Science and Technology Facilities Council

Technical Report
STFC-TR-2026-011

PSDI data lakehouse: technical report

A Pawula Hewage, A Belozerov,
V Bunakov, S Meacham, T Underwood

June 2026

Enquiries concerning this report should be addressed to:

RAL Library
STFC Rutherford Appleton Laboratory
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445577
email: library@stfc.ac.uk

Science and Technology Facilities Council reports are available online at:
https://epubs.stfc.ac.uk

Accessibility: a Microsoft Word version of this document (for use with assistive technology) may be available on request.

DOI: 10.5286/stfctr.2026011

ISSN 2753-5797

STFC Author Identifiers (ORCIDs)

Author ORCIDs are provided where available.

Amali Pawula Hewage	0009-0007-1746-7685
Alexander Belozerov	0000-0001-9506-8718
Vasily Bunakov	0000-0003-3467-5690
Stuart Meacham	0000-0002-3667-6059
Tom Underwood	0000-0001-6431-0612

PSDI Data Lakehouse: Technical Report

Amali Pawula Hewage, Alexander Belozerov, Vasily Bunakov, Stuart Meacham, Tom Underwood

STFC Scientific Computing Department

Table of Contents

Introduction

PSDI (Physical Sciences Data infrastructure) [1] develops tools, services and digital content in support of broadly defined physical sciences. The PSDI data lakehouse offers a solution for FAIR publishing [2] of diverse datasets and for pairing them with programmatic tools, so that all sorts of data wrangling can be done via remote requests to data deposits in the Cloud.

This report briefly walks through the history of PSDI lakehouse implementation before outlining the core technology used, the content currently offered and the user interfaces available.

We then proceed to the discussion of benefits of the lakehouse and of trade-offs made in its implementation and indicate the directions of future work.

The appendices contain descriptions of datasets and give examples of data visualisations generated by requests to the lakehouse.

The generous references section should help a curious reader with a deeper exploration of the lakehouse foundations and possible extensions.

History and evolution of the PSDI data lakehouse

The lakehouse has been developed through several phases indicated in Table 1.

Period	**Activity**
Spring–Summer 2025	Technology and data landscaping
July–October 2025	Technology and data studies; initial report to PSDI
October 2025–May 2026	Core technology hardening; interfaces development; production release candidate
Summer 2026	Expected production release

Table 1. PSDI data lakehouse development phases.

A key milestone was the identification of Apache Iceberg [3][4] as the likely winner in a competition of lakehouse specifications, alongside the maturation of the relevant open-source software ecosystem and the availability of new interesting datasets exemplified by OMat24 [5][6] and OMol25 [7][8].

As of May 2026, the lakehouse has reached production readiness, with the following characteristics:

- Highly available deployment on Kubernetes
- Inexpensive object store backend, ready to scale
- A selection (a “seed”) of popular datasets prepared and tested
- Integration with the PSDI user management solution and with a policy engine
- Documented examples in the form of computational notebooks

The lakehouse pre-production status was reported in a dedicated briefing [9] with relevant research groups in STFC, University of Southampton and University College London.

Core Technology

Apache Iceberg [3][4] serves as the open table format underpinning the lakehouse; it has become a de facto winner over alternative specifications such as Delta Lake [10] or Apache Hudi [11] – in a sense that the community and the amount of open-source software supporting Apache Iceberg are significantly bigger compared to competitors.

Apache Iceberg is a specification or, strictly speaking, a set of a few specifications: for the tabular format [14], for indexes and statistics [15], for a catalogue API [16] and for views [17] that allow interoperability with compute engines such as Trino [18] or Apache Spark [19].

The Iceberg specification(s) can be implemented by different software engines, commercial or open-source. Our implementation consciously follows the open-source route, with the ambition to keep using the open-source building blocks for the delivery of an enterprise-grade solution, where “enterprise-grade” means not only the scale of data or user base but other aspects, too, such as data governance with policy-driven procedures and mechanisms.

The lakehouse is structured around five architectural layers represented by Figure 1.

The data ingestion layer acquires raw scientific datasets from their respective sources, transforms them from their initial formats to a common format, then loads them to the storage layer. Apache Airflow [12] was chosen as the engine for orchestration in the ingestion layer because of its ability to define a combination of data processing tasks as Directed Acyclic Graphs (DAGs) through Python code. The DAGs are easily versioned in a code repository, they are testable and reusable. Airflow control panel features a scheduler that enables retries in case of failure and gives the status of ingestion workflows.

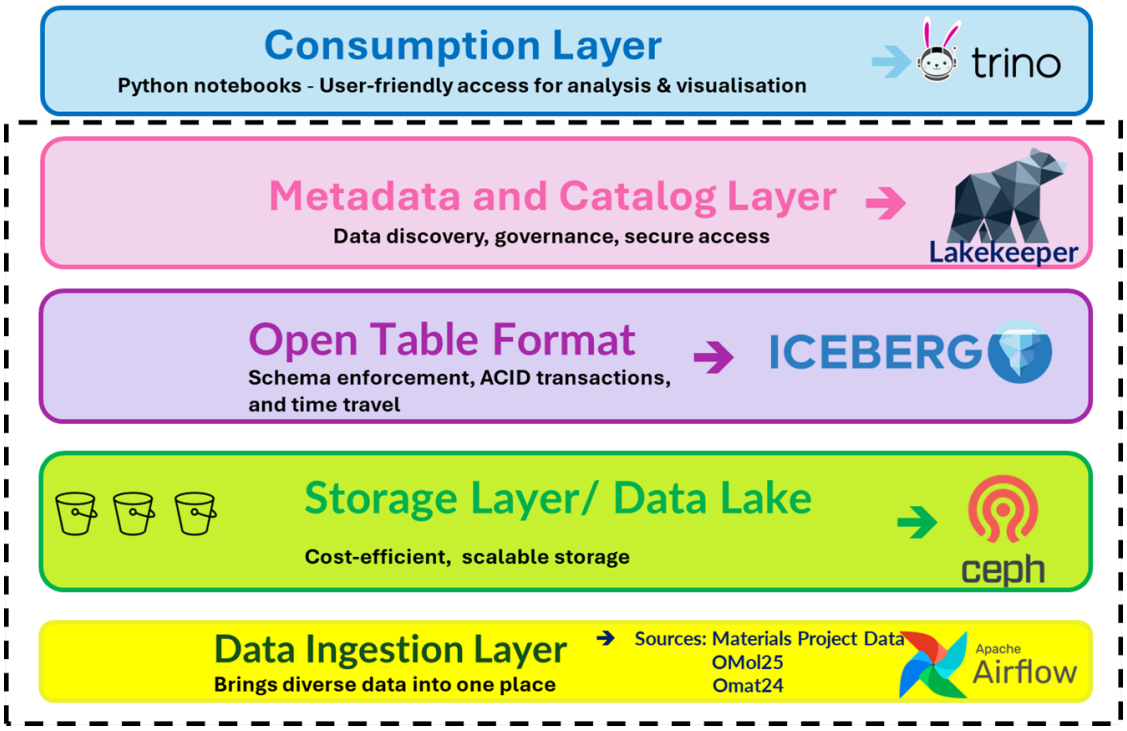

Figure 1. Lakehouse layers.

The storage layer is designed according to "medallion" architecture pattern [13], with the Bronze bucket storing raw data, the Silver bucket storing data transformed in Parquet format [20], and the Gold bucket storing Iceberg data.

All storage is managed by Ceph [21]. Ceph was selected as it is already available within our on-premises infrastructure, provides an S3-compatible API, and is cost-efficient at scale. Other components in the lakehouse stack communicate with the storage via a standard S3 API; if we ever need to migrate to an S3-compatible alternative such as AWS S3 or Azure object storage or MinIO, the existing storage can be easily replaced.

While Ceph S3 object storage provides a capacity to store data, the format in which data is stored within that storage has a significant impact on query performance and usability by downstream applications, e.g. for Machine Learning. Apache Parquet [20] was selected as the common file format across all datasets in our lakehouse; all datasets that we ingest are converted to Parquet and stored in the Silver bucket.

The tabular layer can be considered a "virtual" or "conceptual" one with the actual IT components for it split between the aforementioned Gold bucket and the Iceberg data catalogue. Parquet has no concept of transactions, no schema versioning, no ability to query data as it existed at a previous point in time, and no mechanism for safely updating or deleting records without rewriting entire files. This is where Apache Iceberg tables sitting on top of Parquet files come into play, adding the following capabilities:

1. Schema enforcement and evolution.
2. ACID transactions.
3. Time travel and snapshot isolation: every change to an Iceberg table creates a new snapshot.
4. Partition pruning and predicate pushdown - skipping irrelevant files entirely when executing filtered queries, which significantly improves query performance on large datasets.

For the Iceberg data catalogue, we use Lakekeeper [22] that implements the REST API catalogue specification [16]. The role of the catalogue is to make the lakehouse content findable, manage permissions and route client software requests to Iceberg tables. Lakekeeper was selected over alternative catalogue implementations for the following reasons:

1. It runs as a single pod backed by an external PostgreSQL database, with no additional dependencies.
2. It implements the Iceberg REST catalogue specification, ensuring compatibility with any Iceberg-compliant query engine.
3. It integrates well with fine-grained authorisation systems.
4. It is an actively maintained project with a growing community.

For a consumption layer that exposes Iceberg tables to users, we use Trino [18] that in turn can be queried by various clients; we list some of them in the “User interfaces” section. The architecture of Trino enables the separation of the query execution into a coordinator node that receives queries, plans their execution and coordinates the results, and several worker nodes that execute fragments of queries in parallel. Trino's own Iceberg connector allows communicating with Lakekeeper for tables metadata information and for reading Parquet data from Ceph S3 without full data copying.

The interaction between the user and the lakehouse can be performed through a client software, e.g. with regular Python function calls or a Business Intelligence tool. Users do not see the underlying stack of technologies in the lakehouse, they only interact with the consumption layer: in our IT stack, with Trino.

The general flow of data from the original source is through the sequence of buckets in the storage layer, ending in a registration with data catalogue and the Trino engine that allows queries to Iceberg tables whilst giving respect to permissions defined with the Open Policy Agent [23]. We currently define one policy only that allows all users an equal readonly access to all Iceberg tables.

The data workflow implemented is depicted by Figure 2.

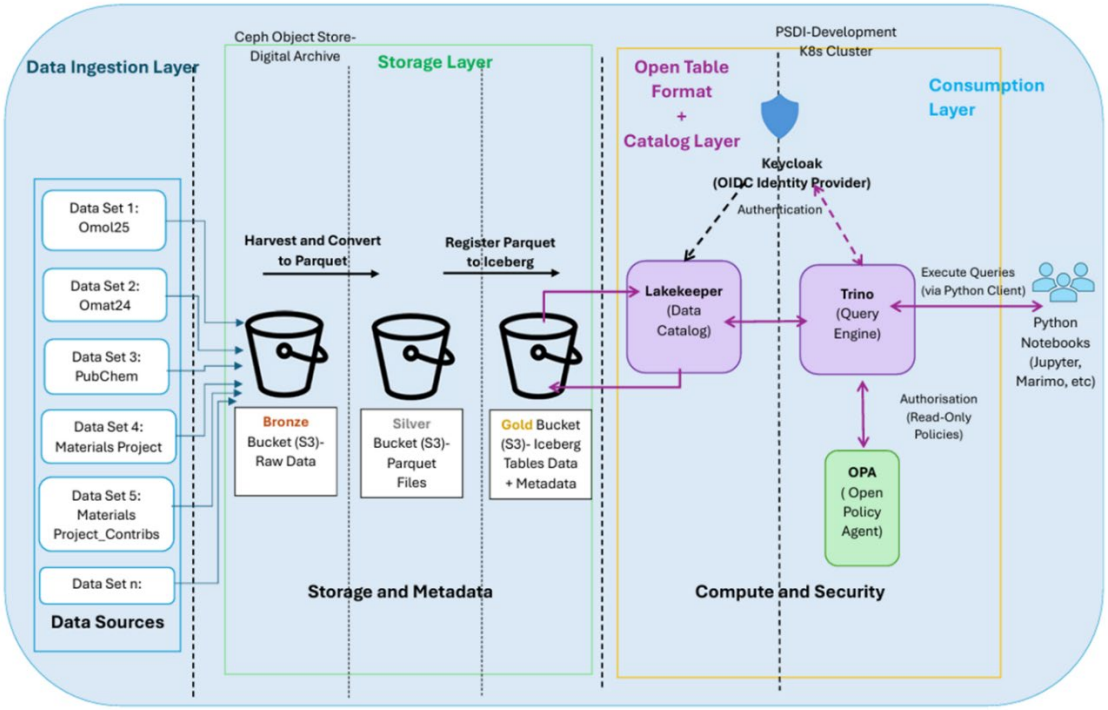

Figure 2. Data workflow in the lakehouse.

The lakehouse is deployed on Kubernetes that provides resilience and scalability, see Figure 3. The deployments propagate through multiple environments: Development, Staging, Production and are implemented as self-documented CI/CD pipelines in GitHub.

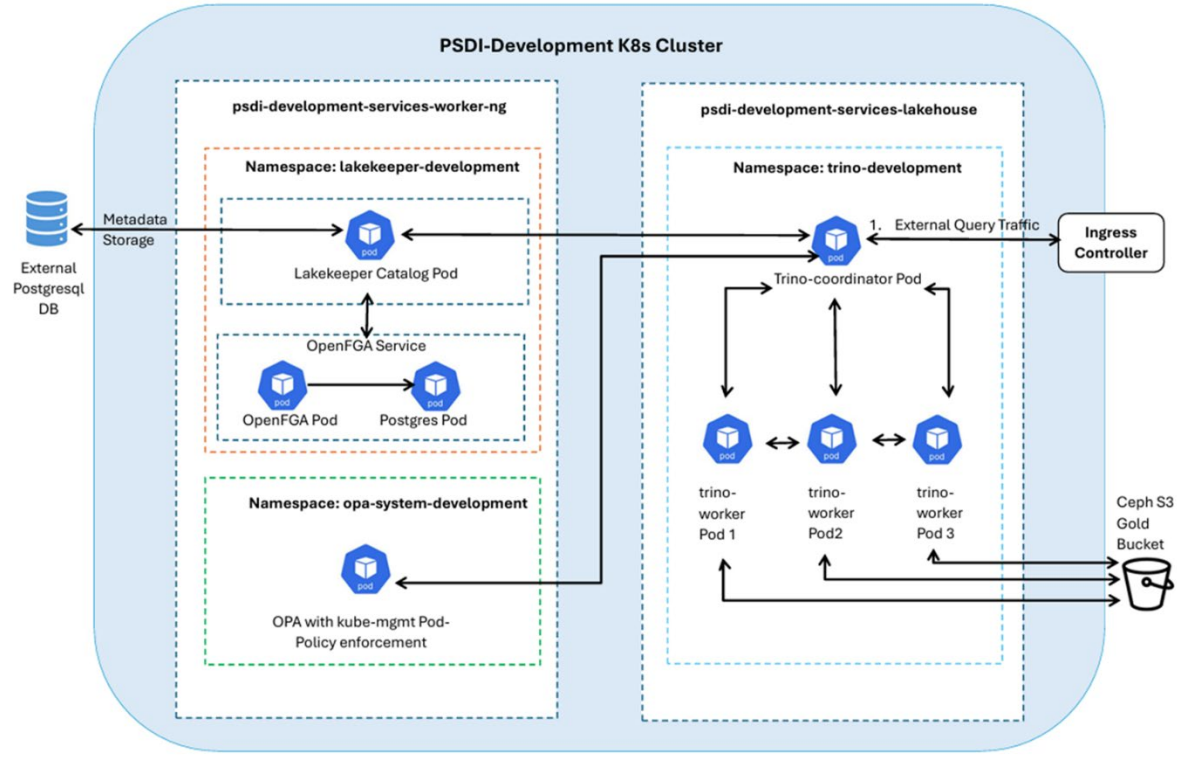

Figure 3. Lakehouse deployment topology.

The Trino deployment is configured on a dedicated Kubernetes node group. This separation ensures that memory-intensive Trino worker pods do not compete for resources with Lakekeeper and OPA (Open Policy Agent) services, and that a resource-intense events in the query layer do not affect the availability of other services. This also provides a clear operational boundary, so that the Trino node group can be scaled independently from other services.

Depending on the architectural point of view, data pipelines should or should not be considered a part of the lakehouse. The pipelines are not in scope of this report; our current approach is using Apache Airflow [12] as the main technology for building pipelines but this may change or may be extended in the future by pairing Airflow with other tools.

Lakehouse content

The IT stack introduced in the "Core Technology" section is only as good as it can serve content (data) that brings value to the lakehouse users. The architecture being layered and modular helps tweaking it if required, e.g. replacing components with their alternatives that implement the same specifications. Design of the lakehouse and its content are not entirely separate: the choice of data to serve can influence the design, and the design can determine what data can or cannot be served efficiently.

The choice of the "seed" datasets to publish in the lakehouse has been driven by the following factors:

- The intention to demonstrate that sizeable datasets of over 100 million records can be served with reasonable performance even if the inexpensive object store backend is slow compared to a file system or a database storage.
- The datasets were chosen so that they could be "flattened" to a columnar representation with not much loss to their original expressivity, so that the columnar representation remains comprehensible and usable.
- The datasets were skewed towards those used by materials science community, as serving this community is prominent in PSDI [1] and in other projects pursued by our group [24], [25].

The "seed" datasets offered with the initial setup of a lakehouse are listed in the Appendix A. We do not claim any additional rights for the data converted in the lakehouse format; we suggest that the lakehouse users respect data licences of the original datasets.

User interfaces

Trino [18] used as our lakehouse consumption layer can serve any Web client, desktop client or machine agent or script or workflow engine that can talk SQL. Over 30 clients

indicated by Trino as belonging to their ecosystem[1] are certainly not a full list of integrations possible.

As the prime audience of our lakehouse are scientists, we focussed on building a set of computational notebooks, as this flavour of a tweakable client software is widely used in research. The notebooks with examples of lakehouse queries to Trino as the consumption layer are offered in two formats: Jupyter [26] and Marimo [27], with broadly similar query examples irrespective of whether the user preference is Jupyter or Marimo.

The queries in the notebooks are formulated in SQL or as dataframes with Ibis library [28]. There is a good chance that many scientists would prefer the dataframes syntax, but giving examples in SQL is important as it is a de facto "lingua franca" of data science, and as SQL auto-generation with AI tools is certainly possible, so some scientists may consider using SQL rather than dataframes.

To access the lakehouse, authentication with PSDI is required, which is prompted upon starting a request to the lakehouse in a notebook. If a user does not have a PSDI account, there is an option to create one.

The notebooks with examples are shared in a GitHub repository that is not public for the time of this report writing but is accessible by members of PSDI project and can be opened for other parties upon request. The notebooks are going to be made public and referenced in PSDI Knowledge Base [31] when a business decision is made about the lakehouse production release.

Appendix B offers data visualisations resulted from queries in the notebooks.

Benefits for a digital infrastructure and users of it

Data lakehouse represents an approach to data publishing that can complement traditional research data repositories in a value proposition by a digital research infrastructure like PSDI. Lakehouse allows direct interaction with datasets published, as users can dig into them with computational tools; it can serve sizeable datasets inexpensively with a cheap backend storage yet a reasonable queries performance; it does not require full downloads of large datasets to experience them: search, filtering or aggregations are conducted via requests to data hosted in the Cloud.

As a result, the lakehouse offers a fuller implementation of FAIR data principles [2] compared to a data repository, see Table 2 – especially but not only for large datasets that just owing to their size are not a good fit for sensibly publishing them in a repository. If one needs a sentence to describe the data lakehouse value proposition to a not necessarily

[1] Trino ecosystem: client applications. https://trino.io/ecosystem/client-application

IT-savvy research data manager or data custodian, this could be “a FAIR data publishing solution for large datasets, with a strong emphasis on Interoperability”.

FAIR Criterion	Data Repository	Data Lakehouse
Findable data	Yes	Yes
Accessible data	Yes	Yes
Interoperable data	No	Yes
Reusable data	Maybe (best effort)	Maybe (supported by design)

Table 2. Implementation of FAIR principles in data lakehouse vs data repository.

The lakehouse makes data truly interoperable and lowers barriers to data reuse by offering uniform machine interfaces for the exploitation of large datasets by research users with average (common) computer skills; it also opens an opportunity for data reuse by all sorts of machine agents including AI-enabled.

The PSDI implementation of a data lakehouse is digitally sovereign, as it is built with open-source components and deployed in STFC Cloud rather than with a public cloud provider. Depending on a policy stance, digital sovereignty can be an important factor to encourage data publishing in a way we are doing it with our lakehouse.

Table 3 summarizes benefits of the lakehouse for the digital infrastructure owner and for a researcher as an infrastructure user.

For the infrastructure owner	For a researcher (infrastructure user)
Large datasets can be published cost-efficiently using an inexpensive and scalable object store as a backend A live API allows users to query datasets directly, appealing to an audience of "doers" who are happy to dig into published data with their own queries User permissions and access policies are well managed	No need to move full datasets into a local research environment A uniform API removes the need to install or learn multiple dataset-specific connectors, or follow their updates "Time travel" is supported, enabling queries against snapshots of evolving data

For the infrastructure owner	For a researcher (infrastructure user)
Digitally sovereign deployment Fuller implementation of FAIR principles (see Table 2)	A separation of concerns: data wrangling can be offloaded onto cloud infrastructure, freeing scientists and their compute resource to focus on science

Table 3. Benefits of data lakehouse.

Trade-offs and limitations

PSDI [1] that inspired and sponsored the lakehouse development is a digital infrastructure aiming at a variety of data needs in broadly defined physical sciences, including but not limited to physical chemistry, materials research or even a molecular biology as long as it relies on experimental or computational methods rooted in physics. This breadth inevitably means that solutions delivered for PSDI, including the lakehouse, are not necessarily aimed at "champion" use cases with very specific requirements to data management or data serving, but rather cater for a broad range of research needs for data selection, filtering and aggregation – for what can be generically labelled as "data wrangling".

Our own experimentation with data and early conversations with researchers that are potential users of the PSDI lakehouse show that the open-source components chosen and assembled in a viable IT stack are adequate for a good range of "data wrangling" cases, with some trade-offs as following:

- **Universality of data representation that allows querying all data via a uniform interface versus "flattening" data in a columnar format with some loss of expressivity.** This trade-off can be later moved in the direction of richer (less flattened) data representations when we base the lakehouse on Apache Iceberg v3. The v3 specification has Variant type [29] that is better suitable for serving "nested" data structures. The current implementation is based on Iceberg v2 that does not support Variant type but is good enough for representation of some scientifically meaningful datasets.
- **Performance versus governance.** Performance is already quite good for a variety of requests (see examples in Appendix B). It could be further improved by replacing Trino [18] with a more performant consumption layer, StarRocks [30] being a good candidate for it, yet this would mean a sacrifice of governance

features that Trino offers, such as its integration with Open Policy Agent [23] that makes Trino respect roles and permissions managed by the Iceberg catalogue. Good governance in a multi-user environment is an important consideration for a universal digital infrastructure compared to bespoke setups for niche use cases serving a few users, that understandably could champion the ultimate performance over sound governance.

- **Digital sovereignty vs turnkey polished service.** PSDI lakehouse is "turnkey" and polished enough in a sense that researchers can use it straight away in computational notebooks or other client software. It is reasonably designed, reasonably tested and reasonably deployed, so it can be called an enterprise-grade IT solution suitable for exploitation at scale. Yet compared to commercial lakehouse offerings, it is not based on the most recent Iceberg specification (still on v2 as indicated above) waiting till the open-source ecosystem of tools has matured enough for the newer specification. Also, it is deployed in STFC Cloud with a lower availability compared to a typical public cloud. These limitations are the price for having a digitally sovereign IT solution with a good control over its code base, functionality, content and associated policies.

The Iceberg-based architecture also has a natural limitation of not directly serving binary formats like images; it is very much aimed at serving voluminous numeric or textual data inexpensively in a cheap object storage backend, with a good performance and with a good traceability such as "time travel" over data snapshots.

A possible strategy for imaging data could be managing metadata in a lakehouse with images stored elsewhere and referenced from metadata. Another strategy could be images segmentation and labelling, so that the lakehouse serves the reduced numeric-and-textual representations of them. Yet of course if managing images and serving them in computation is the prime use case then other solutions but an Iceberg lakehouse can be a better fit.

Directions of future work

We intend to maintain digital sovereignty of the lakehouse by using open-source components and by deployments in STFC Cloud or in other sovereign Cloud if the STFC Cloud availability levels put the lakehouse operation at a significant reputational risk.

We will be growing data collections beyond the initial seed datasets listed in Appendix A. When Iceberg v3 [29] ecosystem of open-source components and tools becomes more mature, we will start migration of the existing datasets on v3, also an acquisition of new datasets for which data conversions in v3 format are a good fit.

We will support building a user community around the lakehouse and datasets published in it. We expect assistance by PSDI project for collecting and analysing this feedback,

with our team's role more focussed on technology. One interesting development in respect to lakehouse user base could be discovering cases that require acquisition of data streams rather than static datasets, e.g. from high-throughput instruments; the Iceberg-based lakehouse is principally suitable to serve such cases.

After a certain period of operating the lakehouse with readonly user access to data, we intend to allow research communities to upload their data, thus transforming the lakehouse into a fully-fledged data publishing platform for communities who consider it a good fit for their needs.

We have already started and will continue to investigate a multi-tenant configuration of the lakehouse, so that it can potentially serve multiple communities in PSDI and beyond PSDI, along with core data offering managed by our team.

We have already invested an effort in the adoption of novel user interfaces such as Marimo notebooks [27] that have certain advantages over traditional Jupyter, and in Ibis library [28] that auto-translates Python requests in SQL, thus bringing an expressive and performant dataframe interface that looks friendly to research users.[2]

We will explore opportunities for requests acceleration; recent developments like Substrait [32] give a principal ability to offload the lakehouse requests execution onto additional high-performance components, e.g. accelerated by GPUs [33].

Feeding lakehouse data in AI is possible right now via its universal machine interface, but AI can be better served with data labelled and annotated. This may entail looking into popular specifications such as Croissant [34] that allows annotations of tabular data, hence might be suitable for annotating Iceberg representations. We can look, too, into query engines like PuppyGraph [35] that allow graph representations of tabular data with all sorts of graph algorithms becoming available then, so that virtual knowledge graphs become "rails" to better direct AI agents through the structure of data.

As the lakehouse content diversifies and grows, we'll have to invest effort in maintenance and in technology for it. What aspects of maintenance are most important is outlined by Apache Iceberg project themselves [36].

One of the drivers for why we may need more attention to maintenance in the future could be the aforementioned possibility of onboarding data streams as data sources: this may require then a specialised data compaction component, or seeing what Trino can do for

[2] Ibis is portable across SQL engines with different SQL dialects in them; this includes Trino we are currently using or its possible future replacements. Hence Ibis has a potential for becoming a valuable "invariant" of user interactions with the lakehouse, even if some components change.
Ibis has a good pedigree, it was conceived by Wes McKinney - "Benevolent Dictator for Life" (BDFL) of the popular Pandas package for data analysis. It is a part of a broader composable data ecosystem envisioned by Wes and others, see https://ibis-project.org/concepts/who.html

compaction and performance optimisation [37], or using compaction engines of EL (Extract and Load) components like OLake [38].

Maintenance technology has always been a part of commercial lakehouse solutions, and now there is a growing interest to this topic in the open-source community [39].

Appendix A. Datasets currently offered in the lakehouse

Name	Description	Records	Reference
Open Materials 2024 (OMat24)	DFT calculations on inorganic bulk materials; 400 million CPU core-hours	>100M	arXiv:2410.12771
Open Molecules 2025 (OMol25)	DFT calculations on small molecules, biomolecules, metal complexes, and electrolytes; 6 billion CPU core-hours	>100M	arXiv:2505.08762
Materials Project (MP) – AWS Open Data	Physical properties derived from DFT calculations (17 datasets)	~200k compounds	doi:10.1038/s41563-025-02272-0
Materials Project – Contribs (MPtrj & MatPes)	DFT datasets for training machine-learning interatomic potentials	>2M	doi:10.1038/s42256-023-00716-3

Table 4. "Seed" datasets offered initially.

Additionally, the processing of PubChem Compounds collection is in progress with about 120 million records in it: https://pubchem.ncbi.nlm.nih.gov/docs/compounds

Appendix B. Visualisations of queries to lakehouse

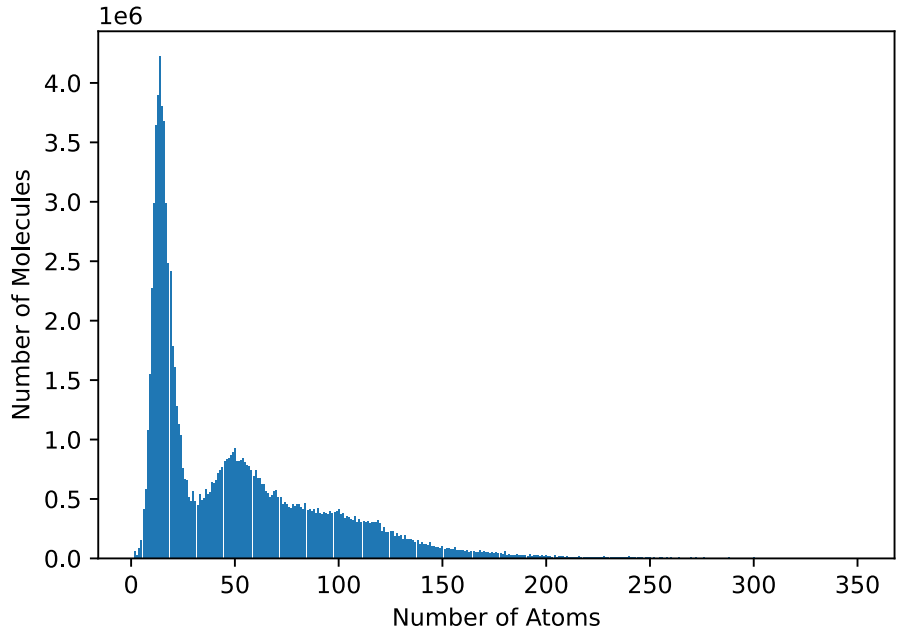

Figure 4. Distribution of molecules by number of atoms in OMol25 dataset. This request takes 5 seconds to complete, against a dataset of > 100 million records hosted in an inexpensive object store.

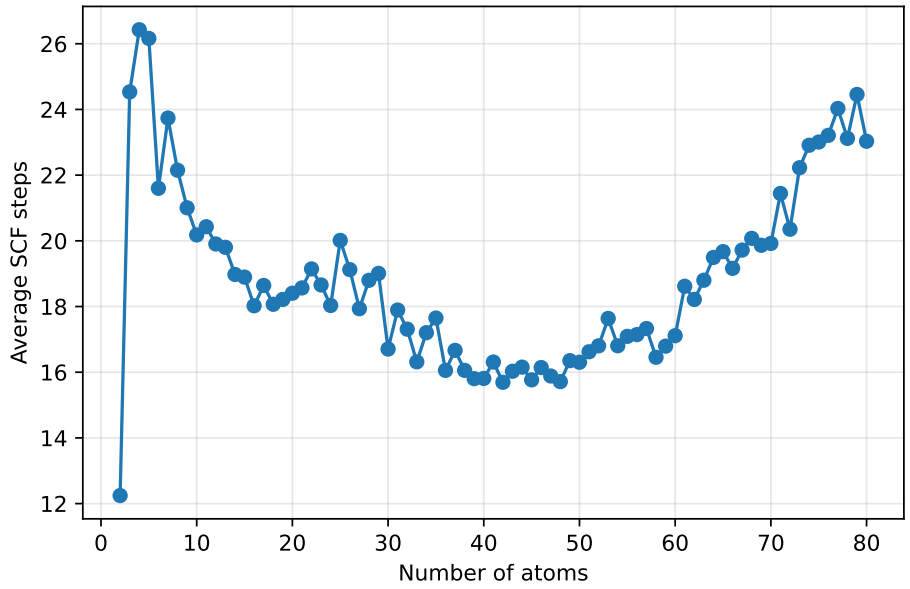

Figure 5. Average SCF iterations vs molecular size in OMol25 dataset. This request takes 12 seconds to complete, with quite heavy grouping and ordering against a dataset of > 100 million records hosted in an inexpensive object store.

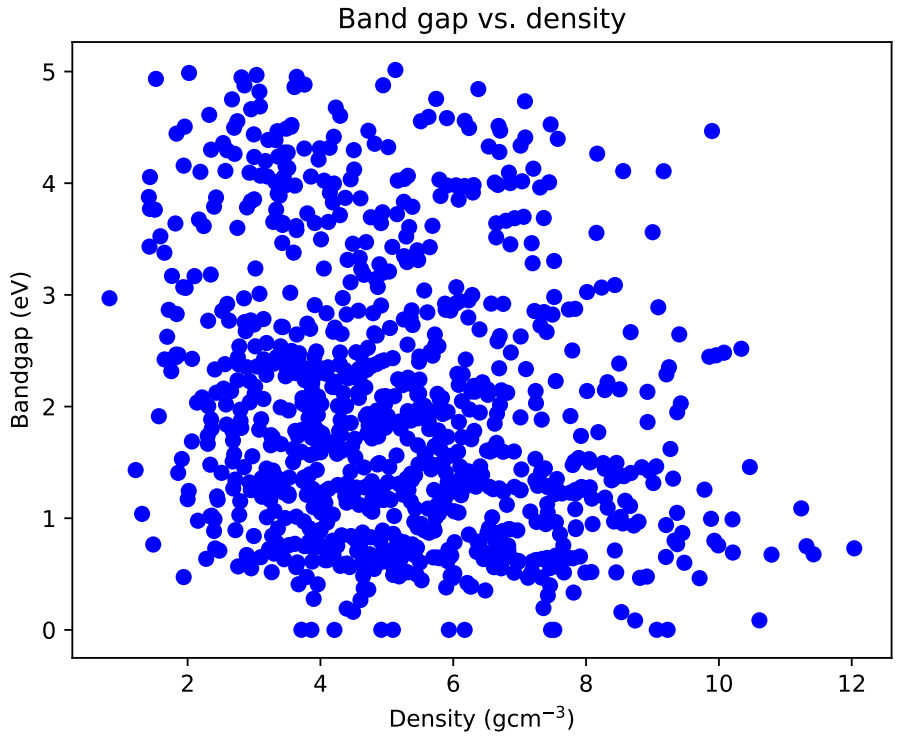

Figure 6. Band gap vs density for the absorption data in Materials Project dataset.

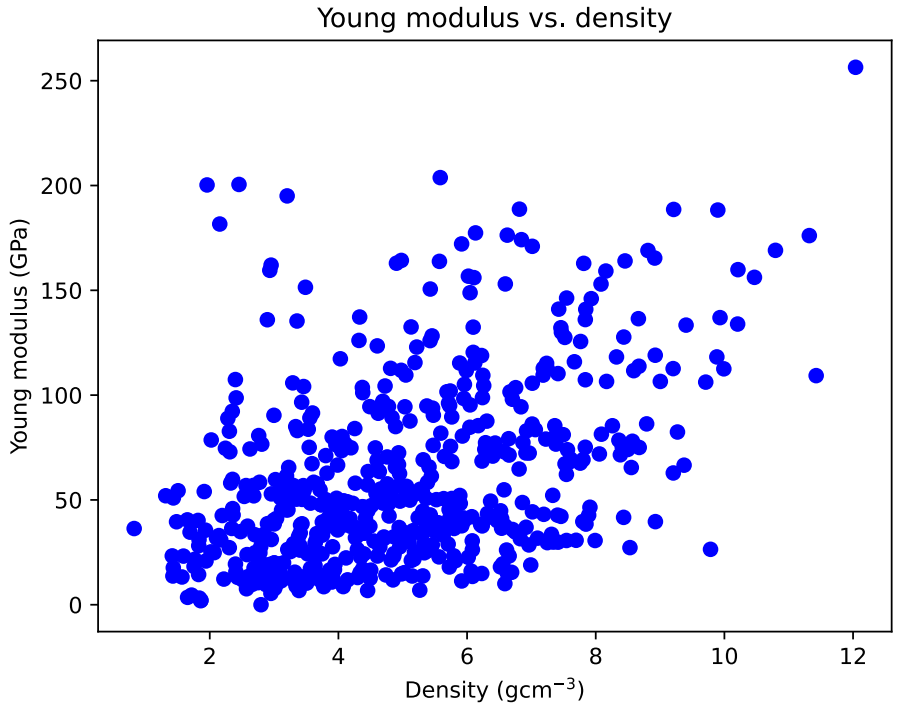

Figure 7. The Young modulus from the elasticity data against density from the absorption data in Materials Project dataset.

Acknowledgements

We acknowledge support by EPSRC grant EP/X032663/1 for PSDI Phase 1b. We are grateful to partners in the University of Southampton and University College London for their feedback on lakehouse deployment in Staging environment.

References

[1] PSDI (Physical Sciences Data Infrastructure). See www.psdi.ac.uk

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

[3] Apache Iceberg. See https://iceberg.apache.org/

[4] Tim Berglund. Apache Iceberg: What It Is and Why Everyone's Talking About It. https://www.youtube.com/watch?v=TsmhRZElPvM

[5] Luis Barroso-Luque, Muhammed Shuaibi, Xiang Fu, Brandon M. Wood, Misko Dzamba, Meng Gao, Ammar Rizvi, C. Lawrence Zitnick, Zachary W. Ulissi. Open Materials 2024 (OMat24) Inorganic Materials Dataset and Models. https://doi.org/10.48550/arXiv.2410.12771

[6] OMat24 in Hugging Face. https://huggingface.co/facebook/OMAT24

[7] Daniel S. Levine, Muhammed Shuaibi, Evan Walter Clark Spotte-Smith, Michael G. Taylor, Muhammad R. Hasyim, Kyle Michel, Ilyes Batatia, Gábor Csányi, Misko Dzamba, Peter Eastman, Nathan C. Frey, Xiang Fu, Vahe Gharakhanyan, Aditi S. Krishnapriyan, Joshua A. Rackers, Sanjeev Raja, Ammar Rizvi, Andrew S. Rosen, Zachary Ulissi, Santiago Vargas, C. Lawrence Zitnick, Samuel M. Blau, Brandon M. Wood. The Open Molecules 2025 (OMol25) Dataset, Evaluations, and Models. https://doi.org/10.48550/arXiv.2505.08762

[8] OMol25 in Hugging Face. https://huggingface.co/facebook/OMol25

[9] Lakehouse briefing in May 2026. http://purl.org/net/epubs/work/66187592

[10] Delta Lake storage framework. https://delta.io/

[11] Apache Hudi data lakehouse platform. https://hudi.apache.org/

[12] Apache Airflow. https://airflow.apache.org/

[13] What is a medallion architecture? https://www.databricks.com/blog/what-is-medallion-architecture

[14] Iceberg Table specification. https://iceberg.apache.org/spec/

[15] Puffin file format. https://iceberg.apache.org/puffin-spec/

[16] Iceberg catalog REST API specification. https://iceberg.apache.org/rest-catalog-spec/

[17] Iceberg View specification. https://iceberg.apache.org/view-spec/

[18] Trino: a distributed SQL query engine. https://trino.io/

[19] Apache Spark. https://spark.apache.org/

[20] Apache Parquet file format. https://parquet.apache.org/

[21] Ceph distributed storage system. https://ceph.io/

[22] Lakekeeper Apache Iceberg REST catalog. https://lakekeeper.io/

[23] Open Policy Agent. https://www.openpolicyagent.org/

[24] Materials Commons. MaterialsCommons4EU

[25] BatCAT project. https://batcat.info

[26] Jupyter project. https://jupyter.org/

[27] Marimo computational notebook. https://marimo.io/

[28] Ibis dataframe library. https://ibis-project.org/

[29] Apache Iceberg v3 key concepts. https://iceberglakehouse.com/iceberg/iceberg-spec-v3/

[30] StarRocks analytical database. https://www.starrocks.io/

[31] PSDI Knowledge Base. https://guidance.psdi.ac.uk/

[32] Substrait: cross-language serialization for relational algebra. https://substrait.io/

[33] Sirius: a GPU-native SQL engine. https://www.sirius-db.com/

[34] Croissant specification. https://docs.mlcommons.org/croissant/docs/croissant-spec.html

[35] PuppyGraph query engine. https://www.puppygraph.com/

[36] Iceberg maintenance. https://iceberg.apache.org/docs/latest/maintenance/

[37] Apache Iceberg with Trino: Performance Optimization Guide. https://lakeops.dev/blog/trino-iceberg-optimization

[38] OLake Extract and Transform framework. https://olake.io/

[39] Apache Amoro: management system for lakehouse. https://amoro.apache.org/