AI Systems Adoption of Unified Research Data Management on Accelerator Computing

A framework for unifying RDM and confidential AI using oneAPI

Peter Darveau
Digital Research Alliance of Canada – Center for Advanced Computing – University of Ottawa
Ottawa, Canada
e-mail: pdarveau@uottawa.ca

Abstract — **Research data is expected to grow exponentially with the adoption of artificial intelligence (AI) and machine learning (ML). Robust data management practices are crucial for ensuring data integrity, provenance tracking, and adherence to ethical and regulatory standards, which is essential for building trustworthy AI systems. This paper explores the adoption of oneAPI, an open standards-based programming model, for streamlining research data management across diverse AI systems. It also explores containerization to ensure consistent execution across heterogeneous Cloud-based environments while providing security over sensitive data-based systems. By leveraging oneAPI's cross-architecture capabilities, including Data Parallel C++ (DPC++) and the other AI toolkits based on oneAPI, researchers can develop secure and performant AI solutions that seamlessly process and analyze sensitive data across heterogeneous computing environments. This unified approach proposes a framework for consistent data handling and reproducibility of research computing results where data confidentiality, security and integrity are concerns notably in the Cloud. Through a case study example, this paper discusses the benefits of adopting oneAPI for AI research data management (RDM), highlighting its potential to accelerate scientific discoveries while maintaining robust security and privacy standards**.

Keywords - AI; data security; research data management; oneAPI, DPC++; accelerator; containers

I. INTRODUCTION

The rapid advancement of artificial intelligence and machine learning technologies has revolutionized various research domains, enabling breakthroughs and discoveries that were once considered unattainable [1][2][3]. However, as these powerful techniques become increasingly ubiquitous in research, the need for secure and reproducible RDM practices has emerged as a critical concern [4][6]. Ensuring the confidentiality, security and integrity of sensitive data while maintaining reproducibility across diverse computing environments is a significant challenge faced by researchers and data scientists [6]. AI and ML systems often rely on large, complex datasets, including personal information, medical records, or proprietary data, which necessitate robust security measures to protect against unauthorized access, data breaches, or unintended leaks. Additionally, the reproducibility and traceability of research data is crucial for scientific integrity, enabling peer review, validation, and knowledge sharing within the research community. Inconsistencies in software environments, dependencies, or hardware configurations can lead to irreproducible results, hindering collaboration and impeding scientific progress. This article's contribution is a framework that sets guidelines beyond descriptive identifiers in data repo metadata and file handling requirements to include computational and confidentiality metadata identifiers to accommodate sensitive and secure data in research. Furthermore, the proposal and findings in this article are found to go beyond research in industry where confidentiality and security of data is a concern. Finance handles personal and proprietary market metrics and sustainability data for environment, social and governance (ESG) portfolio development. In architecture, building Information Management (BIM) systems used for nuclear power plants and water systems are becoming more tightly integrated into modern practices.

This paper begins by discussing existing research data management practices and their limitations in the context of AI and machine learning workflows. It then introduces containerization as a solution for creating secure and reproducible environments, highlighting the benefits of containerization for data security. The work proposes integrating containerization with Data Parallel C++ (DPC++) and the oneAPI unified cross-architecture programming model. This is followed by an exploration of using DPC++ for secure data processing within containers. A case study example is provided to illustrate the approach, along with experimental results demonstrating the effectiveness and promise of the proposed framework. Finally, the paper concludes with a summary of the findings and an outlook on potential future work in this area.

A. Prior work

Traditional approaches to RDM have struggled to keep pace with the rapid evolution of AI and ML technologies,

presenting researchers with a myriad of challenges [7]. These include managing dependencies across diverse hardware and computing architectures, ensuring consistent and reproducible environments, optimizing resource utilization, and maintaining data security throughout the research data lifecycle [8] [9] [10].

B. This work

To address these challenges, this paper proposes the integration of containerization technologies with oneAPI's Data Parallel C++ (DPC++), a powerful language extension for efficient data parallelism and hardware acceleration. Containerization provides isolated and reproducible environments, encapsulating dependencies and configurations, enabling portability across different computing platforms. DPC++, on the other hand, offers a unified programming model that leverages parallel processing capabilities across central processing units (CPUs), graphic processing units (GPUs), and other accelerators, optimizing resource utilization and accelerating computationally intensive tasks. By combining containerization and DPC++, researchers can develop secure and high-performance AI and ML systems that adhere to best practices in RDM. This integration promises to enhance data confidentiality and integrity through robust encapsulation and isolation techniques, while enabling efficient parallel processing and hardware acceleration. Additionally, the reproducibility of research findings is facilitated by consistent and portable runtime environments, fostering collaboration and knowledge sharing within the scientific research community.

II. Existing RDM practices and their limitations in the context of AI and ML.

Existing Research Data Management (RDM) practices and their limitations in the context of AI and Machine Learning (ML) fall into the following areas [7]:

Static Data Management Plans: Traditional RDM often relies on static and rigid data management plans created at the start of a project. These plans may not adapt well to the iterative and experimental nature of AI and ML research. The limitations become evident when project requirements evolve, new data sources emerge, or when the scope of the project changes, rendering the initial data management plan inadequate.

Lack of FAIR Data Principles: The FAIR data principles (Findability, Accessibility, Interoperability, and Reuse) are not always fully embraced in RDM practices. AI and ML algorithms rely on large volumes of high-quality, well-curated data. If data is not findable, accessible, interoperable, and reusable, it can hinder the development and reproducibility of AI/ML models. Inconsistent metadata standards, lack of data documentation, and poor data organization can limit the effectiveness of AI/ML workflows [11]. Specifically, the limitations that are of primary concern in this article are as follows.

1) Insufficient Metadata Discipline:

Metadata, which provides context, description and other ontology [12] for the data, is often underappreciated in RDM, especially in AI and ML. Rich, standardized metadata enables data discovery, understanding biases, and ensuring ethical usage. Inadequate metadata management can lead to data misinterpretation, integration issues, challenges reproducing results, gaps in implementing RDM best practices, and premature data staleness.

2) Lack of Integration with Emerging Data Types:

RDM practices are sometimes slow to adapt to emerging data types, such as real-time data streams, unstructured data, or sensitive data requiring special handling. AI and ML applications often rely on these diverse data sources, and traditional RDM may not provide the necessary tools and guidelines for their effective management, limiting the potential of AI/ML initiatives [1][2]. By integrating a metadata framework inclusive of core computing devices, robust metadata management practices can be seamlessly applied, addressing key RDM challenges of quality, reproducibility, security, and continuous best practice adherence in AI/ML projects.

3) Insufficient Data Ethics and Privacy Considerations:

With the ethical implications of AI and ML under increasing scrutiny, RDM practices need to incorporate robust data privacy, ethical handling, and consent management frameworks. Traditional RDM may not adequately address these concerns, leading to potential legal, ethical, and societal issues when applying AI/ML technologies. The integration of AI tools based on an open library standard such as oneAPI and repo metadata inclusive of data public and private modifiers provides a framework versatile enough to be implemented by developers or code-savvy researchers while removing the technical heavy lifting from the data custodian who typically focuses on data governance.

III. Containerization for Secure and Reproducible Environments

A. Benefits of containerization for system isolation

Containerization offers significant benefits for data security, including robust dependency management and environment consistency, which are crucial in ensuring the confidentiality, integrity, and availability of sensitive data in AI and ML workflows. These workflows often rely on a myriad of software dependencies, such as deep learning frameworks, data preprocessing libraries, and various other tools. Containerization encapsulates all these dependencies within a single, isolated container, ensuring that no external

dependencies or conflicting libraries can inadvertently introduce vulnerabilities or compromise the security of the AI/ML pipeline. By packaging all the required dependencies together, containers eliminate the risk of unintended interactions with other AI/ML systems or compatibility issues that could potentially lead to security breaches or data leaks.

Environment consistency is another key benefit of containerization for data security. AI/ML workflows are often developed and tested in different environments (e.g., local development, staging, production), each with its own unique configuration and system settings. Inconsistencies between these environments can lead to unexpected behavior, security vulnerabilities, or data integrity issues. Containerization solves this problem by ensuring that the same consistent environment is used across all stages of the workflow, from development to production. By encapsulating the entire runtime environment, including system libraries, configuration files, and environment variables, containers guarantee that the AI/ML application will run identically in any environment, reducing the risk of security vulnerabilities introduced by environmental differences.

In the context of AI and ML workflows, data security is of paramount importance, as these applications often deal with sensitive or regulated data, such as personal information, medical records, or financial data. Containerization provides an additional layer of isolation and control, ensuring that sensitive data is processed within a secure and consistent environment, minimizing the risk of unauthorized access, data breaches, or unintended data leaks.

B. *Integration of containerization with DPC++ and oneAPI for security of sensitive data*

Containerization technologies can seamlessly integrate with DPC++ and oneAPI, providing a powerful combination for efficient and secure data processing in AI and ML workflows. DPC++ is a heterogeneous programming model that enables portable, performance-optimized code to be written for various hardware architectures, including CPUs, GPUs, and various accelerators. It is a part of the oneAPI initiative, which aims to provide a unified and open programming model for diverse architectures. DPC++ and oneAPI offer numerous benefits for data-intensive applications, such as AI and ML, by enabling efficient parallelization, optimized memory management, and hardware acceleration. Integrating containerization with DPC++ and oneAPI can provide the following advantages for seamless data processing:

1) *Consistent and Reproducible Environments*:

Containerization ensures that the DPC++ and oneAPI runtime environments, including libraries, dependencies, and configurations, are consistently packaged, and deployed across different systems. This consistency is crucial for reproducibility, as it guarantees that the data processing pipelines will behave identically, regardless of the underlying hardware or software environment.

2) *Portability and Hardware Abstraction*:

DPC++ and oneAPI provide hardware abstraction and portability, allowing code to run efficiently on different architectures, such as CPUs, GPUs, and accelerators. By combining the portability of C++ using DPC++ with containerization, self-contained and portable data processing pipelines can be seamlessly deployed across diverse hardware platforms without modification.

3) *Efficient Resource Utilization*:

Containers are lightweight and can efficiently utilize available hardware resources, including GPUs and accelerators. By integrating DPC++ and oneAPI with containerization, developers can optimize resource utilization by efficiently parallelizing data processing tasks across multiple containers, each using the full potential of the underlying hardware.

4) Simplified Deployment and Scaling:

Containerized DPC++ and oneAPI applications can be easily deployed and scaled across different environments, from local development to cloud-based deployments or high-performance computing (HPC) clusters. This streamlined deployment process facilitates collaboration, enables efficient scaling of data processing pipelines, and accelerates time-to-market for AI and ML solutions.

5) Versioning and Provenance Management:

Containerization tools like Docker and Singularity offer versioning capabilities, allowing developers to track changes to the DPC++ and oneAPI environments, dependencies, and configurations over time. In addition, the program can integrate data provenance tracking mechanisms by using C++ features such as classes, inheritance, and encapsulation.

IV. Data Parallel C++ for Secure Data Processing

The combination of DPC++ (Data Parallel C++) and C++ encapsulation offers significant advantages for secure and efficient data processing in AI and ML systems. These two powerful tools complement each other, enabling developers to build high-performance, scalable, and secure applications while adhering to best practices in software engineering.

DPC++ is a powerful language extension that enables efficient data parallelism and hardware acceleration across various architectures, including CPUs, GPUs, and accelerators. By leveraging DPC++, developers can harness

the full potential of parallel computing, optimizing resource utilization and accelerating computationally intensive tasks such as training deep learning models or processing large datasets. This parallel processing capability is crucial for AI and ML systems, which often require significant computational resources to handle complex algorithms and massive amounts of data.

Complementing DPC++, C++ encapsulation provides a robust approach to organizing and protecting sensitive data and algorithms within AI and ML systems into digital objects [13] [14]. Encapsulation is a fundamental principle of object-oriented programming (OOP) that allows developers to bundle related data and functions into a single unit, called a class. This class acts as a secure container, protecting the internal implementation details from external access or modification, while exposing a well-defined public interface for interacting with the encapsulated functionality. By combining DPC++ and C++ encapsulation, developers can create secure and efficient AI and ML systems that use the power of parallel processing on selected accelerators while adhering to best practices in secure software engineering. DPC++ enables efficient data parallelism and hardware acceleration, optimizing performance and resource utilization, while C++ encapsulation provides a robust framework for organizing and protecting sensitive data and algorithms.

This combination also supports the integration of secure execution environments, such as from a container. By encapsulating critical components within containers, developers can further enhance the security of their AI and ML systems, protecting sensitive data and computations from unauthorized access or modification, even in the presence of privileged software or system administrators.

V. CASE STUDY EXAMPLE

In the field of genomics research, managing and analyzing large datasets of genetic sequences is a computationally intensive task. Researchers often need to process terabytes of data while ensuring the privacy and confidentiality of sensitive patient information [15]. This case study proposes how a combination of C++ encapsulation, oneAPI, and Object Oriented Programming can address these challenges.

1) Data Organization and Encapsulation:

Researchers create a GenomicData class in C++ to encapsulate genomic sequences and associated metadata (e.g., patient information, sample details, consent forms). This class organizes the data into modular units, promoting data provenance tracking and adherence to privacy regulations.

2) Secure Data Processing within Secure Containers:

To ensure the confidentiality of sensitive patient data, the GenomicData class and its methods executes within a container such as Docker or Apptainer. Although containers can run in hardware-based and attested memory, they provide a trusted execution environment protecting the data and computations from unauthorized access or modification, even from privileged software or system administrators.

3) Parallel Sequence Alignment with Data Parallel C++:

One of the core operations in genomic analysis is sequence alignment, which involves comparing genetic sequences against reference databases [15]. The researchers implemented a SequenceAligner class that leverages Data Parallel C++, oneAPI, and hardware acceleration (CPUs, GPUs) to perform parallel sequence alignment operations as per a sample shown in Figure 1.

```
class SequenceAligner {
  public: void alignSequences(GenomicData & data) {
    // Load data into USM (Unified Shared Memory)
    auto sequences = data.getSequences();
    auto referenceDB = loadReferenceDatabase();

    // Parallel alignment using Data Parallel C++
    std::parallel_for(
      std::par_unseq,
      sequences.begin(), sequences.end(),
      [ & ](auto & sequence) {
        alignSequence(sequence, referenceDB);
      }
    );

    // Store aligned sequences back in GenomicData
    data.setAlignedSequences(sequences);
  }

  private: void alignSequence(Sequence & seq,
    const ReferenceDB & db) {
    // Sequence alignment algorithm
    // ...
  }
};
```

Figure 1. A Class implementation of the case study

The alignSequences method utilizes Data Parallel C++ constructs and oneAPI's Unified Shared Memory (USM) to efficiently distribute the alignment tasks across available hardware accelerators, significantly improving performance.

4) Secure Data Storage and Backup:

To maintain data integrity and availability, the researchers implement a DataStorageManager class that handles secure storage and backup of genomic data within the secure container. This class encapsulates methods for encrypting data, performing incremental backups, and securely transferring backups to off-site storage locations.

5) Auditing and Data Access Control:

Adhering to data governance policies, the GenomicData class includes methods for auditing data access and operations. Access control mechanisms ensures that only

authorized researchers could interact with the sensitive genomic data. The GenomicData class encapsulates both the genomic data itself (stored as a vector of strings) and its provenance metadata (stored as an unordered map of key-value pairs).

The class constructor initializes the data and sets the initial provenance metadata with the source of the data as follows. A preprocess method performs any necessary preprocessing steps on the data and updates the provenance metadata with information about the preprocessing method and its parameters. A sample of such and accompanying metadata are shown in Figures 2 and 3.

```
public:
    void preprocessing(const std::vector<std::string>& rawData) {
        // Perform data preprocessing steps
        std::vector<std::string> preprocessedData = removeOutliers(rawData);
        preprocessedData = normalizeData(preprocessedData);
        preprocessedData = encodeData(preprocessedData);

        // Update provenance metadata
        provenance_["preprocessing"].emplace("remove_outliers", true);
        provenance_["preprocessing"].emplace("normalization_method", "min_max")
        provenance_["preprocessing"].emplace("encoding_method", "one_hot");

        // Store the preprocessed data
        processedData_ = preprocessedData;
    }
```

Figure 2. A preprocessing method

```
models:
  - name:  image_model
    launchers:
      - framework: dlsdk
        tags:
          - FP32
        model: ./data/public/model/FP32/image_model.xml
        weights: ./data/public/model/FP32/image_model.bin
        adapter: ssd
        device: CPU # or GPU, GPU.1, GPU.2

    datasets:
      - name: image_model_detection_91_classes
        data_source: ./data/datasets/model_val_data/val2024
        annotation_conversion:
            annotation_file: ./data/datasets/model/annotations/instances_val2024.json
            access_mod: public # or private
            has_background: True
            use_full_label_map: True
            converter: img_detection
        preprocessing:
          - type: resize
            size: 300
        postprocessing:
          - type: resize_prediction_boxes
        metrics:
          - type:precision # or accuracy, top_k, F1_score, etc.
```

Figure 3. Example metadata

By encapsulating both the data and its provenance metadata within the same class, provenance information is propagated throughout the AI/ML workflow [16], enabling comprehensive tracking and documentation of the data's lineage.

A. Explanation

By leveraging C++ encapsulation, oneAPI, and OOP principles, the researchers are able to develop a secure and efficient solution for managing and analyzing genomic data:

- Sensitive patient data is further protected through hardware-based trusted containers, ensuring confidentiality and privacy.
- Computationally intensive sequence alignment operations is accelerated using Data Parallel C++ and oneAPI, enabling efficient analysis of large genomic datasets.
- Modular and encapsulated design promotes code reusability, maintainability, and adherence to RDM best practices.
- Secure data storage, backup, auditing, and access control mechanisms ensures data integrity, availability, and compliance with regulations.

This case study demonstrates how the combination of C++ encapsulation, oneAPI, and OOP can enable secure, efficient, and compliant Research Data Management in the field of genomics and can serve as a template for applications in other data-intensive domains such as BIM and finance. Following is a description of what the public and private methods do in the context of the SequenceAligner class from the genomic data case study:

1) *Public Method:*
    a. *void alignSequences(GenomicData& data):*

This public method is responsible for performing sequence alignment operations on the genomic data encapsulated within the GenomicData class. It takes a reference to a GenomicData object as input. The method loads the genomic sequences and a reference database into Unified Shared Memory (USM) for efficient data sharing among accelerators. It then leverages Data Parallel C++ constructs (std::parallel_for) and oneAPI to parallelize the sequence alignment tasks across available hardware accelerators (CPUs, GPUs). After the parallel alignment is completed, the method stores the aligned sequences back into the GenomicData object.

2) *Private Method:*
    a. *void alignSequence(Sequence& seq, const ReferenceDB& db):*

This private method is a helper function that performs the actual sequence alignment operation for a single genomic sequence (seq) against the reference database (db). It is called in parallel by the std::parallel_for construct within the alignSequences method. The implementation details of the sequence alignment algorithm are encapsulated within this private method. By keeping this method private, the class adheres to the encapsulation principle, hiding the implementation details from external code and providing a

well-defined public interface (alignSequences) for sequence alignment operations.

The separation of public and private methods in the SequenceAligner class follows the principles of encapsulation and information hiding from Object-Oriented Programming: The public alignSequences() method provides a high-level interface for initiating sequence alignment operations on genomic data, abstracting away the underlying implementation details. The private alignSequence() method encapsulates the low-level details of the sequence alignment algorithm, allowing for potential changes or optimizations without affecting the public interface. This design promotes code modularity, maintainability, and extensibility, as the implementation details of the sequence alignment algorithm can be modified or improved without impacting the public interface used by other parts of the application. Additionally, by leveraging Data Parallel C++ and oneAPI within the public alignSequences method, the class can efficiently utilize hardware acceleration and parallel processing capabilities, improving the performance of computationally intensive sequence alignment operations on large genomic datasets. The SequenceAligner class is designed to encapsulate and protect the sensitive genomic data processing logic within the private alignSequence method. It should be note that there are still potential risks if external code can directly access or modify this private method, which could compromise the integrity and confidentiality of the genomic data processing pipeline.

VI. Experimental results

We used oneAPI to handle data loading, preprocessing, and post-processing steps, ensuring consistent and reproducible experiments using openVINO AI inferencing engine for all combinations. A validation dataset consisting of 5000 images was used. Researchers can configure the environment through a metadata configuration file, specifying dataset paths, model paths, access modifiers and evaluation parameters. The experiment generates a detailed accuracy report, easing model and environment comparisons. Additionally, it runs in an Apptainer container, enabling portable and reproducible experiment environments across different computing platforms. This methodology allowed us to implement best practices in research data management and reproducible AI/ML workflows using the framework proposed in this paper. This method goes further suggesting metadata of a dataset include data access modifiers to selectively secure data and define the computing device in the oneAPI software library thereby providing a unified computing device framework.

TABLE I. Experimental results

Results (XPUs)	**Metrics**			
	Reproducibility of computing result	***Transparency***	***Access Modifiers***	***Cloud Environment***
CPU only (Intel)	Same results when executed on same validation data	Metadata-json config file, ONNX, oneAPI	All public	Container, no enclave
GPU (AMD)	Same results when executed on same validation data	Metadata-json config file, ONNX, oneAPI	All public	Container, no enclave
GPU Nvidia	Not fully tested	Metadata-json config file, ONNX, oneAPI	All public	Container, no enclave

Table 1 details the metrics thought most relevant to evaluate the framework on an AI model. The results from this experiment enable a holistic evaluation of the framework and the effort involved in setting it up for a research project. It highlights the AI model evaluation process using oneAPI and DPC++ framework for data repo metadata in a containerized environment, providing consistent, reproducible, and well-documented results aligned with research data management best practices. However, it should be noted that although the tests did not cover a combination of public and private data access modifiers, the results can still be holistically interpreted as successful and promising.

VII. Conclusion and future work

While existing RDM practices provide a foundation for data management, they often fall short in several key areas when applied to AI and ML contexts. Adapting RDM to address these limitations is essential for unlocking the full potential of AI and ML applications and ensuring responsible and effective data-driven innovation. Containerization technologies like Docker and Apptainer offer robust dependency management, environment consistency, and enhanced control over the software supply chain, making them valuable tools for maintaining data security in AI and ML workflows. By encapsulating dependencies, isolating environments, and enabling secure software development practices, containerization contributes to the confidentiality, integrity, and availability of sensitive data throughout the AI/ML lifecycle. The constructive interaction between DPC++ and C++ encapsulation empowers developers to build high-performance, scalable, and secure AI and ML systems that can efficiently process large datasets while maintaining data confidentiality and integrity. This approach promotes code maintainability, extensibility, and collaboration, enabling the development of robust and reliable AI and ML solutions that can be trusted in critical applications. There are, however, some scenarios like the SequenceAligner class could be at risk from external code

requiring access to the private alignSequence method where external code is able to manipulate or inject offending data into the alignSequence method, and potentially compromise the integrity of the sequence alignment algorithm. Future work to mitigate these risks as identified in section V. should include the possibility of splitting the validation dataset into public and private access modifiers then validate accuracy under that scenario and the implementation of enclaves which involves encryption. Encrypted computing [17] is another emerging paradigm that could further address RDM challenges in decentralized systems by encrypting and decrypting at the edge device. This involves the custodian's proprietary data or algorithms encrypted throughout the end-to-end computation process, reducing the risk of unauthorized access or theft.

ACKNOWLEDGMENT

The author thanks the members of the Research IT and Library of the of the University of Ottawa for their continued participation and for helpful discussion on the topics of data-representation and research data management. The work described here was supported by Digital Research Alliance of Canada (Alliance) and the University of Ottawa (uOttawa). The content is solely the responsibility of the author and does not necessarily reflect the official views of the Alliance, uOttawa, or Canadian Government.

CONTRIBUTORSHIP

Simone Darveau (University of Waterloo – Waterloo Canada) contributed to the applicability of the model framework in an international architectural firm's BIM system. Vivianne Darveau (Columbia University, NY USA) contributed to the applicability of the model framework in an investment banking context with an international bank. All authors and contributors discussed the results, the application and contributed to the final manuscript.

DECLARATION OF INTERESTS

The author declares no competing interests.

REFERENCES

[1] Peter Darveau "Decision Trees: Modeling with fast intuition and slow, deliberate analysis." 2023

[2] Peter Darveau "Support Vector Machines: Modeling The Dual Cognitive Processes of an SVM." 2023.

[3] Peter Darveau "Prognostics and Availability for Industrial Equipment Using High Performance Computing (HPC) and AI Technology." 2021

[4] L. Wilson *"RDM Network of Experts"* [Presentation]. DRI Cnnnect National Conference, Halifax, NS, Canada 2024, May 24—25.

[5] F. Pérez-Jvostov *"Overview of the World Data System"* [Presentation]. DRI Cnnnect National Conference, Halifax, NS, Canada 2024, May 24—25.

[6] V. Smith.*"Approaches to sensitive data across the DRI landscape"* [Presentation]. DRI Cnnnect National Conference, Halifax, NS, Canada 2024, May 24—25.

[7] Gail Birkbeck, Tadhg Nagle and David Sammon (2022) Challenges in research data management practices: a literature analysis, Journal of Decision Systems, 31:sup1, pp. 153-167

[8] J. F. Pimentel, L. Murta, V. Braganholo and J. Freire, "A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks," 2019 IEEE/ACM 16th International Conference on Mining Software, Montreal, Canada, 2019.

[9] Gundersen, O. E., and Kjensmo, S. State of the art: Reproducibility in artificial intelligence. In Proceedings of the AAAI conference on artificial intelligence (Vol. 32, No. 1) 2018

[10] Haibe-Kains, Benjamin, et al. Transparency and reproducibility in artificial intelligence. Nature 586.7829: E14-E16 2020.

[11] R. Jenkyns *"Overview of the World Data System"* [Presentation]. DRI Cnnnect National Conference, Halifax, NS, Canada 2024, May 24—25.

[12] Brahaj, Armand, et al. "Ontological Formalization of Scientific Experiments Based on Core Scientific Metadata Model." *Theory and Practice of Digital Libraries*, Springer Berlin Heidelberg, pp. 273–79

[13] Johanne Medina, et al. "*Accelerating the adoption of research data management strategies*", Volume 5, Issue 11, pp. 3614-3642

[14] Petr Ježek and Roman Mouček. "Semantic Framework for Mapping Object-Oriented Model to Semantic Web Languages." *Frontiers in Neuroinformatics*, vol. 9, Feb. 2015, p., doi:10.3389/fninf.2015.00003

[15] S. J. Mack, J. Sauter, J. Robinson et al. "*The genotype list string code syntax for exchanging nomenclature-level genotyping results in clinical and research data management and analysis systems"*. HLA. 2023; 102(4): pp. 501-507.

[16] Lixin Han et al. "AASA: A Method of Automatically Acquiring Semantic Annotations." *Journal of Information Science*, Aug. 2007, pp. 435–450.

[17] Intel Corporation "*Accelerated AI Inference with Confidential Computing"* [White Paper]. Link 2023