -o /work/gaussian/.DATA/SCRIPTS/OUTPUTS
#$ -e /work/gaussian/.DATA/SCRIPTS/ERRORS
#$ -pe smp 2
#$ -t 1-10

export OMP_NUM_THREADS=$NSLOTS

module load apps/gaussian/g09

ICHOR_DATFILE=/work/gaussian/.DATA/JOBS/DATAFILES/gaussian_datafile

arr1=()
arr2=()
while IFS=, read -r var1 var2
do
    arr1+=($var1)
    arr2+=($var2)
done < $ICHOR_DATFILE

ICHOR_N_TRIES=0
export ICHOR_TASK_COMPLETED=false
while [ "$ICHOR_TASK_COMPLETED" == false ]
do
    g09 ${arr1[$SGE_TASK_ID-1]} ${arr2[$SGE_TASK_ID-1]}
```

```
27      let ICHOR_N_TRIES++
28      if [ "$ICHOR_N_TRIES" == 10 ]
29      then
30          break
31      fi
32      eval $(ichor -f check_gaussian_output "${arr1[$SGE_TASK_ID-1]}")
33  done
```

Let us work through the example submission script above (Code S9) line by line. Starting at line 2, ICHOR ensures that the working directory of the script is set explicitly, thereby eliminating any issues from submitting a script in a different location to which it is to be executed. Lines 3 and 4 set the stdout and stderr locations, which is useful for monitoring and debugging the execution of jobs. Line 5 sets the number of cores to assign to each task in the job. This line is both specific to the batch system and to the machine because ICHOR must select the suitable parallel environment given the requested number of cores. More details on parallel environments are found in Section 4.3. Line 6 specifies that the job is to be run as a so-called task array. A task array is an efficient method for running multiple tasks from a single job script. Here ICHOR is requesting a task array of length 10 making clear that the job is to run 10 GAUSSIAN calculations.

Line 8 of the submission script ensures that OpenMP applications use only the number of cores that have been allocated by the batch system. Line 10 is another machine-specific line as the system configuration defines how particular applications are to be run. In this instance, environment modules are used to set up the execution environment. Environment modules are discussed in more detail in Section 4.1. Lines 12 through 20 involve the parsing of an ICHOR datafile into arrays to be used as program inputs. A datafile is used by ICHOR as a flexible method of passing information on which tasks to complete by a particular job. More details on data files are in Section 3.3.

The remainder of the lines (22 to 33) perform the execution of the command, which is the GAUSSIAN command in this case. The actual execution of the command is performed on line 26 with the rest of the lines providing error handling functionality. The error handling is implemented to improve reliability and provide flexibility to the developer, which is explained in more detail in Section 3.4.

3.3 Datafiles

One of the most important sections of ICHOR's submission script is the ICHOR datafile. The datafile provides ICHOR with the flexibility of specifying only a file to write to, rather than writing a set of commands to execute ahead of time. This affords ICHOR the flexibility to change which commands to execute and the number of commands to be executed rather than the alternative of explicitly writing each command in the submission script ahead of time. The datafile is a simple text file with command arguments separated by a delimiter, which is designed to be easily read by the submission script during execution.

Code S11. Example of a GAUSSIAN datafile

```
TRAINING_SET/METHANOL0001.gjf,TRAINING_SET/METHANOL0001.g09
```

```
TRAINING_SET/METHANOL0002.gjf,TRAINING_SET/METHANOL0002.g09
TRAINING_SET/METHANOL0003.gjf,TRAINING_SET/METHANOL0003.g09
TRAINING_SET/METHANOL0004.gjf,TRAINING_SET/METHANOL0004.g09
TRAINING_SET/METHANOL0005.gjf,TRAINING_SET/METHANOL0005.g09
TRAINING_SET/METHANOL0006.gjf,TRAINING_SET/METHANOL0006.g09
TRAINING_SET/METHANOL0007.gjf,TRAINING_SET/METHANOL0007.g09
TRAINING_SET/METHANOL0008.gjf,TRAINING_SET/METHANOL0008.g09
TRAINING_SET/METHANOL0009.gjf,TRAINING_SET/METHANOL0009.g09
TRAINING_SET/METHANOL0010.gjf,TRAINING_SET/METHANOL0010.g09
```

From line 16 in Code S10 it can be seen that the datafile shown in Code S11 will be parsed and each line split by the comma delimiter. This creates two arrays: one for the GAUSSIAN input file and one for the GAUSSIAN output file. Subsequently the array is indexed using the task id of the current job, as can be seen on line 26.

The datafile format is designed with flexibility in mind. Each command is in control of how many arguments may be parsed from the datafile, while the submission script controls how many tasks must be written to the datafile. At execution the submission script is responsible for reading the required number of arguments and indexing the arguments from the parsed arrays to execute the specific task. This flexibility also allows the opportunity for running a number of tasks different to that allocated when initially writing the submission script. For example, if the number of jobs was overestimated when writing the initial script, the script will read in fewer tasks than allocated. When indexing the array, the job will then fail. On the other hand, if the number of tasks was underestimated, previous statically defined submission scripts would be unable to execute the requested number of jobs, thereby leaving ICHOR in an error-state. The datafile method allows for the reindexing of the argument arrays if ICHOR detects that the number of jobs allocated is fewer than the ones written in the datafile, which is handled by the error handling processes described in the next section.

3.4 Error Handling

The HPC clusters ICHOR that is expected to run on are large shared heterogeneous systems that can be unreliable at times. Due to the inhomogeneity, ICHOR may land on a node that is unsuitable for running the specific task or other users may be using one's allocated resource causing one's program to crash. The uncertainty of the environment that ICHOR's tasks wil be executed under makes it difficult to ensure 100% reliability. Therefore when running at scale, it is inevitable that errors will occur. ICHOR handles these errors through routines that check the outputs of specific programs and that make sure that the output is as expected. If not, then ICHOR has two options: (i) rerun the program with different parameters, or (ii) remove the task from the array. The choice will depend on the specific task. How the failure state is handled by ICHOR based upon the job being ran. For the GAUSSIAN example, it is required that the parameters used to run each GAUSSIAN job is fixed (as explained in Section 2.2.1), therefore it makes sense to instead skip the failed GAUSSIAN job. GAUSSIAN jobs are rarely performed in isolation and are more often performed as part of a larger task array. Therefore skipping over an individual failed task often has little effect on the job as a whole.

The result from the decision on how to handle the task (success or failure) is handled by the `eval` statement on line 32 of the submission script (see Code S9). The `eval` statement calls an ICHOR routine that will output a bash command to run. If successful then ICHOR_TASK_COMPLETED will be set to `true`, otherwise `false,` causing the task to be repeated. If the task is to be repeated then a limit is put in place to prevent an infinite loop, which is controlled on a per-job basis via ICHOR control parameters.

The method of message passing between ICHOR routines and the bash script allows for ICHOR to not only control the ICHOR_TASK_COMPLETED variable but also the task id indexing variable. Controlling the indexing parameter allows ICHOR to increment the indexing of the argument arrays providing the opportunity to run more tasks than were allocated to the job. Therefore, during an active learning run, for example, if a file is unintentionally removed or becomes corrupted, then the file can be regenerated in the subsequent iteration by adding the extra tasks to the job.

3.5 Submission Queues

The most common cluster configuration is that of a login node that communicates with many compute nodes. Jobs are submitted to the cluster on the login node and executed on compute nodes, the scheduling of which is controlled by the clustering software. The ICHOR pipeline requires for jobs to be queued up behind each other and executed in series. Hence the name "pipeline". To coordinate this task, ICHOR queues each job by forcing it to hold until the job queued prior has completed. The total number of jobs needed to be queued up in larger active learning runs can quickly amount to the tens of thousands of jobs per run. Adding to this the possibility of other users on the cluster and of multiple active learning jobs running simultaneously, then queueing jobs manually or submitting all jobs ahead of time becomes unfeasible. ICHOR currently implements interfaces to three methods of queueing submission jobs as outlined below.

3.5.1 Hold Queue Wait

The simplest method of queueing all jobs during an active learning run is to submit all jobs ahead of time and holding each job to wait for the previous job to finish. This is the simplest method of submitting jobs to the queueing system and can cause batch system instability if many thousands of jobs are submitted to the queueing system concurrently. A standard per-atom active learning run will typically produce many thousands of jobs and so this method of job submission is only used as a backup if no other option is available.

3.5.2 Submit On Compute

A method for submitting far fewer jobs to the queueing system at any one time is by only submitting the next iteration of jobs after the current iteration of jobs has finished. At the time of writing, an active learning iteration consists of 7 individual steps Therefore each ICHOR instance will have maximum 7 jobs queued in the queueing system at any given moment. This heavily reduces the number of jobs that an ICHOR system is required to queue up concurrently, and therefore puts much less stress on the queueing system.

Unfortunately, each ICHOR job is executed on a compute node within a cluster and therefore the final job in an active learning iteration should submit the next batch of jobs from a compute node. It is standard practice for a queueing system to have only the login nodes configured as submit host and compute nodes typically do not have the required privileges to submit jobs to the queueing system. Therefore, the "submit on compute" method of job submission is only available if it can be confirmed that each compute node is configured as a submit host. If not, another method of submission is required.

3.5.3 Drop Compute

To combat the number of concurrently queued jobs without changing the privileges of any compute node, a system known as "drop compute" was developed. The drop compute utility is a `cron` job that periodically scans a user's local directory for submission scripts to submit. If a submission script is found then the script is submitted to the queueing system by the drop compute system. Instead of submitting the next iteration's jobs, the final job of an active learning iteration will write the next iteration's submission scripts to the local directory scanned by the drop compute system. As the cron job is ran from a node with submission privileges, the submission scripts can be written from a compute node, bypassing the submission privileges usually encountered on a compute node.

The active learning pipeline has the added requirement of holding each job within an iteration in order for the previous job to finish. The drop compute system has the added functionality of specifying such a requirement with the use of a `hold` parameter within the script name.

Code S12. Example list of submission scripts for use by the drop compute system with script id shown in red and hold commands shown in blue.

```
ICHOR_GAUSSIAN.sh+1
GAUSSIAN.sh+2+hold_1
ICHOR_AIMALL.sh+3+hold_2
AIMALL.sh+4+4+hold_3
ICHOR_FEREBUS.sh+5+hold_4
FEREBUS.sh+6+hold_5
ICHOR_ACTIVE_LEARNING.sh+7+hold_6
```

As can be seen in Code S12, each script adds an identifier and a hold command, each separated using a + character. The identifier can be any 32 bit number and the hold command adds the script identifier to hold for.

The drop compute system provides the functionality to submit jobs from a compute node regardless of whether the compute node has submission privileges. The drop compute system is only available on systems that implement it and so is not always an option.

4 Machines

Each machine that ICHOR is required to run on is configured differently. Therefore, in order to ensure that the computer (i.e. machine) is fully utilised, ICHOR must implement machine specific configurations. There are many differences between machines. Each machine will likely differ in configurations of setting up the environment for program execution. There will also be differences in privileges a user may use. ICHOR is designed in a manner that provides a robust default method for program execution and job submission whilst providing options for machine specific optimisations for each task. ICHOR currently supports 3 separate machines:

- `CSF3 (University of Manchester Computational Shared Facility)`
- `FFLUXLAB (Popelier Group's In-House Cluster)`
- `Local (Generic machine with limited functionality)`

ICHOR auto-detects the current machine and user privileges on the given machine to determine what functionality is available to the user such as: available applications, batch systems, hardware information.

4.1 Environment Modules

A common method for controlling environments is through the use of environment modules. Environment modules are scripts written in the high-level programming language Tcl. They set up environment variables required for the execution of a specific program. Environment modules are therefore both system and application because each system will require different environment variables and each application will most likely be executed in a unique way.

ICHOR provides interface-to-environment modules that are added to an application's command line in a submission script. An example of the use of an environment module is line 10 of Code S10 where the GAUSSIAN environment module is loaded before running the GAUSSIAN application. The developer specifies the modules that must be loaded in order to run an external program for a specific system. Then, at runtime, ICHOR selects the relevant environment module from those available.

4.2 Node Privileges

As mentioned in Section 3.5, each machine will have different privileges based upon the node type that the application is executed on. For example, the in-house cluster FFLUXLAB is configured such that all nodes have submission privileges whereas the University-owned Computational Shared Facility (CSF3) allows jobs to be submitted only from the login node. ICHOR must therefore be able to detect both the machine and the node type to enable certain actions. The two node types are login and compute, each enabling certain privileges based upon the machine currently being used.

4.3 Parallel Environments

Each machine has hardware limitations based upon the nodes available to the user. When a user submits a job, the submission script must request system resources required to run the job. These hardware requirements will dictate the hardware assigned to the job by the batch system. The hardware assignment is provided by specifying the given parallel environment, which tells the batch system the nodes to use to run the job. The parallel environments available depend on the machine and the parallel environment required to run a job will depend on the number of cores being requested. ICHOR implements an interface to select the correct parallel environment for a given machine, and provides the hardware limits for each parallel environment on each machine.

4.4 Precompiled Binaries

Not all external programs used by ICHOR are freely available on all machines. For such applications, ICHOR provides precompiled binaries that are validated to run on a given machine preventing user errors from using an incorrect or out-of-date binary. At the time of writing, ICHOR comes packaged with the following in-house binary applications:

- `FEREBUS`
- `TYCHE`
- `DL_FFLUX`

5 Per-Value

ICHOR is a natural successor to the previous GPR pipeline application GAIA prompted by the initial goal to add active learning. Active learning is an iterative approach to improve a machine learning model where each iteration adds training points that will improve the performance of the model. In early versions of ICHOR, this active learning was carried out in a per-system manner by summing each variable over all models within a system.

Performing active learning over an entire system was the simplest implementation but came with certain drawbacks. Not every atom requires the same amount of data to produce high accuracy models. Moreover, not all atoms take the same amount of time to generate data or models. Hence, much time was wasted by waiting for calculations to finish. Future work will move away from the idea of calculation for all of the atoms within a system. Instead, the approach will use only part of a source molecule to create models and so a per-system active learning approach makes little sense in this context.

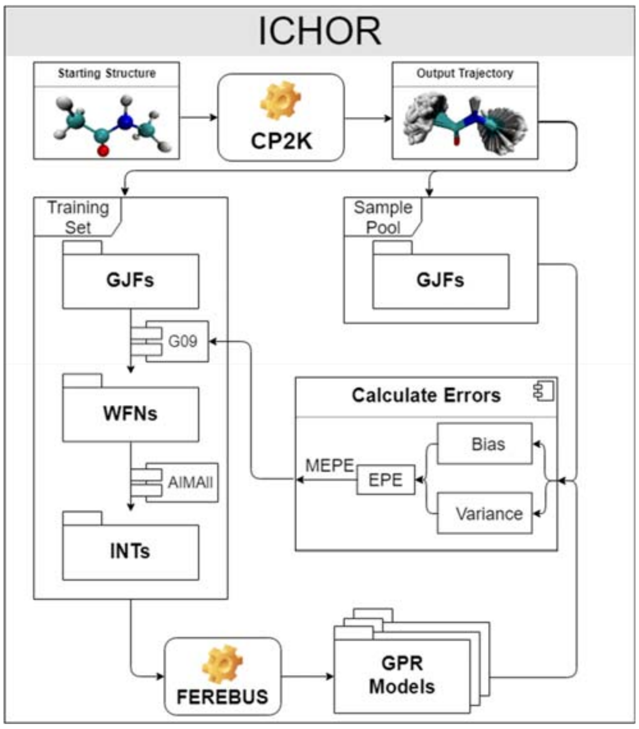


Figure S10. Schematic of traditional ICHOR per-system active learning method using the MEPE method.

The alternative approach introduced in later versions of ICHOR is the idea of per-value active learning. The per-value approach evaluates each model (determined by an atom and a specific property). This approach enables separate and unique training sets as opposed to the per-system approach, which restricts each model to use the same training set for all atoms and properties. At the time of writing, there are three per-value approaches implemented in ICHOR:

- `Per-Atom`
- `Per-Property`
- `Per-Atom Per-Property`

The per-atom approach is the standard per-value approach to use. In this approach a property is chosen to optimise and active learning runs are performed for each atom in the system on the chosen property (usually atomic energy). Then, models for the remaining properties may be generated for each atom using the training set produced by the per-atom active learning runs. Consequently, the property chosen to perform the active learning (defaulted to the IQA energy) controls the generation of the training set, which is used for all atomic properties.

The per-property approach is similar to the per-atom approach but optimises each output property (e.g. atomic energy, multipole moments etc.) either for a specified atom or by creating models for each atom and summing over all atoms of the system. The per-property approach is used less often compared to the per-atom approach because there are often many more properties than atoms to train, which can therefore take longer to produce models. Using the per-property approach provides the option to use all atoms for the active learning (whereby predictions are summed across all atoms of the system). Alternatively, one selects a single atom to use for the generation of the training set and then use this training set to produce models for the remaining atoms of the system.

The per-atom per-property approach is an extension of the per-atom approach by optimising not just a single property but all properties for each atom providing unique training sets for each atom-property combination. The per-atom per-property method takes up the most time and resources by far. Therefore it is seldom used in practice but exists as an option if a user wishes to take advantage of this approach.

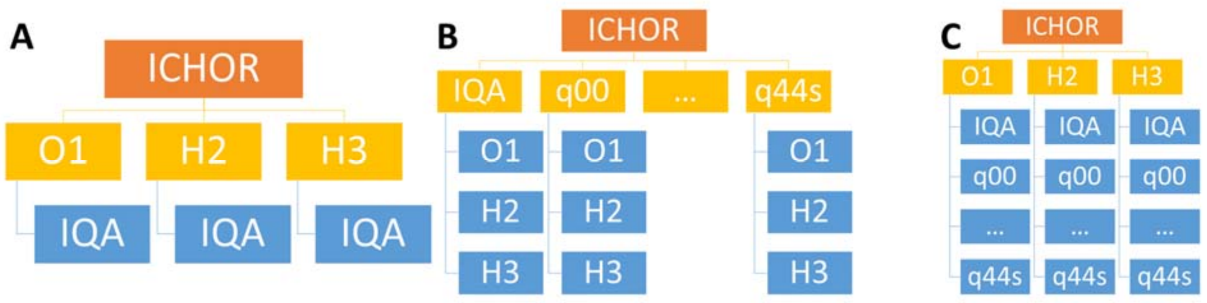


Figure S11. Examples of per-value partitioning schemes for a water model: (a) per-atom approach, (b) per-property approach, and(c) per-atom per-property approach. Each ICHOR instance (orange) controls a set of ICHOR child processes (yellow) to perform active learning run on the GPR models (blue).

ICHOR implements the per-value approach by creating child instances from a parent instance and providing the user with tools to control all child instances simultaneously. Using the parent-child hierarchy allows for each child to become a parent instance and spawn more child processes, which is how the per-atom per-property approach is implemented. The initial ICHOR parent class may then recursively control all child process beneath it, allowing for future expansion if desired.

6 Model Analysis

A vital function of ICHOR is to analyse the GPR models being produced to validate the accuracy and ensuring the model is ready to be used in a simulation. ICHOR implements many tools within the TUI and CLI for performing generic analyses as well as implementing a comprehensive interface with the model which may be used for in-depth and niche analyses via the library interface.

6.1 Model Reading

Providing users with an interface for using GPR models is one of the most used features in ICHOR, therefore it is vital that ICHOR fully supports the model file format standard defined by FEREBUS. An example model file may be found in Code S9, which demonstrates several of the model file features. Firstly the model file requires the ALF used to generate the features so that new points that are to be predicted using the model can use the same ALF when calculating the input features. ICHOR must then implement all mean functions as detailed in the main text.

One of the most important parts of the model file is the definition of the kernel, which can be composite kernel, as shown in Code S9. In order to generate composite kernels, ICHOR must implement a method of adding and multiplying kernels together, and a method for reading and interpreting the composite kernel as written in the config file.

Adding and multiplying kernels is as simple as implementing two composite kernel types: k_{sum} and k_{prod} for adding and multiplying kernels respectively:

$$k_{sum}(\boldsymbol{x}, \boldsymbol{x}^*) = k_1(\boldsymbol{x}, \boldsymbol{x}^*) + k_2(\boldsymbol{x}, \boldsymbol{x}^*) \tag{S1}$$

$$k_{prod}(\boldsymbol{x}, \boldsymbol{x}^*) = k_1(\boldsymbol{x}, \boldsymbol{x}^*) \times k_2(\boldsymbol{x}, \boldsymbol{x}^*) \tag{S2}$$

However, interpreting the composite kernel written in the model (such as the `k1*k2` example shown in Code S9) is slightly more complicated. Firstly, the base kernels must be read and stored in a way that can be referenced using the kernel names, in this case `k1` and `k2`. Secondly, the kernel composition string must be interpreted before combining the base kernels into the requested composite kernel. Interpreters are a well-known research topicin the computer science space and plenty of research has been carried out on it. ICHOR's kernel interpreter only needs to combine a small number of kernels and so does not need a complex design. For this reason, ICHOR's interpreter has been designed in a simple manner following the basic steps of lexical analysis, which is parsing and then interpreting an abstract syntax tree (AST) (see Figure S12 (c) for an example).

The lexical analyser (also known as lexer) simply scans the input characters separating the characters into tokens. A token is a single unit of an expression be that a number, variable name, operator and so on, a full list of tokens defined in the ICHOR interpreter are as follows:

Table S10. All token types used within the ICHOR kernel interpreter with examples.

Token Name	Example
Number	`1, 2.0, 1e-3, etc.`
Add Operator	`+`
Subtract Operator	`-`
Multiply Operator	`*`
Divide Operator	`/`
Left Pathenthesis	`(`
Right Parenthesis	`)`
Variable Name	`k1, k2, myvar1, myvar2, etc.`
End Of File	

Once the input string has been lexed into tokens, they must be parsed into an AST to allow the interpreter to determine the order in which the tokens must be evaluated. The parser forms expressions within a tree that can be evaluated, a single node at a time, in order to return the desired result. The parser is not only responsible for identifying expressions from the tokens but also preserve the correct mathematical order of operations (including parentheses). The parser is implemented using the well known *recursive descent parsing* algorithm. Figure S12 provides a schematic for the workflow of the kernel interpreter.

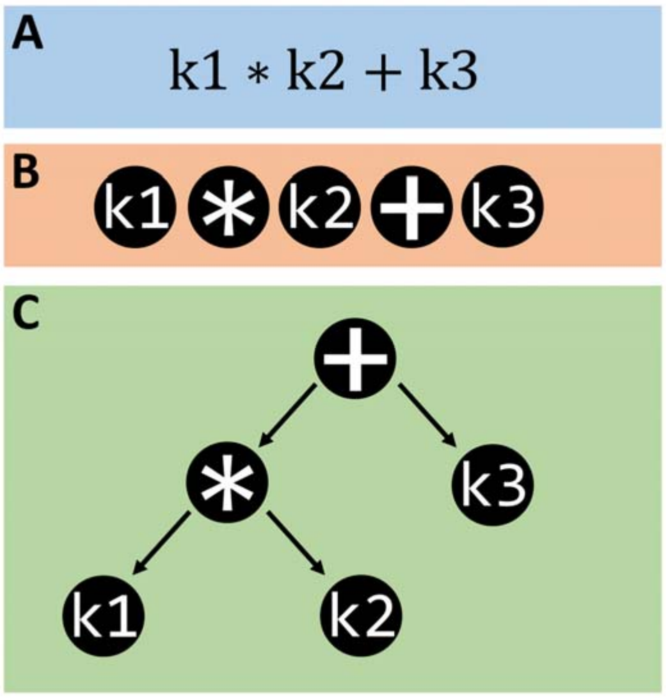


Figure S12. Schematic for interpreting an example composite kernel: (a) Composite kernel input string, (b) Lexed tokens, and (c) AST produced by the recursive descent parser.

The final composite kernel may then be returned by evaluating the composite kernel AST from the head node (first node) downwards whilst substituting any variables with the kernel matching the same variable name from the model file.

6.2 S-Curves

The most used model analysis tool in ICHOR is the S-curve analysis. Much of a model's performance can be determined by its predictive accuracy. Predictive accuracy of a model is often defined in the literature using a single value, typically the root mean squared error (RMSE) or the mean absolute error (MAE). When producing models for atomistic simulations, the average error is frequently less important than the maximum prediction errors. This is because it is the points producing the maximum prediction errors that will break a molecular simulation. For this reason it is also important to know the distribution of the prediction errors in order to determine how many points are close to the maximum prediction error. The more points close to the maximum prediction error, the worse the model.

An S-curve is a cumulative error distribution plot, where the y-axis is expressed in percent and the x-axis as an absolute prediction error. Each point of an S-curve represents a geometry in the validation set. Figures S13 and S14 show examples of S-curves generated by ICHOR. An as example of reading off an S-curve, just over 30% of the geometries have a total prediction energy error of maximum 1 kJ/mol. S-curves for multiple models can be combined to produce a total S-curve displaying the prediction error for an entire system for example, or

$$PE(\boldsymbol{x}) = \left|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right| \tag{S3}$$

$$PE_{total}(\boldsymbol{x}) = \sum_{i=1}^{n_{models}} PE_i(\boldsymbol{x}) \tag{S4}$$

where $f(\boldsymbol{x})$ is the true value of point $\boldsymbol{x}$ and $\hat{f}(\boldsymbol{x})$ is the predicted value of point $\boldsymbol{x}$. ICHOR is responsible for obtaining both the true values and the predictions for each validation point (i.e. geometry) in a validation set, subsequently writing all prediction errors to an excel file, and finally plotting each S-curve.

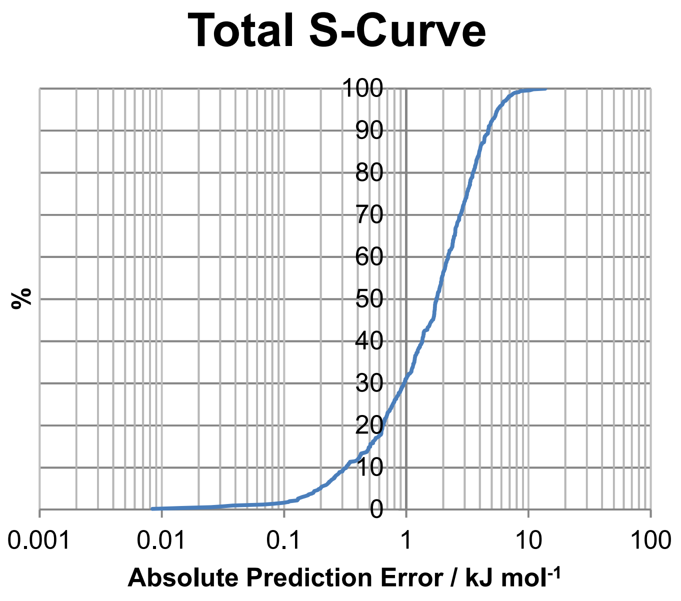


Figure S15. Example of a total S-curve for a glycine IQA energy model produced by ICHOR.

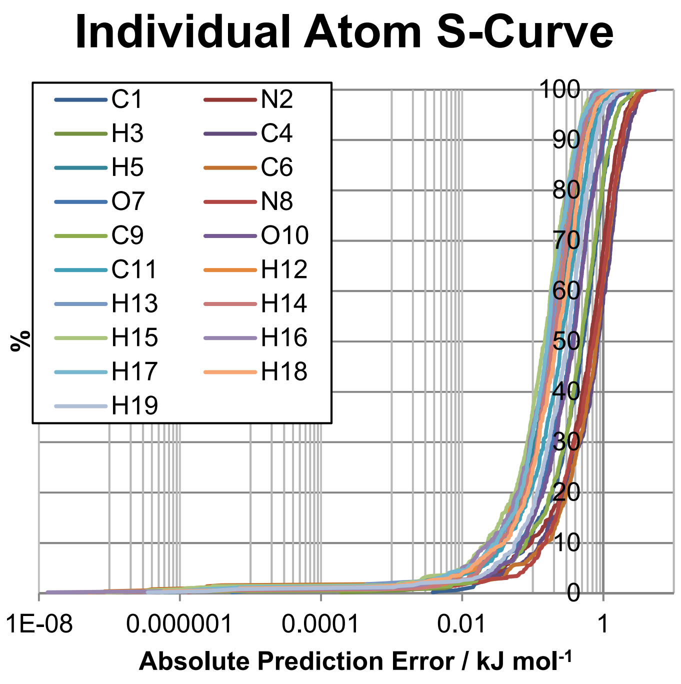


Figure S16. Example of S-curves, of individual atoms in glycine, for IQA energy models produced by ICHOR.

6.3 RMSE Curve

S-curves are ideal for analysising a single model's predictive performance because the entire spectrum of prediction errors can be deduced from a single plot. Unfortunately, plotting many S-curves next to one another makes it difficult to observe underlying trends. Analysing the performance of many models may be desirable when analysing the performance of an active learning run. In an active learning run, a new model is produced every iteration, therefore it is useful to track the progress of each model across the run. Due to the aforementioned difficulties in observing trends with many S-curves, it becomes easier to track the progress of the active learning using a single number to describe the performance of each model. The most typical method of determining the predictive performance of a model with a single number is using an RMSE value. ICHOR is responsible for obtaining the true values of a validation set alongside the predictions for each atom.

$$RMSE(\boldsymbol{X}) = \sqrt{\frac{\sum_{i=1}^{N}\left(f(\boldsymbol{x}_i) - \hat{f}(\boldsymbol{x}_i)\right)^2}{N}} \tag{S5}$$

where $\boldsymbol{X}$ is the validation set consisting of N validation points and $\boldsymbol{x}_i$ is the i^{th} validation point. ICHOR then outputs the RMSE values for each model alongside the number of training points in each model allowing the user to plot the RMSE value against the number of training points.

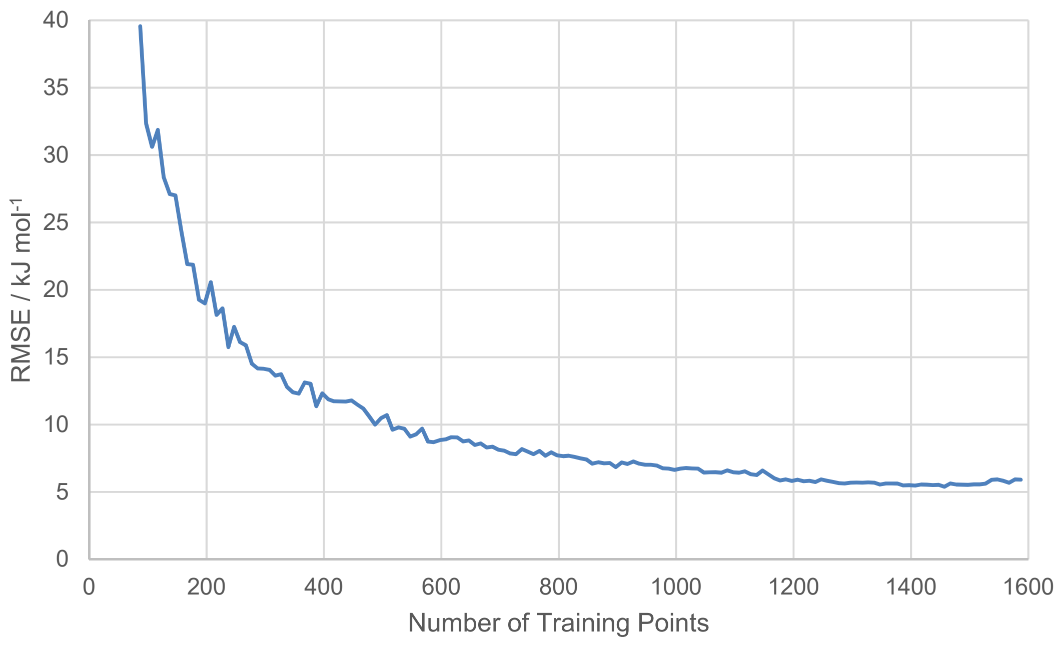


Figure S17. Example of an RMSE curve for an N-mehylacetamide active learning run.

7 Analysis Tools

ICHOR provides the user with several analysis tools for the analysis, not only of models, but also of trajectories used to generate models. It is vitally important that a model is created using data that samples the input space sufficiently well, which may only be ensured via thorough analysis of the input trajectory.

7.1 Geometry Analysis

The most basic form of geometry analysis provided by ICHOR is the calculation of all bonds, angles and dihedrals of each timestep of a trajectory (including originating from TYCHE). If the conformational flexibility of the target system is known, then the flexibility of the input trajectory can be ensured to meet the criteria through the use of the geometry analysis tools provided by ICHOR.

The first step in performing a geometry analysis on a trajectory is to determine the atoms involved in each bond angle and dihedral to allow ICHOR to calculate each feature for all timesteps. Firstly, the connectivity of the system must be determined, as described in the main text, to form a connectivity matrix. Then from this connectivity matrix, a molecular graph is created whereby each node is an atom and each covalent bond depicted by an edge. Determining the bonds of a molecular graph is straightforward as a bond is simply an edge of the graph. Determining the angles of the graph is a little trickier because this involves searching for two edges connected by a common atom. Finding the dihedral angles is an extension of this logic by now looking for two angles joined by a common edge. Once the atoms defining each bond, angle and dihedral have been established, the calculation of each value is a minor task:

$$r(\boldsymbol{A},\boldsymbol{B}) = \left\|\overrightarrow{\boldsymbol{AB}}\right\| \tag{S6}$$

$$\alpha(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \cos^{-1}\frac{\overrightarrow{\boldsymbol{AB}}\cdot\overrightarrow{\boldsymbol{BC}}}{\left\|\overrightarrow{\boldsymbol{AB}}\right\|\left\|\overrightarrow{\boldsymbol{BC}}\right\|} \tag{S7}$$

$$\delta(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{D}) = \mathrm{atan2}(\boldsymbol{m}_1\cdot\boldsymbol{n}_2,\boldsymbol{n}_1\cdot\boldsymbol{n}_2) \tag{S8}$$

where r, α and δ are the bond, angle and dihedral functions respectively, atan2 the 2-argument arctangent, and $\boldsymbol{n}_1$, $\boldsymbol{n}_2$ and $\boldsymbol{m}_1$ are defined by the following:

$$\boldsymbol{n}_1 = \frac{\overrightarrow{\boldsymbol{BC}}\times\overrightarrow{\boldsymbol{CD}}}{\left\|\overrightarrow{\boldsymbol{BC}}\times\overrightarrow{\boldsymbol{CD}}\right\|} \tag{S9}$$

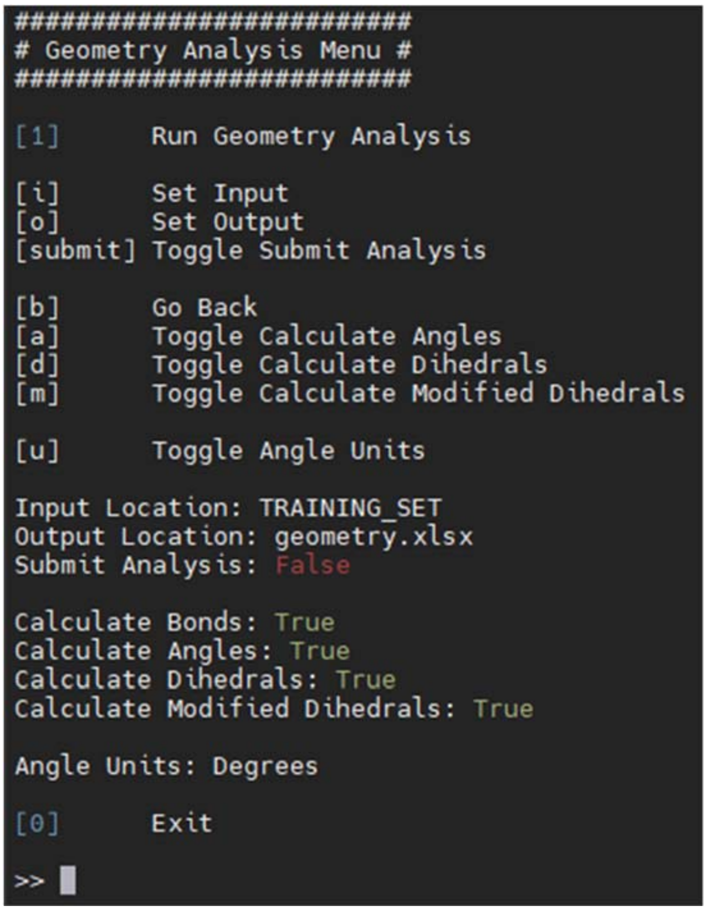
$$\boldsymbol{n}_2 = \frac{\overrightarrow{\boldsymbol{AB}}\times\overrightarrow{\boldsymbol{BC}}}{\left\|\overrightarrow{\boldsymbol{AB}}\times\overrightarrow{\boldsymbol{BC}}\right\|} \tag{S10}$$

$$\boldsymbol{m}_1 = \boldsymbol{n}_1\times\frac{\overrightarrow{\boldsymbol{BC}}}{\left\|\overrightarrow{\boldsymbol{BC}}\right\|} \tag{S11}$$

ICHOR then outputs all "raw" feature values for a given trajectory into an excel spreadsheet allowing the user to perform the necessary analysis. ICHOR also calculates a modified dihedral (δ_{mod}) that extends the value beyond $[0,2\pi]$ because it is often the case, when performing an analysis on a dihedral that performs a full rotation, that the value frequently jumps between 0 and 2π (as seen in Figure S21). Instead ICHOR modifies the dihedral value by setting the value to the cyclic difference between the current dihedral and the previous dihedral in the trajectory. Hence it is obvious to the user how many full rotations a dihedral performs throughout a trajectory. The modified dihedral (δ_{mod}) is defined as follows:

$$\delta_{mod} = \begin{cases} \delta_i, & i = 0 \\ \delta_{i-1} + (\delta_i - \delta_{i-1} + \pi) \bmod 2\pi - \pi, & i > 0 \end{cases} \tag{S12}$$

where i is the current index in the trajectory and mod is the modulus function. The geometry analysis tool is provided to the user through the geometry analysis submenu shown in Figure S18, the output of which may be seen in Figures S19-S22.



```
##########################
# Geometry Analysis Menu #
##########################

[1]      Run Geometry Analysis

[i]      Set Input
[o]      Set Output
[submit] Toggle Submit Analysis

[b]      Go Back
[a]      Toggle Calculate Angles
[d]      Toggle Calculate Dihedrals
[m]      Toggle Calculate Modified Dihedrals

[u]      Toggle Angle Units

Input Location: TRAINING_SET
Output Location: geometry.xlsx
Submit Analysis: False

Calculate Bonds: True
Calculate Angles: True
Calculate Dihedrals: True
Calculate Modified Dihedrals: True

Angle Units: Degrees

[0]      Exit

>> 
```

Figure S18. Example of the geometry analysis menu.

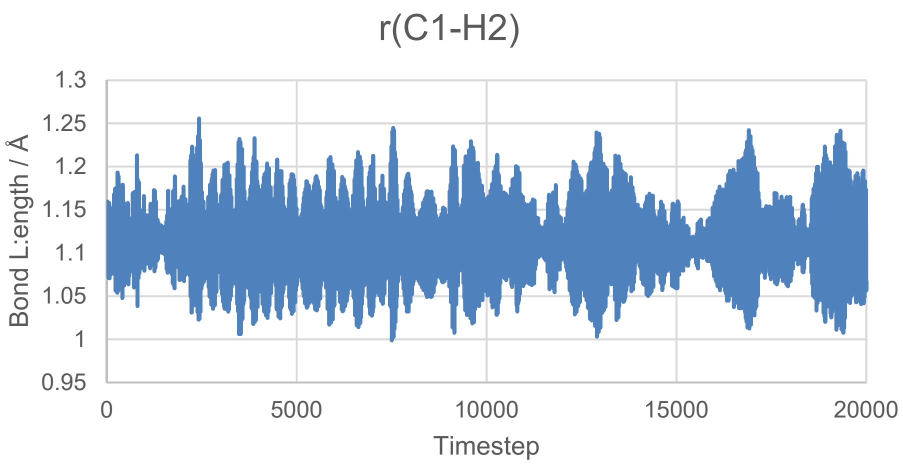


Figure S19. Example of a bond length plot from a geometry analysis on a methanol 1000K CP2K simulation.

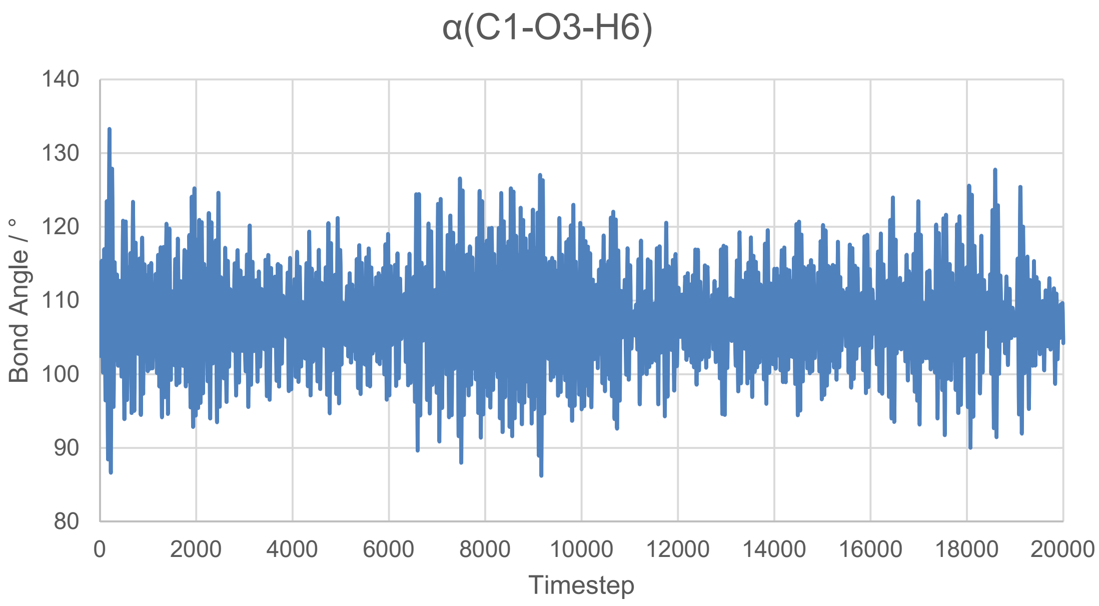


Figure S20. Example of a bond angle plot from a geometry analysis on a methanol 1000 K CP2K simulation.

As can be seen from the difference between Figures S21 and S22, it can be useful to use the modified dihedral as opposed to the original "raw" dihedral value whilst analysing dihedral angles that perform full rotations.

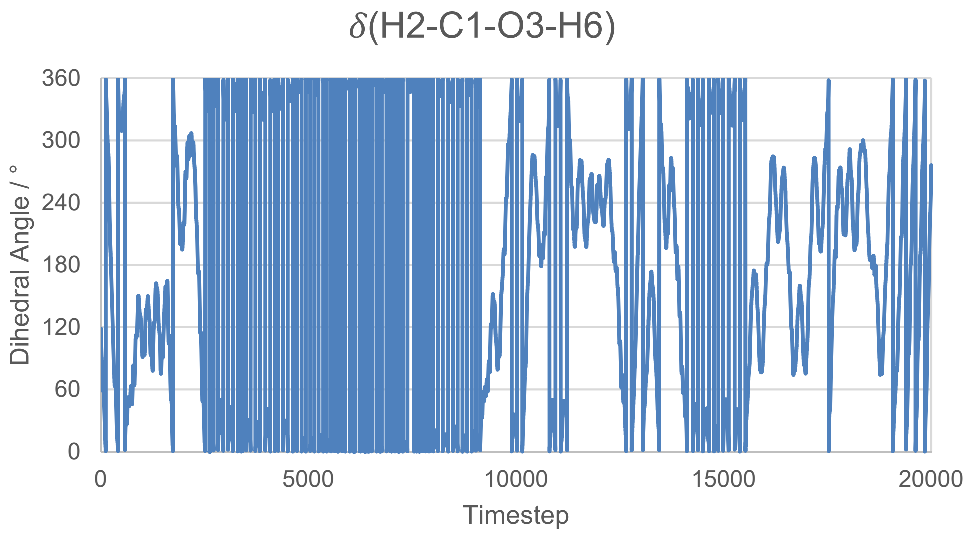


Figure S21. Example of a dihedral angle plot from a geometry analysis on a methanol 1000 K CP2K simulation.

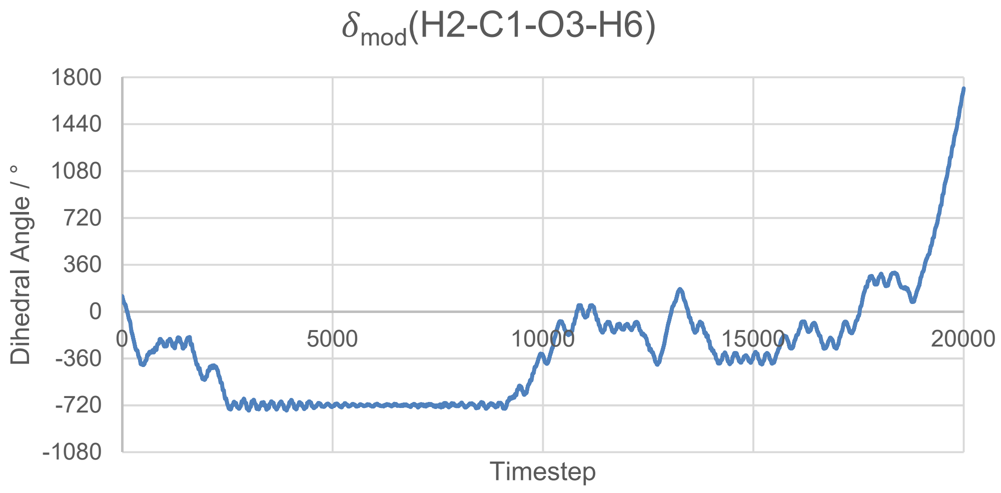
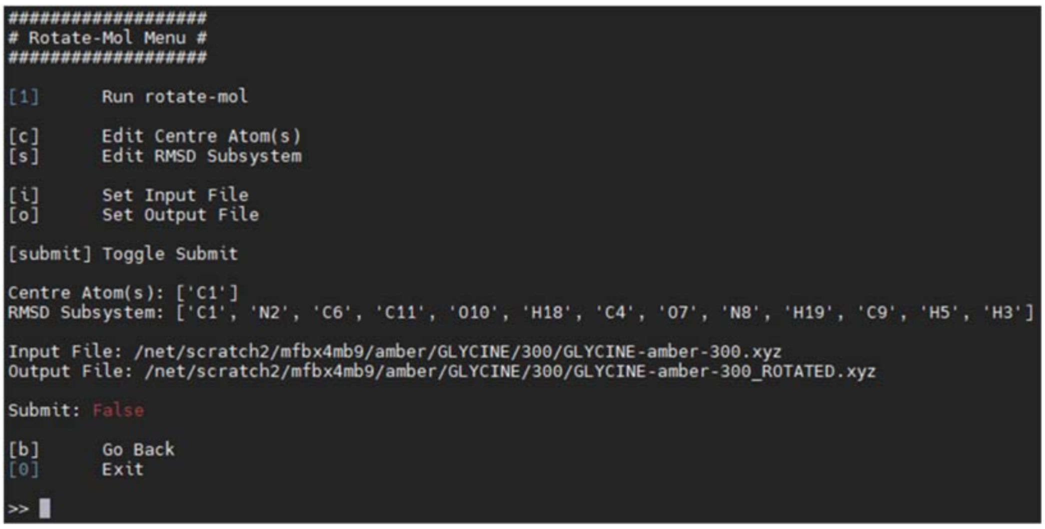


Figure S22. Example of a modified dihedral angle plot from a geometry analysis on a methanol 1000 K CP2K simulation.

7.2 Rotate-Mol

Alongside the numerical data detailing how much a molecule distorts across a trajectory, it is also often quite useful to visualise the data. Generally a good way to visualise the distortions of a molecule is to overlay all of the configurations of a trajectory. Unfortunately, depending on the method of generating the trajectory, geometries may not align as to provide a good visualisation of the molecular distortions due to the molecule translating and rotating in space. ICHOR provides a tool to overlap all or part of a molecule for each timestep in a trajectory. This tool is known as rotate-mol and can be controlled using the rotate-mol submenu shown below in Figure S23.



```
###################
# Rotate-Mol Menu #
###################

[1]      Run rotate-mol

[c]      Edit Centre Atom(s)
[s]      Edit RMSD Subsystem

[i]      Set Input File
[o]      Set Output File

[submit] Toggle Submit

Centre Atom(s): ['C1']
RMSD Subsystem: ['C1', 'N2', 'C6', 'C11', 'O10', 'H18', 'C4', 'O7', 'N8', 'H19', 'C9', 'H5', 'H3']

Input File: /net/scratch2/mfbx4mb9/amber/GLYCINE/300/GLYCINE-amber-300.xyz
Output File: /net/scratch2/mfbx4mb9/amber/GLYCINE/300/GLYCINE-amber-300_ROTATED.xyz

Submit: False

[b]      Go Back
[0]      Exit

>>
```

Figure S23. Example of the rotate-mol ICHOR submenu.

Plotting a unrotated trajectory may result in the geometry translating and rotating in the global frame, an example of which can be seen in Figure S24.

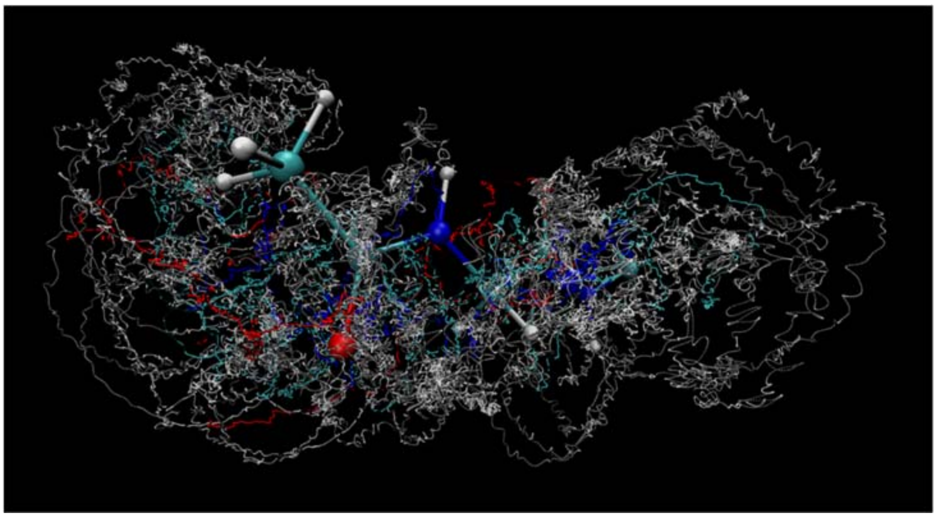


Figure S24. Example of an original trajectory for a 300 K CP2K simulation.

Rotate-mol uses the Kabsch algorithm to calculate the optimal rotation matrix that will overlap two molecules on top of one another. The Kabsch algorithm is a well known algorithm used to find the optimal rotation matrix R in order to rotate points P onto points Q (both of which are centred on the origin) using the following:

$$H_{ij} = \sum_{k=1}^{N} P_{ki} Q_{kj} \tag{S13}$$

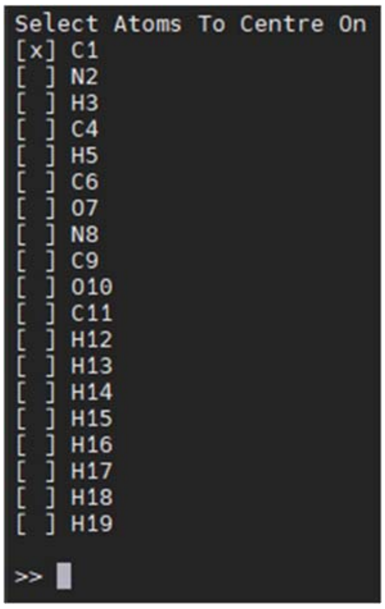
$$H = U \Sigma V^{\mathrm{T}} \tag{S14}$$

$$d = \mathrm{sign}\Big(\det(V U^{\mathrm{T}})\Big) \tag{S15}$$

$$R = V \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{pmatrix} U^{\mathrm{T}} \tag{S16}$$

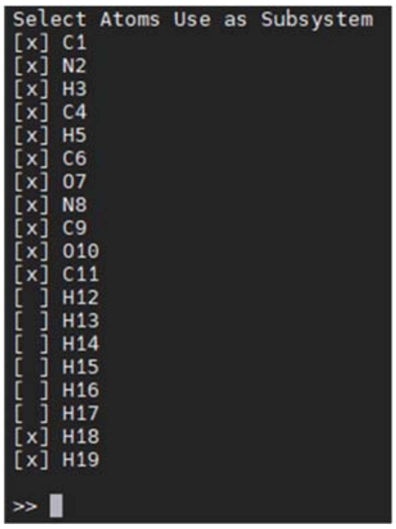
$$P_{rot} = (R P^{\mathrm{T}})^{\mathrm{T}} \tag{S17}$$

where equation S14 denotes the singular value decomposition of the cross covariance matrix H and P_{rot} is the rotated coordinates P such that the root mean squared deviation (RMSD) between the points P and Q is minimised. As mentioned previously, points P and Q must be centred on the origin to find the optimal rotation matrix, which is ensured by subtracting the centroid position of the system from each atom in the system. The centroid is the mean position in each dimension (non-mass weighted) and therefore subtracting this position from each atom places the centroid of the system at the origin. After rotating points P onto points Q, the system may be recentred at a position chosen by the user. Often this point is a particular atom but could be the centre between two atoms or the centre of a ring of atoms, for example. Recentring the system adds additional flexibility when trying to visualise a particular distortion. The centering atoms and subsystem atoms are controlled uising the submenus c and s (from the rotate mol menu shown in Figure S23), examples of which may be seen in Figures S25 and S26.



```
Select Atoms To Centre On
[x] C1
[ ] N2
[ ] H3
[ ] C4
[ ] H5
[ ] C6
[ ] O7
[ ] N8
[ ] C9
[ ] O10
[ ] C11
[ ] H12
[ ] H13
[ ] H14
[ ] H15
[ ] H16
[ ] H17
[ ] H18
[ ] H19

>>
```

Figure S25. Example of the rotate-mol centre atom(s) submenu.



```
Select Atoms Use as Subsystem
[x] C1
[x] N2
[x] H3
[x] C4
[x] H5
[x] C6
[x] O7
[x] N8
[x] C9
[x] O10
[x] C11
[ ] H12
[ ] H13
[ ] H14
[ ] H15
[ ] H16
[ ] H17
[x] H18
[x] H19

>>
```

Figure S26. Example rotate-mol subsys atom(s) submenu.

In ICHOR's case, the points P and Q are the Cartesian coordinates of two configurations of a molecule but it is not necessary to use the Cartesian coordinates of all atoms within a system. Using just a subset of the atoms within a system can improve the clarity of specific molecular distortions of interest. For example, when modelling methanol an important distortion to visualise is the dihedral rotation of the hydroxyl group. Therefore setting the subsystem to overlap the part of the molecule other than the hydroxyl hydrogen and centring on the carbon will provide a clearer view of the true distortion of this hydrogen atom as demonstrated below. An example showing each step of the rotation process for a methanol molecule may be seen below in Figure S27, with an example on a larger glycine molecule shown in Figure S28.

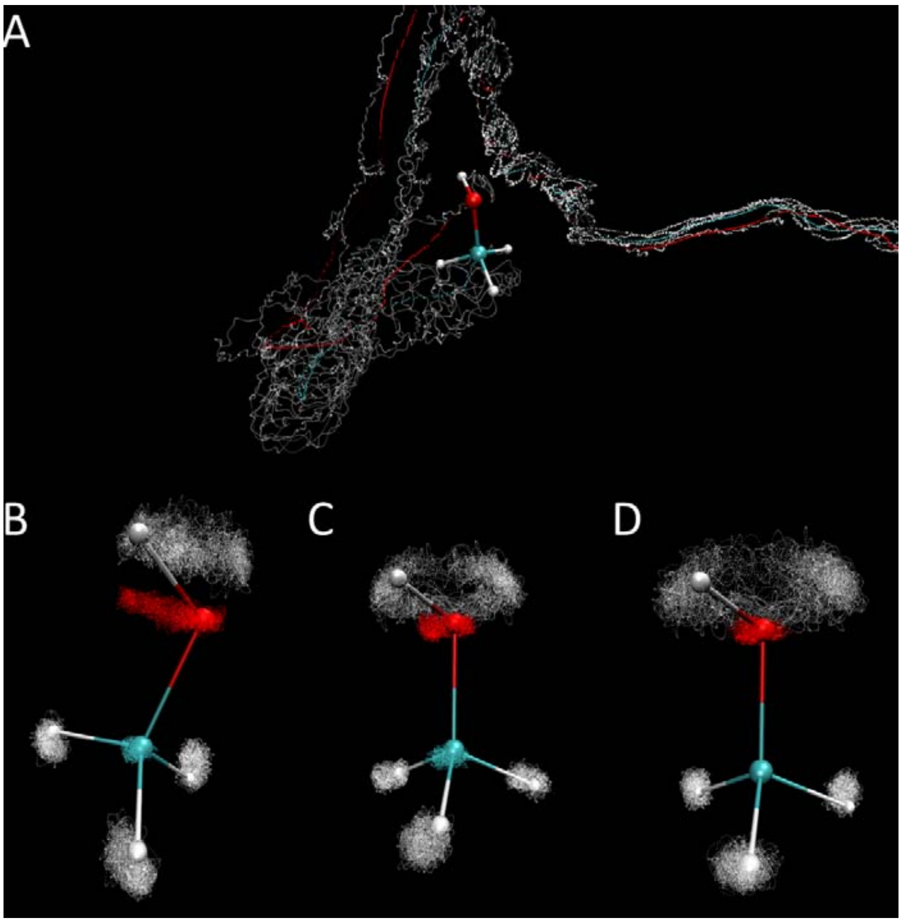


Figure S27. Example of the rotate-mol pipeline for a methanol 1000 K CP2K simulation trajectory showing the (a) raw trajectory data, (b) rotate-mol output using all atoms for the Kabsch algorithm and centring on the origin, (c) rotate-mol output using all atoms except from the hydroxyl hydrogen for the Kabsch algorithm and centring on the origin, and (d) rotate-mol output using all atoms except from the hydroxyl hydrogen for the Kabsch algorithm and centring on the methyl carbon atom. As seen in the steps from (a) to (d), carefully increasing the constraints on the trajectory increases the clarity of the hydroxyl group dihedral rotation whilst visualising on a mist plot.

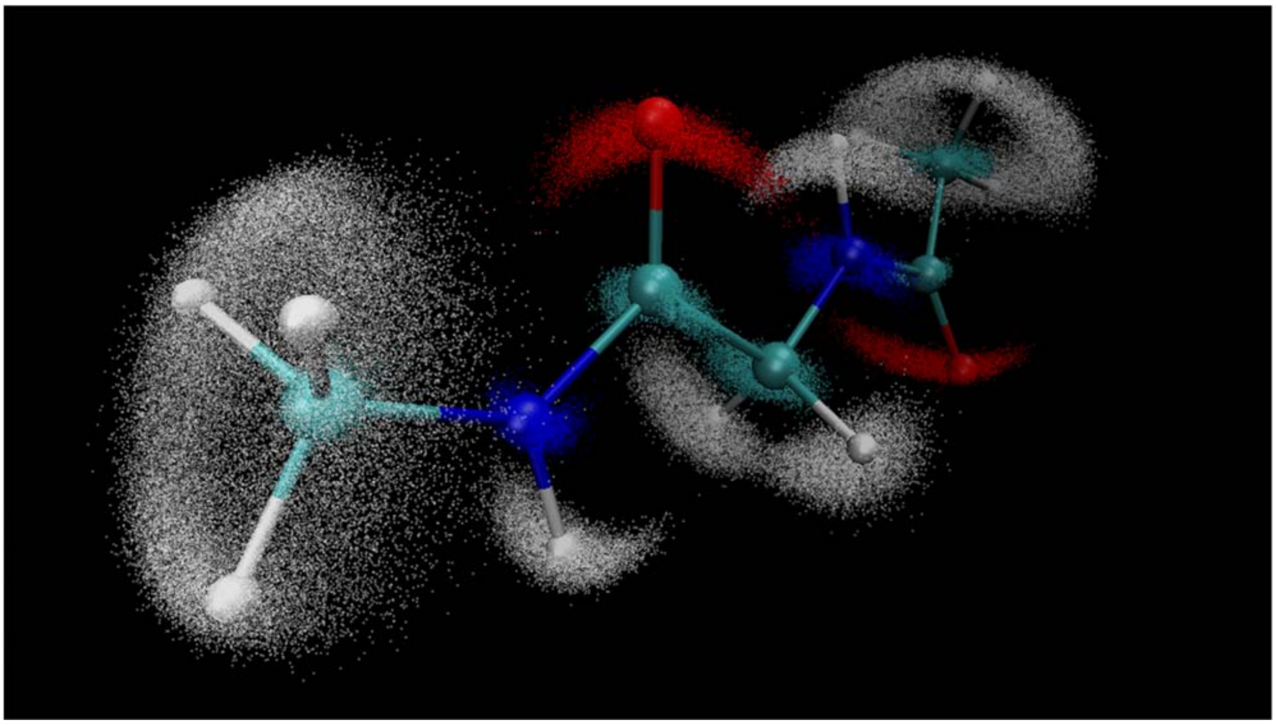
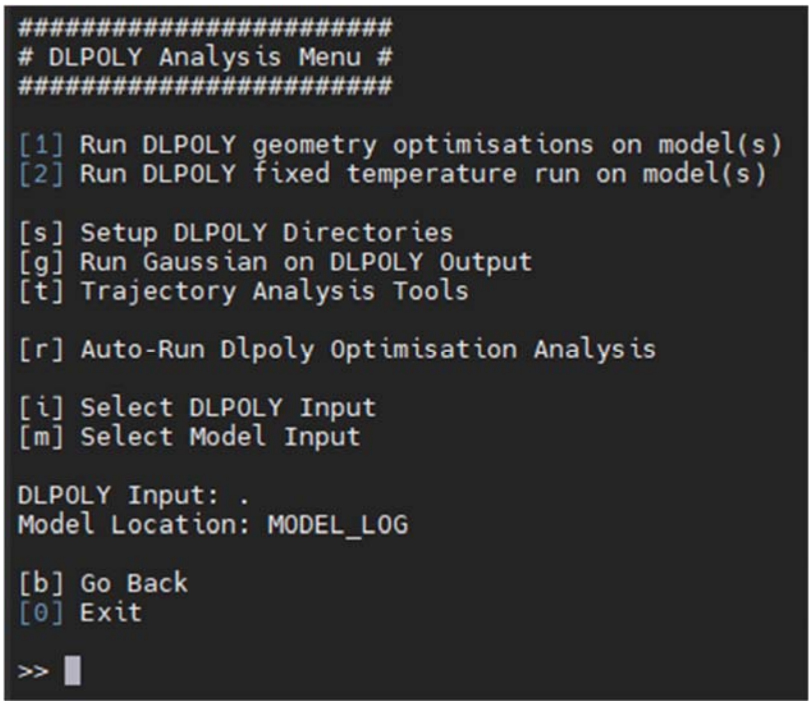


Figure S28. Example of a glycine 300 K CP2K simulation mist plot using the ICHOR default of using all atoms for the Kabsch algorithm and centering on the origin.

7.3 DL_FFLUX Analysis

ICHOR is designed for the purpose of creating models that can be used in atomistic simulations in the novel force field DL_FFLUX. All the other analysis tools in ICHOR are implemented to provide quick feedback on the performance of a model before investing time into performing a full-scale simulation using the models. ICHOR also provides an utility to test the GPR models using DL_FFLUX. As with other analyses, The DL_FFLUX analysis is controlled using a submenu, an example of which can be seen below in Figure S29.



```
########################
# DLPOLY Analysis Menu #
########################

[1] Run DLPOLY geometry optimisations on model(s)
[2] Run DLPOLY fixed temperature run on model(s)

[s] Setup DLPOLY Directories
[g] Run Gaussian on DLPOLY Output
[t] Trajectory Analysis Tools

[r] Auto-Run Dlpoly Optimisation Analysis

[i] Select DLPOLY Input
[m] Select Model Input

DLPOLY Input: .
Model Location: MODEL_LOG

[b] Go Back
[0] Exit

>> 
```

Figure S29. Example DL_FFLUX interface menu.

To run DL_FFLUX, alongside the GPR models to which it has access, 3 files are required: CONFIG, CONTROL and FIELD, which we discuss each in turn. The CONFIG file (Code S13) is the default method of inputting the starting geometry for DL_POLY and contains the Cartesian coordinates for each atom alongside the periodic boundary conditions (PBC) box size. DL_FFLUX also adds the additional functionality of assigning a model to each atom.

Code S13. Example of a DL_FFLUX CONFIG file produced by ICHOR

```
Frame :         1
        0       1
25.0 0.0 0.0
0.0 25.0 0.0
0.0 0.0 25.0
C   1  GLYCINE_C1
-0.390847368421     -0.43032684210     0.33991894736
N   2  GLYCINE_N2
 1.052502631578     -0.54192684210     0.33558894736
H   3  GLYCINE_H3
-0.814447368421     -1.31865684210    -0.13924105263
C   4  GLYCINE_C4
-0.822717368421      0.82495315789    -0.41701105263
H   5  GLYCINE_H5
-0.735257368421     -0.37170684210     1.37720894736
```

```
C   6  GLYCINE_C6
 1.688162631578    -1.60095684210     0.92531894736
O   7  GLYCINE_O7
-0.014167368421     1.62230315789    -0.88932105263
N   8  GLYCINE_N8
-2.189997368421     0.98070315789    -0.50662105263
C   9  GLYCINE_C9
-2.769167368421     2.07455315789    -1.24462105263
O  10  GLYCINE_O10
 1.089572631578    -2.51775684210     1.47933894736
C  11  GLYCINE_C11
 3.189732631578    -1.55068684210     0.85080894736
H  12  GLYCINE_H12
-2.748127368421     1.82664315789    -2.30906105263
H  13  GLYCINE_H13
-2.202097368421     2.99340315789    -1.07321105263
H  14  GLYCINE_H14
-3.802547368421     2.20705315789    -0.91619105263
H  15  GLYCINE_H15
 3.537312631578    -0.72416684210     0.22491894736
H  16  GLYCINE_H16
 3.557282631578    -2.48656684210     0.42120894736
H  17  GLYCINE_H17
 3.593122631578    -1.42242684210     1.85869894736
H  18  GLYCINE_H18
 1.572732631578     0.22148315789    -0.08732105263
H  19  GLYCINE_H19
-2.791047368421     0.21408315789    -0.23041105263
```

The second file required for a DL_FFLUX simulation is the CONTROL file (Code S14), which controls the parameters for the execution of both DL_POLY and DL_FFLUX. ICHOR writes the config file using parameters set by the user in ICHOR's config file.

Table S11 shows that the execution of DL_FFLUX can be controlled with high fidelity, which is important for performing detailed analyses on a variety of systems and providing such control to a novice user. A common task is to perform geometry optimisations of systems using GPR models with FFLUX because this is often a good test of the predictive accuracy of the model. Moreover, it allows for a test of the dynamics of the model (how well the gradients of the predictions match the true derivatives of the PES) and tests the accuracy of the minimum geometry reached using the model.

Table S11. List of parameters used to control the execution of DL_FFLUX.

Parameter	**Description**	**Default Value**
DLPOLY_NCORES	Number of cores to use to run DL_FFLUX	1
DLPOLY_NUBER_OF_STEPS	Specifies the number of steps to run a DL_FFLUX simulation for	500
DLPOLY_TEMPERATURE	Sets the temperature of the DL_FFLUX simulation	0 K
DLPOLY_PRINT_EVERY	Specifies how often DL_FFLUX should write to the output files	1
DLPOLY_TIMESTEP	Sets the length of the timestep in the DL_FFLUX simulation	0.001 ps
DLPOLY_LOCATION	Specifies the path to the DL_FFLUX executable, this is an automated process for certain machines	
DLPOLY_HOOVER	Sets time constant parameter for the Hoover thermostat in DL_FFLUX	0.04
DLPOLY_CHECK_CONVERGENCE	Specifies whether DL_FFLUX should check for convergence criteria whilst performing a geometry optimisation	False
DLPOLY_CONVERGENCE_CRITERIA	Selects the convergence criteria to use from a predefined list of values	4 (Gaussian convergence criteria)
DLPOLY_MAX_ENERGY	Sets the maximum energy convergence criteria threshold	-1 (not set)
DLPOLY_MAX_FORCE	Sets the maximum force convergence criteria threshold	-1 (not set)
DLPOLY_RMS_FORCE	Sets the RMS force convergence criteria threshold	-1 (not set)
DLPOLY_MAX_DISP	Sets the maximum displacement convergence criteria threshold	-1 (not set)
DLPOLY_RMS_DISP	Sets the RMS displacement convergence criteria threshold	-1 (not set)

Code S14. Example of a CONTROL file for a glycine 1 K geometry optimisation

```
Title: GLYCINE

ensemble nvt hoover 0.04

temperature 1


timestep 0.001
steps 1000
scale 100

cutoff  8.0
rvdw    8.0
vdw direct
vdw shift
fflux cluster L1

dump  1000
traj 0 1 0
print every 1
stats every 1
fflux print 0 1
job time 10000000
close time 20000
finish
```

The final file that DL_FFLUX requires is known as the FIELD file (see Code S15). The FIELD file is traditionally used by DL_POLY to define parameters for harmonic bond stretches, angle deformations, etc. However, there is no need for such parameters in DL_FFLUX because the forces are derived from the GPR models only. Therefore the FIELD file need only specify the mass of each atom. All other parameters can be omitted. Unfortunately, due to current implementation constraints, all parameters still must be specified even though they are not used within the simulation. Hence, each bond, angle and dihedral must be defined with spoof parameters for a simulation to run. Producing such a file by hand would be a time consuming and error prone process, so ICHOR provides this functionality by reusing the same technologies to define all internal features as used within the geometry analysis toolkit demonstrated in Section 7.1.

Code S15. Example of a FIELD file for glycine DL_FFLUX simulation

```
DL_FIELD v3.00
Units kJ/mol
Molecular types 1
GLYCINE
nummols 1
atoms 19
C       12.0106000     0.0   1   0
N       14.0068550     0.0   1   0
H        1.0079750     0.0   1   0
C       12.0106000     0.0   1   0
H        1.0079750     0.0   1   0
C       12.0106000     0.0   1   0
O       15.9994000     0.0   1   0
N       14.0068550     0.0   1   0
C       12.0106000     0.0   1   0
O       15.9994000     0.0   1   0
C       12.0106000     0.0   1   0
H        1.0079750     0.0   1   0
H        1.0079750     0.0   1   0
H        1.0079750     0.0   1   0
H        1.0079750     0.0   1   0
H        1.0079750     0.0   1   0
H        1.0079750     0.0   1   0
H        1.0079750     0.0   1   0
H        1.0079750     0.0   1   0
BONDS 18
harm 1 2 0.0 0.0
harm 1 3 0.0 0.0
harm 1 4 0.0 0.0
harm 1 5 0.0 0.0
harm 2 6 0.0 0.0
harm 2 18 0.0 0.0
harm 4 7 0.0 0.0
harm 4 8 0.0 0.0
harm 6 10 0.0 0.0
harm 6 11 0.0 0.0
harm 8 9 0.0 0.0
harm 8 19 0.0 0.0
harm 9 12 0.0 0.0
harm 9 13 0.0 0.0
harm 9 14 0.0 0.0
harm 11 15 0.0 0.0
harm 11 16 0.0 0.0
harm 11 17 0.0 0.0
ANGLES 30
harm 1 2 6 0.0 0.0
harm 1 4 7 0.0 0.0
harm 1 4 8 0.0 0.0
harm 1 2 18 0.0 0.0
harm 2 1 3 0.0 0.0
harm 2 1 4 0.0 0.0
harm 2 1 5 0.0 0.0
harm 2 6 10 0.0 0.0
harm 2 6 11 0.0 0.0
harm 3 1 4 0.0 0.0
harm 3 1 5 0.0 0.0
harm 4 1 5 0.0 0.0
harm 4 8 9 0.0 0.0
harm 4 8 19 0.0 0.0
harm 6 11 15 0.0 0.0
harm 6 11 16 0.0 0.0
harm 6 11 17 0.0 0.0
harm 6 2 18 0.0 0.0
harm 7 4 8 0.0 0.0
harm 8 9 12 0.0 0.0
harm 8 9 13 0.0 0.0
harm 8 9 14 0.0 0.0
harm 9 8 19 0.0 0.0
harm 10 6 11 0.0 0.0
harm 12 9 13 0.0 0.0
harm 12 9 14 0.0 0.0
harm 13 9 14 0.0 0.0
harm 15 11 16 0.0 0.0
harm 15 11 17 0.0 0.0
harm 16 11 17 0.0 0.0
DIHEDRALS 32
harm 1 4 8 9 0.0 0.0
harm 1 2 6 10 0.0 0.0
harm 1 2 6 11 0.0 0.0
harm 1 4 8 19 0.0 0.0
harm 2 1 4 7 0.0 0.0
harm 2 1 4 8 0.0 0.0
harm 2 6 11 15 0.0 0.0
harm 2 6 11 16 0.0 0.0
harm 2 6 11 17 0.0 0.0
harm 3 1 2 6 0.0 0.0
harm 3 1 4 7 0.0 0.0
harm 3 1 4 8 0.0 0.0
harm 3 1 2 18 0.0 0.0
harm 4 1 2 6 0.0 0.0
harm 4 8 9 12 0.0 0.0
harm 4 8 9 13 0.0 0.0
harm 4 8 9 14 0.0 0.0
harm 4 1 2 18 0.0 0.0
harm 5 1 2 6 0.0 0.0
harm 5 1 4 7 0.0 0.0
harm 5 1 4 8 0.0 0.0
harm 5 1 2 18 0.0 0.0
harm 7 4 8 9 0.0 0.0
harm 7 4 8 19 0.0 0.0
harm 10 6 11 15 0.0 0.0
harm 10 6 11 16 0.0 0.0
harm 10 6 11 17 0.0 0.0
harm 10 6 2 18 0.0 0.0
harm 11 6 2 18 0.0 0.0
harm 12 9 8 19 0.0 0.0
harm 13 9 8 19 0.0 0.0
harm 14 9 8 19 0.0 0.0
finish
close
```

With the required DL_FFLUX input files and the relevant model files in the `krig_models` directory, ICHOR is able to submit DL_FFLUX simulations to the HPC cluster in the same manner as any other interfaced external program. The outputs from a DL_FFLUX simulation include trajectories, energies and forces on each atom. ICHOR is able to read such outputs providing an interface with each output file and allowing for the analysis of the MD trajectory and output predictions.

8 Multipole Rotation

As mentioned in the main text, the multipole moments outputted from AIMAll are in the global spherical harmonic representation and for input to the GPR models it is necessary that the multipole moments are represented in the local atomic spherical format. To perform the rotation, a C matrix is computed, as described in the main text, which is used to rotate a set of Cartesian moments into the local frame. Therefore, to perform the rotation, the global spherical moments must be converted to global Cartesian moments then rotated into local atomic cartesian multipole moments before converting back to the local atomic spherical representation. The monopole moment is a scalar quantity and therefore does not need a rotation into the local frame. All other multipole conversions are described using the following sets of equations, organised in Tables S12 to S19.

Table S12. Dipole moment spherical to cartesian conversion.

$\mu_x = Q_{11c}$	(S18)
$\mu_y = Q_{11s}$	(S19)
$\mu_z = Q_{10}$	(S20)

Table S13. Dipole moment cartesian to spherical conversion.

$Q_{10} = \mu_z$	(S21)
$Q_{11c} = \mu_x$	(S22)
$Q_{11s} = \mu_y$	(S23)

Table S14. Quadrupole moment spherical to cartesian conversion.

$\Theta_{xx} = -\frac{1}{2}Q_{20} + \frac{1}{2}\sqrt{3}Q_{22c}$	(S24)
$\Theta_{yy} = -\frac{1}{2}Q_{20} - \frac{1}{2}\sqrt{3}Q_{22c}$	(S25)
$\Theta_{zz} = Q_{20}$	(S26)
$\Theta_{xy} = \frac{1}{2}\sqrt{3}Q_{22s}$	(S27)
$\Theta_{xz} = \frac{1}{2}\sqrt{3}Q_{21c}$	(S28)
$\Theta_{yz} = \frac{1}{2}\sqrt{3}Q_{21s}$	(S29)

Table S15. Quadrupole moment cartesian to spherical conversion.

$$Q_{20} = \Theta_{zz} \tag{S30}$$

$$Q_{21c} = \frac{2}{\sqrt{3}}\Theta_{xz} \tag{S31}$$

$$Q_{21s} = \frac{2}{\sqrt{3}}\Theta_{yz} \tag{S32}$$

$$Q_{22c} = \frac{1}{\sqrt{3}}\left(\Theta_{xx} - \Theta_{yy}\right) \tag{S33}$$

$$Q_{22s} = \frac{2}{\sqrt{3}}\Theta_{xy} \tag{S34}$$

Table S16. Octupole moment spherical to cartesian conversion.

$$\Omega_{xxx} = \sqrt{\frac{5}{8}}Q_{33c} - \sqrt{\frac{3}{8}}Q_{31c} \tag{S35}$$

$$\Omega_{xxy} = \sqrt{\frac{5}{8}}Q_{33s} - \sqrt{\frac{1}{24}}Q_{31s} \tag{S36}$$

$$\Omega_{xyy} = \sqrt{\frac{5}{8}}Q_{33c} - \sqrt{\frac{1}{24}}Q_{31c} \tag{S37}$$

$$\Omega_{yyy} = \sqrt{\frac{5}{8}}Q_{33s} - \sqrt{\frac{3}{8}}Q_{31s} \tag{S38}$$

$$\Omega_{xxz} = \sqrt{\frac{5}{12}}Q_{32c} - \frac{1}{2}Q_{30} \tag{S39}$$

$$\Omega_{xyz} = \sqrt{\frac{5}{12}}Q_{32s} \tag{S40}$$

$$\Omega_{yyz} = -\sqrt{\frac{5}{12}}Q_{32c} - \frac{1}{2}Q_{30} \tag{S41}$$

$$\Omega_{xzz} = \sqrt{\frac{2}{3}}Q_{31c} \tag{S42}$$

$$\Omega_{yzz} = \sqrt{\frac{2}{3}} Q_{31s} \tag{S43}$$

$$\Omega_{zzz} = Q_{30} \tag{S44}$$

Table S17. Quadrupole moment cartesian to spherical conversion.

$$Q_{30} = \Omega_{zzz} \tag{S45}$$

$$Q_{31c} = \sqrt{\frac{3}{2}} \Omega_{xzz} \tag{S46}$$

$$Q_{31s} = \sqrt{\frac{3}{2}} \Omega_{yzz} \tag{S47}$$

$$Q_{32c} = \sqrt{\frac{3}{5}} \left(\Omega_{xxz} - \Omega_{yyz}\right) \tag{S48}$$

$$Q_{32s} = 2\sqrt{\frac{3}{5}} \Omega_{xyz} \tag{S49}$$

$$Q_{33c} = \sqrt{\frac{1}{10}} \left(\Omega_{xxx} - \Omega_{xyy}\right) \tag{S50}$$

$$Q_{33s} = \sqrt{\frac{1}{10}} \left(3\Omega_{xxy} - \Omega_{yyy}\right) \tag{S51}$$

Table S18. Hexadecapole moment spherical to cartesian conversion.

$$\Phi_{xxxx} = \frac{3}{8} Q_{40} - \frac{1}{4}\sqrt{5} Q_{42c} + \frac{1}{8}\sqrt{35} Q_{44c} \tag{S52}$$

$$\Phi_{xxxy} = \frac{1}{8}\left(-\sqrt{5} Q_{42s} + Q_{44s}\right) \tag{S53}$$

$$\Phi_{xxyy} = \frac{1}{8} Q_{40} - \frac{1}{8}\sqrt{35} Q_{44c} \tag{S54}$$

$$\Phi_{xyyy} = -\frac{1}{8}\left(-\sqrt{5}Q_{42s} + \sqrt{35}Q_{44s}\right) \tag{S55}$$

$$\Phi_{yyyy} = \frac{3}{8}Q_{40} + \frac{1}{4}\sqrt{5}Q_{42c} + \frac{1}{8}\sqrt{35}Q_{44c} \tag{S56}$$

$$\Phi_{xxxz} = \frac{1}{16}\left(-3\sqrt{10}Q_{41c} + \sqrt{70}Q_{43c}\right) \tag{S57}$$

$$\Phi_{xxyz} = \frac{1}{16}\left(-\sqrt{10}Q_{41s} + \sqrt{70}Q_{43s}\right) \tag{S58}$$

$$\Phi_{xyyz} = -\frac{1}{16}\left(\sqrt{10}Q_{41c} + \sqrt{70}Q_{43c}\right) \tag{S59}$$

$$\Phi_{yyyz} = -\frac{1}{16}\left(3\sqrt{10}Q_{41s} + \sqrt{70}Q_{43s}\right) \tag{S60}$$

$$\Phi_{xxzz} = -\frac{1}{2}Q_{40} + \frac{1}{4}\sqrt{5}Q_{42c} \tag{S61}$$

$$\Phi_{xyzz} = \frac{1}{4}\sqrt{5}Q_{42s} \tag{S62}$$

$$\Phi_{yyzz} = -\frac{1}{2}Q_{40} - \frac{1}{4}\sqrt{5}Q_{42c} \tag{S63}$$

$$\Phi_{xzzz} = \sqrt{\frac{5}{8}}Q_{41c} \tag{S64}$$

$$\Phi_{yzzz} = \sqrt{\frac{5}{8}}Q_{41s} \tag{S65}$$

$$\Phi_{zzzz} = Q_{40} \tag{S66}$$

Table S19. Hexadecapole moment cartesian to spherical conversion.

$$Q_{40} = \Phi_{zzzz} \tag{S67}$$

$$Q_{41c} = \sqrt{\frac{8}{5}}\Phi_{xzzz} \tag{S68}$$

$$Q_{41s} = \sqrt{\frac{8}{5}}\Phi_{yzzz} \tag{S69}$$

$$Q_{42c} = 2\sqrt{\frac{1}{5}}\left(\Phi_{xxzz} - \Phi_{yyzz}\right) \tag{S70}$$

$$Q_{42s} = 4\sqrt{\frac{1}{5}}\Phi_{xyzz} \tag{S71}$$

$$Q_{43c} = 2\sqrt{\frac{2}{35}}\left(\Phi_{xxxz} - 3\Phi_{xyyz}\right) \tag{S72}$$

$$Q_{43s} = 2\sqrt{\frac{2}{35}}\left(3\Phi_{xxyz} - \Phi_{yyyz}\right) \tag{S73}$$

$$Q_{44c} = \sqrt{\frac{1}{35}}\left(3\Phi_{xxxx} - 6\Phi_{xxyy} + \Phi_{yyyy}\right) \tag{S74}$$

$$Q_{44s} = 4\sqrt{\frac{1}{35}}\left(\Phi_{xxxy} - \Phi_{xyyy}\right) \tag{S75}$$

As mentioned previously, in between the conversions from spherical to Cartesian and back, the cartesian moments must be rotated using the C matrix via an n-rank Cartesian tensor rotation demonstrated by the following sets of equations:

$$\mu_i' = \sum_a C_{ia}\mu_a \tag{S76}$$

$$\Theta_{ij}' = \sum_a \sum_b C_{ia}C_{jb}\Theta_{ab} \tag{S77}$$

$$\Omega_{ijk}' = \sum_a \sum_b \sum_c C_{ia}C_{jb}C_{kc}\Omega_{abc} \tag{S78}$$

$$\Phi_{ijkl}' = \sum_a \sum_b \sum_c \sum_d C_{ia}C_{jb}C_{kc}C_{ld}\Phi_{abcd} \tag{S79}$$

9 Atomic Constants

Table S20. Table of atomic constants used within ICHOR.

Atom Symbol	Atomic Mass[1] (A.U)	Atomic Radius[2] (Å)	Van der Waals Radius[3] (Å)	Electronegativity[4]
H	1.00783	0.37	0.430	2.20
He	4.0026	0.32	0.741	0.00
Li	7.01601	1.34	0.880	0.98
Be	9.01218	0.90	0.550	1.57
B	11.0093	0.82	1.030	2.04
C	12.0000	0.77	0.900	2.55
N	14.0031	0.75	0.880	3.04
O	15.9949	0.73	0.880	3.44
F	18.9984	0.71	0.840	3.98
Ne	19.9924	0.69	0.815	0.00
Na	22.9898	1.54	1.170	0.93
Mg	23.9850	1.30	1.300	1.31
Al	26.9815	1.18	1.550	1.61
Si	27.9769	1.11	1.400	1.90
P	30.9738	1.06	1.250	2.19
S	31.9721	1.02	1.220	2.58
Cl	34.9689	0.99	1.190	3.16
Ar	39.9624	0.97	0.995	0.00
K	38.9637	1.96	1.530	0.82
Ca	39.9626	1.74	1.190	1.00
Sc	44.9559	1.44	1.640	1.36
Ti	47.9480	1.36	1.670	1.54
V	50.9440	1.25	1.530	1.63
Cr	51.9405	1.27	1.550	1.66
Mn	54.9381	1.39	1.555	1.55
Fe	55.9349	1.25	1.540	1.83
Co	58.9332	1.26	1.530	1.88
Ni	57.9353	1.21	1.700	1.91
Cu	62.9296	1.38	1.720	1.90
Zn	63.9291	1.31	1.650	1.65
Ga	68.9256	1.26	1.420	1.81
Ge	73.9212	1.22	1.370	2.01
As	74.9216	1.19	1.410	2.18
Se	79.9165	1.16	1.420	2.55

Br	78.9183	1.14	1.410	2.96
Kr	83.9115	1.1	1.069	3.00
Rb		2.11	1.670	
Sr		1.92	1.320	0.95
Y		1.62	1.980	1.22
Zr		1.48	1.760	1.33
Nb		1.37	1.680	
Mo		1.45	1.670	2.16
Tc		1.56	1.550	
Ru		1.26	1.600	2.2
Rh		1.35	1.650	2.28
Pd		1.31	1.700	2.20
Ag		1.53	1.790	1.93
Cd		1.48	1.890	
In		1.44	1.830	1.78
Sn		1.41	1.660	
Sb		1.38	1.660	2.05
Te		1.35	1.670	2.10
I		1.33	1.600	2.66
Xe		1.30	1.750	
Cs		2.25	1.870	
Ba		1.98	1.540	0.89
La		1.69	2.070	
Ce			2.030	1.12
Pr			2.020	
Nd			2.010	
Pm			2.000	
Sm			2.000	
Eu			2.190	
Gd			1.990	1.20
Tb			1.960	
Dy			1.950	
Ho			1.940	
Er			1.930	
Tm			1.920	
Yb			2.140	
Lu		1.6	1.920	
Hf		1.5	1.770	
Ta		1.38	1.630	

W	1.46	1.570	12.36
Re	1.59	1.550	
Os	1.28	1.570	
Ir	1.37	1.520	
Pt	1.28	1.700	
Au	1.44	1.700	2.54
Hg	1.49	1.900	
Tl		1.750	
Pb		1.740	
Bi	1.48	1.740	
Po	1.47	1.880	
At	1.46	0.200	2.02
Rn	1.45	0.200	
Fr		0.200	
Ra		2.100	
Ac		2.080	
Th		1.990	
Pa		1.810	
U		1.780	
Np		1.750	
Pu		0.200	
Am		1.710	
Cm		0.200	
Bk		0.200	
Cf		1.730	

References

1. Meija, J.; Coplen, T. B.; Berglund, M.; Brand, W. A.; De Bièvre, P.; Gröning, M.; Holden, N. E.; Irrgeher, J.; Loss, R. D.; Walczyk, T.; Prohaska, T., Isotopic compositions of the elements 2013 (IUPAC Technical Report). *Pure and Applied Chemistry* **2016,** *88*, 293-306.

2. Clementi, E.; Raimondi, D. L., Atomic Screening Constants from SCF Functions. *The Journal of Chemical Physics* **1963,** *38*, 2686-2689.

3. Alvarez, S., A cartography of the van der Waals territories. *Dalton Transactions* **2013,** *42*, 8617-8636.

4. Allred, A. L., Electronegativity values from thermochemical data. *Journal of Inorganic and Nuclear Chemistry* **1961,** *17*, 215-221.