

PERSPECTIVE OPEN ACCESS

Toward Full Interoperability in Materials Science: Integrating Workflows With Knowledge Graphs

Jan Janssen¹  | Osamu Waseda¹ | Bernd Bayerlein²  | Simon Stier³  | Steffen Brinckmann⁴  |
 Abril Azócar Guzmán⁵  | Sarath Menon⁶  | Matthias A. Popp³  | Jesper R. Pedersen⁷  | Ivano E. Castelli⁷  |
 Jörg Waitelonis⁸ | Wolfgang Wenzel⁹  | Jörg Neugebauer¹  | Jörg Schaarschmidt⁹  | Tilmann Hickel² 

¹Max-Planck-Institut für Nachhaltige Materialien GmbH, Düsseldorf, Germany | ²Bundesanstalt für Materialforschung und -prüfung (BAM), Berlin, Germany | ³Fraunhofer-Institut für Silicatforschung ISC, Würzburg, Germany | ⁴Institute of Energy Materials and Devices, Structure and Function of Materials (IMD-1), Forschungszentrum Jülich GmbH, Jülich, Germany | ⁵Institute for Advanced Simulations – Materials Data Science and Informatics (IAS-9), Forschungszentrum Jülich GmbH, Jülich, Germany | ⁶Interdisciplinary Centre for Advanced Materials Simulation (ICAMS), Ruhr University Bochum, Bochum, Germany | ⁷Department of Energy Conversion and Storage, Technical University of Denmark, Kongens Lyngby, Denmark | ⁸Leibniz Institute for Information Infrastructure (FIZ Karlsruhe), Eggenstein-Leopoldshafen, Germany | ⁹Karlsruhe Institute of Technology (KIT), Institute of Nanotechnology (INT), Eggenstein-Leopoldshafen, Germany

Correspondence: Jörg Schaarschmidt (joerg.schaarschmidt@kit.edu) | Tilmann Hickel (tilmann.hickel@bam.de)

Received: 12 May 2026 | **Revised:** 27 July 2026 | **Accepted:** 20 August 2026

Keywords: interoperability | knowledge graph | workflows

ABSTRACT

Interoperability is a central challenge in modern materials science as scientific workflows increasingly need to integrate heterogeneous computational methods, experimental infrastructures, data platforms, and AI-assisted systems. This perspective presents interoperability concepts from European digitalization initiatives in materials science namely, Platform MaterialDigital, NFDI-MatWerk, and MaterialsCommons, which focus on workflow management systems (WfMSs), semantic representations, and knowledge graph technologies, while relating these developments to the broader landscape of materials informatics. Since various WfMSs address different requirements, interoperable abstraction layers such as the Python Workflow Definition are needed and enable workflows to be exchanged between systems while preserving reproducibility. Beyond technical interoperability, the review highlights the growing importance of semantic interoperability through ontology-driven frameworks, including atomRDF, the Abstract Workflow Language, and semantikon. These approaches enrich workflows with machine-readable semantic annotations, provenance information, and Resource Description Framework-based knowledge graphs, enabling workflows to become semantically transparent and reusable scientific objects. The growing importance of semantically annotated workflows for the integration of computational and experimental workflows within Materials Acceleration Platforms and the emerging role of large language models and AI-assisted workflow orchestration are also emphasized. Together, these developments outline a multilayered interoperability ecosystem that combines workflow execution, semantic reasoning, provenance tracking, and AI-driven scientific automation within unified materials research infrastructures.

1 | Introduction

Materials science is intrinsically interdisciplinary, spanning phenomena across multiple temporal and spatial scales, from ultrafast electronic processes occurring on femtosecond timescales to the long-term evolution and degradation of engineered structures over

decades. At the same time, the materials life cycle encompasses raw material extraction, synthesis, processing, manufacturing, operation, repair, recycling, and re-integration into production chains. In addition, each class of materials introduces distinct experimental, computational, and data-management challenges, making it beyond the scope of any single methodology or software environment to

This is an open access article under the terms of the [Creative Commons Attribution](https://creativecommons.org/licenses/by/4.0/) License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2026 The Author(s). *Advanced Engineering Materials* published by Wiley-VCH GmbH.

capture the full complexity of modern materials research. Consequently, scientific progress increasingly depends on the interoperability of heterogeneous workflows. Rather than presenting a comprehensive survey of all existing approaches [1, 2], this perspective presents the complementary technologies developed within selected European digitalization initiatives in materials science, namely, Platform Material Digital (PMD) [3], National Research Data Infrastructure for Materials Science and Engineering (NFDI-Matwerk) [4], and Materials Commons [5], and discusses how they integrate with the broader ecosystem of workflows and semantic standards. In such an ecosystem, interoperability is not merely a technical requirement for exchanging files or orchestrating tasks but also a semantic challenge requiring a shared and machine-interpretable representation of data, processes, and provenance within knowledge graphs.

The growing diversity of workflow management systems (WfMSs) in materials science reflects the wide spectrum of user requirements and computational environments [6]. Some frameworks focus on high-throughput atomistic simulations and efficient HPC execution, whereas others prioritize graphical user interfaces, laboratory integration, provenance tracking, or semantic expressiveness. While this diversity has accelerated innovation, it has also led to fragmentation: workflows implemented in one ecosystem are often difficult to transfer, reuse, or reproduce in another. Recent efforts toward FAIR workflows—making workflows findable, accessible, interoperable, and reusable—have therefore shifted attention from isolated workflow environments toward interoperable workflow representations and shared semantic infrastructures [7].

A key development in this context is the transition from purely technical interoperability to semantic interoperability [8]. Technical interoperability ensures that workflows can exchange data and be executed across different platforms, whereas semantic interoperability guarantees that exchanged data retain their scientific meaning and contextual interpretation. This distinction becomes particularly important in materials science, where workflows routinely couple experimental procedures, atomistic simulations, continuum-scale models, and data-driven analysis pipelines. Semantic technologies based on the Resource Description Framework (RDF) and Web Ontology Language for knowledge graphs and ontologies provide a promising solution by enabling workflows, metadata, and provenance information to be represented in a machine-interpretable and formally structured way. Approaches such as Abstract Workflow Language (AWL) [9], semantikon [10], and atomRDF [11] demonstrate how ontology-driven workflow descriptions can connect workflow execution with semantically annotated research data, thereby transforming isolated workflows into interoperable and reusable scientific knowledge objects.

In parallel, workflow interoperability has evolved beyond conventional directed acyclic graph (DAG) abstractions. Emerging interoperability formats such as the Python Workflow Definition (PWD) [12] enable workflows to be exchanged across different WfMS, while knowledge-graph-based approaches such as AWL and semantikon provide bridges between executable workflows and ontology-driven semantic representations. These developments establish the foundation for interoperable workflow ecosystems in which computational and experimental workflows can be seamlessly connected across software boundaries, institutions, and scientific disciplines.

The increasing importance of semantically annotated knowledge graphs is further reinforced by the emergence of Materials Acceleration Platforms (MAPs) and AI-assisted scientific automation. In MAP environments, automated laboratories, simulation workflows, and data infrastructures must exchange information continuously and reproducibly. Knowledge graphs provide the semantic backbone required to integrate these heterogeneous data streams while preserving their provenance, contextual meaning, and workflow traceability. At the same time, large language models (LLMs) and AI-driven workflow orchestration systems increasingly rely on semantically annotated workflow components and machine-interpretable metadata to discover, compose, and validate scientific workflows automatically. Thus, semantic workflow representations are becoming essential not only for reproducibility and FAIR data management but also for enabling intelligent and autonomous scientific infrastructures.

This perspective, therefore, focuses on developments in European digitalization initiatives in materials science to enable interoperability in workflows with particular emphasis on semantic data representation and knowledge graph integration. As outlined in Figure 1, we examine how existing workflow systems address interoperability at conceptual (Section 2), technical (Sections 3 and 4), and semantic levels (Sections 5 and 6) and how ontology-driven approaches can bridge fragmented workflow ecosystems. Starting from workflow abstraction layers and interoperable workflow definitions (Section 7), we discuss the role of semantic annotation, provenance modeling, and RDF-based knowledge graphs in enabling reusable and machine-actionable workflows across computational (Section 8) and experimental domains (Section 9). Finally, we outline emerging directions toward converging workflow standards, AI-assisted orchestration (Section 10), and fully integrated semantic infrastructures for collaborative and data-centric materials research (Sections 11 and 12).

2 | Conceptual Interoperability and Parameter Design in Workflows

As outlined above, both *workflow infrastructure* and *ontology solutions* are decisive to achieve interoperability in materials science. The workflow infrastructure guarantees *numerical interoperability*, i.e., that the output data of one workflow step commonly referred to as a node in the workflow graph can be correctly interpreted as the input of the subsequent node. The ontology solution guarantees *semantic interoperability*, meaning that the output of one node can be assigned to the input of another node in a semantically consistent manner. To achieve full interoperability, however, *conceptual interoperability* must also be ensured. Conceptual interoperability concerns the proper definition and handling of method-specific parameters that must be passed between workflow nodes. For example, a stress–strain curve can be obtained using atomistic, continuum, or experimental methods. Atomistic simulations require structural information, continuum approaches require mesh definitions, and experimental methods rely on the details of sample preparation. Each of these inputs is specific to the underlying method and cannot be automatically inferred. If such method-specific information is left entirely open to the workflow developer, full interoperability between tools becomes unattainable. Although workflows may remain numerically and semantically valid, their practical

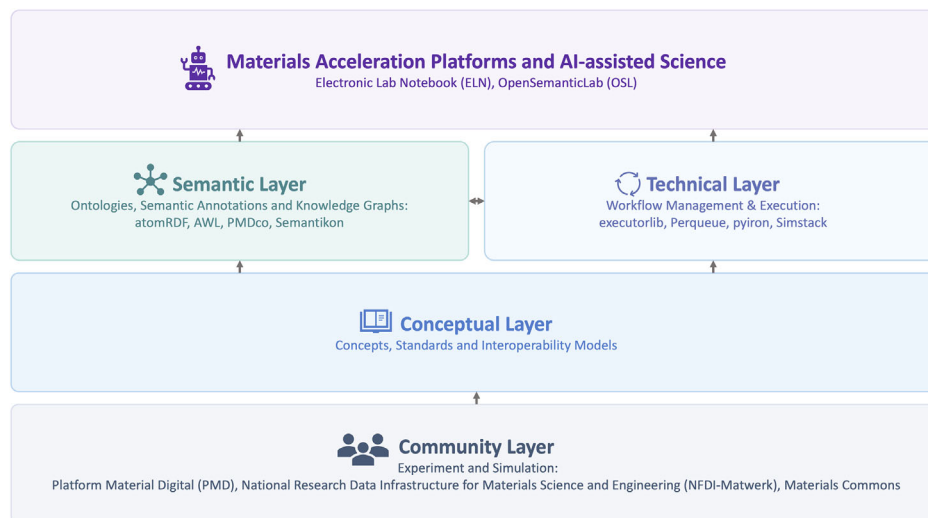


FIGURE 1 | Overview of the interoperability ecosystem for materials science developed within European digitalization initiatives. The combination of the conceptual layer, the technical layer, and the semantic layer forms the basis for Materials Acceleration Platforms (MAPs) and AI-assisted science, bringing together the materials science community including both experimentalists and simulation experts.

usability degrades because users are required to reason about parameters that are either implicit consequences of higher-level choices or irrelevant to the scientific question at hand. As scientific domains differ substantially in their conventions and modeling assumptions, it is difficult to formulate a universally applicable solution. Nevertheless, certain recurring patterns can be identified across various workflows. This section deals with these patterns and their prototypical solutions.

One of the most prominent patterns observed in the workflow development within European digitalization initiatives in materials science is the lack of structure in parameter interfaces. In many cases, workflow nodes expose long, flat lists of parameters that are technically complete but conceptually unorganized. In order to illustrate the problem, we take the continuum-scale simulation tool DAMASK [13]. A typical example from DAMASK-based workflows includes elasticity constants, plasticity model parameters, solver settings, grid definitions, rotations, and output controls specified side by side, without distinction as to their physical role or dependency structure. While such interfaces may be manageable for expert users, they do not scale well as reusable workflow components. In particular, they obscure which parameters represent genuine scientific choices and which merely reflect the internal requirements of a specific method or software implementation. Introducing a hierarchy of parameters provides a way to address this issue and thereby supports conceptual interoperability.

A useful first distinction is between *independent* and *dependent* parameters. Independent parameters represent genuine user choices rooted in physical understanding, such as the material system itself (e.g., aluminum). The dependent parameters, by contrast, are consequences of those choices. Elastic constants such as C_{11} , C_{12} , and C_{44} are not independent inputs if the material is fixed; they should instead be derived automatically or provided through carefully chosen defaults.

This distinction has direct implications for interface design: independent parameters should generally be required or explicitly visible, while dependent parameters should be optional at most.

Optional accessibility is critical for an expert who wants to use these tools.

Beyond dependency, parameters can also be grouped according to their role in the physical and computational model. In computational materials science, a pragmatic categorization is as follows:

1. Material parameters (e.g., tensile strength and modulus of elasticity) elastic constants, slip systems), etc.
2. Model parameters (e.g., constitutive law choices),
3. Process parameters (e.g., loading paths and strain rates),
4. Technical parameters (e.g., internal identifiers, bookkeeping labels, and convergence parameters).

Technical parameters are particularly important to identify because they should almost never be exposed to users. In DAMASK, for example, the so-called `composition` field often serves merely as an internal phase label. Its value is irrelevant from a physical standpoint provided that it is used consistently. Requiring users to reason about such parameters shifts an unnecessary cognitive load from the computer to the scientist.

From this perspective, most technical parameters should be hidden unconditionally, while dependent material and model parameters should default to sensible values unless the user explicitly overrides them.

Defaults are often controversial, but they are unavoidable if workflows are meant to be usable. A helpful guiding question is whether a parameter is *fundamental* to the workflow, that is, whether it constitutes a physically meaningful input that a domain expert would immediately recognize as essential. Parameters that do not meet this criterion should, wherever possible, be assigned defaults.

There are several strategies for doing so responsibly:

1. **Embed logic directly into the workflow.** If certain parameters depend on others in a predictable way, the

workflow can infer them automatically. For instance, the set of output variables required by DAMASK depends on whether plasticity is enabled. Rather than forcing users to specify an `output_list`, the workflow can derive it internally based on the presence or absence of a plasticity model.

2. **Provide lookup functions.** Frequently used physical data, such as ground-state crystal structures, can be encapsulated in lookup utilities. This allows users to specify only the composition, while the lattice information is resolved automatically.
3. **Additional workflow steps.** In some cases, defaults can be generated through auxiliary calculations, such as computing elastic constants instead of requiring them as an input. While this increases the workflow complexity, it can significantly improve usability.
4. **Leverage curated libraries.** For high-dimensional parameter spaces—plasticity models being a prime example—manual specification is often impractical. When reference parameter sets exist, exposing them as a searchable library is far more effective than exposing every parameter individually. This approach is only viable, however, if the library includes rich metadata (e.g., composition and model assumptions), a requirement that is frequently underestimated.
5. **Assign conventional defaults with transparency.** Some defaults, such as a temperature of 300 K in molecular dynamics or a strain range of 5%–10% in elastic calculations, are not universally correct but are widely understood. In such cases, defaults can be justified as long as their use is documented clearly and, where appropriate, is accompanied by warnings.
6. **Split workflows when defaults diverge fundamentally.** If different use cases require qualitatively different parameter regimes—such as adsorption versus surface diffusion—it is often cleaner to provide separate workflows rather than forcing a single interface to accommodate incompatible defaults.
7. **Leave parameters open, but document them rigorously.** Certain parameters, such as energy barriers in kinetic Monte Carlo simulations, resist meaningful default values. In these cases, the responsibility shifts to documentation: the workflow should explain how such parameters are determined, provide references, and point to working examples.

Across all of these strategies, documentation plays a central role. Regardless of how defaults are chosen, or whether they are chosen at all, the rationale behind them and their potential limitations should be clearly stated and updated as new issues are discovered. The conceptual organization of workflow parameters described here forms the basis for the technical workflow abstractions discussed in Sections 3 and 4. At the semantic level, these concepts are subsequently formalized through an ontology (Section 7) and exposed within executable workflows using *semantikon* (Section 8).

3 | Workflows as a Layer of Abstraction

While this section focuses on workflow execution, the following sections demonstrate how execution-oriented workflows can

be exchanged through common workflow representations (Section 4) and subsequently enriched with semantic information (Sections 5–8). WfMSs provide the infrastructure required to define, execute, monitor, and reproduce complex computational or experimental workflows. In materials science, workflows frequently couple heterogeneous tasks and concepts, such as simulations, data analysis, experimental measurements, and high-performance computing (HPC) jobs. To manage this complexity, a workflow typically introduces an abstraction layer between the scientific application and the underlying computational or laboratory infrastructure. This abstraction generally consists of three tightly connected components: (i) a workflow interface, defining workflow steps and their dependencies, (ii) a data management layer for handling intermediate results, provenance, and persistence, and (iii) a resource interface connecting workflows to computational platforms, storage systems, or experimental devices. Together, these components enable scalable, reproducible, and reusable scientific workflows while shielding users from many low-level technical details.

The diversity of scientific workflows in materials science has led to the development of multiple workflow systems with complementary design philosophies and target user groups [14]. Within the European digitalization initiatives in materials science [1, 15], several representative WfMSs are used, reflecting different priorities such as HPC integration, semantic interoperability, graphical usability, or interactive workflow development. Rather than converging toward a single universal workflow framework, these systems demonstrate how different abstraction layers can address the needs of distinct scientific communities.

1. **PerQueue** [16] was designed to simplify workflow orchestration for researchers already familiar with Python scripting. Internally, it represents workflows as graph-based structures with custom abstractions capable of supporting both static and dynamic workflows. PerQueue provides integrated job scheduling for HPC systems, automatic forwarding and persistence of intermediate data through a lightweight SQLite backend, and management of Python execution environments within workflows. By minimizing additional configuration overhead, PerQueue allows researchers to transform existing Python-based research scripts into reproducible workflows with relatively little effort. PerQueue has been demonstrated for both orchestrating ML-bridged multiscale simulations [16] and self-driving laboratories [17].
2. **executorlib** [18], originally developed within the Exascale Computing Project, focuses strongly on efficient execution in HPC environments. In contrast to higher-level workflow abstractions, executorlib closely follows the Python standard library and native parallel-processing paradigms that are familiar to simulation specialists. The framework emphasizes low-overhead execution by interfacing directly with computational resources and by employing efficient binary caching formats for workflow data. This lightweight architecture maximizes scalability and resource utilization, making executorlib particularly suitable for large-scale simulation campaigns and high-throughput workflows.
3. **pyiron** [19] approaches workflow management from the perspective of an integrated scientific development environment. Closely integrated with the JupyterLab ecosystem, pyiron combines workflow orchestration, simulation management,

visualization, and documentation within a unified interactive workspace. Originally developed for atomistic simulations, pyiron provides users with a higher-level abstraction layer tailored to computational materials science workflows while remaining accessible through familiar Python interfaces. More recently, pyiron has been extended toward integration with experimental instrumentation and laboratory workflows [20], thereby establishing a foundation for MAPs that combine computational and experimental processes within a single workflow environment.

4. **SimStack** [21] emphasizes accessibility and usability for multiscale simulation workflows through a graphical client-server architecture. Unlike workflow systems centered on scripting or Python APIs, SimStack provides a drag-and-drop graphical environment in which workflows are assembled from reusable components called Workflow Active Nodes. The framework automatically manages parameter exposure, data transfer, remote execution, and HPC resource interaction behind the graphical interface, substantially reducing the technical barrier for users without extensive scripting or HPC expertise. At the same time, SimStack preserves flexibility for advanced workflow customization and supports the modular integration of third-party simulation codes, enabling reproducible and scalable workflows across academic and industrial materials research.

These workflow management systems illustrate the broad spectrum of requirements present within the materials science community. Different user groups require different balances between flexibility, automation, semantic richness, scalability, and usability. Consequently, enforcing a single workflow framework across all domains is neither realistic nor desirable. Instead, interoperability between heterogeneous workflow systems therefore forms the technical foundation of the broader interoperability stack presented in this review. The PWD introduced in the next section provides a corresponding exchange format, while later sections show how these executable workflows can be transformed into semantically interoperable knowledge objects.

4 | Python Workflow Definition

The increasing diversity of WfMSs in computational materials science has created a growing demand for interoperable workflow representations that can be exchanged between different workflow ecosystems [22] without requiring workflows to be rewritten manually. The Python Workflow Definition (PWD) [12] was developed to address this challenge for Python-based workflow environments commonly used in computational materials science, including AiiDA [23], jobflow [24], and pyiron [19]. Rather than introducing yet another workflow engine, the PWD addresses technical interoperability between workflow management systems. In contrast, Sections 5–8 focus on semantic interoperability by attaching machine-interpretable meaning to these portable workflow representations.

Conceptually, PWD separates workflow logic from workflow execution. It defines workflows through three complementary components: (i) a conda environment describing the required software dependencies, (ii) a standard Python module containing the

Python functions that define the workflow nodes, and (iii) a workflow graph encoded in JavaScript Object Notation (JSON). Together, these components provide a lightweight and portable representation of Python-based workflows that remains independent of the internal implementation details of a specific WfMS. A scientist can therefore construct a workflow in AiiDA, export it to the PWD representation, and subsequently import and execute the same workflow in jobflow, pyiron, or other compatible systems without modifying the underlying workflow logic. Although the PWD representation is human readable and can technically be edited directly, it is primarily intended as an exchange and interoperability format generated automatically by workflow systems.

This interoperability is particularly important in the context of FAIR workflows [25], where workflows should not only be reproducible but also portable and reusable across different computational environments. While some workflow managers support both import and export of PWD workflows, support can also be partial. For example, AiiDA, jobflow, and pyiron provide full bidirectional interoperability, whereas executorlib [18] currently supports importing workflows from the PWD but not exporting workflows into the format. Nevertheless, by publishing workflows in the PWD representation, researchers can make their workflows accessible to users across multiple workflow ecosystems with minimal additional effort.

The structure of the PWD can be illustrated through a simple workflow for calculating kinetic energy. The computational logic itself is expressed as ordinary Python functions:

```
1 def get_speed(distance: float, time: float) -> float:
2     speed = distance / time
3     return speed
4
5 def get_kinetic_energy(mass: float, velocity: float) -> float:
6     return 0.5 * mass * velocity**2
```

Because the PWD relies on standard Python modules rather than workflow-specific decorators or APIs, these functions remain directly executable as ordinary Python code:

```
1 from workflow import get_speed, get_kinetic_energy
2
3 velocity = get_speed(distance=1.0, time=2.0)
4 kinetic_energy = get_kinetic_energy(mass=3.0, velocity=velocity)
```

The software environment required for reproducing the workflow is described separately through a lightweight conda specification. In this minimal example, only the Python version is required:

```
1 channels:
2 - conda-forge
3 dependencies:
4 - python=3.14
```

The third component of the PWD is the workflow graph itself, represented in JSON using the Eclipse Layout Kernel graph format. The graph contains workflow metadata together with lists of nodes and edges that formally define the workflow structure. Nodes represent workflow components such as functions, inputs, or outputs, while edges describe the dependencies and data flow between them. Function nodes reference Python functions through their import paths, thereby preserving the connection between the graph representation and the

executable Python code. Since workflow functions may contain multiple inputs and outputs, edges additionally specify source

and target ports corresponding to individual parameters and return values.

```

1  {
2    "version": "1.0.0",
3    "nodes": [
4      {"id": 0, "type": "function", "value": "workflow.get_speed"},
5      {"id": 1, "type": "function", "value": "workflow.get_kinetic_energy"},
6      {"id": 2, "type": "input", "value": 1.0, "name": "distance"},
7      {"id": 3, "type": "input", "value": 2.0, "name": "time"},
8      {"id": 4, "type": "input", "value": 3.0, "name": "mass"},
9      {"id": 5, "type": "output", "name": "result"}
10   ],
11   "edges": [
12     {"source": 2, "sourcePort": null, "target": 0, "targetPort": "distance"},
13     {"source": 3, "sourcePort": null, "target": 0, "targetPort": "time"},
14     {"source": 4, "sourcePort": null, "target": 1, "targetPort": "mass"},
15     {"source": 0, "sourcePort": null, "target": 1, "targetPort": "velocity"},
16     {"source": 1, "sourcePort": null, "target": 5, "targetPort": null}
17   ]
18 }

```

A central advantage of the PWD is that workflows represented in this format can be imported directly into different workflow managers with only minimal WfMS-specific code. For simple workflows, importing and executing a workflow requires only a few lines.

AiiDA:

```

1  from aiiida import load_profile
2  load_profile()
3
4  from python_workflow_definition.aiida import load_workflow_json
5  wg = load_workflow_json(file_name="workflow.json")
6  wg.run()

```

Jobflow:

```

1  from python_workflow_definition.jobflow import load_workflow_json
2  from jobflow.managers.local import run_locally
3
4  flow = load_workflow_json(file_name="workflow.json")
5  run_locally(flow)

```

Pyiron:

```

1  from python_workflow_definition.pyiron_base import load_workflow_json
2
3  delayed_object_lst = load_workflow_json(file_name="workflow.json")
4  delayed_object_lst[-1].pull()

```

In all cases, the workflow logic itself remains unchanged. This dramatically reduces the effort required to share workflows between different WfMS compared to reimplementing workflows individually for each environment. More recently, additional workflow systems have started adopting PWD. For example, PerQueue can now import PWD workflows directly:

```

1  from perqueue import PersistentQueue
2  from perqueue.external.python_workflow_definition import read_PWD
3
4  wf, undef_scripts = read_PWD("workflow.json", write_scripts = True)
5
6  with PersistentQueue() as pq:
7      pq.submit(wf)

```

Compared to more general workflow standards such as the Common Workflow Language (CWL) [26] or the AWL, the PWD places a stronger emphasis on HPC workflows. CWL is intentionally programming-language independent and exchanges data between workflow steps primarily through files. In contrast, PWD transfers Python objects directly between workflow nodes whenever possible. This design choice is particularly advantageous for Python-dominated high-throughput simulation

workflows common in atomistic materials science, where avoiding unnecessary file system operations can substantially improve the workflow performance and scalability.

Overall, PWD provides the execution-oriented interoperability layer of the PMD ecosystem. As discussed in the following sections, this portable workflow representation also serves as the starting point for semantic enrichment using atomRDF (Section 5), AWL (Section 6), PMDco (Section 7), and semantikon (Section 8).

5 | The atomRDF Framework

While WfMSs provide the technical infrastructure required to execute and orchestrate scientific workflows, they do not inherently guarantee semantic interoperability. In computational materials science, workflows often generate large and highly heterogeneous datasets whose scientific value depends critically on the availability of structured metadata, provenance information, and domain-specific context. This challenge becomes particularly pronounced in atomistic simulations, where workflows may involve multiple simulation codes, preprocessing steps, postprocessing tools, and parameter spaces. Without a shared semantic representation, workflow outputs remain difficult to compare, reuse, or integrate across platforms and research domains. While the PWD introduced in Section 4 standardizes workflow execution, atomRDF addresses the complementary challenge of representing workflow outputs as semantically interoperable knowledge graphs.

The atomRDF framework [11] addresses this problem by coupling computational workflows with ontology-based semantic annotation at the application level. Rather than treating workflows and metadata as separate entities, atomRDF transforms workflow-generated metadata directly into RDF knowledge graphs aligned with domain-specific ontologies. In its current implementation, the framework focuses primarily on atomistic simulations and builds on two complementary application ontologies: the Computational Materials Sample Ontology (CMSO) [27], which describes material structures and simulation systems, and the Atomistic Simulation Method Ontology (ASMO) [28], which captures methodological aspects of atomistic modeling. Together, these ontologies provide a semantically consistent representation of simulation inputs, computational methods, and generated outputs. The resulting RDF triples enable

machine-readable workflow descriptions that support automated querying, provenance tracking, semantic validation, and integration into broader materials science knowledge infrastructures.

A key design principle of atomRDF is the separation between workflow execution and the semantic representation. This approach is illustrated in the following example using a

```

1  from workflows.build import bulk
2  from workflows.evcures import relax_structure
3  from conceptual_dictionary import ConceptualDict
4
5  pair_style = "eam/alloy"
6  pair_coeff = "* * Fe_Ack.eam Fe"
7
8  kg = ConceptualDict()
9
10 structure = bulk('Fe', cubic=True, kg=kg)
11 relaxed_structure = relax_structure(structure, pair_style, pair_coeff, kg=kg,)
12 kg.to_yaml('relaxed.yaml')
```

During workflow execution, metadata describing the structure, simulation method, interatomic potential, and workflow provenance are automatically stored in the `conceptual dictionary`. It thereby provides a bridge between execution-oriented workflow representations such as the PWD (Section 4) and ontology-driven semantic representations introduced in the following sections. These metadata can subsequently be transformed into a semantically annotated RDF knowledge graph using atomRDF:

```

1  from atomrdf import WorkflowParser, KnowledgeGraph
2
3  kg = KnowledgeGraph()
4  wf = WorkflowParser(kg)
5
6  wf.parse('relaxed.yaml')
```

This conversion step generates RDF triples that formally describe both the workflow execution and the resulting simulation data. By relying on ontology-driven representations rather than workflow-specific metadata schemas, atomRDF enables semantically interoperable workflow outputs that can be queried, validated, and integrated across heterogeneous computational environments.

One of the primary advantages of semantic annotation at the application level is the preservation of a complete scientific context. Unlike purely technical workflow descriptions, ontology-based representations capture not only the workflow structure but also the meaning of workflow entities, simulation methods, and generated data. As a result, all metadata required for reproducibility and reuse can be retained without the loss of information. Provenance relationships remain explicitly represented within the knowledge graph, ensuring that both workflows and resulting datasets are transparent, traceable, and machine-understandable.

A second advantage lies in the use of a controlled and semantically consistent set of ontological artifacts. By restricting metadata generation to carefully aligned ontologies such as the CMSO and ASMO, atomRDF reduces the risk of introducing incompatible semantic concepts or logically inconsistent annotations. Although interoperability with external ontologies and

molecular dynamics structure relaxation workflow [29, 30]. All relevant metadata from the workflow node are stored in a conceptual dictionary [31], a lightweight JSON structure containing the key-value pairs required for generating RDF triples. The conceptual dictionary is designed to be interoperable across workflow environments and can, therefore, serve as a common exchange layer independent of the workflow engine.

standards remains important, semantic harmonization must be performed in a controlled manner to preserve consistency within the resulting knowledge graph.

Another major strength of atomRDF is its interoperability with the broader atomistic simulation software ecosystem. The framework already supports metadata extraction and semantic annotation for widely used simulation packages, including VASP [32–34], LAMMPS [35], and Quantum ESPRESSO [36, 37], together with common preprocessing and postprocessing tools. Beyond workflow interoperability, atomRDF also aims to integrate with existing materials databases and standards such as the Materials Project [38] and OPTIMADE [39], thereby facilitating semantic data exchange across platforms and repositories.

Although atomRDF is designed in a modular way and can be generalized to new workflows and use cases, extending ontology coverage requires substantial effort. The ontologies must be expanded to represent modeling approaches that lie outside the current scope of CMSO and ASMO and therefore require continued ontology development for the expansion and refinement of the ontological framework. This challenge is particularly relevant for workflow provenance, where atomRDF must not only describe computational steps consistently but also remain interoperable with external ontology frameworks used across the materials science community. At present, workflow provenance in atomRDF relies primarily on the PROV-O ontology [40], which is a W3C recommendation for provenance representation. However, interoperability with broader materials science ontologies such as PMDco [41] requires alignment with the Basic Formal Ontology (BFO) [42], which can be achieved with an explicit mapping between PROV-O and BFO [43].

Scalability also becomes increasingly important as workflows grow in complexity and throughput. High-throughput simulation campaigns can generate extremely large RDF graphs, especially when workflow metadata are captured at a fine granularity. The efficient management of such large-scale knowledge graphs requires optimized triple stores, scalable

graph databases, and potentially distributed storage infrastructures. A promising complementary strategy is to retain large homogeneous datasets in their native storage while exposing semantically relevant metadata through ontology-based interfaces, thereby reducing the size of the RDF graph without sacrificing interoperability.

Future developments of atomRDF, therefore, focus on several complementary directions, including scalable benchmarking of semantic representations and evaluation of alternative knowledge graph architectures. One objective is the expansion of ontology coverage toward additional atomistic phenomena such as defect interactions, thermodynamic properties, microstructure evolution, and phase diagram calculations. Another important direction is the extension toward workflows operating at larger length scales and across different simulation paradigms. Improvements to the conceptual dictionary abstraction are also expected to facilitate standardized metadata capture across heterogeneous workflow systems, including pyiron [19], CWL-based workflows [26], and jobflow [24]. In the longer term, the integration of semantic reasoning engines could enable automated consistency checking, inference of missing metadata, semantic validation of workflows, and even intelligent recommendations for workflow composition and selection.

The atomRDF framework demonstrates how semantic enrichment can transform workflow outputs from isolated simulation artifacts into interoperable and machine-actionable research objects. By combining ontology-driven metadata annotation with RDF-based knowledge graph generation, atomRDF enables reproducible, semantically consistent, and reusable representations of atomistic simulation workflows and their resulting data. This illustrates that interoperability in computational materials science extends beyond workflow execution and workflow definition toward the semantic data representation itself. Such semantically enriched workflows naturally complement the ontology-driven workflow representations introduced by AWL (Section 6), PMDco (Section 7), and semantikon (Section 8), enabling interoperable workflow execution and semantic reasoning within a unified ecosystem.

6 | AWL: AST and JSON-LD Based Abstract Workflow Language

Whereas atomRDF focuses primarily on the semantic annotation of workflow execution and generated data, the AWL instead captures the semantic structure of the workflow program itself. It follows this direction by providing a machine-readable representation of programmatic workflows based directly on the programming language syntax. Rather than defining workflows exclusively through workflow-specific abstractions, AWL treats source code as a first-class semantic object that can be automatically transformed into interoperable graph representations.

The primary goal of AWL is to generate a detailed, semantically structured, and language-independent representation of computational and experimental workflows from the executable source code. To achieve this, AWL combines concepts from Abstract Syntax Trees (ASTs), JSON-LD, and object-oriented linked data (OO-LD) [44] to map program structures into RDF-based knowledge graphs. This enables workflows to be represented not only as execution graphs but also as semantically queryable software artifacts.

More specifically, AWL aims to provide the following:

1. language-agnostic JSON serialization of workflows,
2. bidirectional transformation between source code and structured workflow representations,
3. support for common programming constructs such as conditions, loops, and function calls,
4. schema-based validation using JSON-Schema or semantic constraint languages such as SHACL [45],
5. and transformation into RDF knowledge graphs through JSON-LD contexts for semantic querying and analytics.

In contrast to workflow representations focused primarily on the execution order, AWL preserves the logical and structural properties of the original program. This allows workflows to retain programming constructs such as branching logic and nested function calls that are often difficult to express in conventional DAG-based workflow systems.

The central mechanism underlying AWL is the transformation of AST into semantically enriched graph representations. An AST provides a structured representation of program syntax independent of textual formatting, making it suitable for machine interpretation and semantic analysis. AWL applies object-oriented linked data concepts to AST structures in order to create interoperable workflow descriptions that can subsequently be transformed into RDF. A reference implementation for Python is provided by awl-python [46], and an interactive demonstration is available through the AWL-Playground.

To illustrate this approach, consider the following simple Python workflow:

```
1 x = func_a()
2 if x == 1:
3     y = func_b(x)
4 else:
5     y = func_c(x)
6 return y
```

Using Python's standard ast module, the program can be transformed into a normalized AST representation. After removing nonessential metadata, the resulting structure can be serialized as JSON or YAML:

```

1  _type: Module
2  body:
3    - _type: Assign
4      targets: [{_type: Name, id: x}]
5      value: {_type: Call, args: [], func: {_type: Name, id: func_a}, keywords: []}
6    - _type: If
7      test: {_type: Compare, left: {_type: Name, id: x}, ops: [{_type: Eq}], comparators: [{_type: Constant}]}
8      body:
9        - _type: Assign
10         targets: [{_type: Name, id: y}]
11         value: {_type: Call, func: {_type: Name, id: func_b}, args: [{_type: Name, id: x}], keywords: []}
12      or_else:
13        - _type: Assign
14         targets: [{_type: Name, id: y}]
15         value: {_type: Call, func: {_type: Name, id: func_c}, args: [{_type: Name, id: x}], keywords: []}
16    - _type: Return
17      value: {_type: Name, id: y}

```

Although more verbose than the original source code, this AST representation captures all structural elements of the workflow in a normalized and syntax-independent form. Importantly, the representation preserves the semantic program structure, including conditional execution paths and function dependencies, which are often lost in simplified workflow abstractions.

The next step in AWL is the transformation of this AST representation into RDF. This is achieved by applying a

JSON-LD context that maps AST entities and relationships to semantic graph concepts. The resulting RDF graph can then be serialized in formats such as Turtle and queried using semantic technologies, including SPARQL [47].

A simplified RDF representation of the workflow is shown below (IRIs as placeholders for actual ontology mappings):

```

1  @prefix awl: <https://oo-ld.github.io/awl-schema/> .
2  @prefix ex: <https://example.org/> .
3
4  ex:x a awl:Variable .
5  ex:y a awl:Variable .
6  ex:func_a a awl:Function .
7  ex:func_b a awl:Function .
8  ex:func_c a awl:Function .
9
10 [] a awl:Module ;
11    awl:HasPart
12      [ a awl:Assign ;
13        awl:HasTarget ex:x ;
14        awl:HasValue [ a awl:Call ; awl:HasFunctionCall ex:func_a ]
15      ] ,
16
17      [ a awl:If ;
18        awl:HasCondition [
19          a awl:Compare ;
20          awl:HasLeftHandComparator ex:x ;
21          awl:HasOperator [ a awl:Eq ] ;
22          awl:HasRightHandComparator [ a awl:Constant ; awl:HasValue 1 ]
23        ] ;
24        awl:IfTrue [
25          a awl:Assign ;
26          awl:HasTarget ex:y ;
27          awl:HasValue [ a awl:Call ; awl:HasArgument ex:x ; awl:HasFunctionCall ex:func_b ]
28        ] ;
29        awl:IfFalse [
30          a awl:Assign ;
31          awl:HasTarget ex:y ;
32          awl:HasValue [ a awl:Call ; awl:HasArgument ex:x ; awl:HasFunctionCall ex:func_c ]
33        ]
34      ] ,
35
36      [ a awl:Return ; awl:HasValue ex:y ] .

```

This example demonstrates how AWL transforms executable program logic into a semantically structured knowledge graph. Variables, functions, conditions, comparisons, assignments, and control-flow constructs become explicitly represented entities and relations within the RDF graph. Because the representation is based on generic AST concepts rather than workflow-engine-specific abstractions, AWL is largely independent of individual workflow systems and scientific domains.

Beyond static code analysis, AWL also supports tracing actual code execution using, e.g., Python's tracing module, capturing called functions, control-flow paths taken, variable states, and timing information. Combining the static workflow definition (T-Box) with individual execution traces (A-Box) within a unified RDF graph enables detailed provenance tracking and post-hoc workflow analysis.

Furthermore, when workflow functions are annotated with semantic type information (e.g., using OO-LD dataclasses as input/output type hints), AWL can represent these type relationships within the knowledge graph. This enables SPARQL-based workflow composition, where valid execution chains between typed functions can be discovered automatically, as well as semantic validation of existing workflows against declared type constraints [9].

One of the major strengths of AWL is its generality. Unlike workflow standards tailored specifically to computational materials science or HPC execution, AWL operates at the level of programming language semantics. In principle, any programming language with an available AST representation can be mapped into the AWL framework. This opens the possibility of interoperable workflow representations that extend beyond Python and beyond materials science itself.

At the same time, this level of abstraction introduces important trade-offs. Because AWL preserves program structure rather than workflow execution semantics, it does not inherently provide the execution-oriented interoperability targeted by standards such as the Python Workflow Definition (PWD) (Section 4). Instead, AWL represents workflows as semantically analyzable software artifacts that can be validated, queried, transformed, and integrated into broader knowledge graphs. It therefore complements the execution-oriented interoperability of PWD and the application-level semantic annotations of atomRDF (Section 5). Together with PMDco (Section 7) and semantikön (Section 8), AWL contributes an additional layer of semantic interoperability.

7 | Semantic Representation of Workflows With the PMD Core Ontology

The technologies introduced in the previous sections require a shared vocabulary describing materials, processes, and workflow entities. The Platform MaterialDigital Core Ontology (PMDco) [41, 48] provides this semantic foundation for interoperable workflow and data representation in Materials Science and Engineering (MSE) [8, 48]. In contrast to workflow-centric approaches that primarily focus on execution or workflow portability, PMDco addresses interoperability at the conceptual and ontological level. Its goal is to provide a shared semantic framework capable of consistently representing materials, processes, properties, simulations, experiments, and their relationships across heterogeneous scientific domains and software ecosystems.

PMDco is designed as a mid-level ontology tailored specifically to the needs of materials science. The current version (3.0) is built upon the BFO [49], a widely adopted upper-level ontology that provides formal distinctions between entities such as objects, processes, qualities, and information artifacts. In addition, PMDco aligns with established ontological standards including the Relation Ontology (RO) [50], the Information Artifact Ontology [51], and the Ontology for Biomedical Investigations (OBI) [52]. Through this layered architecture, PMDco establishes a semantically robust and interoperable framework that can be extended through domain- or application-specific ontologies [53, 54].

At its core, PMDco aims to provide a unified representation of the core pillars of MSE:

1. Processes, including materials manufacturing, characterization, simulation, and experimental procedures,
2. Material structures, encompassing substances and engineered materials, as well as material specifications such as compositions and multiscale structural features,
3. Material properties, including both intrinsic physical quantities and experimentally or computationally derived measurements, with a focus on process-structure-property relationships.

Beyond these domain-specific concepts, PMDco also introduces reusable semantic constructs for entities commonly required throughout the material data life cycle, such as devices,

functions, specifications, roles, and information artifacts. This enables workflows, experiments, simulations, and associated datasets to be represented within a common semantic framework, spanning multiple stages of materials research and development.

A defining characteristic of PMDco is its explicit grounding in the ontological distinctions introduced by BFO [55]. Within this framework, processes are modeled as `bfo:occurents`, meaning entities that unfold over time. Examples include chemical reactions, manufacturing operations, simulations, or experimental procedures. In contrast, physical entities such as materials, instruments, or laboratory equipment are modeled as `bfo:continuants`, which persist through time while participating in different processes. This distinction is fundamental for representing scientific workflows because it separates the evolving execution of a workflow from the physical or informational entities involved in that workflow.

Processes in BFO possess a temporal structure and can therefore contain subprocesses, phases, or sequential steps. PMDco leverages this capability to model complex scientific workflows with varying levels of granularity. In addition, processes are always linked to participating entities through semantic relations such as `ro:has participant`, thereby connecting workflow execution with the materials, instruments, or data objects involved. These process definitions provide the ontological foundation for the workflow representations generated by semantikön (Section 8), while also supporting the semantic integration of workflow outputs represented with atomRDF (Section 5).

Another important distinction introduced by BFO is the separation between processes and process boundaries. Process boundaries represent instantaneous temporal markers, such as the precise start or end of a workflow step, while the process itself occupies a temporal interval. This distinction enables PMDco to represent workflows with precise temporal semantics, making it particularly suitable for describing multistep and time-dependent materials science workflows.

Within this semantic framework, PMDco models workflows through formally defined process classes and their associated inputs and outputs. Importantly, the ontology distinguishes between physical processes involving material entities and computational or simulation processes operating on information artifacts represented as `iao:information content entities`. This separation enables experimental and computational workflows to coexist within a shared ontological structure while preserving their conceptual differences. Using Description Logic syntax and OBI properties, a class of manufacturing processes can be represented as follows:

<i>Process</i>	$\sqsubseteq$ <i>Occurrent</i>
<i>MaterialEntity</i>	$\sqsubseteq$ <i>Continuant</i>
<i>ManufacturingProcess</i>	$\sqsubseteq$ <i>Process</i>
	$\sqcap \exists$ <i>has_specified_input.MaterialEntity</i>
	$\sqcap \exists$ <i>has_specified_output.MaterialEntity</i>

These axioms express that a manufacturing process is a process that consumes and produces material entities. Such semantic definitions naturally enable workflow composition because the

output of one process can serve as the input to another. Unlike simple workflow graphs, this representation also preserves the semantic meaning of the participating entities.

PMDco further refines workflow semantics through the concept of roles. In many workflows, entities participate in a process with distinct functional meanings, even if they both belong to material input. For example, in polymer manufacturing, one material may act as a filler, while another serves as a matrix. PMDco represents these distinctions explicitly using `bfo:role` assignments:

```
PolymerManufacturingProcess
  ⊆ ManufacturingProcess
  ⊧ ∃ has_specified_input. (MaterialEntity ⊧ ∃ has_role.Filler)
  ⊧ ∃ has_specified_input. (MaterialEntity ⊧ ∃ has_role.Matrix)
```

This capability is particularly important for semantic interoperability because it allows workflows to distinguish between entities based not only on their physical type but also on their functional participation within a process.

To represent the workflow structure and execution order, PMDco extends the relational framework provided by BFO and RO. In addition to properties such as `has part`, `starts with`, `ends with`, and `precedes`, PMDco introduces the relation `simultaneous with` to represent subprocess structures, sequential and parallel activities, and complex process patterns.

For example, a parent process consisting of multiple coordinated subprocesses can be represented as follows:

```
process_parent : Process
process_step_1 : Process
process_step_2a : Process
process_step_2b : Process
process_step_3 : Process
  has_part(process_parent, process_step_1)
  has_part(process_parent, process_step_2a)
  has_part(process_parent, process_step_2b)
  has_part(process_parent, process_step_3)
  starts_with(process_parent, process_step_1)
  ends_with(process_parent, process_step_3)
  precedes(process_step_1, process_step_2a)
  precedes(process_step_1, process_step_2b)
  precedes(process_step_2a, process_step_3)
  precedes(process_step_2b, process_step_3)
  simultaneous_with(process_step_2a, process_step_2b)
```

This representation illustrates how PMDco captures both hierarchical workflow structure and temporal execution semantics within a formally defined ontological framework. Parallel and sequential workflow components become explicitly represented semantic entities rather than implicit properties of an execution graph.

A major strength of PMDco lies in its ability to bridge experimental workflows, simulation workflows, and industrial processes

within a unified semantic model. Because all workflow entities are grounded in shared ontological concepts, workflows originating from different domains or software environments can be integrated consistently within the same knowledge graph. This interoperability is particularly important for FAIR data infrastructures and Materials Acceleration Platforms (MAPs), where experimental and computational workflows must exchange semantically meaningful information across heterogeneous systems.

At the same time, PMDco is intentionally designed as a modular and extensible framework. Rather than prescribing fixed domain models, it provides a semantic backbone onto which specialized ontologies can be layered. This allows the ontology ecosystem to evolve alongside emerging materials science methods, simulation paradigms, and experimental technologies while preserving semantic consistency across domains.

In summary, PMDco demonstrates how ontology-driven semantic modeling can elevate workflow interoperability beyond technical workflow execution toward conceptual interoperability. By formally representing processes, materials, properties, workflow structure, temporal relations, and participant roles within a shared ontological framework, PMDco builds a semantic foundation and enables workflows originating from PWD-based execution (Section 4), AWL (Section 6), atomRDF (Section 5), and semantikon (Section 8) to share a common conceptual model.

8 | semantikon: Knowledge-Graph-Based Semantic Workflows

semantikon combines the execution-oriented workflow abstraction introduced by the PWD (Section 4) with the semantic foundations established by PMDco (Section 7), thereby connecting executable workflows directly to knowledge graphs. Although ontological frameworks such as PMDco [48] provide formal semantic definitions for materials science concepts and processes, workflow implementations themselves are often developed independently by different research groups using distinct coding styles, workflow management systems, and data conventions. Consequently, connecting workflow nodes with ontological knowledge typically requires additional manual mapping efforts, introducing inconsistencies and limiting interoperability between workflows, datasets, and knowledge graphs.

The semantikon framework addresses this challenge by introducing a lightweight semantic layer directly into Python-based workflow development [10]. Rather than separating workflow execution from semantic annotation, semantikon embeds ontological metadata directly into workflow node definitions using Python-type annotations. In this way, workflow nodes become semantically self-describing components that can be interpreted not only computationally but also semantically within RDF-based knowledge graphs.

This approach enables workflows to be organized, queried, validated, and reused according to their scientific meaning rather than solely according to their technical implementation. As a result, semantikon provides a direct bridge between executable workflows and ontology-driven knowledge representation,

allowing computational workflows to become interoperable components within larger semantic infrastructures.

The following example illustrates this concept using a simple pyiron-based workflow consisting of two workflow nodes: one computing a velocity from distance and time and another calculating kinetic energy from mass and velocity.

```

1 from rdflib import Namespace
2 from semantikon.metadata import u
3 from semantikon.ontology import get_knowledge_graph
4 from pyiron_workflow import Workflow
5 from pyiron_workflow.knowledge import parse_workflow
6
7 EX = Namespace("http://www.example.org/")
8
9 @Workflow.wrap.as_function_node
10 def get_speed(
11     distance: u(float, uri=EX.distance, units="meter"),
12     time: u(float, uri=EX.time, units="second")
13 ) -> u(float, uri=EX.velocity, units="meter/second"):
14     speed = distance / time
15     return speed
16
17 @Workflow.wrap.as_function_node
18 def get_kinetic_energy(
19     mass: u(float, uri=EX.mass, units="kilogram"),
20     velocity: u(float, uri=EX.velocity, units="meter/second")
21 ) -> u(float, uri=EX.kineticEnergy, units="joule"):
22     return 0.5 * mass * velocity**2
23
24 wf = Workflow("kinetic_energy")
25 wf.velocity = get_speed(distance=1.0, time=2.0)
26 wf.kinetic_energy = get_kinetic_energy(mass=3.0, velocity=wf.velocity)
27
28 wf.run() # not required - only if values should be contained
29
30 pwd = parse_workflow(wf)
31
32 knowledge_graph = get_knowledge_graph(pwd)
33

```

A central feature of semantikon is the use of semantic-type metadata embedded directly within Python type hints. Introduced in Python 3.9, this mechanism allows workflow developers to annotate workflow inputs and outputs with ontological identifiers, units, and semantic meaning without modifying the computational logic of the workflow itself. In the example above, quantities such as distance, time, velocity, and kinetic energy are associated with explicit URIs and physical units, thereby linking workflow variables directly to semantic concepts within a knowledge graph.

Importantly, semantikon does not require developers to implement custom parsers or ontology translation layers manually. Instead, workflows are first converted into the PWD introduced in Section 4, which provides a workflow-system-independent exchange format before semantic enrichment. The resulting PWD workflow graph is then processed by semantikon to generate RDF-based semantic representations. Consequently, semantikon remains largely independent of a specific workflow management system: any WfMS capable of exporting workflows into the PWD representation can, in principle, integrate with semantikon.

The semantic workflow representation generated by semantikon can be exported either as a recipe ontology (T-box) describing the semantic workflow structure or as an execution graph (A-box) representing a specific workflow execution instance. Both representations are generated in the RDF form using rdflib and can subsequently be validated against external ontologies or reasoning frameworks.

This semantic representation enables several forms of automated reasoning and validation:

1. **Ontological type compatibility checks**, ensuring that semantically incompatible workflow nodes cannot be connected incorrectly,

2. **Python datatype validation**, verifying consistency between workflow inputs and outputs at the programming level,
3. **Unit compatibility and conversion checks**, ensuring physically meaningful data exchange between workflow components,
4. **Automatic extraction of RDF knowledge graphs**, enabling semantic querying and integration with external knowledge infrastructures.

Unlike conventional workflow validation approaches that operate purely on technical interfaces, semantikon performs validation at the semantic level of the workflow graph. This allows workflows to be checked for conceptual consistency before execution rather than solely during runtime.

Another important consequence of semantikon's ontology-driven representation is the seamless integration of computational workflows with experimental data. Because workflow variables are linked to globally defined semantic URIs, computational and experimental datasets can reference identical concepts within the same knowledge graph. For example, a quantity such as `EX.velocity` may simultaneously describe simulation outputs and experimentally measured data, enabling unified semantic querying across heterogeneous data sources (cf. Figure 2 shows the results).

At the same time, semantikon remains consistent with the ontological distinctions introduced by BFO. Within this framework, workflow-generated data are represented as information content entities rather than physical objects themselves. Consequently, the semantic representation captures the informational specification of quantities and workflow relationships while remaining ontologically distinct from the physical entities it describe.

A major strength of semantikon lies in its ability to combine semantic reasoning with lightweight workflow integration. By embedding ontological metadata directly into workflow node definitions, semantikon avoids the need for separate semantic modeling stages while preserving the modularity and readability of the original Python code. Workflow developers can therefore add semantic interoperability incrementally without restructuring existing computational workflows.

Several key advantages emerge from this approach:

1. **Provenance and traceability:** semantikon generates provenance-rich workflow graphs that preserve workflow lineage, execution history, and semantic context, thereby supporting reproducibility and auditability.
2. **Interoperability:** Because most semantic operations are performed directly within RDF graphs, semantikon enables interoperability not only across Python-based workflow systems but also with external semantic tools, databases, and experimental infrastructures.
3. **Static semantic reasoning:** Workflow validation occurs at the semantic recipe level before execution, allowing conceptual inconsistencies to be detected during workflow composition rather than only at runtime.
4. **Modularity and minimal intrusiveness:** Semantic annotations are stored entirely within Python metadata fields

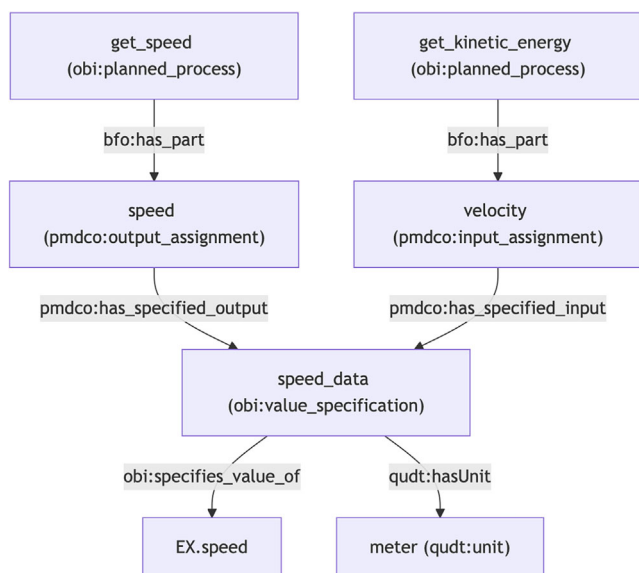


FIGURE 2 | Workflow part of the knowledge graph for the example workflow.

and remain separated from the computational implementation itself. This preserves workflow readability while avoiding the need for dedicated ontology parsers or domain-specific workflow languages.

In summary, semantikron represents an important step toward semantically enriched workflow ecosystems in scientific computing. By tightly integrating ontology-based annotations with executable Python workflows, it enables workflows to become semantically transparent, interoperable, and machine-actionable components within broader knowledge graph infrastructures, providing the final integration step between workflow execution, ontology-based semantics, and the AI-enabled applications discussed in Sections 9 and 10.

9 | Generalized Workflows for Material Acceleration Platforms Including Lab Environments

The interoperability technologies introduced in the previous sections become particularly valuable when computational workflows are combined with experimental procedures within MAPs [56]. While computational workflows need to be digitally formalized in order to be executable by a WfMS, experimental procedures are still often documented through informal protocols or instrument-specific software, limiting reproducibility and interoperability.

To address this gap, MAP workflows must support hybrid experimental–computational processes that combine automated simulations, laboratory instrumentation, and human interactions within a shared provenance framework. In this context, standard operating procedures (SOPs) can be treated similarly to computational workflow nodes as both define parameterized transformations of inputs into outputs together with associated metadata and provenance information.

Several data management systems and orchestration platforms already contribute toward standardized and interoperable experimental workflows in materials science.

1. The Electronic Lab Notebook (ELN) file format is an archive format for the exchange of experimental results and data [57].
2. FINALES enables orchestration between geographically distributed experimental infrastructures through standardized APIs and broker architectures [58].
3. Nomad-CAMELS combines experimental workflows with live user feedback and instrument abstraction layers based on ophyd [59].
4. OSL integrates ontology-driven semantic annotation with user interfaces and knowledge graphs [60].

Together, these approaches demonstrate that experimental and computational workflows can be represented through interoperable workflow abstractions that differ primarily in their execution modality: computational tasks execute automatically, whereas SOPs may involve human operators or laboratory equipment.

In the near future, fully autonomous laboratories are unlikely to replace human operators entirely. Instead, most MAPs will operate in partially automated settings where human expertise remains necessary for calibration, interpretation, troubleshooting, and instrument handling. As a result, human-in-the-loop workflows have become a central design requirement for interoperable laboratory workflows.

One promising approach is the integration of graphical interfaces and ELNs that automatically annotate user inputs with semantic metadata [61]. Platforms such as OSL provide examples of ontology-driven interfaces that combine workflow execution with automatic semantic annotation and provenance capture.

The following simplified workflow illustrates how human–interaction steps can be represented explicitly within a workflow, building on the Abstract Workflow Language (AWL) [9] for defining typed workflow nodes with human-in-the-loop support:

```

1 @hitl
2 def set_machine_params(params: MachineParams):
3     # transfer params to machine
4     return params
5
6 @hitl
7 def document_result(params: ProcessDocumentation):
8     # validate documentation
9     return params
10
11 @hitl
12 def document_more(params: YesNo) -> YesNo:
13     return params
14
15 def archive_data(params: Any):
16     # store documentation in database
17     pass
18
19 @entry_point(gui=True, jupyter=True)
20 def workflow():
21     machine_params = set_machine_params() # prompts user
22     do_document_more = True
23     while(do_document_more):
24         result_evaluation = document_result() # prompts user
25         do_document_more = (document_more()).answer
26     archive_data(machine_params) # runs automatically
27     archive_data(result_evaluation) # runs automatically
28
29 workflow()
  
```

In this workflow, user interactions become explicit workflow nodes rather than external or undocumented activities. Graphical interfaces can then automatically generate sequential dialogs guiding the operator through the workflow while simultaneously capturing provenance and metadata (cf. Figure 3 shows the results. Such approaches bridge the gap between executable workflow systems and traditional laboratory procedures.

A promising direction for hybrid workflows is the introduction of lightweight procedural layers that complement existing workflow engines rather than replacing them. Recent prototype implementations build upon workflow systems such as pylon workflow [19] while adding minimal abstractions for representing manual laboratory procedures.

In these approaches, SOPs are represented as concise, human-readable Markdown documents enriched with structured parameter definitions encoded as key-value pairs using lightweight markup conventions (cf. Figure 4 shows the results. Parameters, instrument states, metadata, and output locations can then be extracted automatically and integrated into workflow provenance systems.

These recent developments demonstrate that lightweight procedural layers, semantic workflow annotations, ontology-driven metadata integration, and human-centered workflow interfaces can successfully bridge the gap between automated computation and laboratory experimentation. Such hybrid workflows demonstrate how the interoperability stack developed can be applied to connect computational workflows, laboratory automation, semantic knowledge graphs, and AI-assisted scientific discovery [62].

10 | Interoperability With and Through Large Language Models

LLMs represent a natural application of the presented interoperability stack. They benefit from execution-oriented workflow representations (Section 4), ontology-based semantic descriptions (Sections 5–8), and provenance-rich workflow knowledge graphs when discovering, composing, validating, and executing scientific workflows. Their effectiveness depends on receiving actionable, semantically annotated interfaces to code and workflows and on accessing findable, reusable building blocks. This section outlines how structured Python interfaces and semantic annotations improve LLM operability, how LLMs recombine existing

components into new workflows or code, and how emerging LLM-native workflow engines relate to existing scientific tooling.

10.1 | LLM-Actionable Representations

Dataclass-based, semantically annotated Python functions (e.g., using OO-LD [44]–flavoured pydantic) provide directly actionable tools for LLMs (e.g., via LangChain, an open-source framework for constructing LLM-based applications and agentic workflows). By connecting code functionality (program syntax) with structured annotations (comments, descriptions, and semantic links), interpretability improves for both LLMs and humans. In this context, a semantic description of the interface (input/output) and functionality not only aids the user but also the LLM to properly utilize and connect these tools in a meaningful way. LLMs may also use workflow examples to generate custom code on the fly [63].

10.2 | Findability and Reuse

Re-usability depends critically on the findability of existing workflows and workflow nodes. Historically, missing interoperability has often stemmed from the fact that finding and installing existing workflows was more time-consuming than creating new, non-interoperable versions. LLMs can both magnify and mitigate this problem: generating new code becomes trivial, risking fragmentation; conversely, LLMs can act as search agents over knowledge graphs or workflow databases such as the PMD workflow store [1] (workflow-RAG), composing new workflows from verifiable, human-understandable, and well-tested elements. Semantic annotation therefore contributes to the safer use of AI.

10.3 | Recomposition of Workflows

LLMs can enable reusability through recomposition of the (Python) code at multiple stages:

1. *Dynamic calling of fixed workflows at runtime:* An LLM calls one tool at a time, interprets intermediate results, and flexibly selects the next tool and its parameterization. Advantage: highest flexibility and potential for discovering useful new compositions. Disadvantage: increased probability of deviation from standardized workflows.

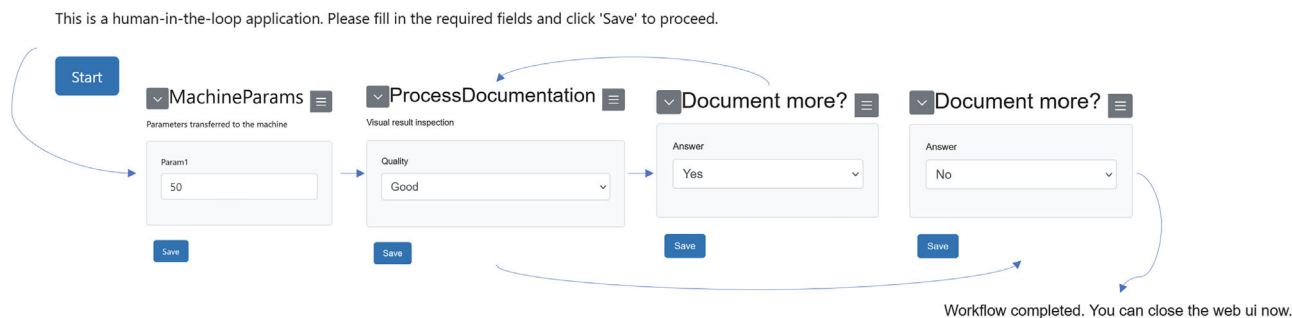


FIGURE 3 | Sequential user dialog auto-generated from the workflow above.

```
# Zeiss SEM
- Vent
- Open chamber
- Mount sample and close chamber
- Use it at |voltage|20| kV
- Use a aperture of |aperture|3|

Author: Steffen Brinckmann, IMD-1, FZJ
<!-- version:1.0 |filename|| -->
```

FIGURE 4 | Mock-up of an SOP for experimental workflows using the Markdown format and reStructuredText for key-value pairs.

2. *Workflow creation by an LLM*: An LLM composes a workflow of fixed nodes that is then executed as a whole. Advantage: repeatability after workflow creation, scalability, and performance for batches of inputs with a constant format. Disadvantage: LLMs are trained primarily on established programming languages rather than workflow definitions.
3. *Code creation by an LLM*: An LLM composes a workflow of fixed nodes within an established programming language such as Python. Advantage: good coding capabilities without additional requirements. Disadvantage: semantic annotation is optional, yielding executable but non-annotated code with an unclear context of applicability.

10.4 | Examples

LangSim [64] and DREAMS [65] illustrate LLM agents constructing workflows. At the current stage, individual workflow steps are developed by scientists, while the LLM combines these steps based on user requests. In addition, LLM converts natural-language inputs into workflow inputs, acting as a translation layer. These developments are based on the LangChain framework introduced above, remaining independent of specific LLM backends.

10.5 | LLM-Native Workflow Engines

The AI community has identified the need for structured execution of AI workflows and agents, leading to LLM-native environments, such as LangGraph [66]. Such systems orchestrate interactions among LLM agents over a shared state, support automated documentation of intermediate results, and interface with UI frameworks for debugging and analysis. Consequently, LLMs should not be treated as an independent interoperability technology but rather as beneficiaries of the semantically enriched workflow ecosystem described in the preceding sections.

11 | Complementarity and Synergy Effects

The presented technologies form complementary layers of a common interoperability architecture rather than competing alternatives. Their combined development illustrates that interoperability in computational and experimental materials science cannot be

solved by a single tool but instead requires coordinated abstractions that connect executable workflows, semantic metadata, ontological reasoning, and AI-assisted orchestration into a coherent ecosystem for both computational and experimental workflows.

At the technical level, workflow interoperability is primarily addressed through execution-oriented representations such as the PWD. It enables workflows to be exchanged between heterogeneous WfMSs while preserving executable workflow logic and software environments. This provides a practical foundation for FAIR and reusable workflows across Python-based materials science infrastructures. However, PWD intentionally focuses on workflow logic and portability rather than conceptual meaning or scientific interpretation. Consequently, PWD alone cannot fully capture the semantic relationships between workflow entities, scientific quantities, experimental procedures, or provenance structures.

Similarly, the ELN file format provides a standardized mechanism for exchanging experimental data and results [57]. In addition, extending workflow definitions to incorporate standard operating procedures (SOPs) for experimentally relevant workflow steps involving human interaction enables the structured representation of experimental workflows within MAPs [62].

The missing semantic relationships are addressed by semantic frameworks such as atomRDF [11], semantikon [10], and OpenSemanticLab (OSL) [60], which extend workflow interoperability toward machine-interpretable scientific knowledge. atomRDF enriches workflow-generated metadata through ontology-based RDF representations aligned with CMSO and ASMO, thereby transforming atomistic simulation workflows into semantically queryable knowledge graphs. Semantikon is an alternative approach that embeds semantic annotations directly into workflow node definitions through lightweight Python typehints. Importantly, semantikon can be directly connected to the PWD representation, demonstrating a synergy between execution-level interoperability and semantic interoperability. PWD provides the portable workflow graph, while semantikon adds ontology-aware semantic meaning, validation capabilities, and RDF integration. Finally, OSL integrates ontology-driven semantic annotation for experimental workflows with user interfaces and knowledge graphs. Together, they establish a workflow representation that remains both executable and semantically machine-actionable, covering both computational and experimental workflows, as illustrated in Figure 5.

A shared technical foundation underlying several of these approaches is Object-Oriented Linked Data (OO-LD) [44], which provides a framework for combining JSON-Schema-based data modeling with JSON-LD-based semantic annotation. OO-LD is used within AWL for mapping AST structures to RDF, within OpenSemanticLab for auto-generated ontology-driven interfaces, and within the LLM tool interfaces described in Section 10 for semantically annotated pydantic dataclasses.

AWL [9] occupies a complementary position within this ecosystem by addressing interoperability at the level of program logic itself. Unlike workflow-centric approaches that primarily represent execution graphs, AWL transforms abstract syntax trees into semantically structured RDF graphs that preserve the program

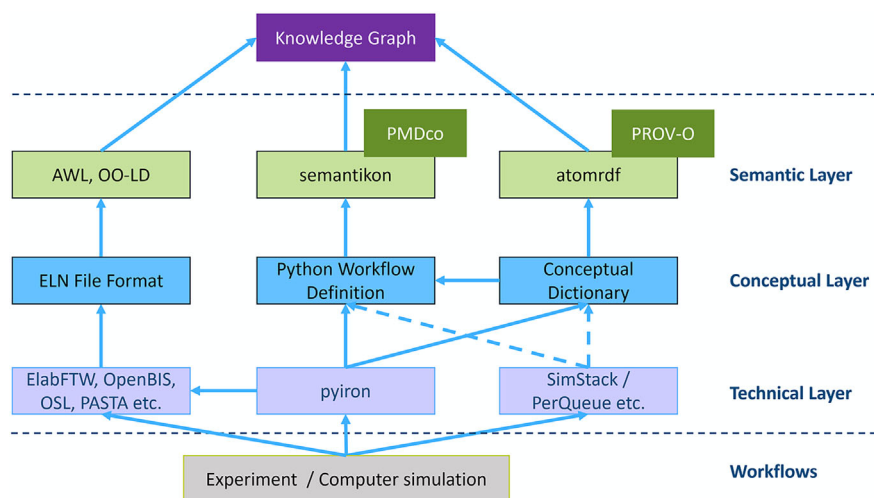


FIGURE 5 | Multilayered digital interoperability ecosystem from workflow execution to knowledge graph integration.

structure, control flow, and logical dependencies. As a result, AWL provides a semantic representation of workflows that is largely independent of specific workflow management systems or scientific domains. This enables workflows to become analyzable software artifacts within knowledge graphs, complementing the more execution-oriented abstractions of PWD and the domain-oriented semantic annotations provided by semantikon and atomRDF. In this sense, AWL can be interpreted as a bridge between source-code semantics and workflow-level semantic infrastructures.

The strongest synergy effects arise when these approaches are combined into integrated workflow ecosystems. PWD enables workflows to move between workflow systems; semantikon enriches these workflows with semantic annotations; atomRDF converts workflow metadata into ontology-aligned RDF knowledge graphs; AWL captures executable program logic as semantically queryable structures; and PMDco [48] provides a shared conceptual framework capable of integrating workflows, experiments, simulations, and materials data into coherent semantic infrastructures. Together, these components establish the basis for interoperable, provenance-rich, and machine-actionable workflow environments that extend far beyond simple workflow execution.

These complementarities become particularly important in the context of MAPs), where experimental devices, computational workflows, laboratory procedures, and AI systems must interact continuously [56]. Here, interoperability can no longer be limited to data exchange alone. Instead, workflows must preserve the semantic meaning, provenance, parameter context, and conceptual consistency across heterogeneous infrastructures. Ontology-driven semantic layers enable automated validation and workflow composition, while execution-oriented workflow abstractions ensure scalability and reproducibility across distributed computational environments. Human-in-the-loop laboratory workflows further demonstrate that interoperability must encompass not only automated software components but also human interactions, instrumentation, and procedural knowledge [62].

The convergence of these approaches is also highly relevant for emerging AI-assisted scientific infrastructures. Large language models and autonomous workflow agents increasingly depend on semantically annotated and machine-readable workflow components in order to discover, compose, validate, and execute workflows automatically. In this context, semantic workflow representations become a prerequisite for trustworthy AI-driven scientific automation. PWD provides reusable executable building blocks, semantikon, and atomRDF expose semantic relationships and provenance, AWL enables the semantic interpretation of program logic, and PMDco supplies the conceptual consistency required for integrating workflows into domain knowledge graphs. Their combination therefore creates an ecosystem in which workflows are simultaneously executable, semantically interpretable, discoverable, and reusable by both humans and AI systems.

Beyond further technological developments, broad community adoption will require objective benchmarking and accessible pathways for incremental integration. Since the presented technologies address complementary layers of interoperability rather than competing implementations, future benchmark suites should focus on evaluating workflow portability, semantic expressiveness, provenance completeness, reasoning capabilities, and integration effort across representative computational and experimental workflows. Such benchmarks would help external groups identify appropriate technologies for their specific use cases while also guiding future development. Equally important is minimizing the entry barrier. The PMD ecosystem is designed to support incremental adoption: existing workflows can first be exchanged through interoperable representations such as the Python Workflow Definition, followed by optional semantic enrichment using frameworks such as semantikon or atomRDF, with more comprehensive ontology-based integration through PMDco becoming beneficial as workflows are shared across institutions or MAPs. Consequently, most users are expected to interact primarily with high-level workflow libraries and reusable semantic components, while detailed ontology development remains the responsibility of infrastructure providers and domain experts.

12 | Conclusions

Collectively, the presented developments establish a layered interoperability architecture. Conceptual interoperability (Section 2) informs workflow design, WfMSs and the PWD (Sections 3 and 4) provide execution interoperability, atomRDF, AWL, PMDco, and semantikon (Sections 5–8) introduce progressively richer semantic representations, while MAPs and LLM-assisted workflows (Sections 9 and 10) demonstrate how these complementary technologies enable the next generation of autonomous materials research infrastructures.

The workflow management systems discussed in this review demonstrate that different scientific communities require different balances between scalability, usability, semantic richness, and automation. Consequently, interoperability is more realistically achieved through compatible abstraction layers and shared semantic representations than through the adoption of a single universal workflow system. In this context, execution-oriented interoperability approaches provide portable and reusable workflow representations that allow workflows to move between heterogeneous workflow ecosystems while preserving executable logic and reproducibility. Complementarily, AWL captures the program structure and workflow logic as semantically analyzable RDF graphs without any specific prerequisites.

At the same time, semantic interoperability requires workflows and datasets to retain their scientific meaning independently of the software environment in which they were generated. Ontology backbones such as PMDco and ontology-driven approaches such as atomRDF, OSL, and semantikon therefore extend interoperability toward machine-readable and semantically structured knowledge representations. Together, these approaches transform workflows from isolated execution artifacts into semantically transparent, provenance-rich, and machine-actionable scientific objects.

The growing importance of MAPs further reinforces the need for such integrated semantic workflow infrastructures. Future materials discovery environments will increasingly combine simulations, automated laboratories, experimental devices, human-in-the-loop interactions, and AI-driven orchestration systems within unified workflows. That is, LLMs and AI-assisted workflow systems introduce both opportunities and new requirements for interoperability. Achieving this new level of interoperability in these environments requires semantically annotated workflows capable of preserving provenance, conceptual meaning, and parameter context across distributed infrastructures. Hybrid experimental–computational workflows therefore represent one of the most important future application areas for semantic workflow technologies.

In consequence, several practical recommendations emerge. First, interoperability should be considered already during workflow design rather than being treated as a later integration problem. Second, semantic annotations and provenance capture should be integrated as close as possible to workflow execution and data generation. Third, ontology-driven representations should be developed incrementally and modularly, allowing workflows to remain usable while progressively improving the semantic interoperability. Finally, we outline emerging directions toward converging workflow standards, including community benchmarks for interoperable workflows and semantic

technologies, as well as practical pathways for adoption by external laboratories.

Collectively, the workflow and semantic tools developed in European digitalization initiatives in materials science operationalize the proposed guidelines by embedding interoperability, provenance capture, and reusable ontology-based annotations while exposing machine-readable interfaces for future AI-assisted automation.

Acknowledgments

ChatGPT 5.5 has been used to revise the text of the finalized manuscript and to improve the consistency between the different paragraphs. The authors thank the German Federal Ministry of Research, Technology and Space (BMFTR) for financial support of the project Innovation-Platform MaterialDigital through project funding Grants no. 13XP5094E (BAM), 13XP5094C (MPI SusMat), 13XP5094F (FIZ), and 13XP5094A (KIT). This work also benefitted from the support by the NFDI-MatWerk consortium, funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the National Research Data Infrastructure – NFDI 38/1 – project number 460247524.

Open Access funding enabled and organized by Projekt DEAL.

Funding

This work was supported by the Bundesministerium für Bildung und Forschung (13XP5094A, 13XP5094C, 13XP5094E, and 13XP5094F) and Deutsche Forschungsgemeinschaft (460247524).

Data Availability Statement

Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

References

1. S. Bekemeier, C. R. C. Rêgo, H.-L. Mai, et al., “Advancing Digital Transformation in Material Science: The Role of Workflows Within the Materialdigital Initiative,” *Advanced Engineering Materials* 27, no. 8 (2024): 202402149.
2. L. M. Ghiringhelli, C. Baldauf, T. Bereau, et al., “Shared Metadata for Data-Centric Materials Science,” *Scientific Data* 10, no. 1 (2023): 626.
3. MaterialDigital, accessed May 9, 2026, <https://www.material-digital.de>.
4. NFDI-MatWerk, accessed May 9, 2026, <https://nfdi-matwerk.de>.
5. MaterialsCommons, accessed May 9, 2026, <https://www.materialscommons.eu>.
6. awesome-workflow-engines, accessed May 9, 2026, <https://github.com/meirwah/awesome-workflow-engines>.
7. S. R. Wilkinson, M. Alokala, K. Belhajjame, et al., “Applying the FAIR Principles to Computational Workflows,” *Scientific Data* 12, no. 1 (2025): 328.
8. B. Bayerlein, J. Waitelonis, H. Birkholz, et al., “Concepts for a Semantically Accessible Materials Data Space: Overview over Specific Implementations in Materials Science,” *Advanced Engineering Materials* 27 (2024): 2401092.
9. S. P. Stier, L. Gold, N. Wilhelm, and D. Nusko, “The Abstract Workflow Language (AWL) and It’s Linked Data Extension (AWL-LD) - Application of OO-LD on Abstract Syntax Trees in Order Share Workflow Descriptions and Map Them Bidirectional to RDF,” (2025), <https://github.com/OO-LD/awl-schema>.

10. Semantikon, accessed May 9, 2026, <https://github.com/pyiron/semantikon>.
11. A. A. Guzman, S. Menon, T. Hickel, and S. Sandfeld, "Ontology-Based Knowledge Graph Infrastructure for Interoperable Atomistic Simulation Data," (2026), <https://arxiv.org/abs/2604.06230>.
12. J. Janssen, J. George, J. Geiger, et al., "A Python Workflow Definition for Computational Materials Design," *Digital Discovery* 4 (2025): 3149.
13. F. Roters, M. Diehl, P. Shanthraj, et al., "DAMASK – the Düsseldorf Advanced Material Simulation Kit for Modeling Multi-Physics Crystal Plasticity, Thermal, and Damage Phenomena from the Single Crystal up to the Component Scale," *Computational Materials Science* 158 (2019): 420.
14. J. Schaarschmidt, J. Yuan, T. Strunk, et al., "Workflow Engineering in Materials Design Within the Battery 2030+ Project," *Advanced Energy Materials* 12, no. 17 (2021): 2102638.
15. M. Bruns, S.-M. Wi, J. Zhao, et al., "Supporting AI Readiness through Digital Workflows in Materials Science," *Advanced Engineering Materials* 28, no. 20, <https://doi.org/10.1002/adem.71262>.
16. B. H. Sjölin, W. S. Hansen, A. A. Morin-Martinez, et al., "PerQueue: Managing Complex and Dynamic Workflows," *Digital Discovery* 3 (2024): 1832.
17. S. K. Steensen, P. W. Butler, J. R. Pedersen, et al., "The Necessity of Dynamic Workflow Managers for Advancing Self-Driving Labs and Optimizers," *Advanced Intelligent Discovery* 2 (2026): 202500067.
18. J. Janssen, M. G. Taylor, P. Yang, J. Neugebauer, and D. Perez, "Executorlib – Up-Scaling Python Workflows for Hierarchical Heterogenous High-Performance Computing," *Journal of Open Source Software* 10, no. 108 (2025): 7782.
19. J. Janssen, S. Surendralal, Y. Lysogorskiy, et al., "pyiron: An Integrated Development Environment for Computational Materials Science," *Computational Materials Science* 163 (2019): 24.
20. M. Stricker, L. Banko, N. Sarazin, et al., "Computationally Accelerated Experimental Materials Characterization – Drawing Inspiration From High-Throughput Simulation Workflows," *npj Computational Materials* 12 (2025): 2, <https://doi.org/10.1038/s41524-025-01919-5>.
21. C. R. C. Rêgo, J. Schaarschmidt, T. Schlöder, et al., "SimStack: An Intuitive Workflow Framework," *Frontiers in Materials* 9 (2022): 877597.
22. S. K. Steensen, T. S. Thakur, M. Dillenz, et al., "The Interoperability Challenge in DFT Workflows Across Implementations," *Advanced Intelligent Discovery* (2026): e202500232.
23. G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, "AiiDA: Automated Interactive Infrastructure and Database for Computational Science," *Computational Materials Science* 111 (2016): 218.
24. A. S. Rosen, M. Gallant, J. George, et al., "Jobflow: Computational Workflows Made Simple," *Journal of Open Source Software* 9, no. 93 (2024): 5995.
25. C. de Visser, L. F. Johansson, P. Kulkarni, et al., "Ten Quick Tips for Building FAIR Workflows," *PLOS Computational Biology* 19, no. 9 (2023): 1.
26. M. R. Crusoe, S. Abeln, A. Iosup, et al., "Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language," *Communications of the ACM* 65, no. 6 (2022): 54–63.
27. A. A. Guzman, "Computational Material Sample Ontology (CMSO)," (2024), <https://doi.org/10.5281/zenodo.10805536>.
28. A. A. Guzman, "Atomistic Simulation Methods Ontology (ASMO)," (2024), <https://doi.org/10.5281/zenodo.10805591>.
29. A. A. Guzman, H.-T. Luu, S. Menon, T. Hickel, N. Merkert, and S. Sandfeld, "Towards Knowledge-Based Workflows: A Semantic Approach to Atomistic Simulations for Mechanical and Thermodynamic Properties," (2026), <https://arxiv.org/abs/2602.01358>.
30. A. A. Guzman, H.-T. Luu, and S. Menon, "semantic-workflows: Semantic Workflows for Atomistic Simulations," 2025, GitHub Repository, accessed May 11, 2026, <https://github.com/pyscal/semantic-workflows>.
31. S. Menon, and A. Azocar Guzman, "conceptual_dictionary," (2026), https://github.com/OCDO/conceptual_dictionary/.
32. G. Kresse and J. Furthmüller, "Efficient Iterative Schemes for Ab Initio Total-Energy Calculations Using a Plane-wave Basis Set," *Physical Review B* 54 (1996): 11169.
33. G. Kresse and J. Hafner, "Ab Initio Molecular Dynamics for Liquid Metals," *Physical Review B* 47, no. 1 (1993): 558.
34. G. Kresse and J. Furthmüller, "Efficiency of Ab-Initio Total Energy Calculations for Metals and Semiconductors Using a Plane-Wave Basis Set," *Computational Materials Science* 6, no. 1 (1996): 15.
35. A. P. Thompson, H. M. Aktulga, R. Berger, et al., "LAMMPS - A Flexible Simulation Tool for Particle-Based Materials Modeling at the Atomic, Meso, and Continuum Scales," *Computer Physics Communications* 271 (2022): 108171.
36. P. Giannozzi, O. Andreussi, T. Brumme, et al., "Advanced Capabilities for Materials Modelling With Quantum ESPRESSO," *Journal of Physics: Condensed Matter* 29, no. 46 (2017): 465901.
37. P. Giannozzi, S. Baroni, N. Bonini, et al., "QUANTUM ESPRESSO: A Modular and Open-Source Software Project for Quantum Simulations of Materials," *Journal of Physics: Condensed Matter* 21, no. 39 (2009): 395502.
38. A. Jain, S. P. Ong, G. Hautier, et al., "Commentary: The Materials Project: A Materials Genome Approach to Accelerating Materials Innovation," *APL Materials* 1, no. 1 (2013): 011002.
39. M. L. Evans, J. Bergsma, A. Merkys, et al., "Developments and Applications of the OPTIMADE API for Materials Discovery, Design, and Data Exchange," *Digital Discovery* 3 (2024): 1509.
40. T. Lebo, S. Sahoo, D. McGuinness, et al., "PROV-O: The PROV Ontology," W3C Recommendation (2013), <http://www.w3.org/TR/2013/REC-prov-o-20130430/>.
41. GitHub, "Platform MaterialDigital Core Ontology (PMDco)," accessed December 18, 2025, <https://github.com/materialdigital/core-ontology/>.
42. R. Arp, B. Smith, and A. D. Spear, *Building Ontologies With Basic Formal Ontology* (MIT Press, 2015).
43. T. Prudhomme, G. De Colle, A. Liebers, et al., "A Semantic Approach to Mapping the Provenance Ontology to Basic Formal Ontology," *Scientific Data* 12, no. 1 (2025): 282.
44. S. Stier and L. Gold, "OO-LD/Schema: The Object Oriented Linked Data Schema Based on JSON-LD and JSON-Schema," (2025), <https://github.com/OO-LD/schema>.
45. H. Knublauch and D. Kontokostas, "Shapes Constraint Language (SHACL), W3C Recommendation," (W3C, 2017), <https://www.w3.org/TR/shacl/>.
46. S. Stier and S. Feiler, "OO-LD/AWL-Python: Python Implementation of the Abstract Workflow Language (AWL/AWL-LD)," (2025), <https://github.com/OO-LD/oold-python>.
47. E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF, W3C Recommendation," (W3C, 2008), <https://www.w3.org/TR/rdf-sparql-query/>.
48. B. Bayerlein, M. Schilling, H. Birkholz, et al., "PMD Core Ontology: Achieving Semantic Interoperability in Materials Science," *Materials & Design* 237 (2024): 112603.
49. GitHub, "Basic Formal Ontology (BFO)," accessed December 18, 2025, <https://github.com/BFO-ontology/BFO/>.

50. GitHub, "OBO Relations Ontology (RO)," accessed December 18, 2025, <https://github.com/oborel/obo-relations/>.
51. GitHub, "Information Artifact Ontology (IAO)," accessed December 18, 2025, <https://github.com/information-artifact-ontology/IAO/>.
52. GitHub, "Ontology for Biomedical Investigations (OBI)," accessed December 18, 2025, <https://github.com/obi-ontology/obi/>.
53. M. Schilling, B. Bayerlein, P. von Hartrott, et al., "FAIR and Structured Data: A Domain Ontology Aligned with Standard-Compliant Tensile Testing," *Advanced Engineering Materials* 27, no. 8 (2024): 2400138.
54. B. Bayerlein, M. Schilling, M. Curran, et al., "Natural Language Processing-Driven Microscopy Ontology Development," *Integrating Materials and Manufacturing Innovation* 13 (2024): 915–926, <https://doi.org/10.1007/s40192-024-00378-y>.
55. International Organisation for Standardization, "Information Technology - Top-Level Ontologies (TLO) - Part 2: Basic Formal Ontology (BFO)," (International Organisation for Standardization, 2021), Standard, <https://www.beuth.de/de/norm/iso-iec-21838-2/348948268>.
56. S. Stier, C. Kreisbeck, H. Ihssen, et al., "Materials Acceleration Platforms (MAPs): Accelerating Materials Research and Development to Meet Urgent Societal Challenges," *Advanced Materials* 36 (2024): 45.
57. ELN File Format, accessed May 9, 2026, <https://github.com/TheELNConsortium/TheELNFileFormat>.
58. M. Vogler, J. Busk, H. Hajiyani, et al., "Brokering Between Tenants for an International Materials Acceleration Platform," *Matter* 6, no. 9 (2023): 2647.
59. A. D. Fuchs, J. A. f. Lehmeyer, H. Junkes, H. B. Weber, and M. Krieger, "NOMAD CAMELS: Configurable Application for Measurements, Experiments and Laboratory Systems," *Journal of Open Source Software* 9, no. 95 (2024): 6371.
60. S. P. Stier, L. Gold, A. Räder, M. A. Popp, A. Triol, and L. Scheuplein, "The Opensemanticalab Platform," 2023, <https://github.com/OpenSemanticLab>.
61. M. Schilling, S. Bruns, B. Bayerlein, et al. "Seamless Science: Lifting Experimental Mechanical Testing Lab Data to an Interoperable Semantic Representation," *Advanced Engineering Materials* 27, no. 8 (2025): 2401527, <https://doi.org/10.1002/adem.202401527>.
62. S. Brinckmann, S. Menon, R. Röske, et al., "A Lightweight Procedural Layer for Hybrid Experimental–Computational Workflows in Materials Science," *Advanced Engineering Materials* (2026): e202503176, <https://doi.org/10.1002/adem.202503176>.
63. "Skills," accessed May 11, 2026, <https://claude.com/blog/skills>.
64. Y. Zimmermann, A. Bazgir, A. Al-Feghali, et al., "32 Examples of LLM Applications in Materials Science and Chemistry: Towards Automation, Assistants, Agents, and Accelerated Scientific Discovery," *Machine Learning: Science and Technology* 6, no. 3 (2025): 030701.
65. Z. Wang, H. Huang, H. Zhao, et al., "Dreams: Density Functional Theory Based Research Engine for Agentic Materials Simulation," 2025, <https://arxiv.org/abs/2507.14267>.
66. Langgraph, accessed May 11, 2026, <https://www.langchain.com/langgraph>.