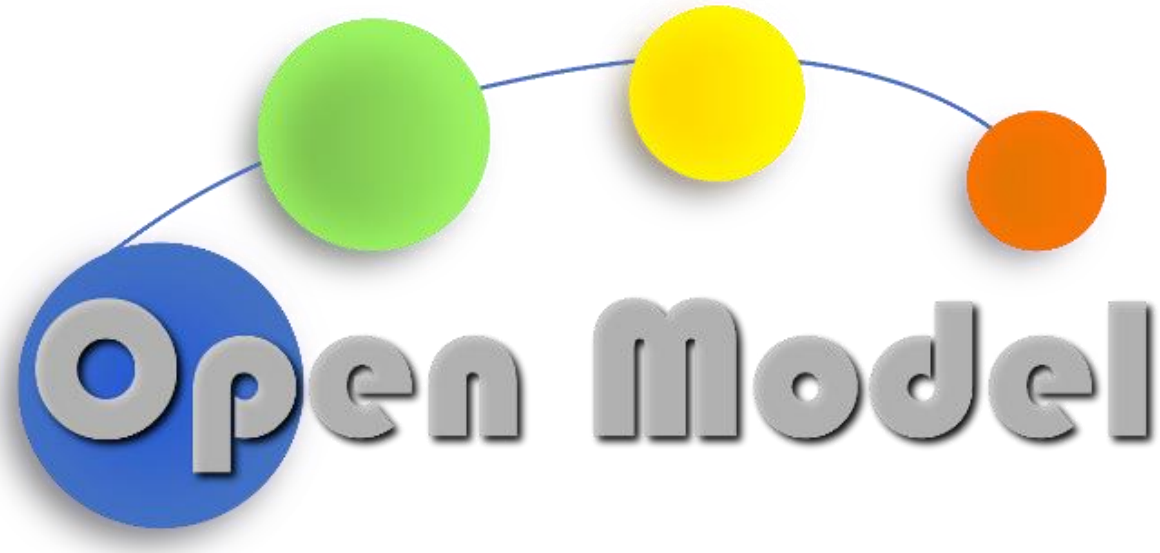
2023



ADHAM HASHIBON , OWAIN BEYNON, OTELLO ROSCIONI

UCL

D4.9 - CREATION OF A VALIDATION AND BENCHMARKING DATABASE

DOCUMENT CONTROL

Deliverable ID	D4.9
Deliverable title	Creation of a validation and benchmarking database
WP	WP4
Task	Task 4.4
Document Type	Other
Dissemination level	Public
Status	Draft
Version	0.4
Lead beneficiary	UCL
Contributing beneficiaries	
Author(s)	Adham Hashibon ,Owain Beynon (UCL) and Otello Roscioni (GCL)
Due date	2023-01-31 (M24)
Date delivered	2023-11-31 (M34)

ABSTRACT

This document reports the work, developments and ongoing activity in delivering verification and benchmarking services (D4.9) for the entire OpenModel eco system. Herein, we present the first version of a working validation and verification framework. It describes a verification and validation Python package for the OpenModel OIP that can be easily extended and expanded with additional data representing various ongoing use cases in the project. We discuss the key aspects of the service, namely, the conversion of simulation data to a common universal data structure (CUDS), the ability to store CUDS in a knowledge base, the activation of AI based V&V backend, and the retrieval of simulation data as well as verification and validation data via connection to the

knowledge base backend. Furthermore, we illustrate the capabilities of the V&V services with examples from other OpenModel use cases.

CHANGE HISTORY

Version	Date	Comment
0.1	2023-05-11	First Draft
0.2	2023-05-12	Adapted using comments from Welchy Leite Cavalcanti (IFAM)
4.0	2023-11-10	Updated draft with latest devs (OB)
4.1	2023-11-11	Review (AH)
4.2	2023-11-21	Review (technical manager)
4.3	2023-11-27	Updates by Owain Beynon
4.4	2023-11-27	Second review (technical manager)
1.0	2023-11-30	Final

DISSEMINATION LEVEL

PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

TABLE OF CONTENTS

LIST OF FIGURES

1. AIMS AND OBJECTIVES

Work package 4 (WP4) objectives are:

1. Development of novel semantic based OntoFlow workflow builder with BDSS support
2. Automated AI- and MCO-enhanced model and method selection for advanced workflows based on industry requirements.
3. Provide advanced AI based Validation and Verification services for materials modelling and materials data.
4. Integration and Applications for BDSS systems

For the V&V services, verification is defined as confirmation through objective evidence to determine whether specified requirements have been fulfilled, where the objective evidence for verification may be determined through performing alternative and additional calculations.[1] For OpenModel, verification addresses such user question as "are we implementing the model right?". Validation is defined as confirmation through the provision of objective evidence which ensure the requirements for specified application have been fulfilled, which may correspond to performing additional calculations.[1] In that respect, validation addresses the more overarching question of "are the results correct?" which are important concepts when attempting to take well informed decisions for the design of new materials based on simulations.

First, one needs to make sure that the right models are used *(the Validation)*, i.e., models that are,

1) suitable for the case, able to describe relevant phenomena and processes,
2) reliable, able to provide the desired level of details (e.g., quantitative accuracy).
3) robust, able to describe the evolution of the system without loosing relevance,
4) solvable, i.e., can be solved within reasonable computational resources.
5) holistic, able to describe the interplay between the various phenomena across relevant time and length scales

Second, one needs to be sure that the simulation of modelling in general is implemented right *(the Verification)*, e.g.

1) whether convergence is achieved,
2) whether the right numerical integration parameters are used.
3) whether sufficient sampling is made (e.g. k point sampling, size of time step etc).
4) whether convergence is achieved, or how far from convergence,
5) finite size effects
6) effects of discretisation's, usually in the form of systematic errors
7) other numerical errors, etc

While verification of simulation workflows is a technical task that can be catered for relatively easily by internal cycles [2,3]. the reliance on simulation data in materials modelling has inherent risk factor due mainly to the lack

[1] https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:en
[2] B.H.Thacker, S.W.Doebling, F.M.Hemez, M.C. Anderson, J.E. Pepin, and E.A. Rodriguez, Concepts of Model Verification and Validation, No. LA-14167, 835920, 2004.
[3] J. H. Panchal, S. R. Kalidindi, and D. L. McDowell, Key Computational Modeling Issues in Integrated Computational Materials Engineering, Computer-Aided Design 45, 4 (2013).

of validation, from e.g. experimental data. Subsequently, decision making workflows are compromised by un-validated data, which hinders integrated materials modelling. Task 4.4 "Validation and Benchmarking by using AI-based classification approaches" of WP4 requires the development of both verification and validation and benchmarking services of data by using AI-based classification services. In that regard, machine learning (ML) methods, such as neural networks (NN), provide pathways for validation in materials modelling. Furthermore, implementation of semantics methods i.e., ontologies, are required for managing the large number of datasets with complex dependencies used for ML-based validation and benchmarking. Consequently, for the creation of validation and benchmarking (D4.9) the development of a database with semantic layer is necessary. Moreover, this provides a pathway to creating training and test sets for ML wrapper development.

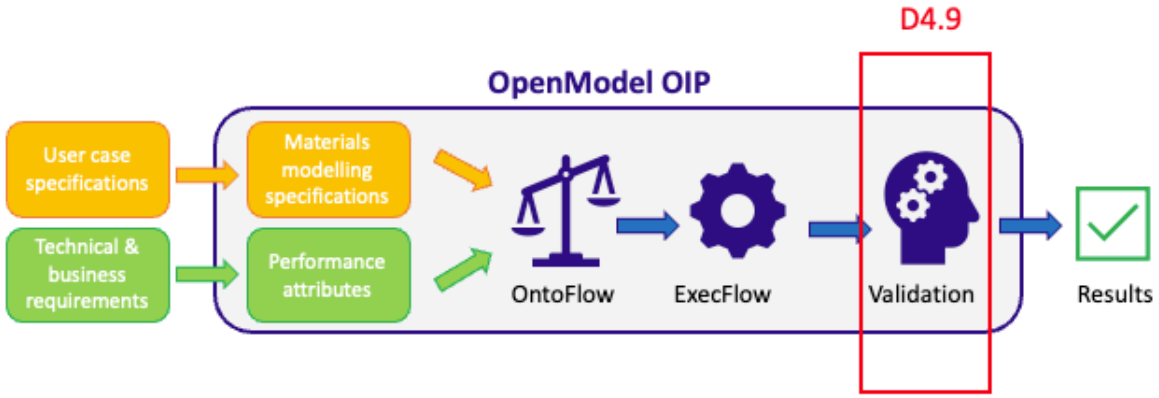


Fig. 1: Depiction of D4.9 in the OpenModel workflow.

For D4.9, the main aim of this deliverable is the creation of a validation and benchmarking database resulting from work done in Task 4.4. This task has three parts, the first consists of delivery of a database allowing validation services to be accessed. The second part is the validation services themselves, which is a software package, based on SciKit-Learn[4], of machine learning methods and models to perform verification, validation, and benchmarking on simulation data available from the data (knowledge) base. Thirdly, the development of interfaces and API are necessary to ensure integration between the database backend and the verification and validation methods available through the service. In that regard, the deliverable focuses the OpenModel demonstration success stories, where a database with API services (section 2.1), triplestore backend based on SimPhoNy-OSP CORE are provided and a set of initial simulation data from the success stories (section 3). Of which, the source code and examples of using the V&V services are found on the OpenModel GitHub repository: https://github.com/H2020-OpenModel/Validation_Benchmarking.

We emphasize that the goal of this deliverable is to describe the underlying V&V service which includes mechanisms for collecting and storing data (curation), however, this delivery does not yet cover all success stories as data for these is not yet available. The deliverable though enables straightforward integration and inclusion of such data once its available and consequently the support for V&V specific for these use cases is supported.

[4] https://scikit-learn.org/stable/

For the successful completion of D4.9 and deployment of validation and benchmarking services, the following workplan was completed:

1. Delivery of a graph database to store simulation data
2. Development of AI-based verification and validation services
3. Access of the data through the deployed database

2. OVERVIEW OF THE V&V SERVICES

As previously mentioned, the V&V services contain three key components (Fig. 2): firstly, an API for managing requests between the platform and the core of the OpenModel services. Secondly, the tools for performing validation and verification, which are a series of AI methods that can perform, machine learning (ML), statistical and analysis on simulation data. Lastly, the CUDS interface are methods for connecting with the knowledge base backend (currently based on the open source APACHE Jena Fuseki backend), which is the CUDS database.

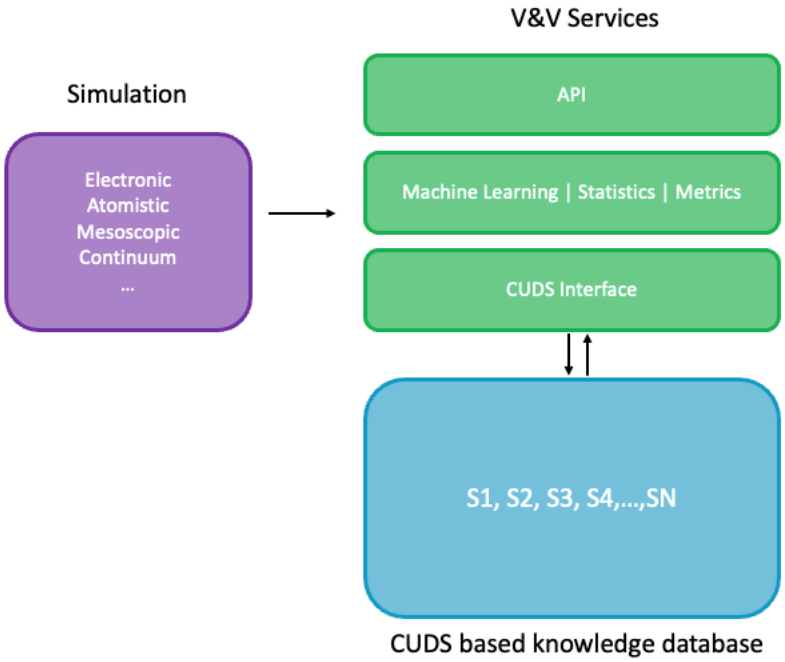


Fig. 2: Overview of the architecture of the V&V services.

For D4.9, the V&V service is deployed as a Python based OpenModel App which allows facile access to the tools in the service. A breakdown of the different scripts is as follows:

- data.py: tools for retrieving simulation data, uploading and retrieving datasets in the knowledge base.
- query.py: tools for performing queries on data stored in the CUDS database.
- validation.py: contains ML methods for performing verification and validation.
- view.py: contains visualization tools.

Furthermore, the relationship of the packages mentioned above to the architectural design depicted in Fig 2 is illustrated further in Fig 3. The entire V&V app is deployable as a service on the OpenModel Infrastructure and will be deployed in the next release planned. It will enable any OpenModel workflow or service to access the V&V app both through an API (own Restful API, and Fuseki API), as well as functionally through pythonic API, all with proper access right s provisioning provided by the advanced middleware technology implemented in the OpenModel Infrastructure (OMI).

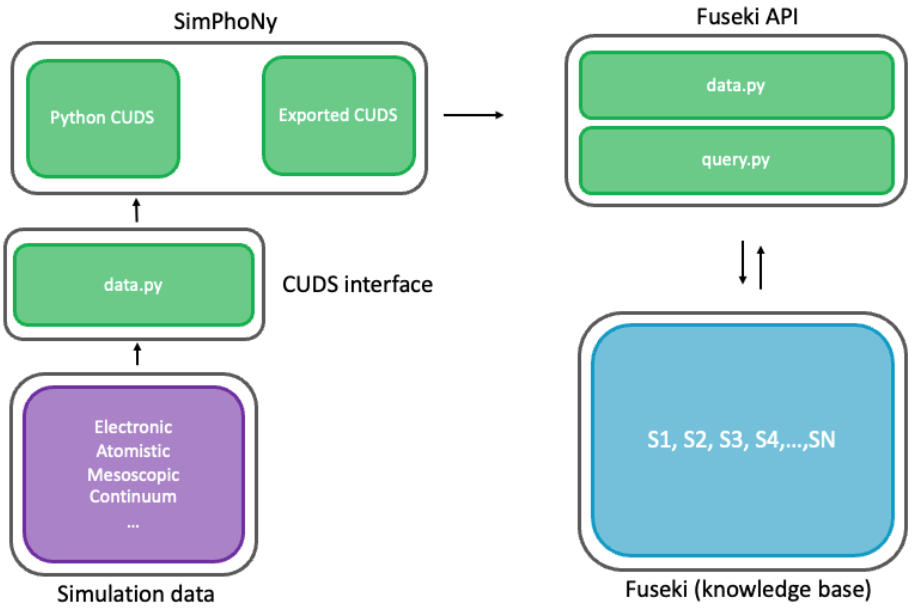


Fig. 3: Depiction of how the Python packages in V&V relate to the architecture.

Currently, *data.py* contains specific functions for parsing and retrieving data from LAMMPS and Quantum Espresso files. However, with the development of wrappers for SimPhoNy CUDS and AiiDA data nodes in WP3 (Task 3.4, D3.5), along with existing wrappers for DLite and AiiDA, integration between the declarative workflows and the V&V services is possible. Furthermore, it is important to note that data from DLite or SimphoNy can be parsed as the V&V services are agnostic to the type of CUDS provided. Regardless, through using the methods found in the V&V services, it is possible to manage and stored simulation data for verification and validation which we demonstrate herein. This integration with the wrappers is the integration of the OpenModel use cases into the V&V service as mentioned above, where further work will continue with D4.10 (V&V wrappers).

2.1 A GRAPH DATABASE FOR SIMULATION DATA

There are numerous approaches to implement a data base accessible from the network, and hence integrate able into a web or other platforms such as the OpenModel workflow system. These include traditional SQL database systems, including open-source solution such as Maria DB and PostgreSQL. These rely on predefined linked data tables designed according to strict schemas developed on a case per case basis as they need to

closely map the actual transactions made in a domain. As OpenModel caters for a wide range of complex computational problems with vastly dynamic schemas this approach is not the best. Other types of so called "NoSQL" databases including MongoDB for example, allow for freer, entity-attribute-value models to be used. These allow more easily to account for the hierarchical interrelated structure of the data needed for materials innovation workflows in general and computational modelling in particular. An Entity Attribute Value model (E-A-V) is essentially a triplet of data items that map an entity to a value through simple relation (usually a has-a). While useful for many applications, it is rather limited to elements relating a property to a value. An extension of E-A-V models, that may be seen as a generalisation is the use of Subject-Predicate-Object triplets which can relate a subject (say an atom, or a material) to an object).

For this deliverable, Apache Jena Fuseki is used as a knowledgebase for storing simulation data and simulation workflows, in alignment with the Open Model Infrastructure (OMI). Fuseki natively supports RDF data and contains a SPARQL endpoint, which means it's a natural choice for storing and retrieving simulation CUDS for the V&V services. Furthermore, integration with the APACHE Jena Triple Store TDB[5] which provides persistent and robust storage layer for the V&V services. Since the CUDS is semantic-based (Fig. 4), Fuseki offers advantages over using other unstructured and document-orientated databases such as MongoDB, which primarily supports NoSQL. Furthermore, Fuseki is open source, allowing for flexibility when developing APIs and other functionally over commercial triple stores such as GraphDB, making Fuseki a good choice for the knowledgebase in the V&V services. We note that Fuseki[6] is merely a middle ware to provide a SPARQL endpoint to TDB.

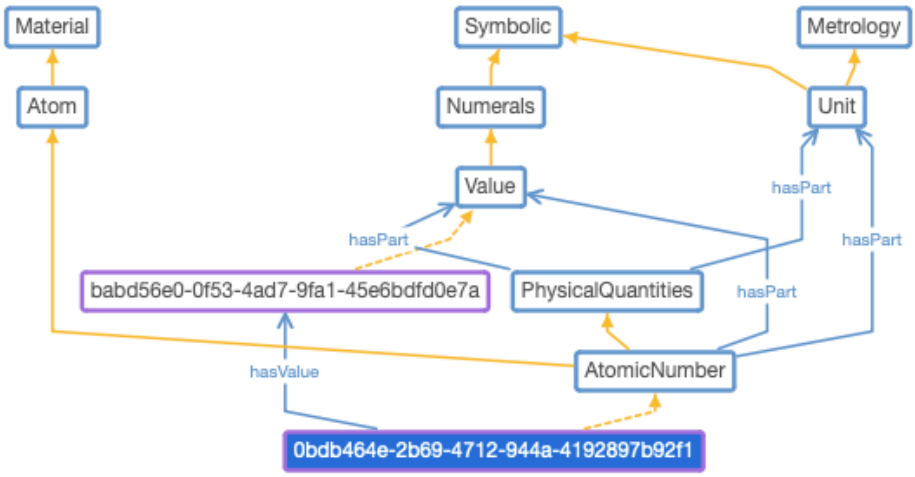


Fig. 4: Example of triples stored in the Fuseki knowledge base.

For the demonstrations, the data from the simulation is managed using the open simulation platform SimPhoNy,[7] where ontologies are used for creating triplets. Moreover, the engine for SimPhoNy works with SQlite, RDFlib,[8] and renders any existing engine to become a triple store that directly takes data models from an ontology. The generated CUDS is stored in the knowledge base (CUDS databased) and is retrievable through SPARQL

[5] https://jena.apache.org/documentation/tdb/index.html
[6] https://jena.apache.org/documentation/fuseki2/index.html
[7] 'SimPhoNy OSP Core'. 2023. [Online]. Available: https://simphony.readthedocs.io/en/v4.0.0/
[8] 'SQLite'. 2023. [Online]. Available: https://www.sqlite.org/index.html

query, which are performed via basic API functions found in *data.py* and *query.py* (Fig.5). Where a user can programmatically create and upload data to a new dataset in Fuseki, add data to an existing dataset, and retrieve data for verification and validation.

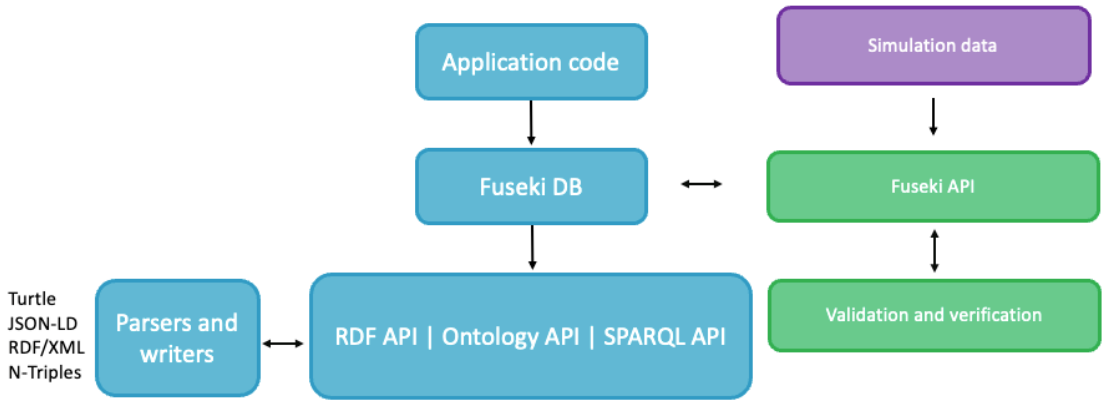


Fig. 5: Overview of the Jena Apache architecture and integration with V&V services through a Fuseki API.

2.2 VALIDATION AND BENCHMARKING

Our approach for V&V relies on the availability of data and suitable machine learning (data based) models. The approach does two things:

1. Taking a modelling workflow as input, attempts to identify an existing workflow in the knowledge data base, either from previous experiments or modelling.
    a. If found, it uses all available data to return a measure of uncertainty

b. If not found it does the following:
    i. It collects nearest results.
    ii. It creates on the fly a machine learning model.
    iii. Verifies the model internally.
    iv. Predicts the result of the workflow using the model.
    v. Returns the result.

We note that this is a crude simplistic description of a state-of-the-art ongoing research. The basic functionality is continuously updates and mode enhanced models will be added including conformal prediction.

With the development of frameworks and methods for storing and managing data from simulation, many ML methods for validation exist which would need to be explored. For example:

1. Linear Regression: Linear regression is a statistical method for modelling the relationship between a dependent variable and one or more independent variables. The goal of linear regression is to find the best-fit line through the data points that minimizes the sum of the squared residuals.
2. Logistic Regression: Logistic regression is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best-fit line through the data points that maximizes the likelihood of observing the data given the model.
3. Support Vector Machines (SVM): SVM is a machine learning algorithm used for classification and regression analysis. It works by finding the hyperplane that best separates the classes in the feature space. SVM tries to maximize the margin between the hyperplane and the nearest data points from both classes.
4. Decision Trees: Decision trees are a type of machine learning algorithm used for both regression and classification tasks. The algorithm works by recursively partitioning the data into subsets based on the values of the input variables. At each node, the algorithm selects the input variable that produces the most homogeneous subsets with respect to the target variable.
5. Random Forest: Random Forest is an ensemble machine learning method used for classification, regression and other tasks. It works by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.
6. Gradient Boosting: Gradient boosting is a machine learning method used for both regression and classification tasks. The algorithm works by building an ensemble of decision trees sequentially, where each tree tries to correct the errors of the previous tree.
7. Neural Networks: Neural networks are a class of machine learning algorithms modelled after the structure and function of the human brain. They are composed of multiple layers of interconnected nodes.

Furthermore, the Data-Driven Materials Discovery group at UCL, are developing implantations of the delta (δ) method[9] [10]. In this method, an error between experimental and simulation (δ) data is calculated and through

[9] : Liew, J., *Metamodel Verification & Validation Approach for Energy Materials Applications* (2021) masters' dissertation, UCL

[10] K. Atz, C. Isert, M. N. A. Böcker, J. Jiménez-Luna, and G. Schneider, Δ-Quantum Machine-Learning for Medicinal Chemistry, Phys. Chem. Chem. Phys. 24, 10775 (2022).

ML unknown δ values are predicted, Fig. 6. Notionally, this approach would mean that error approximations could be applied to simulation results which would provide a metric on the accuracy of the data. This would provide an accessible method for the validation for simulation data produced for the V&V services owing to its simplicity. However, with the ongoing work for other deliverables, such as deep learning and data scraping wrappers, which are necessary for full development of the V&V services, other ML models will be employed for the purpose of this demonstration.

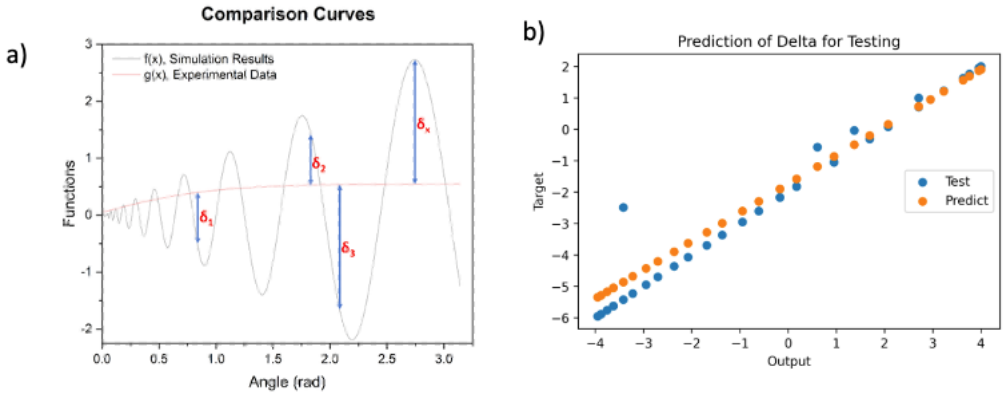


Fig. 6: a) δ between experimental and simulation data b) prediction of δ.

For the verification and validation, a ML model should be able to answer the following types of user questions: i) are the results converged? ii) will the simulation parameters selected give reasonable/ expected results? iii) how does simulation results compare to experiment or any other benchmark data? The examples in these results emphasise ii) and iii), where predictions of target observables (band gap and density) are made using data stored in the knowledge base. The predicted observables are then compared to experiment, which provides insight on the validity of the simulation parameters chosen and verifies the target observables.

Currently, *validation.py* in the V&V services offers four machine learning methods, namely, Gaussian process regression, random forest regression, Bayesian regression, and neural networks. These methods have been tested and compared (Fig. 7) for predicted the band gap of silicon which is further demonstrated in Example 2. For benchmarking the ML methods, the models were trained on the band gap energy of silicon as a function of different kinetic energy cut-off values (10-20 eV) from Quantum Espresso calculations. Subsequently, the models made predictions at a new cut-off value of 21 eV based on the training data, where the prediction was benchmarked against the band gap value at 21 eV from a simulation. The models had predictions errors of 2.47, 2.46, 2.43, and 2.81% for random forest, Gaussian, Bayesian regressions, and neural network, respectively. The ML models give reasonable accuracy, but it's important to emphasise that the small training set compromises the overall accuracy of the models, but for the purpose of demonstrating the capabilities of the V&V services, is sufficient. In that regard, given the close accuracy values of the different models, random forest regression is chosen as the ML model for the use case examples demonstrated below. Furthermore, the continual exploration and development of other sophisticated and possibly more suitable ML methods will be undertaken.

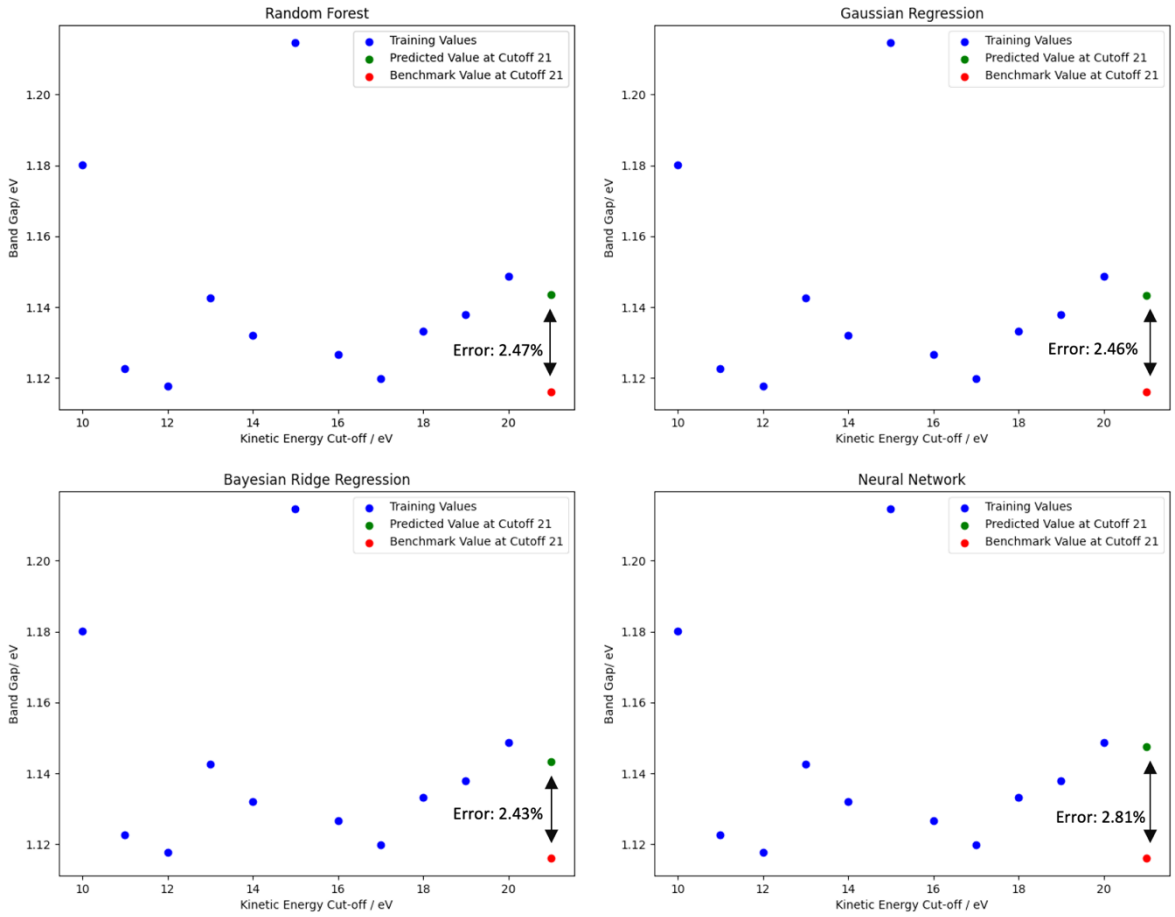


Fig. 7: Comparison between different ML methods on predicting the band gap value of silicon.

For the examples and demonstration Random Forest regression (RFR) will be used (Fig. 8). RFR assembles multiple decision trees during training and outputs the average prediction of individual trees, where the averaging of multiple trees affords more stable predictions than usage of a single tree. Additionally, the ability to handle high-dimensional data and mitigate overfitting makes RFR an excellent choice of prediction of simulation observables where multiple variables and parameters maybe needed for testing.

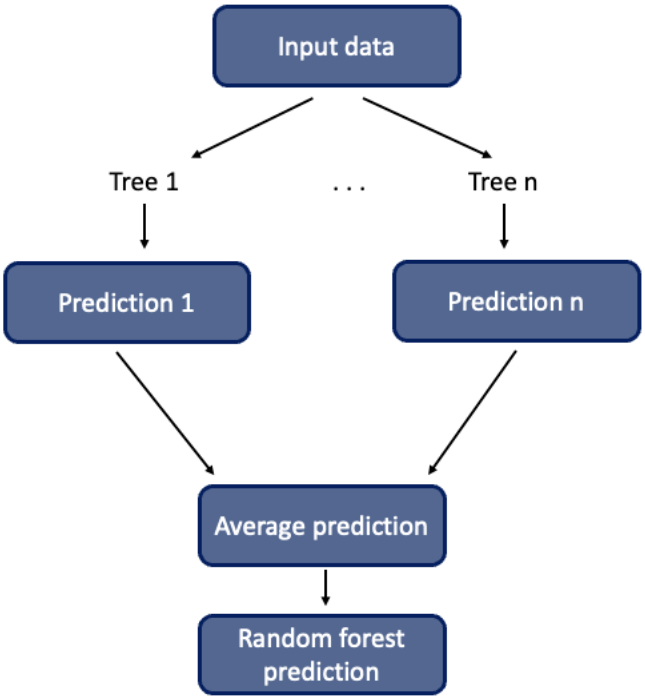


Fig. 8: Diagram of Random Forest regression method.

3. EXAMPLES AND DEMONSTRATIONS

For this deliverable, construction of a suitable database system to serve as the knowledge base for the V&V services, ML methods and the development of APIs and CUDS interfaces for uploading and retrieving simulation data from the knowledge base for verification and validation have been accomplished. Therefore, in this section examples from OpenModel use cases are used to illustrate the capabilities of the verification and validation services. Herein, two examples are used the D5.5 use case (density of water with LAMMPS) and Success Story 1 (SS1, Quantum Espresso calculations). The source code for the examples shown is found on the GitHub repository: https://github.com/H2020-OpenModel/Validation_Benchmarking/tree/main/examples.

3.1 EXAMPLE 1: D5.5 USE CASE

For the creation of simulation data for validation and benchmarking, atomistic simulations were conducted on water using the Large-scale Atomistic/Molecular Massively Parallel Simulator (LAMMPS) package[11] and the Moltemplate wrapper[12]. Moltemplate is a generalized cross-platform text-based molecule builder for LAMMPS. Utilization of such a wrapper provides flexibility and simplicity for simulation execution and data management for subsequent validation and benchmarking as outlined in Fig. 9. Although this example is also employed for D5.5, adoption of the same use case ensures ease of integration between different work packages, and we note that simulation data for other systems will be generated to ensure robustness of the systems developed in D4.9.

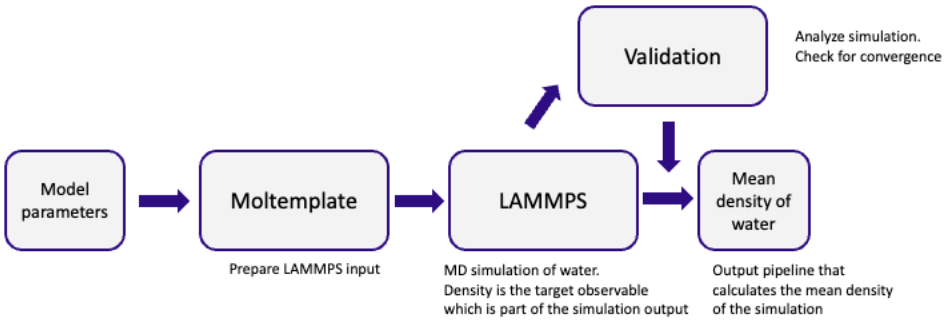


Fig. 9: Workflow of LAMMPS/Moltemplate simulation of water.

Moltemplate constructs a template file '.lt' for the parsing of variables which are executed by the LAMMPS engine (Fig. 10). This is beneficial as the nature of variable input for simulation via .lt files allows the mass generation of data for verification and validation through flexible control of input parameters. Currently, the variable

[11] A. P. Thompson *et al.*, 'LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales', *Computer Physics Communications*, vol. 271, p. 108171, Feb. 2022, doi: 10.1016/j.cpc.2021.108171.
[12] A. I. Jewett *et al.*, 'Moltemplate: A Tool for Coarse-Grained Modeling of Complex Biological Matter and Soft Condensed Matter Physics', *Journal of Molecular Biology*, vol. 433, no. 11, p. 166841, May 2021, doi: 10.1016/j.jmb.2021.166841.

definition is very syntactic to Moltempalte, with efforts ongoing for the creation of semantic definitions in other work packages.

```
# Input variables.
variable ts     equal 2       # timestep
variable tf     equal 398.15  # equilibrium temperature
variable p      equal 1       # equilibrium pressure
variable cl     equal 400     # correlation length for averaging
variable s      equal 5     # sample interval for averaging
variable prod   equal 20000   # Production steps
```

Fig. 10: Input variable for water_aa.lt.

For this specific example of the density of water, which is the desired physical observable is a value in the output log for LAMMPS (Fig 11). Therefore, the development of functions that can extract the relevant simulation data from the output log have been developed, providing integration with the next stage which the development of semantic methods for data storage. These are prototypes for the wrappers which will be developed in WP3, such as AiiDA to SimPhony and AiiDA to DLite plugins, which are out of the scope of this deliverable.

Step	TotEng	E_vdwl	E_coul	E_long	KinEng	PotEng	Temp	Press	Volume	Density	CPU
0	-419.70623	89.376432	1931.495	-2667.2573	226.68062	-646.38684	598.79252	-7636.8547	1911.2405	1.0017348	0
2000	-401.4154	87.83485	2041.3504	-2667.8445	138.04466	-539.46006	364.6545	-2354.5298	2232.6213	0.85753736	2.4650463
4000	-391.21397	105.98784	2032.2899	-2667.4651	137.97341	-529.18738	364.46631	1605.0224	2000.2915	0.95713859	4.969937
6000	-384.78549	102.35944	2014.9197	-2667.0582	164.98563	-549.77112	435.82095	-1161.0233	2299.3575	0.83264833	7.2830894
8000	-441.88651	137.34474	1948.7169	-2666.9042	138.95606	-580.84257	367.06204	2455.2766	2022.0978	0.94681683	9.7123788
10000	-404.27732	101.69982	2011.3463	-2667.2448	149.92136	-554.19868	396.02763	437.08668	2085.1824	0.91817205	12.00146
12000	-405.88102	102.38276	2024.4785	-2667.4133	134.67107	-540.55209	355.74294	-1719.5669	2199.6345	0.87039741	14.452
14000	-416.02801	104.2239	2015.2175	-2666.9198	131.45844	-547.47845	347.25655	-2484.1889	2217.0445	0.86356238	16.817759
16000	-416.12896	111.76335	2005.4482	-2667.542	134.20148	-550.33044	354.50248	-1009.7566	2073.6784	0.92326574	19.219284
18000	-372.36112	108.32542	2017.2257	-2667.7745	169.86227	-542.22339	448.70292	-754.75512	2343.8907	0.81682827	21.663647
20000	-391.53299	99.836819	2028.2228	-2667.0462	148.25362	-539.78661	391.62217	1168.7485	1943.712	0.98499994	23.942197

Fig. 11: LAMMPS simulation output log.

For this example, the simulation data is post processed for verification and validation but is in envisaged that with the further development of the OpenModel platform, simulation CUDS for V&V will be generated from the OTEAPI pipelines or data nodes from AiiDA, which will be developed for the V&V services in D4.10. In this example, data for 12 LAMMPS simulations at varying temperatures (273.15-383.15 K) are stored as a data for validation, verification, and benchmarking (Fig 12).

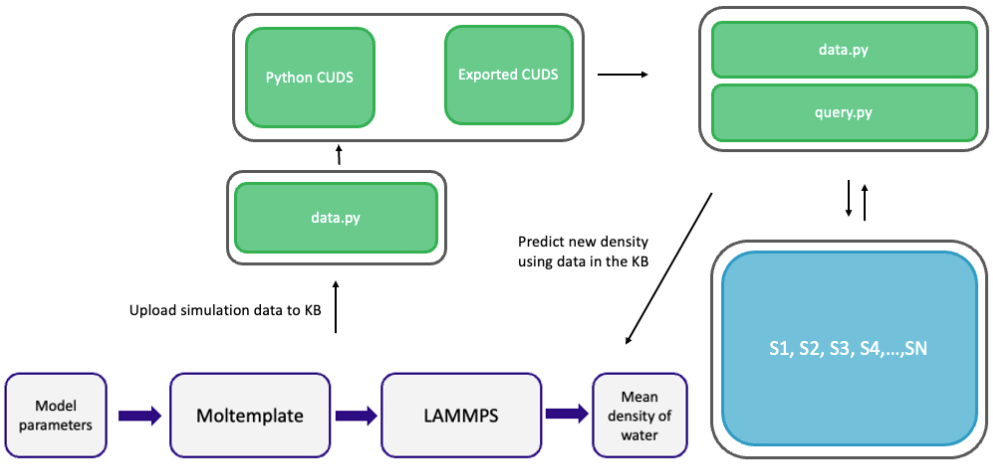


Fig. 12: Workflow for the prediction of water density using LAMMPS and the V&V architecture.

Using SimPhony-osp, a CUDS is generated from the simulation using Python and a custom ontology. In this example, the relevant input parameters from the LAMMPS simulations, such as temperature, pressure etc., along with desired output, density, are retrieved from the LAMMPS log full using methods found in *data.py*. The CUDS (Fig. 13) is then stored in the SimPhoNy backend through committing the session. Again, it is important to emphasize that these are for the demonstration purposes, as the V&V services can work with any format of CUDS (Tutrle, Json, YML) and any ontology, such as EMMO. Since the V&V services utilises CUDS, they connects to the OpenModel interoperability framework via the open source dlite-cuds package[13] developed in the OntoTrans and OpenModel projects. This will be further demonstrated in D4.10.

[13] https://github.com/EMMC-ASBL/dlite-cuds

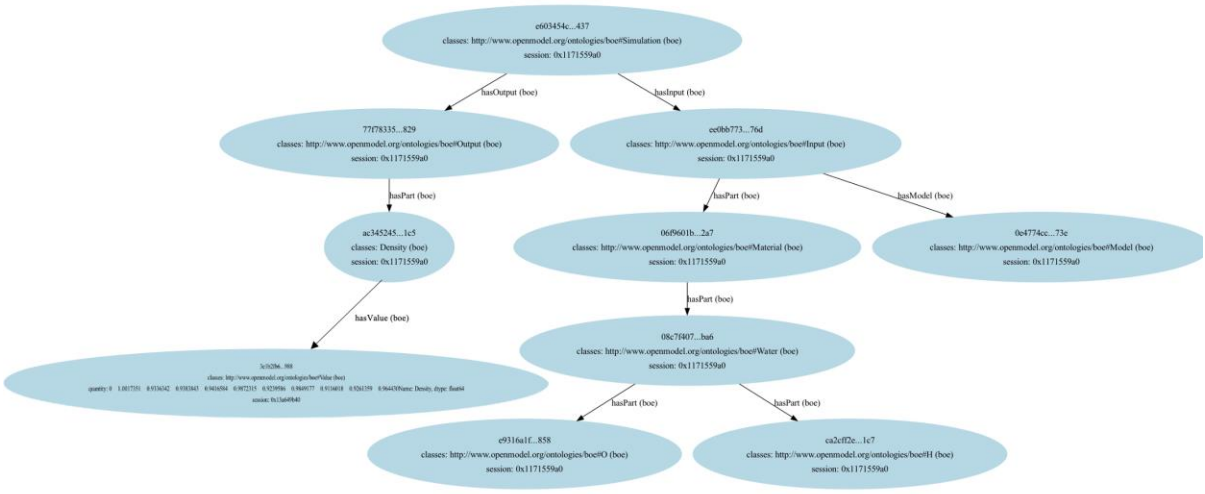


Fig. 13: Graph of LAMMPS simulation CUDS.

Furthermore, the CUDS is also exported in Turtle syntax with an example CUDS given in Fig. 14. Additionally, it is possible to export the CUDS into multiple different RDF supported syntaxes, which allows flexible usage of simulation CUDS. It is also important that the knowledgebase can work with multiple formats of CUDS generated with SimPhoNy or other software such as DLite. Therefore, *data.py* hosts functions for conversion of data to facilitate the conversion of CUDS to desired form for the knowledgebase. In which case, a predefined CUDS, for example from OTEAPI pipelines or AiiDA data nodes, can be provided to the V&V services and stored in the knowledge base.

```
@prefix boe: <http://www.openmodel.org/ontologies/boe#> .

<https://www.simphony-osp.eu/entity#04e9f9d3-79cd-46e0-ba56-060b6fe29c8f> a boe:OPLS .

<https://www.simphony-osp.eu/entity#0571d8e8-d979-43ed-851e-d6dc2f4b07a5> a boe:Simulation ;
    boe:hasInput <https://www.simphony-osp.eu/entity#0abf649b-1572-4679-9fb7-4e1ea315d045> ;
    boe:hasOutput <https://www.simphony-osp.eu/entity#38cdedaa-c5ad-4478-b8ad-fa955cc968d6> .

<https://www.simphony-osp.eu/entity#0dccf227-b0e2-4a5f-9d86-f8c082a19530> a boe:OPLS .

<https://www.simphony-osp.eu/entity#10c3d644-c068-4644-8bc5-152e4e6a416b> a boe:Simulation ;
    boe:hasInput <https://www.simphony-osp.eu/entity#50da1ce4-4399-495f-8a6d-6dbf30367578> ;
    boe:hasOutput <https://www.simphony-osp.eu/entity#5fc0f308-dc84-4360-a7d5-7b55cb077655> .

<https://www.simphony-osp.eu/entity#11405d1d-fe25-4639-bf92-8d9942cf3de6> a boe:OPLS .

<https://www.simphony-osp.eu/entity#1490d047-8e89-4242-88c7-239508b3711b> a boe:MolecularDynamics .

<https://www.simphony-osp.eu/entity#1afd400e-6587-471f-8c93-7c0e61a25afb> a boe:CorrelationLength ;
    boe:hasValue <https://www.simphony-osp.eu/entity#8659d2ae-a5e6-4c8b-baba-0bed669ee379> .

<https://www.simphony-osp.eu/entity#1f6028ad-3766-4bbf-9471-35bc178ccdb1> a boe:Pressure ;
    boe:hasValue <https://www.simphony-osp.eu/entity#263b7e70-806b-4136-be71-a1f2ebb05e76> .

<https://www.simphony-osp.eu/entity#24123c80-f111-4882-8237-f45441cbab50> a boe:Simulation ;
    boe:hasInput <https://www.simphony-osp.eu/entity#c2e7a27b-6d21-4b01-b368-d642d9c409cc> ;
    boe:hasOutput <https://www.simphony-osp.eu/entity#1640def6-8729-4fa8-a7b4-ddf175392fa0> .

<https://www.simphony-osp.eu/entity#300871c9-d844-4d4a-9a7c-f0c9597ddac1> a boe:OPLS .

<https://www.simphony-osp.eu/entity#302e2a51-726a-4733-8f88-1b9cee1f116f> a boe:Temperature ;
    boe:hasValue <https://www.simphony-osp.eu/entity#61f80272-0cdc-4488-b51c-f750869e31e0> .

<https://www.simphony-osp.eu/entity#32079b67-3218-4b18-a213-c773af6daca5> a boe:Pressure ;
    boe:hasValue <https://www.simphony-osp.eu/entity#3d9fc684-375b-4213-9188-05d0a2236962> .

<https://www.simphony-osp.eu/entity#3323c9d9-7c1f-4a56-8292-d0a91ed46c56> a boe:Time ;
    boe:hasValue <https://www.simphony-osp.eu/entity#fa9688e8-974b-4fe4-9e78-63303efec7b5> .

<https://www.simphony-osp.eu/entity#336b5adc-0498-4581-8ef3-c680b03a1b77> a boe:Time ;
    boe:hasValue <https://www.simphony-osp.eu/entity#287a35e1-be0f-402e-8582-c86b318d31c9> .

<https://www.simphony-osp.eu/entity#3545de57-4bbb-4059-ad8f-27d84090b18e> a boe:Time ;
    boe:hasValue <https://www.simphony-osp.eu/entity#97d33940-eb28-4a7d-a887-014d1e698cca> .

<https://www.simphony-osp.eu/entity#37617b44-66da-429c-b5fa-a0ead4292d97> a boe:Simulation ;
    boe:hasInput <https://www.simphony-osp.eu/entity#6d190b03-c2dc-4788-901e-a2cf55e38c29> ;
    boe:hasOutput <https://www.simphony-osp.eu/entity#12ebccc4-d6e0-4ef8-a6b1-36e385df689b> .
```

Fig. 14: CUDS (Turtle) for LAMMPS simulation data created with SimPhoNy-osp.

Through functions in data.py, the CUDS.ttl is pushed to the knowledge base, which is queried and retrieved by the V&V methods in *validation.py*. The management of large dataset and databases created when employing validation services for simulation results necessitates creation of methods for fast and reliable data retrieval. In that regard, the Fuseki API (Fig .15) in the V&V services, facilitate efficient creation of datasets and retrieval of CUDS and simulation data.

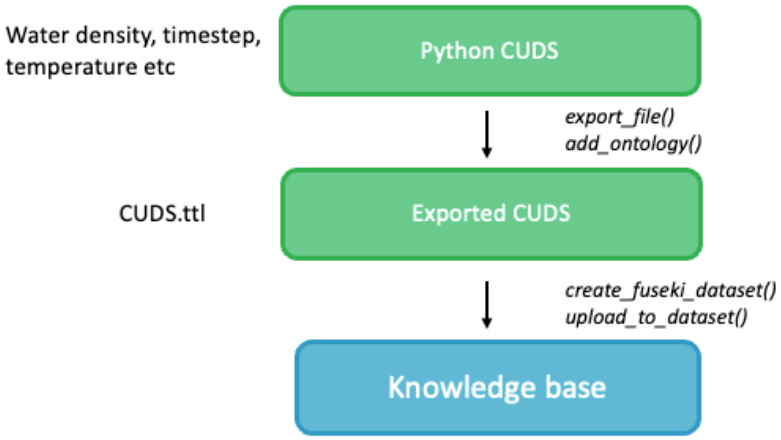


Fig. 15: Example workflow for managing data through the Fuseki API.

In Fig. 15, methods for managing CUDS data stored in the knowledge base are depicted. Through the API and functions found in *data.py*, the user can create a new dataset (ds0) for storing the CUDS, *create_fuseki_dataset().* Furthermore, the CUDS is then pushed to the new dataset through *upload_to_dataset()*. Where the relevant simulations data is retrieved through a SPARQL query on the dataset. In *query.py* a function for creating generic SPARQL queries exists, *sparql_query()*, which allows a user retrieve data from any CUDS supplied by the ontology namespace, subject, predicted and object. In conjunction with *query_data_from_fuseki()* the relevant simulation data, in this case density, is retrieved for verification and validation.

Importantly, the CUDS stored in the knowledge base is a graph database (Fig. 16), where the LAMMPS simulation data is stored as triplets.

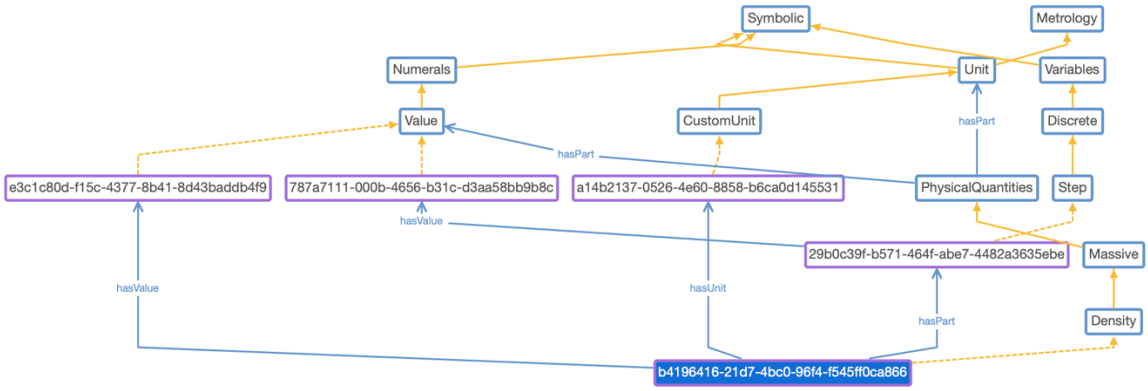


Fig. 16: Graph of relationships and classes in the ontology stored in the knowledge base.

The density data retrieved from the knowledge base is then subject to random forest regression (RFR) available in *validation.py* (Fig 17). Through RFR, the density data for simulations at temperature 273.15-373.15 K is used for training and a prediction for the density of water at a new simulation at 383.15 K is made.

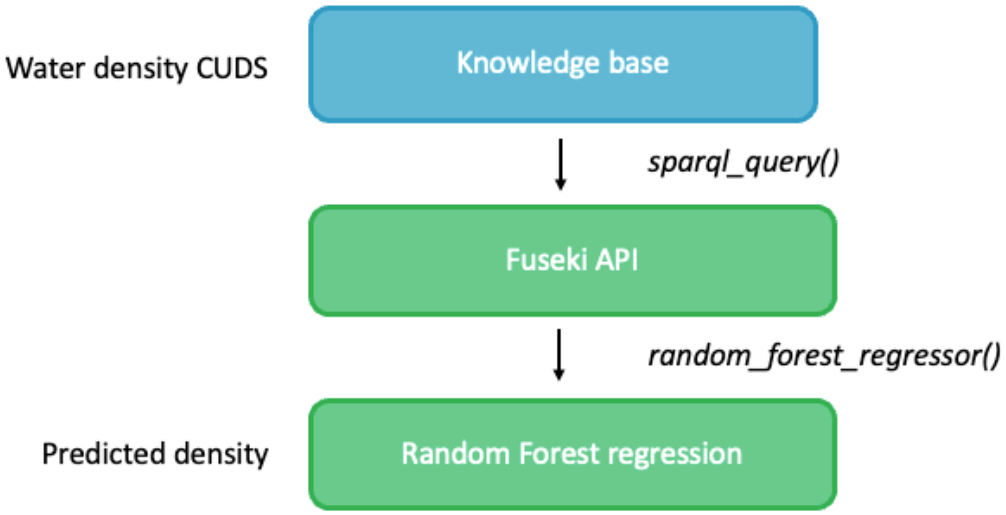


Fig. 17: Example workflow for performing ML on CUDS stored in the knowledge base.

The RFR model predicts that the density at 383.15 K is 0.963 gcm^{-1}, and an error against a benchmark experimental value at this temperature (0.958 gcm^{-1})[14] is given as 0.48%. In that regard, it is possible to infer that using the simulation methods and parameters provided in the CUDS to simulate the density of water at 383.15 K has an error of 0.48% compared to experiment, validating the simulation methods used. Although an adequate prediction, it is important to note that the accuracy of the data would be improved with a far large dataset to train the model on, but for this example it demonstrated the capabilities of the verification and validation services. Furthermore, through using *add_meta_data()* function used in data.py, the dataset on the knowledge base is updated with the error of the model as calculated with the V&V services.

[14] https://www.engineeringtoolbox.com/water-density-specific-weight-d_595.html

3.2 EXAMPLE FOR SS1

For this example, we consider the band gap of bulk silicon as calculated with Quantum Espresso, which represents key elements of SS1 (Fig 18). The band gap of silicon at different kinetic energy cutoff values (ecutwfc), 10-20 eV, is calculated and is used to train a prediction of the band gap at 21 eV.

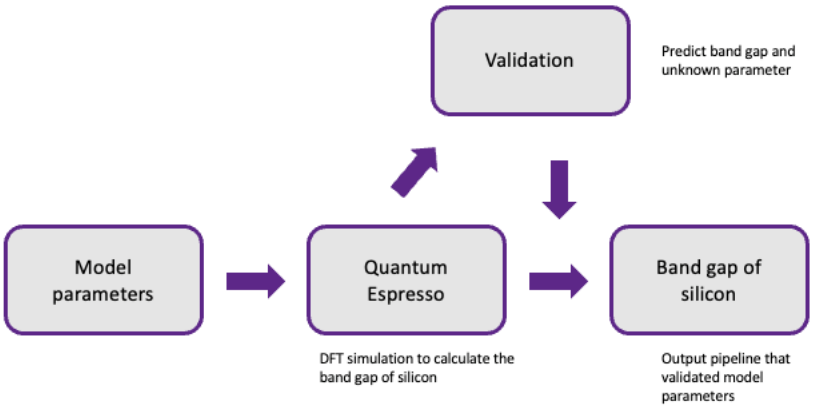


Fig. 18: Workflow for V&V on Si band gap calculated with Quantum Espresso.

As with the previous example, the CUDS is generated in Python using SimPhoNy and stored in the Fuseki knowledgebase through the API available in the V&V services (Fig. 19).

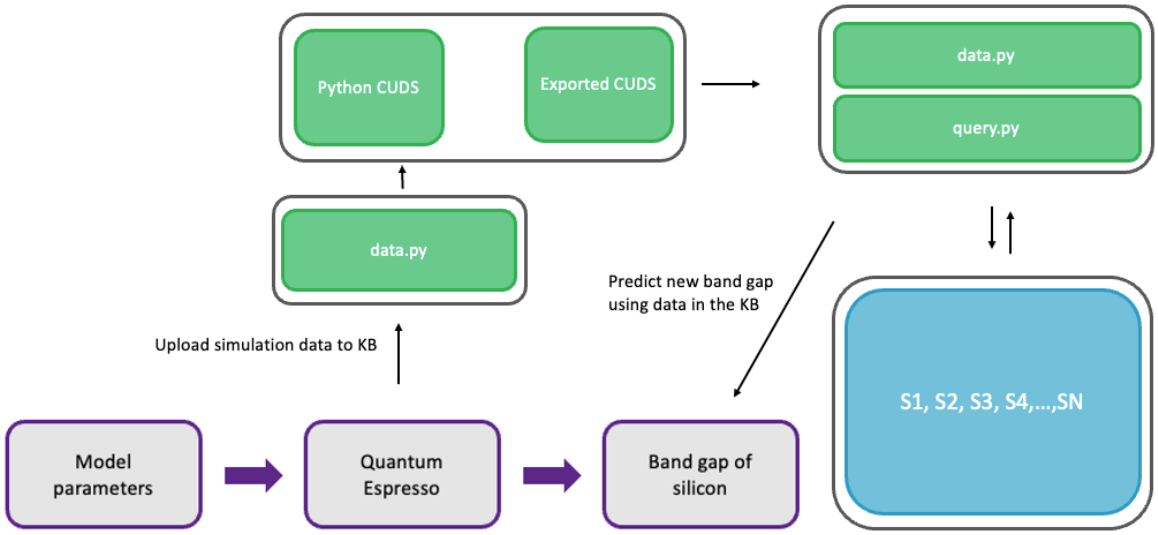


Fig. 19: Workflow for Quantum Espresso simulation CUDS as generated through SimPhoNy.

The band gap data from the QE output file ‘scf.out’, is parsed through SimPhoNy, which is ontologized to create a Python CUDS and triplets (Fig. 20). Again, it is necessary to emphasize that with continual developments of other work packages and deliverables, this step may not be necessary, and band gap data retrieved from AiiDA data notes could be retrieved by SimPhoNy and pushed to the knowledge base.

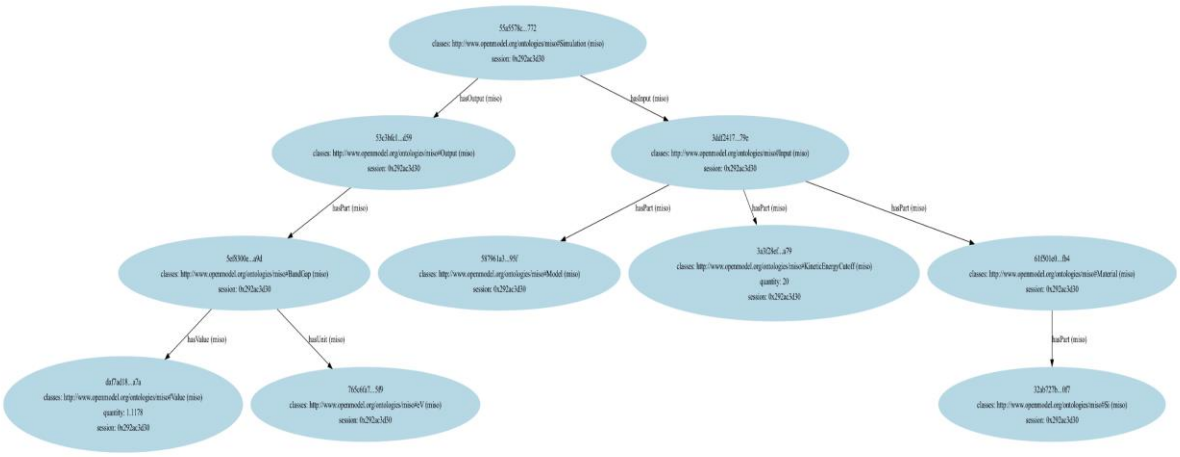


Fig. 20: Graph of Quantum Espresso simulation CUDS.

The Python CUDS (is exported to Turtle format, which contains the triplets, IRI, and simulation data (Fig. 21).

```
@prefix miso: <http://www.openmodel.org/ontologies/miso#>

<https://www.simphony-osp.eu/entity#1245b237-0839-4fb5-84ec-d3fec5e550cf> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#2891210d-283d-4338-a58c-2368f4583f1a> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#99c09f8f-38b4-4cd9-a0e0-acdb9d684dca> .

<https://www.simphony-osp.eu/entity#2224eb36-1521-40a6-a6db-1dc73d5bacbf> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#4e2db831-e9e5-44e8-b019-b89b8bff17a8> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#780028a7-a359-40db-8690-aa79b283df68> .

<https://www.simphony-osp.eu/entity#3dd83ecc-823e-4035-b3af-5d6b8aee3947> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#12c652a3-d6c1-4fc8-b1ef-f1d9bc5bf772> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#df3f9278-9948-4ed5-b0c2-c1b7ccc11da1> .

<https://www.simphony-osp.eu/entity#4d27f6c7-0916-424c-82e8-2b11f2472000> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#8648b0ac-de5b-4434-959d-ddea4c0bf294> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#1750bb19-e444-45db-b56f-9925604e64f9> .

<https://www.simphony-osp.eu/entity#75e76dee-5288-4a1f-aecf-4bb876520c38> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#eefc6384-766e-401b-8961-76115cc03a54> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#1c210cdb-ab2d-4cc6-b6e3-57420331d544> .

<https://www.simphony-osp.eu/entity#9f4c696f-188f-4336-9af9-313b02784f1a> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#e7938dc3-ea40-4509-8fd9-efb2101d4b14> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#3df5f07b-6706-4688-843c-a965c7e556d9> .

<https://www.simphony-osp.eu/entity#c0f41872-77c9-40fb-9a62-aa071bf2b633> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#80a38509-d720-4193-bf0b-323023f0e218> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#edbc921b-fe0c-4b66-b515-bccfb75aa94c> .

<https://www.simphony-osp.eu/entity#c11c9a08-352f-4e75-b85d-0f8ee8cf79cb> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#44069e58-c382-4734-8d58-4d2d633730bb> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#b264dd36-963a-4d91-bbe9-144a7e050603> .

<https://www.simphony-osp.eu/entity#cfd5be50-293d-4b73-baed-60478f912a04> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#ede089fb-753b-427b-985b-aa7c1a47341c> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#f55b9d87-273c-4acc-b839-2fc74c59c587> .

<https://www.simphony-osp.eu/entity#ed75d71a-16a2-4f22-af9b-ea10a185cb02> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#f7349449-6a8f-4572-aae8-85f54b30df12> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#0880e8d6-1322-46a4-b536-160b7cf2ee83> .

<https://www.simphony-osp.eu/entity#efab68ed-c425-4070-a44b-cb607ec747ab> a miso:Simulation ;
    miso:hasInput <https://www.simphony-osp.eu/entity#e4461414-f301-45e2-8b7f-1a2a2686b942> ;
    miso:hasOutput <https://www.simphony-osp.eu/entity#33319f2d-4645-4435-9703-354150749958> .

<https://www.simphony-osp.eu/entity#079a5189-3a37-4d40-9c64-7be10c47d4d9> a miso:Material ;
    miso:hasPart <https://www.simphony-osp.eu/entity#d844020e-f099-4190-8c3d-7cde2b7434d8> .

<https://www.simphony-osp.eu/entity#0880e8d6-1322-46a4-b536-160b7cf2ee83> a miso:Output ;
    miso:hasPart <https://www.simphony-osp.eu/entity#5f867d0d-16d3-452a-a2f2-90e01f8adc24> .

<https://www.simphony-osp.eu/entity#111f601a-7bcf-46cc-9f05-8861f3184c91> a miso:Value ;
    miso:quantity "1.1265999999999998" .

<https://www.simphony-osp.eu/entity#11250e34-fc7b-47dc-9fc1-06f0be580f83> a miso:Model .

<https://www.simphony-osp.eu/entity#12c652a3-d6c1-4fc8-b1ef-f1d9bc5bf772> a miso:Input ;
    miso:hasPart <https://www.simphony-osp.eu/entity#28dffc68-6015-429c-95d8-a53f41d5ad43>,
        <https://www.simphony-osp.eu/entity#2b782560-bfe2-4345-b2c9-97cb066490a9>,
        <https://www.simphony-osp.eu/entity#a7401c36-cbbd-4b64-aee1-2e5575327498> .
```

Fig. 21: CUDS (Turtle) for Quantum Espresso simulation data created with SimPhoNy-osp.

The simulation data is then uploaded to the knowledge base using the functions of the Fuseki API found in *data.py* (Fig. 22). Importantly, a new dataset can be created for the band gap data (in this case ds1), which allows the separation of datasets in the knowledge base, affording efficient data management.

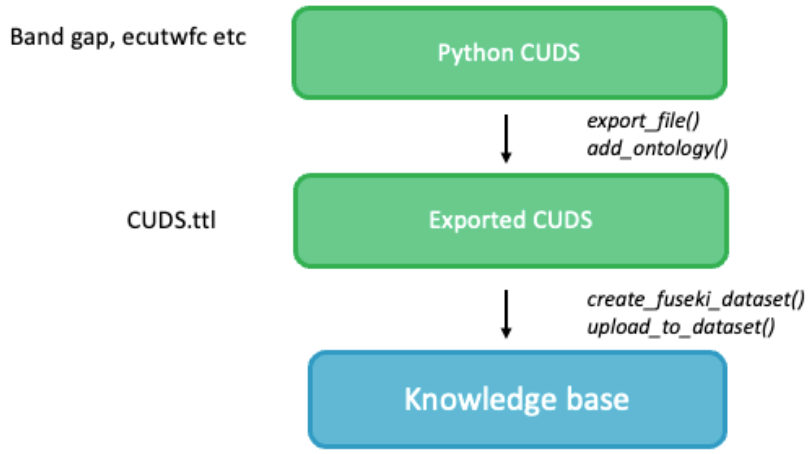


Fig. 22: Workflow of managing data through the Fuseki API.

Using the API in *query.py*, the band gap data stored in the knowledge base is retrievable and queried for verification and validation. Again, the data stored in the knowledge base is stored as triplets (Fig. 23) where ontological relationships are represented, affording the capability for semantic retrieval of data.

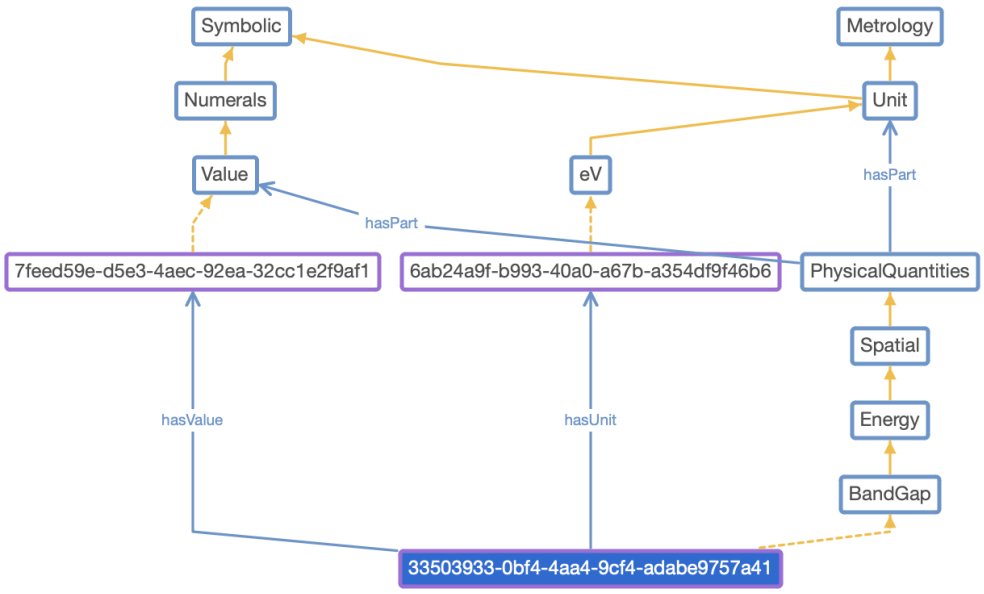


Fig. 23: Graph of relationships and classes in the ontology stored in the knowledge base.

The CUDS stored in the knowledge based is then queried using SPARQL through the Fuseki API (Fig. 24). The retrieved data from the knowledge base is subject to random forest regression, found in *validation.py*, for predicting the band gap energy at a new cutoff energy of 21 eV.

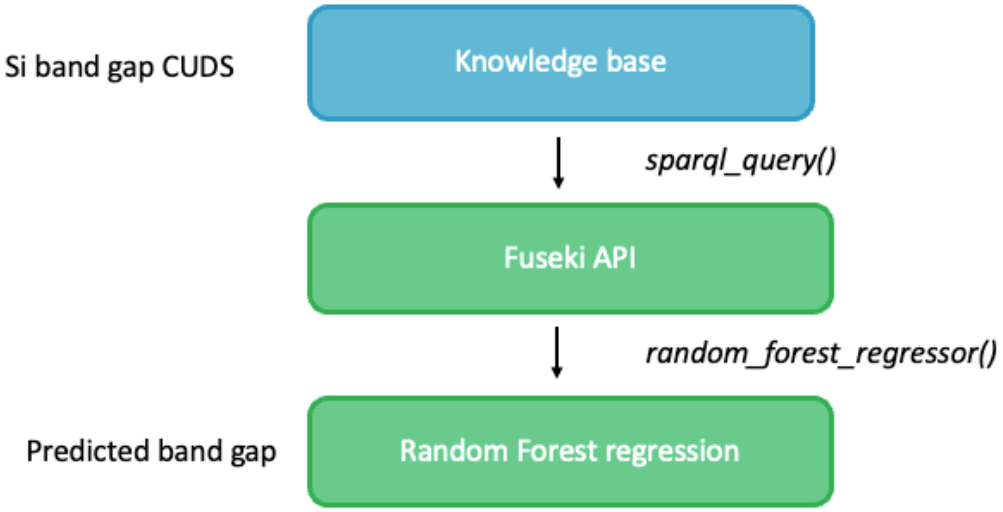


Fig. 24: Example script for performing ML on simulation data.

The model predicts that the new bandgap at ecutwfc = 21 eV is 1.14 eV, which an error estimation of 1.96 % compared the experimental band gap of silicon which is 1.12 eV.[15] From this, it can be inferred that a simulation with a kinetic cutoff energy at 21 eV deviates by around 2% from experiment. As with example 1, the error predicted from the V&V is added to the database using *add_meta_data()*. Again, the model makes a reasonable prediction compared to benchmarked values, but better accuracy would be achieved through training larger datasets, but the example here demonstrated the capabilities and functionality of the V&V services.

[15] https://toshiba.semicon-storage.com/eu/semiconductor/knowledge/faq/diode_sic-sbd/sic-sbd001.html#:~:text=Si%20(Silicon)%20has%20a%20band,wide%2Dband%2Dgap%20semiconductors.

4. NEXT STEP

In this report we have demonstrated the V&V services available in OpenModel. The main objectives of this deliverable, namely, delivery of a database system to store simulation data has been met, with examples of functionally shown.

Further development of the V&V will be undertaken in task 4.5 and D4.10 (deep learning wrappers). In this task, wrappers for the V&V service will be developed, which will facilitate integration of verification and validation methods with the workflows and OTEAPI pipelines developed in other tasks. Furthermore, completion of D3.5 will also provide integration between SimPhoNy and AiiDA plugins as stated in the description of action, which will provide seamless data transfer between the V&V services and pipelines. Moreover, due to the JupyterHub interface of the OMI platform, the possibility of deploying the V&V services in the launcher (Fig. 25), which will be explored during D.10.

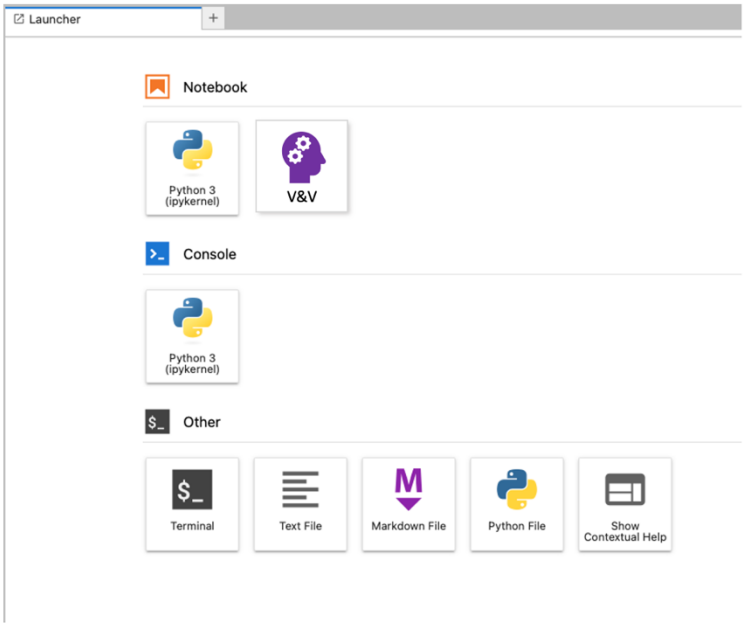


Fig. 25: Example of V&V services app in the launcher of OMI.

5. ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 953167.