Simulation Science Laboratory 2020

Analysis Tool for a Materials Design Database

Students:

Sijie Luo
Zhipeng Tan
Miao Wang

Supervisors:

Stefan Blügel Prof.Dr.
Daniel Wortmann.Dr.
Jens Bröder
Johannes Wasmer
Forschungszentrum Jülich

Abstract — AiiDA, a robust open-source high-throughput infrastructure, can fully record automated workflows and data sources. An AiiDA database can deal with any simulations. When such a database grows reasonably large, suitable tools are needed to analyze and visualize it. Here we present a statistical tool which can be used to analyze such databases. Our tool builds on one of the distinguishing features of AiiDA databases. Namely, they track the provenance of simulation data, by storing inputs, calculations and results in a directed acyclic graph and a directed graph for the logic. A materials scientist who uses AiiDA for his or her simulations can use our tool to quickly produce interactive visual summaries of a database via Jupyter notebooks. One such notebook produces a statistical birds-eye view of the database. Another provides an interactive structure-property visualizer for materials simulation workflows.

Keywords exploratory data analysis, visual data exploration, scientific data management, simulation data provenance, computational materials science, AiiDA

AICES
Schinkelstr. 2
Rogowski
Building 4th
Floor
52062 Aachen

Acknowledgments

The authors are deeply indebted to the following members of the section Quantum Theory of Materials in the Peter-Grünberg Institute at the Forschungszentrum Jülich: our supervisors Prof. Dr. Stefan Blügel and Dr. Daniel Wortmann for their helpful input and useful suggestions; Jens Bröder and Johannes Wasmer for their instructive advice.

Contents

1 Introduction **1**

Problem Statement 1
Motivation and Requirements 2
Project Steps 3

2 Theoretical Background **4**

3 Implementation **8**

Statistical birds-eye view of the contents in an AiiDA database. 8
Structure-property visualizer 13

4 Applications **16**

Statistical birds-eye view of the contents in an AiiDA database. 16
Structure-property visualizer. 24

5 Conclusion & Outlook **28**

Chapter 1

Introduction

Problem Statement

In computational materials science, we aim to understand various properties and phenomena of materials, such as magnetism, and achieve designing and making better materials for society. It is amazing how much computing power and computational methods have developed over the past few decades. But besides the high-performance hardware, some powerful infrastructure or tools are also required to assist researchers, helping them with retrieving, handling, analyzing and interactively visualizing complex data associated with modern computational science.

The tool we created is based on AiiDA (Automated Interactive Infrastructure and Database for Computational Science), a robust open-source high-throughput infrastructure, which is widely applied in computational materials science. It aims to address the challenges arising from the need for automated workflow management and data provenance recording[1]. With AiiDA, one can easily reconstruct the complete history of each calculation or scientific result, and for instance, visualize these things in a provenance graph.

However, when the AiiDA database is too large, or when the simulation progress is relatively complex, the provenance graph is not sufficient to visualize the results or content in the database clearly and concisely. Although the built-in facilities of this infrastructure are great for inspecting and traversing through inputs, workflows and outputs for individual sets of simulations, it yet lacks the ability to produce quick statistical overviews over larger sets of hundreds to millions of stored simulations.

Therefore, we developed a Python package with user frontends in the form of Jupyter notebooks (deliverables), which implements the functions of (1) getting a statistical birds-eye view of the contents in an AiiDA database, (2) generating structure-property visualizations over large sets of simulations. We call them deliverable 1 and deliverable 2. The whole pipeline, from raw data querying from an AiiDA database, data transformation with de-/serialization, to interactive selection and visualization of selected data, is addressed in this project.

Motivation and Requirements

The goal of the project is to develop an analysis tool, which is able to extract all the necessary information from an AiiDA database in a preprocessing pipeline, store the output into files, which is then utilized to implement the exploratory data analysis.

This imposes several requirements on the software:

a) Functional requirements

- Serialization and deserialization: The tool should be able to write/read the data to/from a file. The serialization helps to avoid tedious retrieving for the next time, and also enables (faster) data selection, filtering, etc.
- Interactive plots: The visualization produced with the tool should be interactive to get extra helpful information and implement real-time parameters change.
- Export features: Plots should be exportable in common formats (HTML, PDF, PNG). Simulation property output files should be human-friendly to read, whenever this makes sense (JSON, EXCEL).

b) Non-functional requirements

- Runtime performance: In AiiDA, the size of a large database can easily exceed one million nodes. Therefore, the preprocessing and visualization technique should be as efficient as possible, and scale to large databases. The measurement of the runtime is broken into the following steps:
  - ✧ data acquisition (i.e., querying),
  - ✧ de-/serialization,
  - ✧ data visualization.
- User-friendliness: The tool should be as easy to use as possible for both solid-state physics researchers and non-expert common users. This involves,
  - ✧ building a clear and intuitive interface for users,
  - ✧ producing useful error messages,
  - ✧ providing some basic default/predefined variables,
  - ✧ designing well-arranged control panels in the interactive plots.
- Easy to access: The tool should be able to run in different environments smoothly with minimum additional setup.
- Easy to extend and maintain: The deliverables and modules we created are maintained via the Github library aiida-jutools[2]. To keep pace with external improvements and new tools, the code should adhere to the common Python code style guide PEP8 [3].

Project Steps

The project was organized into several steps:

- Understanding the problem: Get acquainted with AiiDA. Gain knowledge of the terminology and components (e.g. provenance, data node format, etc.), and get familiar with basic operations (e.g. working with small databases, grouping nodes, querying for data, etc.).
- Querying (data reduction): Write functions for more complex queries to transform AiiDA data using appropriate projections.
- Serialization and deserialization: Create a Python module that could achieve the goal of writing and reading the query results to/from a file. Decide on data transformation, serialization format.
- Data cleansing: Investigate possible challenges that might occur during visualizing the data source we get. Come up with methods which can deal with problems such as, points on the plot are covering each other, missing values in the data source, different workflow versions have different projection attributes.
- Visualization: Utilize the Bokeh package to implement the interactive visualization of the data[4].
- Testing and debugging: The usability of the deliverables should be expanded to general AiiDA databases, of which the workflows and the number of nodes remain unknown. Thus, the tool should be tested on different toy and real-world databases, and the portability to different databases should be taken into consideration.
- Runtime performance measurement: Run the Jupyter notebooks intended for use by the materials scientist (the deliverables) on various databases and record the timings.

Chapter 2

Theoretical Background

AiiDA

To introduce our laboratory work, the AiiDA framework must be first discussed as well as their basic concepts. The automated Interactive Infrastructure and Database for Computational Science, shortly AiiDA, is an open source infrastructure designed especially for computational science to manage scientific workflows in a more efficient and reproducible way. AiiDA will record calculation workflows into a provenance graph and store the calculation results in a relational database automatically, mainly with Postgresql backend. Thus it can provide researchers easier management of their data and complies with the FAIR principles[11] for scientific data.

Some key features of AiiDA make it widely used in computational materials science, such as the automation of high-throughput calculations on HPC systems, which support the high computational workload of many institutes and research centers. Hence AiiDA databases can be easily managed although the database sizes can scale to over millions of nodes. AiiDA's python interface also provides developers the convenience when querying the AiiDA databases in the development stage of this project.

AiiDA Plugins

The AiiDA framework provides developers a flexible plugin-based python environment [1]. An AiiDA plugin is a developer-defined Python package which adds new classes to AiiDA's unified interface. In this project, there mainly used plugins are aiida-fleur and aiida-kkr which was developed by the supervisors of the current projects from PGI-1/ IAS-1 of Forschungszentrum Jülich.

Those plugins connect the computational framework AiiDA and Materialscloud with the simulation code that the Institute has developed, namely the FLEUR program for density functional theory (DFT) and juKKR for the full-potential relativistic Korringa-Kohn-Rostoker Green function (KKR) method [5]. With the help of plugins, those FORTRAN simulation code executions can be simplified by the AiiDA Python interface while gaining the advantage of recording the provenance automatically.

Data Provenance

As was mentioned before, data provenance is one of the most important information stored in an AiiDA database. It will be explained in more detail in this part. Firstly, Figure 2.1 shows three different representations of a provenance graph of a representative toy calculation. The figure 1.a) shows how the data are calculated. D1-D5 represents different data, triangle C1 and C2 represent calculation and multiplication of two data. Figure 1.b) shows the information recorded in the calculation node with solid line representing the creation of data, while figure 1.c) shows workflow with dashed lines and the corresponding calculation steps are recorded in figure 1.b).

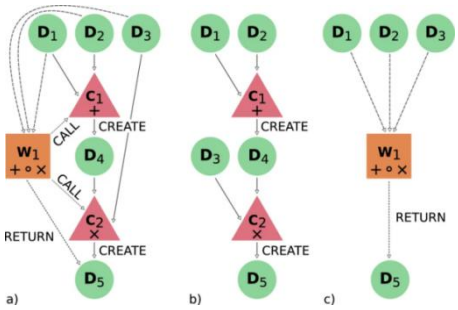


Figure 2.1: Simple provenance graphs recorded by AiiDA. [1]

As one can see from the picture above, data provenance represents a record trail that accounts for the origin of a piece of data together with an explanation of how and why it got to the present place. Thus with the provenance recorded, it becomes easier to answer questions such as “How and why are the results created”, “Who created the results” and “where are they created”.

As the image also showed, there are nodes and links in such graphs. Nodes here represent Data and Processes in AiiDA, whereas links connect inputs, calculations and outputs of a process or create and return the new data back to the workflow. Each node can be queried specifically by its UUID (universally unique identifier) [1].

Data and Process

The node types of AiiDA are divided into a hierarchy as the Figure 2.2 shows.

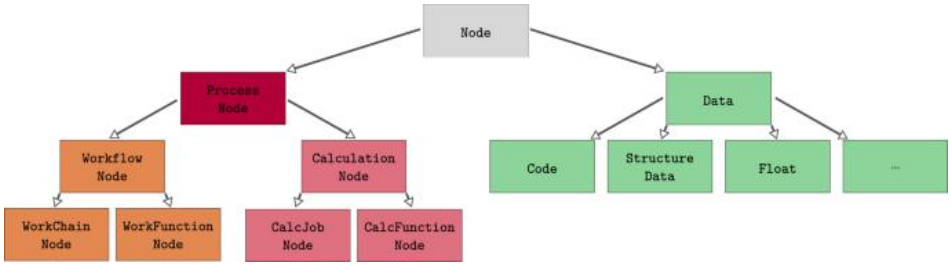


Figure 2.2: Nodes hierarchy. [1]

Data nodes are regarded as input or output of process nodes, while process nodes represent simple calculations or more complex workflows.

Data

The data types of AiiDA are very similar to the data types of other programming languages. AiiDA offer on the one hand wrapped versions of basic data structures,such as `Int`, `Float`, `Dict` etc., and on the other hand, more complex types needed for simulation codes, such as wrappers for numpy arrays, `StructureData` as interface for various crystal `StructureData` formats, or `FolderData` for referencing output files stored in a file repository. The `Computer` and `Code` types are special types which connect the system with compute services such as HPC clusters, and the underlying codes installed there which perform the actual simulations.

Process

Any representation of a computation in AiiDA is an instance of the Process class. The Process class contains all the information and logic to tell, which service is handling it, how to run it to completion. When a process is executed, a node is created in the database as its representative (`CalculationNode`, `WorkflowNode`). While the process ends at some time, the node remains. As calculations and workflows represent different provenance concepts, this distinction must be further elucidated.

Calculations create the new data with 2 subclasses as figure 2 shows. To be stored in the provenance graph automatically, the calculation function must be decorated by `@caclfunction` Python decorator. When running calculations that require an external code or run on a remote machine, a simple calculation function is no longer sufficient. For the purpose of recording external code, AiiDA provides the `CalcJob` process class[2].

Workflow automates the execution of processes, and stores the provenance graph for reproducibility. As the figure 2 shows, the subclasses of Workflow are `workfunction` and `workchain` node, see the figure 2.2 above. `Workfunction` represent those not computationally intensive Python functions decorated with corresponding decorator `@workfunction`, whereas Workchain recorded multiple logical steps and will save the progress once the operation succeed. Once a `CalcJob` process is represented by a `CalcJobNode` in the database, while a `workchain` process is represented by a `WorkchainNode`.

The AiiDA plugins mentioned before enable the execution of external simulation software and they come with specific `workchain` subclasses, specific `calcjob` subclasses and additional data types implemented by the plugin developers. Thus AiiDA supports researchers by recording the simulation provenance which was executed in more low level programming languages such as FORTRAN and C++.

Materials science specific data types

AiiDA was firstly developed for computational solid-state physics and computational chemistry. So some data types are specific to this area and not yet refactored into a plugin, such as `StructureData`, `UpfData`, `KpointsData` etc. Among them, `StructureData` are especially important in the context of this project, so it will also be explained here in more detail.

`StructureData` represents the structure of a crystal unit cell, with specific atoms on specific crystal lattice positions. After the atoms and the positions are defined, there are other variables that must be recorded, such as the elements, cell volume etc. All those information are recorded in an object of a `StructureData` class in AiiDA.

Chapter 3

Implementation

In this chapter, we will solve the two deliverables mentioned in the introduction. The task code of this project stable and well in a small database. As it is tested, by the supervisors on a larger database so that they can be run on any database. We complete the task code on the 'deliverable' python notebooks, and run the code in the AiiDA[6] environment using the virtual machine. We mainly use the database provided by the institute and superviors..

The task is divided into two parts, the statistical analysis of the AiiDA database and the structural attribute visualization tool.The two user frontends (deliverables) have different target audiences. Deliverable 1 (D1, which generates statistical summary) is usable by any AiiDA user. Since AiiDA-wise, it only relies on the aiida-core package. Deliverable 2 (D2, structure property visualizer) is, for now, implemented for aiida-fleur, aiida-kkr plugins users. However, it will be easily extended to support the workflows of other plugins, such as Quantum ESPRESSO.

We need to query the required data in the database, improve the running time by serializing the data, and visualize the data in the form of interactive plots. In this chapter, we will introduce how we use the Bokeh tool to achieve the tasks in the project.

Deliverable1 Statistical birds-eye view of the contents in an AiiDAdb

Performance Plots

Figure 3.1 shows the performance scaling with different database sizes for each part in this project, where the databases we had lying around. Generally one can see a linear scaling except for provenance analysis, which may result from the querying of incoming and outgoing nodes of each node. What one can observe is also an fixed outlier with the second largest database size for some parts, the extra run time may result from

relatively more processes or larger structures.

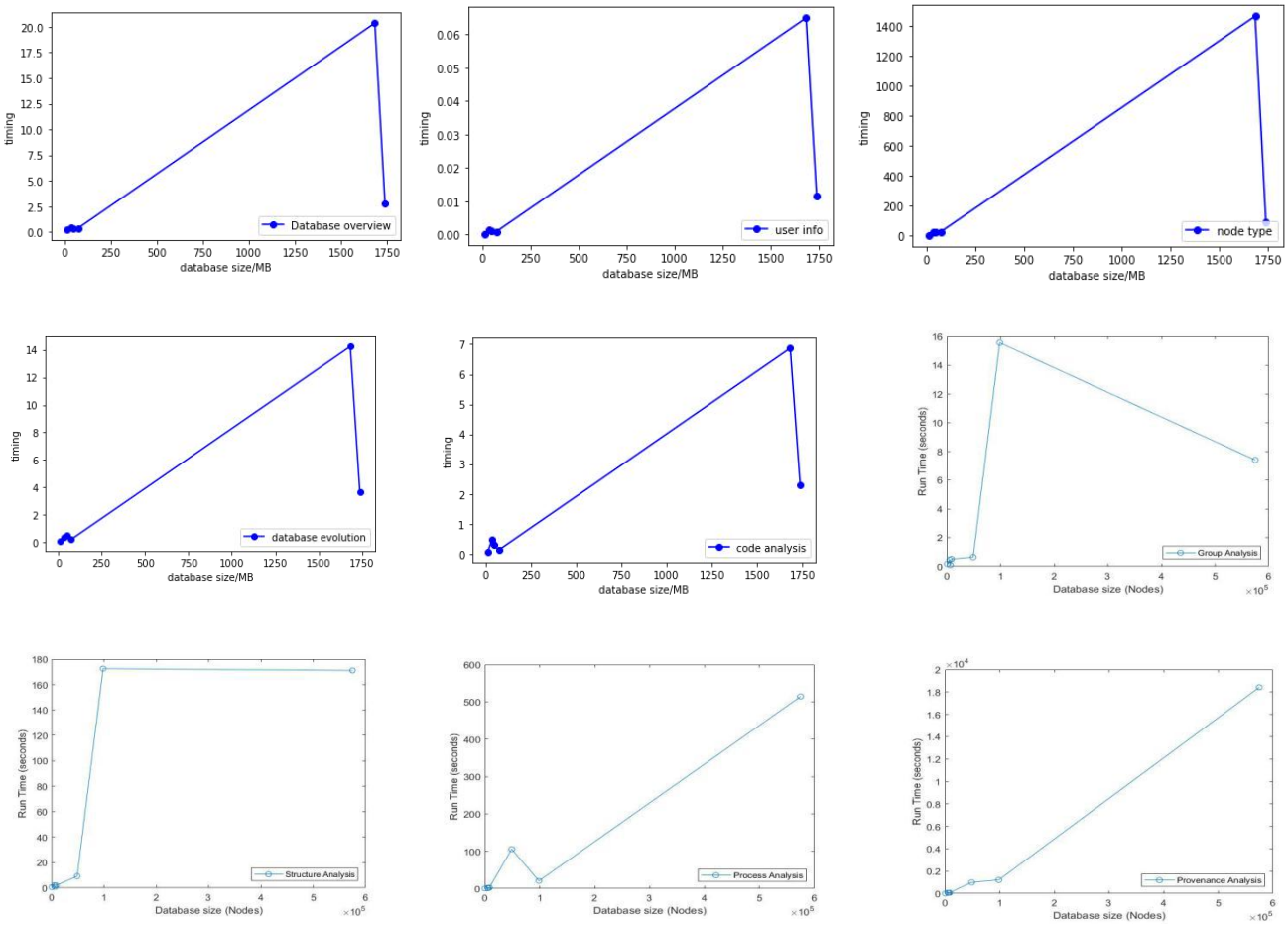


Figure 3.1: Run time scaling for different subtasks

Database Overview

When the data stored in the database is very large, we need to use AiidA. The data in the AiiDA database is stored as a graph of connected entities [7]. One can query the data stored by AiiDA in a Postgresql database directly via SQL commands, however for more complex queries on a graph this is not very user friendly. AiiDA provides here a tool--the QueryBuilder. Through QueryBuilder we can find the specified content. We need to import QueryBuilder into the code, add the node class to the searched list, and attach it to the graph to search, which is equivalent to finding the vertice of the graph. Count the total number of nodes in the database.

User Information

A database can have contributions by many users. It is important to see who contributed how much data and when. That is why the scond part of the statistical summary is a user list. Add the User class to the search list, narrow the search scope to the User type. The User class contains the user's information, name, and the number of nodes created. List the names of all users in the database, and count the number of nodes created accordingly.

Node types distribution

Nodes have many different types, and different types are projected to different functions. As shown in Figure 2.2, we can see that the node types are mainly divided into two categories, namely data nodes and process nodes. In this task, we will divide the queried nodes into two categories by node name, and then classify the two categories in more detail. By distinguishing the types of data nodes, count the corresponding types and the total number of nodes. Process nodes are divided into two categories, Calculation nodes, and Workflow nodes. Simple workflows will use work functions, and complex workflows will be connected by work chains. The process node types are distinguished, and the corresponding sub types and the total number of nodes are counted. Nodes are the vertices of the graph, and links can be regarded as edges in the graph. We query the incoming link label of the Dict node in the data node and count it. We will use the Bokeh tool to complete the interactive visualization of the statistical data with pie plots.

Database time evolution

The visualization libraries used for Python are usually Matplotlib, Seaborn, Plotly, Bgplot, Bokeh, Pyecharts, etc. Seaborn and Matplotlib have relatively limited functions and generate static graphs. Although Pyecharts has beautiful dynamic effects, the controls cannot be customized. Bokeh is an interactive visualization Python library specifically for the rendering function of web browsers, which supports modern web browser display (It can be output as a JSON object, HTML document or interactive web application), this is the core difference between Bokeh and other visualization libraries. It provides an elegant and concise graphical style of D3.js and extends this functionality to high-performance interactive data sets and data streams[8]. On the premise that the user name and corresponding node number have been obtained, we query the creation time and modification time of the node. Count the number of nodes according to the changes in time and list them according to user names. We will use the Bokeh tool to complete the interactive visualization of the statistical data with a line plot.

Codes

Add the Code class to the searched list, as part of the graph, they are linked to the calculations. As Code nodes formed by some calculations, we want to retrieve the calculations between them. We query CalcJobNode and select the appropriate filter on node-class. Count the number of Calculation jobs, the name of the Code, and the name of the running computer.

Performance

The main requirement of the preprocessing and query function should be the performance, such that the explosion of running time must be avoided. To achieve the performance requirement, serialization and deserialization of was implemented to all those parts. Namely one queries firstly the useful data from database and then save the data to output file and visualize them, the database will only be queried when there is no output file with the .json format. See the following figure of those functions:

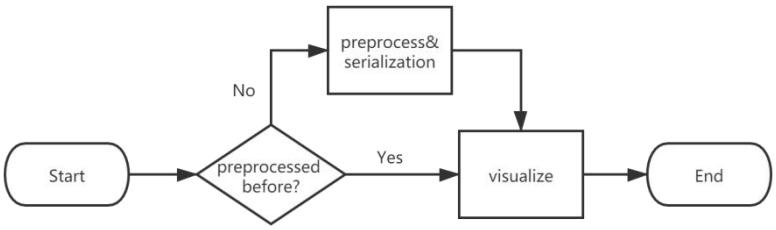


Figure 3.2: Serialization process

The performance gain of implementing serialization is quite satisfying. See the following chart of speedup. The speed-up of the huge computational functions process and provenance analysis will go up to 200, which can save the shareholders much time when they tried to analyse those parts.

	Groups analysis	Structures analysis	Processes analysis	Provenance analysis
Unserialized	0.6411(s)	9.2704(s)	106.0234(s)	1002.4026(s)
Serialized	0.1500(s)	0.6174(s)	1.4370(s)	4.2827(s)
Speed-Up	4.3	15.1	73.8	234.1

Table 3.1: Run time comparison of serialization.

Interactive Plots

As mentioned in the introduction part, interactive plots are also the required visualization outputs of this analysis tool. Bokeh hover tool is implemented in the interactive plots for the following advantages:

1. Bokeh is an open source package with Python API;
2. The zoom in and drag tool allow image to show the desired part robust to outliers;
3. When mouse hovers on the points or bar, extra information will be showed;

The results of all the interactive plots will be shown in chapter 4.

Chapter 3.Implementation

Process of the tasks

For each task, the general process of implementation is firstly extracting the desired data from the database or reading data from the output file, and then visualizing these data. The differences between those functions are the desired data and the corresponding ways of visualization. The implementation of each function part is listed below:

- **Groups Analysis:** Analyze all group names with how many nodes they contain, exclude certain nodes we don't want to count.
- **Structure Analysis**: Further analyze what structures are in the DB, their chemical formulas and their compositions.
- **Process Analysis**: Detail analysis of calculations and workflow. We want to analyze their exit status, exit message and exit code.
- **Provenance Analysis**: Analyze the health of the DB. Display the number of nodes that have no incoming links (any number outgoing), no outgoing links (any number incoming), and neither.

Deliverable2 Structure-property visualizer

The second deliverable is a structure-property visualizer. Considering that the visualization of the result nodes is relatively easier than the visualization of the input nodes, our challenge for deliverable 2 (D2) is to connect the input "structure" to the properties in the output plot. To realize the goal, there are two steps (see Figure. 3.3 a): (1) Data acquisition. Extract float data in certain nodes and transform it into Pandas objects[9], which will be then used as a data source. (2) Interactive plot. Create an interactive scatter plot with linked histograms by a Bokeh server application.

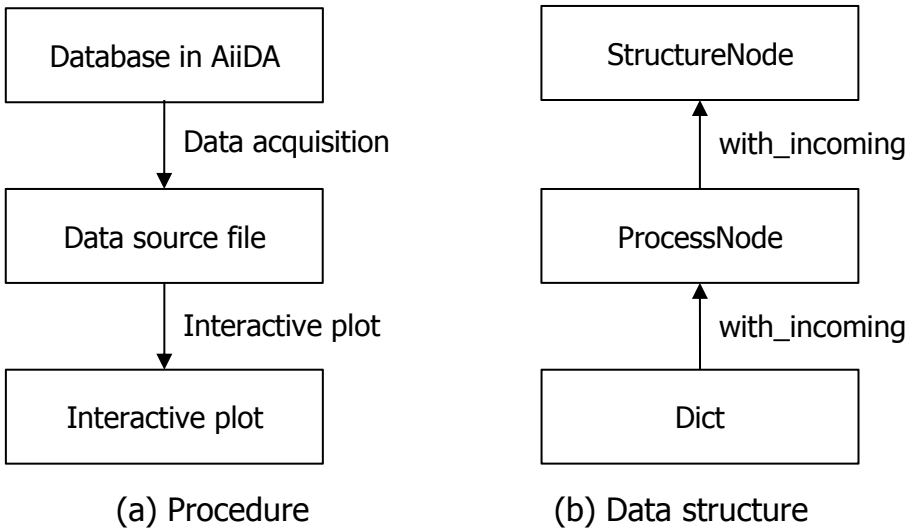


Figure 3.3: Procedure and data structure. The arrow indicates dataflow in (a) and dependency in (b).

Data acquisition

1. Data structure & Data source

In AiiDA, a typical case of data structure is employing a `StructureNode` as the input, which is then appended by arbitrary number of processes denoted by `ProcessNode` for simulation implementation, and then appended by a `DictNode` to store the calculation results, see Figure 3.3 b. It serves as the data structure of querying in D2.

The data extracted from AiiDA is stored into a file as the data source, which would then be used for the interactive visualization. There are two available formats for data storage, JSON and XML. JSON was developed with the goal of simplifying the parsing engine. Thus, JSON is faster than XML, and it is a great choice when transferring small amounts of data that is short-lived, not complex, and verified correctness is not a concern. In D2, we adopt JSON for storing data of a single workflow version. However, in the scenario of multiple workflow versions, JSON could not meet the requirement of reading/loading various Pandas dataframes simply, and it is not robust. While XML could achieve the target of saving dataframe of each workflow version in each of a single sheet, which makes it human-readable, and it is also a better choice to reject invalid data before it causes a defect. Hence, we apply XML for all workflow versions of

data storing.

2. Performance

In AiiDA, the databases could be easily over several million nodes, i.e. easily over several million nodes. Therefore, the data acquisition procedure should be as efficient as possible.
Here we record the unserialized and serialized runtime for a medium-sized (431 MB) database, which has approximately around 50168 nodes in total, with 800 impurities (defect atoms) embeddings into different elemental host crystals with aiida-kkr plugin.
As shown in Table 3.2, for the serialized scenario, the timings for both three of the `generate_structure_property` function, `generate_dict_property` function and `generate_combined_property` function are shorter than that of the unserialized scenario, with 47.77%, 4.96%, 12.27% respectively, within which the gain of the `generate_structure_property` function is the most significant. Therefore, we could roughly see the necessity of utilizing a serializer. And we could see that the serialized timing performance results for the procedure of acquiring all the data from a medium-sized database (0.6971s, 1.0151s, 0.5114s) are relatively efficient.

Type	**Database size (MB)**	**Nodes**	**Process nodes**	**Data nodes**	**Structure timing (sec)**	**Dict timing (sec)**	**Combine timing (sec)**
Unserialized	431	50168	15529	34639	1.3346	1.0655	0.5829
Serialized	431	50183	15534	34649	0.6971	1.0151	0.5114
Speedup	-	-	-	-	47.77%	4.96%	12.27%

Table 3.2: Runtime for data acquisition of unserialized and serialized scenarios

Interactive plot

Since all along we have been using Python to develop and run our codes, the visualization tool would also be constructed based on Python as well. Bokeh gives users the flexibility and convenience of hosting real-time applications on its own servers. Within the context of the Bokeh Server, the Python code the users write would be converted to a JSON document, which then would be rendered in JavaScript by the client library BokehJS so that the users could view the application in the browser[10]. It serves as a bridge connecting Python and the browser in which we can conveniently host our applications, and no prior knowledge of JavaScript is required for users. Therefore, we decide on implementing the interactive visualization by the Bokeh Server application in D2, see Figure 3.4.

In the interactive visualization pipeline (Figure 3.4 a), we filter out the unavailable dataframes by thresholds of the number of scatter points nodes and the number of plottable attributes. In the Bokeh application (Figure 3.4 b), we construct the structure in three steps: (1) create scatter plots with linked histogram and set up useful widgets, (2) define callback functions, add them on objects to trigger when attributes are changed. (3) arrange the control panel and create layouts for plots and widgets.

(a) Interactive visualization pipeline

(b) Structure of Bokeh application

Figure 3.4: Implementation of interactive plot

Chapter 4

Applications

D1 and D2 of the Python notebook can be executed on any database we have, and a series of interactive plot outputs can be achieved through the Bokeh tool. For the results of D1, we mainly focus on statistical data (such as :query, deserialization, analysis/visualization), and for the results of D2, we mainly focus on interactive plots.

Deliverable1 Statistical Analysis of an AiiDA database

Database Overview

Use Querybuilder to query the database and make a preliminary understanding. In the output Figure 4.1, we can know that the last execution time is Tue Feb 2021, and there is 48733 node in the database.

```
Information on nodes in the DB:

last executed on Tue Feb  9 13:19:03 2021
Total number of nodes in the database: 48733 (retrieved in 1.4403438568115234 s.)
```

Figure 4.1: Database overview

User Information

We specify the name of the user and the number of nodes to find the content and find the relevant data in the database. In the output Figure 4.2, we can see the user name in the database is johannes.wasmer@gmail.com and the number of corresponding nodes is 48733.

```
Users:
- johannes.wasmer@gmail.com created 48733 nodes
```

Figure 4.2: User Information

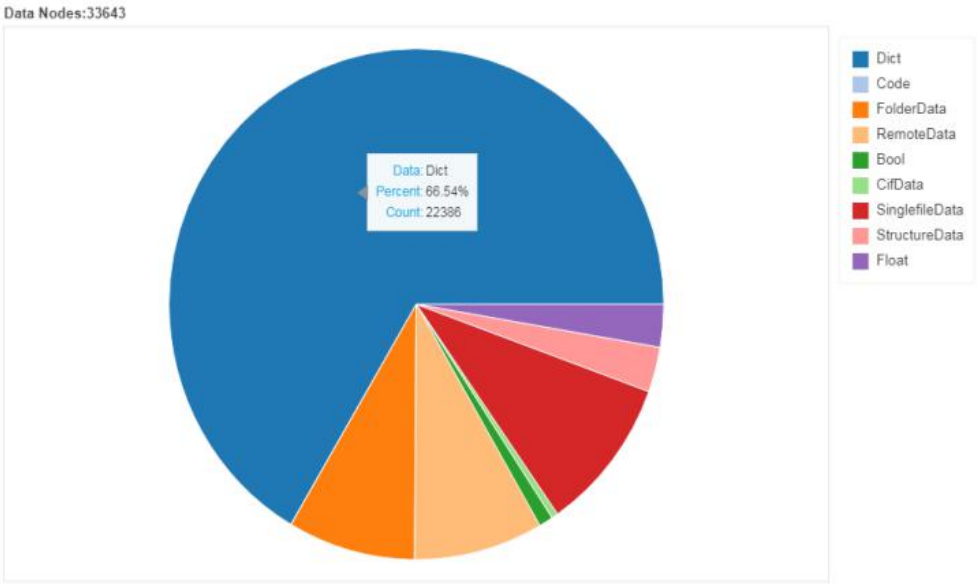
Node Types Distribution

Find all the node types from the database, as shown in the output Figure 4.3, the node type and the corresponding node number.

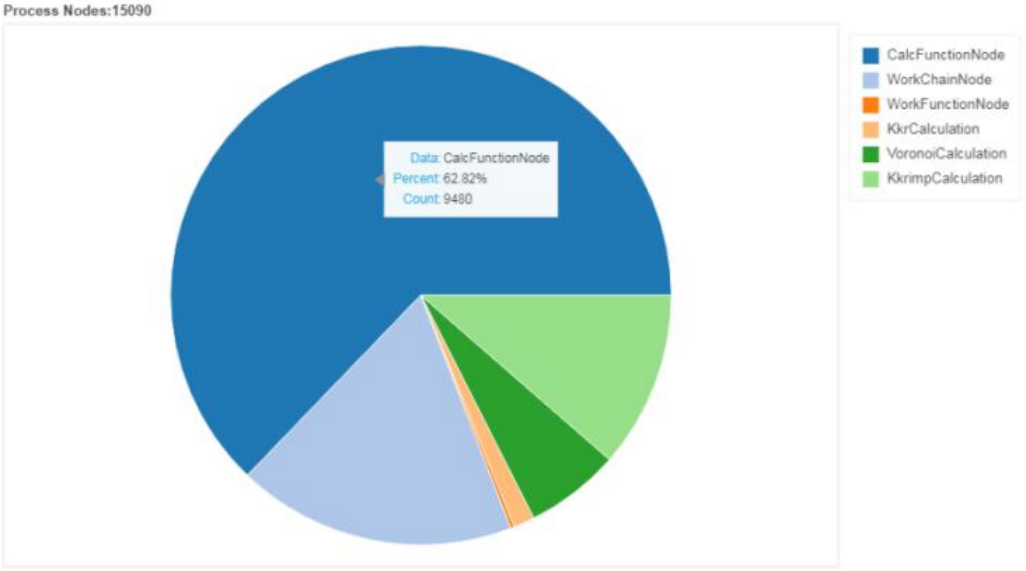
```
Node types:
- data.dict.Dict. created 22386 nodes
- process.calculation.calcfunction.CalcFunctionNode. created 9480 nodes
- data.singlefile.SinglefileData. created 3291 nodes
- process.calculation.calcjob.CalcJobNode. created 2853 nodes
- data.remote.RemoteData. created 2836 nodes
- data.folder.FolderData. created 2806 nodes
- process.workflow.workchain.WorkChainNode. created 2720 nodes
- data.structure.StructureData. created 956 nodes
- data.float.Float. created 912 nodes
- data.bool.Bool. created 309 nodes
- data.cif.CifData. created 142 nodes
- process.workflow.workfunction.WorkFunctionNode. created 37 nodes
- data.code.Code. created 5 nodes
```

Figure 4.3: Node types

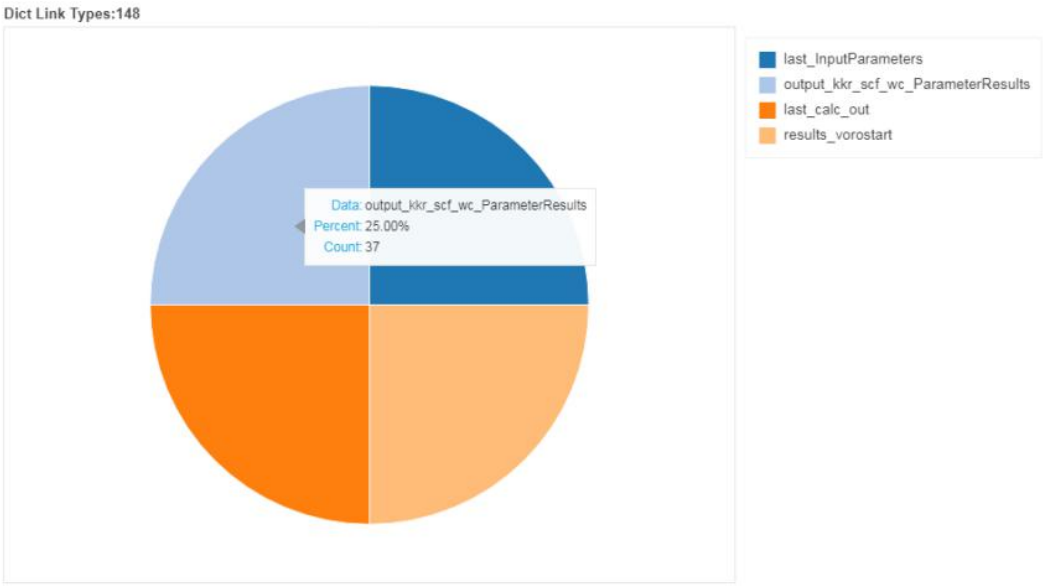
The node types are divided into data nodes and process nodes, as shown in Figures 4.4 (a) and (b). In Figure 4.4(a), the total number of data nodes is 33643. We can see the types of data nodes contained in the pie chart. When the mouse is placed on the pie plot, we can see the corresponding number of nodes, node names, And account for the total percentage of data nodes. In Figure 4.4(b) and (c), the total number of process nodes is 15090. The total number of dict nodes is 148.Pie plot has the same characteristics as (a).



(a)



(b)



(c)

Figure 4.4: (a)data nodes in pie plot. (b)process nodes in pie plot. (c)dict nodes with incoming link labels in pie plot

Database time evolution

Used to look up the creation time and modification time of database nodes, and the creation time and modification time of each user's associated node. In the output Figure 4.5, a line plot is used to show the trend of the number of nodes over time.

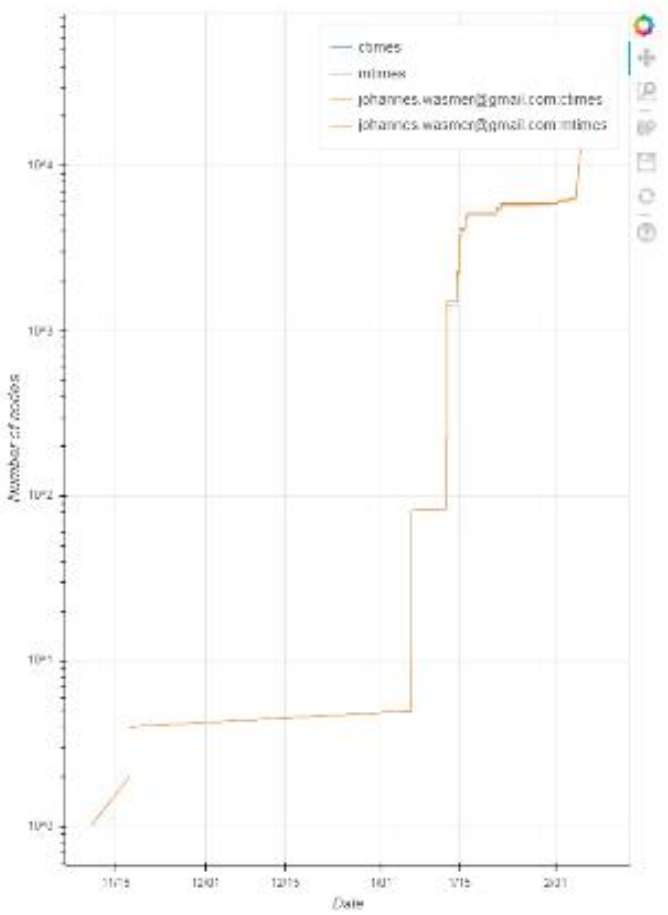


Figure 4.5: Line plot by ctime & mtime of all nodes over time.

Codes

List the code names in the database. In Figure 4.6 we can see the number of calcjobs run each other and sort them from highest to bottom.

	code@computer	CalaJobcount
0	kkrimp@claix18	1726
1	voronoi@localhost	924
2	kkrhost@claix18	204
3	kkrimp@localhost	0
4	kkrhost@localhost	0

Figure 4.6: Codes name

Group Analysis

In AiiDA database, users are allowed to define nodes groups freely, and assign any number of nodes to the groups for their convenience. Thus when facing an unknown database, an often desired information might be the groups in this database. From the group names and group sizes, one can infer what kind of work the users have done and the workload distribution of these groups. The following figure 4.7a shows the groups and sizes of a medium sized database.

	User	Group_Name	Node	type_string
0	j.broeder@fz-juelich.de	20200520-130520	44	core.import
1	j.broeder@fz-juelich.de	20200520-130929	21	core.import
2	j.broeder@fz-juelich.de	20200520-130940	20	core.import
3	j.broeder@fz-juelich.de	20200520-131009	19	core.import
4	j.broeder@fz-juelich.de	20200520-131155	193	core.import
5	j.broeder@fz-juelich.de	20200520-131156	149	core.import
6	j.broeder@fz-juelich.de	20200520-131156_1	52	core.import
7	j.broeder@fz-juelich.de	20200520-131156_2	21	core.import
8	j.broeder@fz-juelich.de	20200520-131157	21	core.import
9	j.broeder@fz-juelich.de	20200520-131157_1	26	core.import
10	j.broeder@fz-juelich.de	20200520-131157_2	26	core.import
11	j.broeder@fz-juelich.de	20200520-131157_3	47	core.import
12	j.broeder@fz-juelich.de	20200520-131157_4	47	core.import
13	j.broeder@fz-juelich.de	20200520-131157_5	2	core.import
14	j.broeder@fz-juelich.de	Element_structures_from_ICSD	1271	core
15	j.broeder@fz-juelich.de	Binary_structures_from_ICSD	30448	core
16	j.broeder@fz-juelich.de	20200617-123459	31719	core.import
17	j.broeder@fz-juelich.de	20200925-132400	47	core.import
18	j.broeder@fz-juelich.de	delta_structures_gustav	71	core
19	j.broeder@fz-juelich.de	delta_parameters_gutstav_soc	71	core
20	j.broeder@fz-juelich.de	20201204-094451	142	core.import
21	j.broeder@fz-juelich.de	20201204-103340	142	core.import
22	j.broeder@fz-juelich.de	Full Database	81470	core

Figure 4.7: (a) Group information

As one can see from the image, many groups here are probably named by the created time and they were actually imported. So occasionally, the researcher may want to filter out some of the groups by name or group type. This tool allow the filter of some node types by specify the type_string. For example, figure 4.7b shows the result when the “import” nodes are filtered.

Group names:	sizes:
Element_structures_from_ICSD	1271
Binary_structures_from_ICSD	30448
delta_structures_gustav	71
delta_parameters_gutstav_soc	71
Full Database	81470

Figure 4.7: (b) Group information after filtering

With the groups, one can understand the work of the users and the database easily.

Structure Analysis

As explained in chapter 2, StructureData are special material science data containing material information, including cell volume, chemical formula, and chemical composition etc. For material structure, the important information would be how large the structures are, and which elements are there in the database. For those 2 purposes the composition should be analyzed.

For the size of the structure, the rough idea was firstly to analyze the number of atoms, since the size of an atom would be hard to define and also differ between different structures. So firstly all StructureData are queried and useful information like

composition is stored. Through the composition, number of atoms of each node is calculated and counted to visualize. The final visualization is shown below in figure 4.8. As one can see from the figure, with x axis the number of nodes and y axis the number of atoms, in this database most of the nodes have only one or two atoms.

With the help of Bokeh hover tool, one can see not only more information but can also interact with the plot. For example if one hover the mouse on the bar, one see there are 416 nodes containing 2 atoms with 5 examples showing UUIDs and formulas of the 416 nodes.

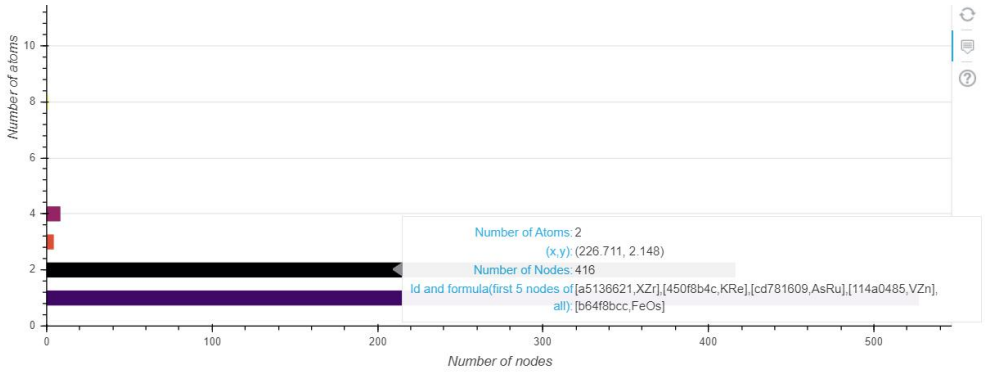


Figure 4.8: Number of atoms

For the elements in the database, the function analyzes the composition similarly. But count how many times each element appears instead of the number of atoms. As the figure 4.9 below shows, the result shows elements versus the number of nodes containing this element. As one can see from the figure, this database has for example 240 nodes containing oxygen element. With this visualization, it is possible to find out what elements are there in the database and also infer the usage of the database. Whether there are more metal elements or more non-metal elements, heavy elements or light elements. Some special elements like magnetic elements iron, nickel and cobalt can also be analyzed accordingly.

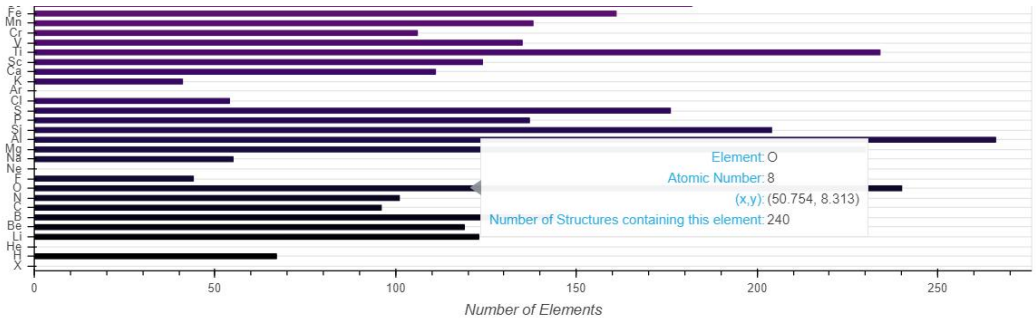


Figure 4.9: Elements in the database

Process Analysis

As expounded in chapter 2, the process nodes of AiiDA define the calculation and workflow of the data nodes. So analyzing the process nodes gives the insight in the work process. Among that information from process nodes, one interesting aspect will be the exit code and exit messages. An example figure 4.10 below shows this information of process nodes. One can see that the exit messages correspond to the exit status, where 0 means succeed. Whenever an “Error” is found in exit message, one can dig deeper into the database and find out what caused the error.

	Node_uuid	Process_State	Exit Status	Exit_Message	node_type	Called by	label
0	460a43bb-7a9e-420b-93df-62e6c68633d0	ProcessState.FINISHED	130.0	ERROR: Last calculation is not in finished state	process.workflow.workchain.WorkChainNode.	a447432c-206f-4d29-b6e1-180e823067f3	kkr_imp_sub_wc
1	dab86744-ddd2-4b8e-8249-90f0bb8355ce	ProcessState.FINISHED	233.0	ERROR: last_remote could not be set to a previ...	process.workflow.workchain.WorkChainNode.	None	kkr_scf_wc
2	f8eb4558-81be-4af3-aaaf-4d91e6867b6b	ProcessState.FINISHED	0.0	None	process.workflow.workchain.WorkChainNode.	52bc0324-c21b-43fe-8bfc-da8be7106da1	kkr_startpot_wc
3	4458ce0d-a4d7-4c5d-8a2c-ee2381d91b50	ProcessState.FINISHED	0.0	None	process.workflow.workfunction.WorkFunctionNode.	b3551aad-745f-4d51-84b8-e10b606bebc5	create_scf_result_node
4	c17dffcd-bc91-478d-a4f5-9ad1d1e643e9	ProcessState.FINISHED	0.0	None	process.workflow.workchain.WorkChainNode.	dab86744-ddd2-4b8e-8249-90f0bb8355ce	kkr_startpot_wc

Figure 4.10: Exit status and exit messages

The next step is naturally to count the percentage of the process nodes that succeed and didn't succeed so that a more general view of the database can be observed. As one can see from the figure 4.11, there most of the workflow nodes have succeed but still some of them failed. With the Bokeh hover tool, one can see more detailed information about how many nodes have succeeded or not as well as the percentages.

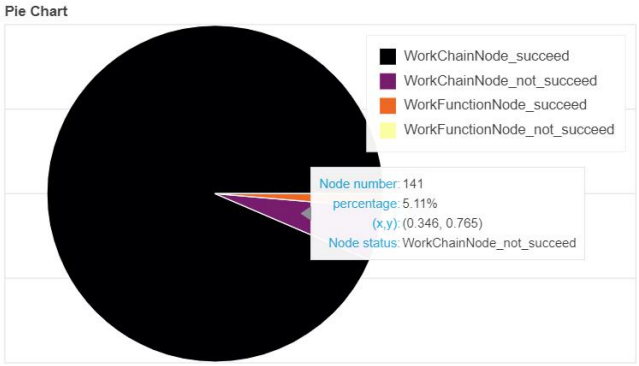


Figure 4.11: Distribution of workflow nodes

Provenance Analysis

Provenance is the automatically recorded directed graph, indicating the complete history of the calculation or workflow. Provenance Analysis gives researchers database-health indicating information besides the process analysis. For any provenance, the most important information must be the connection between nodes. Thus for some general view of the provenance health, this tool will firstly extract the connections of each nodes, creating a table as the image 4.12 below.

	Node_Type	PK	FirstInput	FirstOutput
0	data.code.Code.	50	None	[eddcf30c-0d0f-4aac-93ac-d4bc4dfbd975, {'name'...
1	data.dict.Dict.	10502	None	None
2	data.dict.Dict.	12786	[cd8241a3-b30c-4c5a-b085-3e799ce2b40f, {'name'...	None
3	data.dict.Dict.	10503	None	None
4	data.code.Code.	42	None	[eddcf30c-0d0f-4aac-93ac-d4bc4dfbd975, {'name'...
...	...	...	...	...
6100	data.float.Float.	20015	None	[dac97232-92ed-4ea2-9b8b-5e7df42de12c, {'name'...
6101	data.singlefile.SinglefileData.	20017	[dac97232-92ed-4ea2-9b8b-5e7df42de12c, {'name'...	[a14bd078-8335-4d67-8a66-3b957cb53dd9, {'name'...
6102	process.calculation.calcfunction.CalcFunctionN...	20016	[7b38a16c-fe09-477e-bc63-1d08bdf721f3, {'name'...	[2d019387-4d76-4f6f-872b-36961b177564, {'name'...
6103	process.workflow.workchain.WorkChainNode.	20013	[5df5439b-fbe1-443e-8d69-31779ca3adac, {'name'...	[4c4f373f-ba6f-46d3-8529-3cc355a93f7c, {'name'...
6104	process.calculation.calcjob.CalcJobNode.	20018	[2d019387-4d76-4f6f-872b-36961b177564, {'name'...	[cc12efc6-accc-45fc-8172-5d20ef91b1d6, {'name'...

Figure 4.12: Connection information of nodes

As the table shows, some nodes have only input connections, other nodes have only

outgoing connections and some even have no connections at all. Here especially interesting are those data and process nodes without connections, since they are not used or there are problems with recording. Thus one needs to find out the data and process nodes without connections, this goal is achieved by a stacked bar plot with the data type. See the figure 4.13 of the connection states.

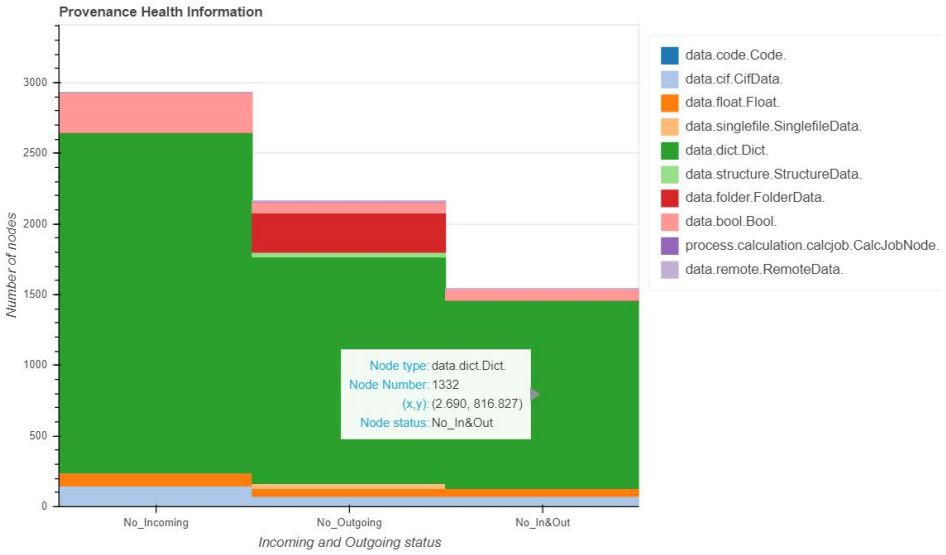


Figure 4.13: Connection states and data type

With the stacked bar plot and Bokeh hover tool, one can specify number of nodes with arbitrary connection states and data type. For example there are 1332 dictionary data nodes that have no connections, which should be analyzed further to see if those 1332 dictionary data nodes are still necessary in the database.

Deliverable2 Structure-property visualizer

User Interface

Endeavoring to be more user-friendly, we design a user interface for users to plug in constants and set the value of some basic variables (see Figure 4.14).
A workflow may contain WorkflowNodes of various versions, which have very different output attributes, which means that we could not retrieve the same attributes from all WorkflowNodes in the database. To deal with this problem, we define in the Helper package the predefined_workflow, where the attributes of commonly-used workflow and parser for two AiiDA plugins, aiida.fleur and aiida.kkr, could be found (see Figure 4.15). It enables users to easily get access to the default Dict projections attributes for plotting, and it avoids the time-consuming setting for projections.

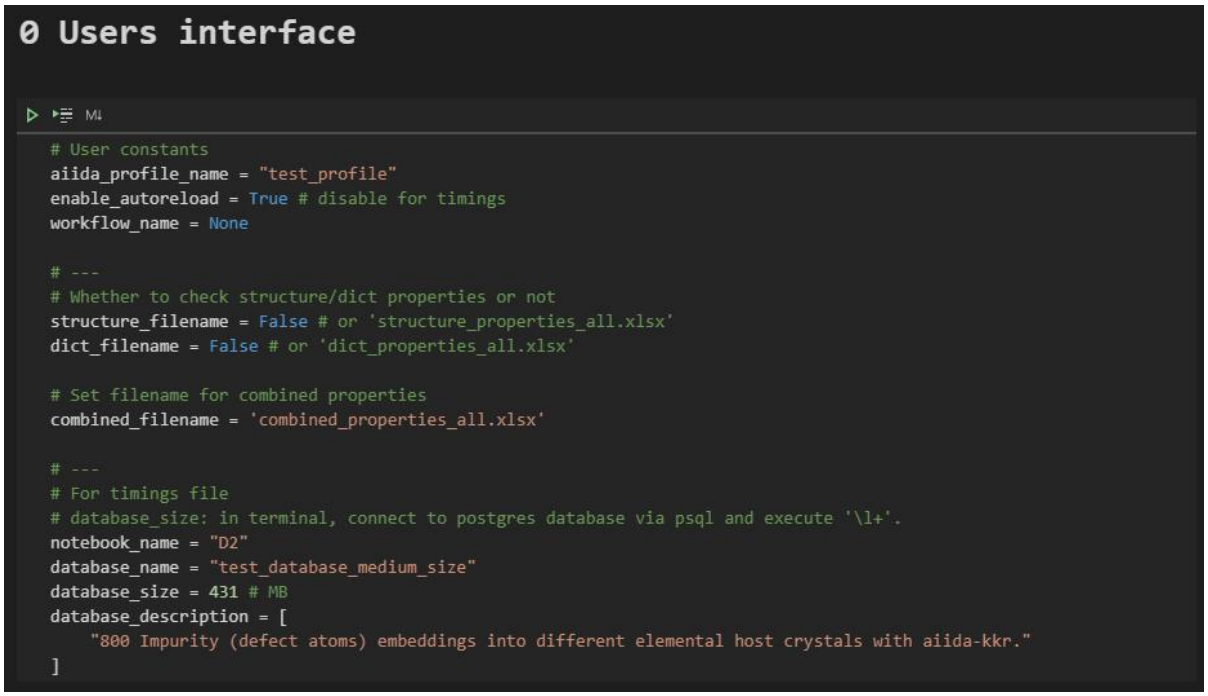


Figure 4.14: Users interface

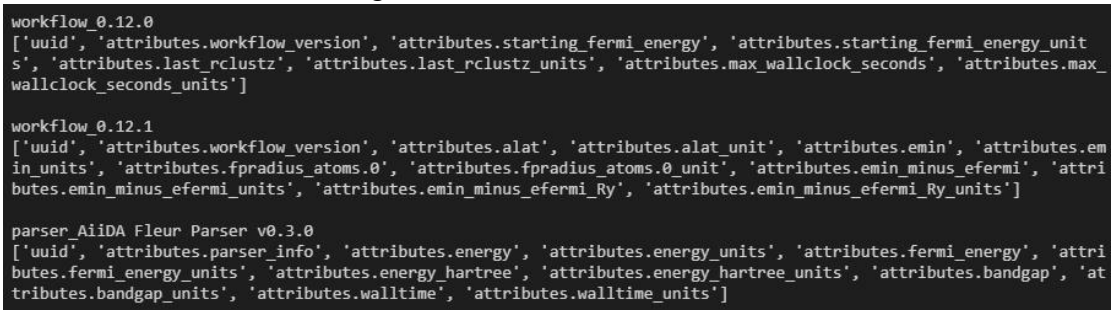


Figure 4.15: Screenshots of default projection attributes in predefined_workflow

Interactive visualization by Bokeh Server

1. Overview

To activate the interactive visualizer application we need to run the Bokeh serve command in the terminal. The default layout could be seen in Figure 4.16.
By the control panel on the right, users could select the workflow version on the radio button widget. The attributes for plotting are selected on the drop-down menu widgets below. The slider widget enables users to freely choose the number of bins of the histograms. The upper text tablet on the panel shows the numerical statistics for the selected attributes of output DictNodes. The lower text tablet shows the categorical statistics for input StructureNodes.

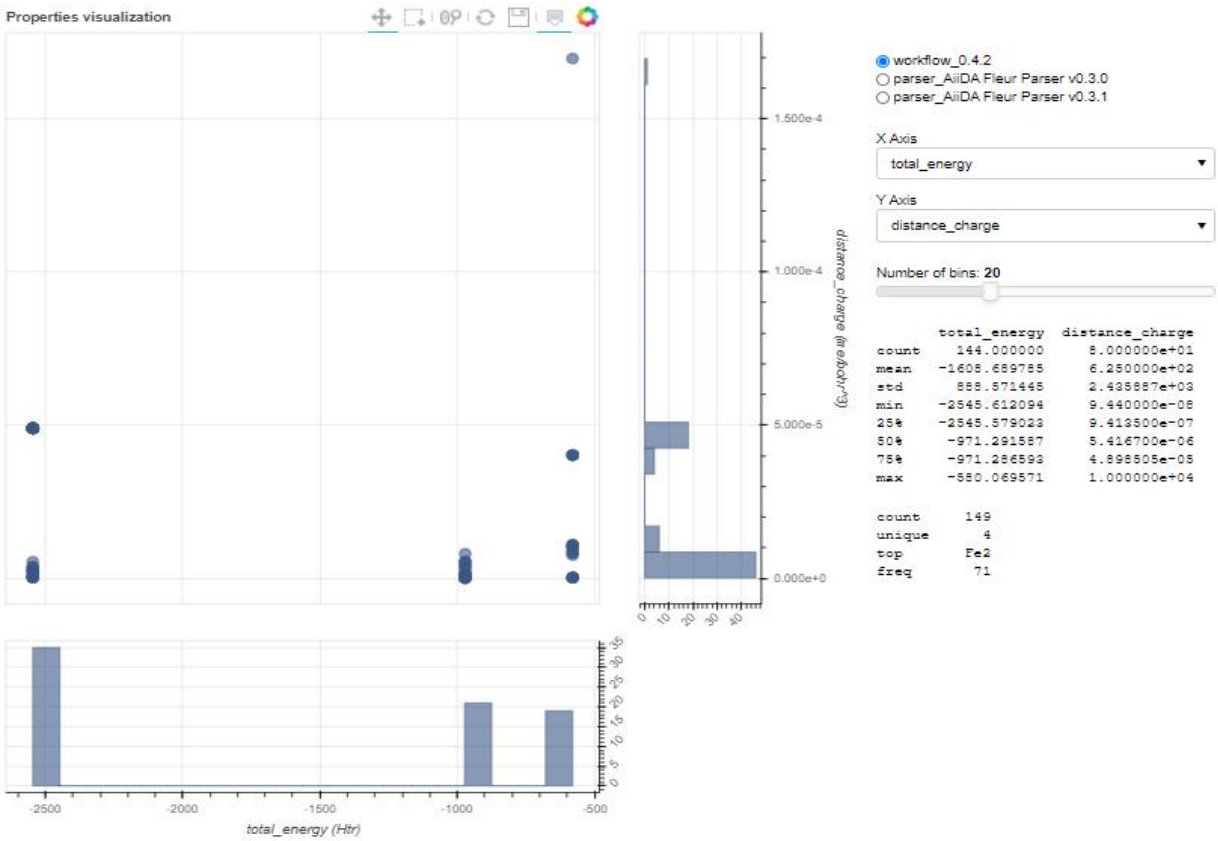
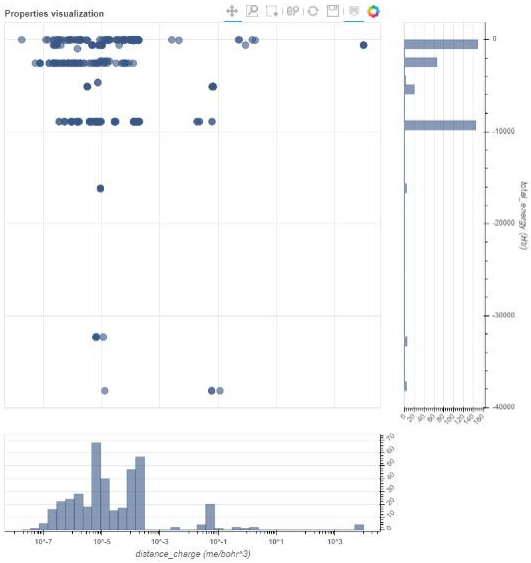


Figure 4.16: Overview of the interactive visualization of Bokeh server application.

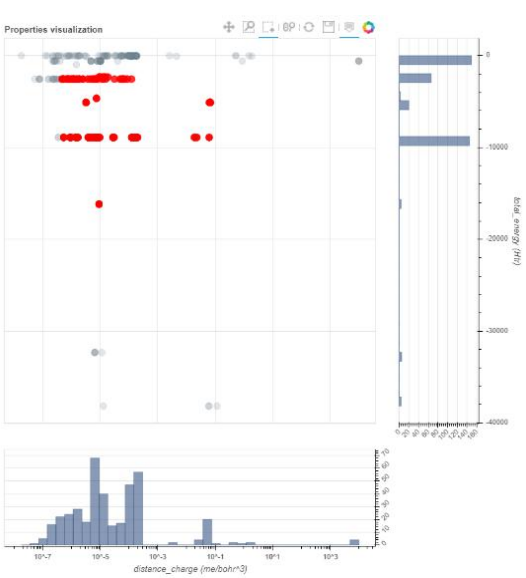
2. Tooltips and Hover tools

Besides the interactiveness achieved by Bokeh widgets on the control panel, users could also utilize tooltips, which is on the top right of the scatter plot, or hover over plots to derive more information, as is shown in Figure 4.17, the interactive plots of 'AiiDA Fleur Parser v0.3.2' with 502 nodes.

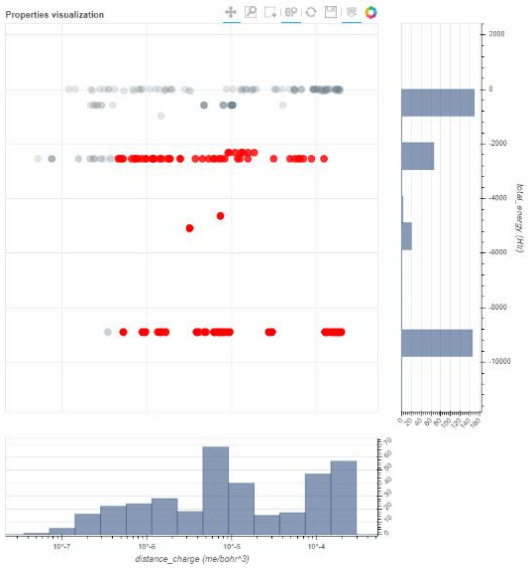
The tooltips bar includes the tool of pan, wheel zoom, box select, reset and save (see Figure 4.17 a, b, c). Some of them could be simultaneously selected to realize more functions.



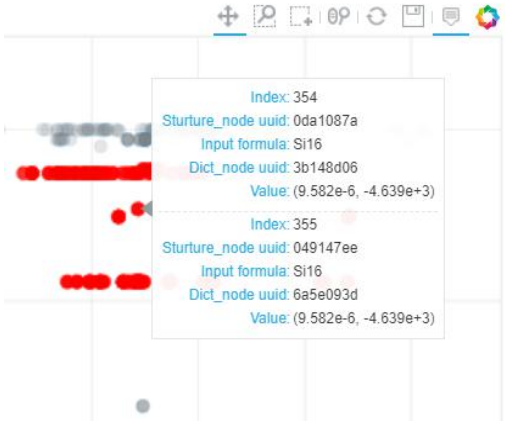
(a)‘AiiDA Fleur Parser v0.3.2’



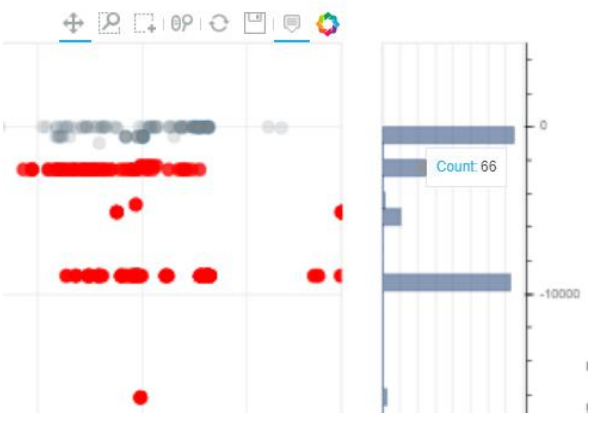
(b) Selecting nodes on scatter plot



(c) Zooming and panning



(d) Hovering over nodes on scatter plot



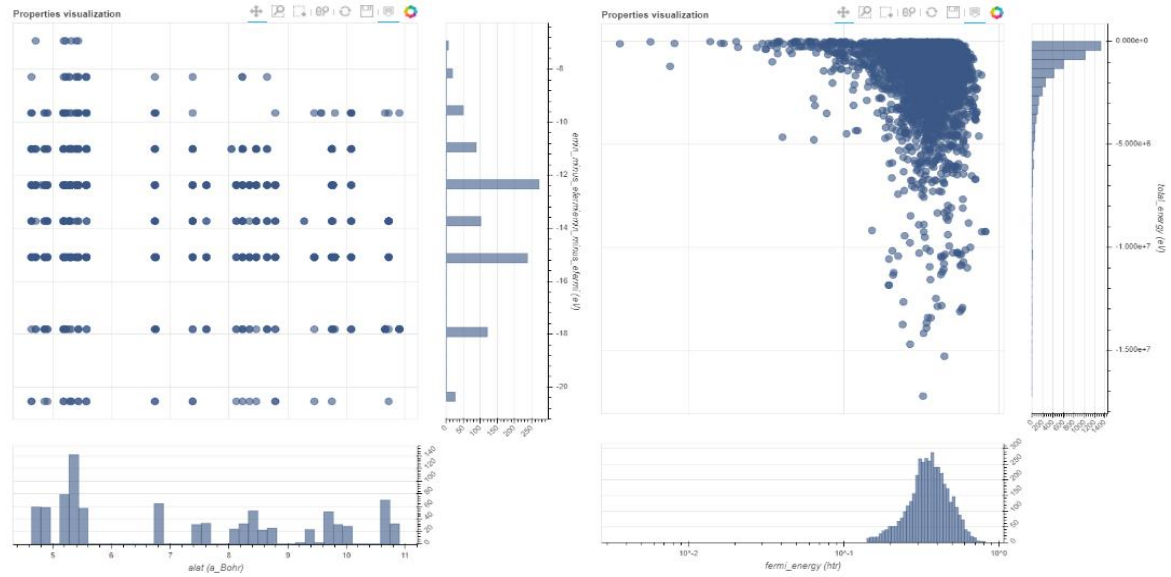
(e) Hovering over histograms

Figure 4.17: Interactive plots of ‘AiiDA Fleur Parser v0.3.2’ with 502 nodes, x axis: distance_charge (me/bohr^3), y axis: total_energy (Htr). (a) Tooltips bar. (b) Box select tool. (c) Wheel zoom and pan. (c)(d) HoverTools.

When users hover on nodes on scatter plot, the information of node index, `StructureNode` uuid, input formula and value of the node would be displayed (Figure 4.17 d). This would allow a user to further investigate a certain simulations result of interest for a structure in the database. When users hover on histograms, the count of the corresponding bar would be shown (Figure 4.17 e).

3. Other plotting examples

More plotting examples are shown in Figure 4.18, an interactive plot with 918 nodes of aiida.kkr workflow (Figure 4.18 a) and an interactive plot with 7862 nodes of aiida.fleur workflow (Figure 4.18 b). As it is shown, users now could easily get access to the interactive visualization of an AiiDA database and derive useful statistics plots by the structure-property visualizer. Furthermore, through the interaction, one can zoom in to explore what every data point is and from which calculation it is originated, enabling further investigations.



(a)'kkr_startpot_ps_0_3_2' (b) 'AiiDA Fleur Parser v0.1beta'

Figure 4.18: Another interactive plotting examples: (a) 'kkr_startpot_ps_0_3_2' with 918 nodes, x axis: alat(a_Bohr), y axis: emin_minus_efermimin_minus_efermi(eV); (b) 'AiiDA Fleur Parser v0.1beta' with 7862 nodes, x axis: fermi_energy(htr), y axis: total_energy(eV).

Chapter 5

Conclusion & Outlook

In this project, we learned a lot about computer science. Through the development of analysis tools, users can statistically view the output, workflow or input data visualization.

To conclude, so far, we have

- Successfully created deliverables that manage to (1) have a statistical birds-eye view of the contents in an AiiDA database, (2) perform structure properties visualizer.
- Developed a tool to retrieve, analyze and interactively visualize large databases on AiiDA.
- Reduce the runtime by the serializer.

For a further outlook about this project, the following attempts could be taken into consideration:

- Improve the scalability of large databases.
- Implement more complex data analyses (e.g. clustering, dimension reduction...).
- Implement interactiveness customizability of plotting (e.g. log scaling).
- Improve the tool by making it more user-friendly.

Bibliography

[1] Huber, Sebastiaan P., et al. "AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance." Scientific data 7.1 (2020): 1-18.

[2] JUDFTteam (2020) aiida-jutools [Source code]. https://github.com/JuDFTteam/aiida-jutools

[3] Van Rossum, Guido, Barry Warsaw, and Nick Coghlan. "PEP 8: style guide for Python code." Python. org 1565 (2001).

[4] Bokeh Development Team (2018). Bokeh: Python library for interactive visualization URL http://www.bokeh.pydata.org.

[5] Philipp Rüßmann, Fabian Bertoldo, and Stefan Blügel, *The AiiDA-KKR plugin and its application to high-throughput impurity embedding into a topological insulator*, arXiv:2003.08315 [cond-mat.mtrl-sci] (2020); https://arxiv.org/abs/2003.08315

[6] Giovanni Pizzi et al. “AiiDA: automated interactive infrastructure and database for computational science”. In: *Computational Materials Science* 111 (2016), pp. 218– 230. issn: 0927-0256. doi: https://doi.org/10.1016/j.commatsci.2015.09 . 013. url: http:// www. sciencedirect. com/ science/ article/ pii/S0927025615005820.

[7] S.P. Huber et al., Scientific Data 7, 300 (2020)(Online); https://aiida.readthedocs.io/projects/aiida-core/en/latest/index.html

[8] Sial, Ali Hassan, Syed Yahya Shah Rashdi, and Abdul Hafeez Khan. "Comparative Analysis of Data Visualization Libraries Matplotlib and Seaborn in Python." *International Journal* 10.1 (2021).

[9] McKinney, Wes. "Data structures for statistical computing in python." Proceedings of the 9th Python in Science Conference. Vol. 445. 2010.

[10] [6]Jolly, Kevin. Hands-On Data Visualization with Bokeh: Interactive Web Plotting for Python Using Bokeh. Packt Publishing Ltd, 2018.

[11] Wilkinson, M., Dumontier, M., Aalbersberg, I. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3,** 160018 (2016). https://doi.org/10.1038/sdata.2016.18

[12] J. Broeder, D. Wortmann, and S. Blügel, Using the AiiDA-FLEUR package for all-electron ab initio electronic structure data generation and processing in materials science,In Extreme Data Workshop 2018 Proceedings, 2019, vol 40, p 43-48