



MDMC

Master in Data Management
and Curation

MASTER IN DATA MANAGEMENT AND
CURATION

FAIR data management of
quantum-mechanical calculations
for the spin states dynamics in
SiC materials

Supervisor(s):

Antonino LA MAGNA,
Matteo BIAGETTI

Candidate:

Giovanni CASTORINA

2024–2025





Finanziato
dall'Unione europea
NextGenerationEU



Ministero
dell'Università
e della Ricerca



Italiadomani
PIANO NAZIONALE
DI RIPRESA E RESILIENZA

This Pilot training activity has been funded by the European Union – NextGenerationEU within the project PNRR "PRP@CERIC" IR00000028 and «NFFA-DI" IR00000015 - Missione 4, "Istruzione e Ricerca" – Componente 2, "Dalla ricerca all'impresa" – Linea di investimento 3.1, "Fondo per la realizzazione di un sistema integrato di infrastrutture di ricerca e innovazione" – Azione 3.1.1, "Creazione di nuove IR o potenziamento di quelle esistenti che concorrono agli obiettivi di Eccellenza Scientifica di Horizon Europe e costituzione di reti".

The supporting projects:

- [NFFA-DI](#) : Nano Foundries Fine Analysis - Digital Infrastructure
- [PRP@CERIC](#) : Pathogen Readiness Platform for Ceric - Eric Upgrade



nffa-di



prp

Author's Declaration

I, Giovanni Castorina, declare that this thesis entitled, 'FAIR data management of quantum-mechanical calculations for the spin states dynamics in SiC materials' and the work presented therein are my own.

I certify that:

- This work was performed wholly or principally during MDMC internship in CNR-IMM, Catania Section.
- If any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have relied on the published work of others, this is always clearly attributed.
- Where the thesis is based on work I have done in collaboration with others, I have made clear exactly what was done by others and what I contributed.
- With respect to AI technologies:
 - I acknowledge the use of [gemini](#) to identify improvements in the writing style.
 - I acknowledge the use of [gemini](#) to generate or correct code for the NOMAD plugin.
- Where I have quoted from the work of others, the source is always cited. Except for such quotations, this thesis is entirely my own work.
- I have acknowledged all major sources of help.

Signed: Giovanni Castorina

Date: 30/04/2025

*“There’s plenty of ~~Room~~ **Data** at the Bottom.”*

Richard P. Feynman (With a small modification)

Acknowledgments

First and foremost, I am deeply grateful to my supervisors: Dr. Matteo Biagetti, Dr. Antonino La Magna, and Dr. Tommaso Rodani. Their help was fundamental in navigating me (and this thesis) through all phases of the research and data management. I thank Antonino for guiding my learning of new concepts and their application in my work as a theoretical physicist. Matteo and Tommaso were instrumental in making the data management aspects possible, offering crucial support when I was lost through the complexities of code, schemas, and data.

Secondly, my sincere thanks to the entire theoretical research team at CNR-IMM Catania. Their support and lunch table chats were a small, but fundamental part day after day. In particular, I wish to thank Dr. Gaetano Calogero and Dr. Rosario Gaetano Viglione for generously sharing their insights into the inner workings of Silicon Carbide.

Special thanks go to the FAIRmat community. Without their assistance, this project would have faced considerably greater challenges. I am particularly thankful to Dr. Esma Birsan Boydas and Dr. Andrea Albino for their dedicated support and close collaboration during the development of the plugin.

I would like to thank my colleagues in the master's program for their wonderful support and friendship every day. I offer special thanks to Lucia Li Donni for her continuous and precious advices throughout the entire duration of the master's. My thanks also go to Filippo Ruberto and Matteo F. Bontorno for their friendship, which made every day in Trieste funnier than the one before.

Lastly, I wish to express my deep gratitude to Prof. Giuseppe G.N. Angilella. His support was crucial in helping me push through difficult periods, when the workload seemed overwhelming.

Abstract

This thesis addresses the application of FAIR data management principles to quantum-mechanical simulations, using spin state dynamics in Silicon Carbide (SiC) materials for quantum sensing as a case study.

It begins by explaining the importance of FAIR data management, emphasizing significance and benefits of adopting such practices.

Subsequently, a practical research case concerning a quantum enhanced magnetometer in SiC is presented. This case served as a basis for identifying the specific types of data and metadata involved in such simulations.

The core of the thesis describes the practical implementation methodology, detailing how FAIR compliance of simulation's data was achieved using the NOMAD platform through the development of a specialized plugin for the QuTiP based calculations.

Finally, the thesis demonstrates a FAIR-by-design workflow through an example incorporating all types of data and metadata structured by the plugin's schemas.

Contents

1	Introduction	8
1.1	FAIR Data and Open Science in the scientific research process	8
1.2	Quantum Science & Technologies	13
1.3	Thesis Goals	14
2	Data Management Background	15
2.1	Data Management	15
2.1.1	Research data lifecycle	16
2.2	FAIR principles	19
2.2.1	FAIR for data	20
2.2.2	The value of FAIR	23
2.2.3	FAIR for code	24
2.2.4	Interplay between FAIR and Data Management	26
2.3	FAIRNESS in Material Science: NOMAD and FAIRmat	27
2.3.1	NOMAD schemas	28
2.4	Summary	32
3	Scientific Background	33
3.1	Theoretical Minimum	33
3.1.1	Quantum mechanics & Information	34
3.1.2	Ab initio Methods & DFT	37
3.2	Previous literature	40
3.2.1	Diamond Nitrogen Vacancy	40
3.2.2	Silicon Carbide	43
4	Research Work	46
4.1	Python and QuTiP	46
4.2	System Description	51
4.2.1	Description of Hamiltonians	52

4.2.2	Previous ab-initio calculations	53
4.3	Simulations	53
4.3.1	Results	55
4.4	Future research directions	57
5	Data Management Process	67
5.1	Data Management Plan	67
5.2	Developing of a NOMAD Plug-in	68
5.2.1	Metadata and Schemas	68
5.3	Data management pipeline	70
5.4	Summary: challenges, solutions and future improvements . . .	70
6	Conclusions	72
A	Code	74
A.1	Eigenvalues finding	74
A.2	Plugin Schemas	76
A.3	Parsers	88

Chapter 1

Introduction

1.1 FAIR Data and Open Science in the scientific research process

The principles of Open Science are increasingly recognized as vital across all scientific disciplines. The concept extends beyond merely providing open access to published knowledge; it advocates for transparency throughout the entire *process of knowledge creation*, a necessity more critical now than ever.

Applying open licenses to hardware, data (where FAIR principles are integral), and software (guided by FAIR4RS principles) is fundamentally important. This approach not only enhances the quality and quantity of research, benefiting the scientific community, but also grants society more inclusive and transparent access to the scientific process. This aligns with and helps fulfil Article 27.1 of the Universal Declaration of Human Rights [1].

The detrimental effects of insufficient openness in science are evident. Examples include slowed progress in life-saving research during the COVID-19 pandemic, disparities in medical knowledge due to historical exclusion in research (such as differing knowledge between male and female bodies), and a significant number of paper retractions or corrections stemming from errors, misuse of generative AI, or other issues. As succinctly put, "*The opposite of open science is not closed science but **bad science***" [2].

Open Science operates on several core tenets, underpinned by what might be considered a **zeroth** principle: **As open as possible, as closed as necessary**. This guiding statement highlights the imperative for openness to address global challenges, such as the climate crisis, while respecting constraints related to intellectual property, privacy, national security, industrial confidentiality, and other potential restrictions inherent in the research pro-

cess.

Further elaborating on its foundation, Open Science is defined by four key values and six guiding principles [1]. The four values are:

- **QUALITY AND INTEGRITY:** Ensuring high-quality, rigorously scrutinized science by integrating diverse knowledge sources and enhancing the transparency and accuracy of evaluating scientific methods and outputs.
- **COLLECTIVE BENEFIT:** Recognizing science as a global public good that should benefit all of humanity.
- **EQUITY AND FAIRNESS:** Ensuring equitable and fair access to scientific knowledge and participation in its creation for all, irrespective of location, nationality, race, age, gender, income, socio-economic status, career stage, discipline, language, religion, disability, ethnicity, or migratory status.
- **DIVERSITY AND INCLUSIVENESS:** Embracing a wide range of knowledge systems, practices, workflows, languages, research topics, and outputs.

The six guiding principles are:

- **Transparency, scrutiny, critique and reproducibility:** Reinforcing the rigour of scientific findings, enhancing science's societal impact, and improving our collective ability to solve complex problems.
- **Equality of opportunities:** Ensuring all scientists and interested parties have an equal chance to access, contribute to, and benefit from science.
- **Responsibility, respect and accountability:** Upholding public accountability, managing conflicts of interest, maintaining intellectual integrity, and considering the potential social and ecological impacts of research.
- **Collaboration, participation and inclusion:** Fostering collaborations that transcend geographical, linguistic, and resource boundaries, and incorporating knowledge from marginalized communities to address significant societal challenges.
- **Flexibility:** Acknowledging diverse pathways to practising Open Science while adhering to its core values, recognizing that no single approach fits all situations.

- **Sustainability:** Building upon long-term practices, services, infrastructures, and funding models to ensure efficiency, impact, and the participation of scientists from less-privileged settings.

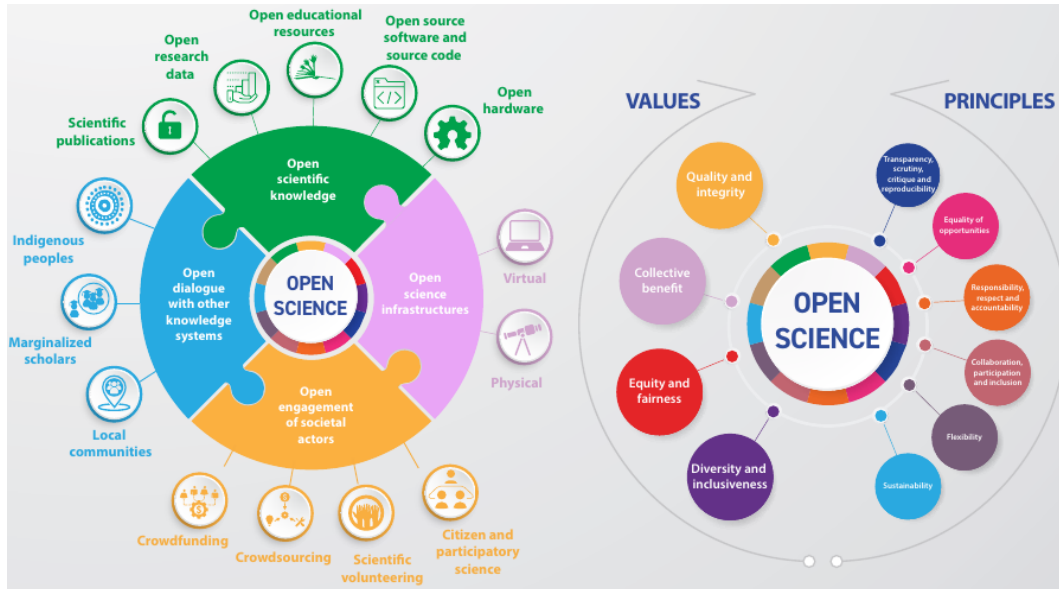


Figure 1.1: Open science goals[1]

Open Science involves several key practices aimed at making the entire research process and its outputs more transparent, accessible, and reusable. Some common ways it is practised include[1]:

- **Early and Open Sharing:**
 - **Preregistration:** Researchers publicly register their research plans (hypotheses, study design, analysis plan) in a repository *before* collecting data. This increases transparency and credibility. Examples of platforms include OSF and As Predicted.
 - **Preprints:** Scientific manuscripts are shared publicly on preprint servers (like arXiv, bioRxiv, medRxiv, SocArXiv) *before* formal peer review and journal publication, allowing for faster dissemination and feedback.
- **Managing and Sharing Research Outputs:**
 - **Data Management Plans (DMPs):** Implementing DMPs to properly manage research data throughout its lifecycle .

- **Open Data:** Making research data Findable, Accessible, Interoperable, and Reusable (FAIR) and depositing it in trusted repositories with clear access policies and licenses .
- **Open Software/Code:** Sharing software and code used for analysis, often using platforms like GitHub, SourceForge, or others. FAIR principles are also being adapted for software (FAIR4RS) .
- **Open Workflows and Protocols:** Sharing detailed experimental workflows and protocols, potentially using platforms like Protocol Exchange or protocols.io .
- **Open Access Publishing:** Making final research publications freely available to read, often through specific journals or repositories [2].
- **Open Peer Review:** Participating in peer review processes where reviewer identities and/or review reports are made public.
- **Ensuring Reproducibility:** Implementing measures specifically designed to ensure research outputs can be reproduced by others.
- **Involving Society:** Engaging citizens, civil society, and end-users in the research process (e.g., through citizen science).

These practices aim to make various components of the research lifecycle, including experimental setup, data, analysis strategies, code, and publications available and reusable [2].

As previously noted, the FAIR data principles are a cornerstone of the Open Science process. Although they will be explored in detail later, this section provides a brief overview[3]. FAIR principles are needed in order to support the reuse of scholarly data. They have been designed by a joint effort of academia, industry, publishers and other stakeholders and in this way they act as guidelines for the reusability of data holdings. A key aspect of the FAIR principles, particularly relevant to current applications, is their emphasis on enabling machines to automatically find and utilize data, complementing reuse by human researchers. This concept is sometimes encapsulated as making data *FAIR as Fully AI Ready*.

As illustrated in Figure 1.2, the FAIR principles are:

- **Findability:** Ensuring digital resources can be easily discovered by both humans and computers. This relies heavily on rich metadata descriptions and the assignment of globally unique and Persistent Identifiers (PIDs) within suitable repositories.

Box 2 | The FAIR Guiding Principles**To be Findable:**

- F1. (meta)data are assigned a globally unique and persistent identifier
- F2. data are described with rich metadata (defined by R1 below)
- F3. metadata clearly and explicitly include the identifier of the data it describes
- F4. (meta)data are registered or indexed in a searchable resource

To be Accessible:

- A1. (meta)data are retrievable by their identifier using a standardized communications protocol
 - A1.1 the protocol is open, free, and universally implementable
 - A1.2 the protocol allows for an authentication and authorization procedure, where necessary
- A2. metadata are accessible, even when the data are no longer available

To be Interoperable:

- I1. (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.
- I2. (meta)data use vocabularies that follow FAIR principles
- I3. (meta)data include qualified references to other (meta)data

To be Reusable:

- R1. meta(data) are richly described with a plurality of accurate and relevant attributes
 - R1.1. (meta)data are released with a clear and accessible data usage license
 - R1.2. (meta)data are associated with detailed provenance
 - R1.3. (meta)data meet domain-relevant community standards

Figure 1.2: FAIR guiding principles [4]

- **Accessibility:** Defining the conditions and protocols required to access the data. This includes specifying authorization procedures, acknowledging that access to certain data (e.g., sensitive medical data) may be restricted.
- **Interoperability:** interoperability defines the need for the data and metadata to use an accessible and broadly applicable language for data representation, with references connecting meta(data) explicitly states. In this point is fundamental to have machine readable and actionable ready data.
- **Reusability:** Specifying the conditions under which data can be reused, including clear and accessible data usage licenses (e.g., Creative Commons). It also stresses the importance of detailed provenance of data (how the data was generated).

Adhering to these principles facilitates knowledge discovery, reduces redundant research efforts, and enhances the scientific infrastructure by promoting data integration and reuse.

The FAIR principles are not limited to data; their application to research software (FAIR4RS) is also gaining traction and will be discussed later in this thesis.

1.2 Quantum Science & Technologies

This thesis investigates the crucial contribution of Open Science principles to advancing the complex field of Quantum Technologies. Key practices like Open Access, Open Data, and Open Methodology directly enhance the transparency, reproducibility, and trustworthiness of quantum research. Although quantum hardware production remains largely concentrated in the private sector, open hardware initiatives are under active development [5]. Emphasizing open-source approaches for both software and hardware design is expected to democratize access, stimulate collaboration across disciplines, minimize duplicated work, and significantly accelerate quantum innovation for Europe and globally.

Quantum technologies represent a rapidly advancing field at the intersection of physics, chemistry, and engineering, attracting significant interest from the global scientific community. The European Union, among others, is [actively supporting](#) research in this domain .

The core principle driving this field is harnessing unique phenomena from quantum science, such as superposition, uncertainty, tunnelling and entanglement to develop technologies capable of achieving performance unattainable with conventional, classical approaches. The term quantum technologies was first introduced in 1997 Milburn's book [6] and 2003 paper [7]. Milburn's thesis was that a second quantum revolution is starting in this millennia. The *First quantum revolution*, that was attained at the start of the last century mainly thanks to the blackbody radiation problem, brought knowledge the quantum nature of matter. A *second quantum revolution* involves the use of the aforementioned quantum phenomena in order to "actively employing quantum mechanics to alter the quantum face of our physical world"[7]. Perhaps the most widely recognized example is the quantum computer: These machines leverage quantum effects like entanglement and superposition to perform computations and simulations of complex systems that are intractable for even the most powerful classical computers due to limitations in time, memory, or precision. In fact [one of the first](#) quantum technologies was a 2-qubit nuclear magnetic resonance quantum computer, invented in 1988. However, quantum technologies extend beyond computing. Quantum effects enable the development of various other innovative devices. Another prominent example are sensors that employ quantum effects in order to improve their performances, which have potential applications in diverse areas including medical diagnostics, environmental monitoring, and fundamental science.

This thesis focuses on the study and characterization of a specific type of

quantum sensor through quantum mechanical computational calculations of the spin state dynamics. Specifically, a Silicon Carbide (SiC) magnetic field sensor based on quantum effects, a material of considerable current interest ([8] [9]).

1.3 Thesis Goals

Established community-wide efforts promoting Open Science and FAIR principles within the quantum technology and information fields are currently limited, with the notable exception of a recent position paper addressing this problem [10].

Therefore, the primary goal of this thesis is to implement an initial FAIR-by-design data management framework for a specific project, done at CNR-IMM Catania, involving quantum-mechanical calculations of spin state dynamics in a potential Silicon Carbide (SiC) based magnetic field sensor. Achieving this objective will involve collaboration and communication with the FAIRmat community, an established community initially focused on ab initio calculations in materials science. FAIRmat is now expanding its scope to encompass diverse data from simulations and experiments stemming from materials science and related disciplines in physics and chemistry.

Chapter 2

Data Management Background

This chapter explores the fundamentals of data management and the increasingly important FAIR (Findable, Accessible, Interoperable, Reusable) data principles. Initially, it discusses general data management strategies before delving into the specifics of the FAIR approach. To illustrate these concepts, the chapter examines the FAIRmat community and the NOMAD platform as an example, showcasing their successful application in the field of material science. Introducing robust data management based on FAIR principles is foundational to embedding open science practices within this quantum information project. Therefore, we investigate the approaches and tools pioneered by FAIRmat and NOMAD. Despite their material science origins they offer a valuable framework for understanding, and then achieving, FAIR data treatment in this project.

2.1 Data Management

Legally speaking, research data are defined in Italian legislation¹ as:

«electronic documents, other than scientific publications, collected or produced in the course of scientific research and used as evidence in the research process, or commonly accepted in the research community as necessary to validate the conclusions and the results of the research»

In general, research data represent the union of raw, processed and analysed data collected or generated during the research process. It also encompasses

¹Guidelines AGID (Agency for Digital Italy) (agid.gov.it)

all information necessary to reproduce the data generation, replicate results, and interpret the data contextually. Examples include computer code and instruments, for means necessary to generate data, and specifications of variables as an example of contextual information [11]. Research data can exist in both digital and non-digital forms.

Specifically:

- **Raw Data:** The original data collected but not yet subjected to analysis, such as data directly recorded from experiments.
- **Processed Data:** Data that have undergone manipulation, including cleaning, translation, anonymization, or other transformations.
- **Analysed Data:** Data derived from raw and processed data, intended to present research conclusions, often visualized as graphs and tables.

Research Data Management (RDM) is an essential element in any project that generates or re-uses data. It can be defined as [11]:

The process of making informed decisions about the handling of research data and implementing those decisions throughout the data's lifecycle.

The core idea is to manage data effectively for organization, retrieval, clarity, transparency, and, critically, preservation [12].

2.1.1 Research data lifecycle

It is useful to consider the **research data lifecycle**, which represents the stages of research concerning data and the associated [tools](#). This process is briefly illustrated in Figure 2.1.

The research data lifecycle can be broadly divided into the following key stages (see also Figure 2.2), which will be discussed subsequently.

Design and Planning

The initial step involves addressing fundamental questions about the research conduct (e.g., "What is the research question?") and considering aspects like reproducibility, integrity, and FAIRness (e.g., "How can the research process be prepared for reproducibility and reuse?").

A primary tool in this stage is the creation of a **Data Management Plan (DMP)**. The DMP is a formal document outlining all aspects of the research data lifecycle from the outset and should be updated throughout the research process[11]. Registering the DMP as an open-access public deliverable is often beneficial.

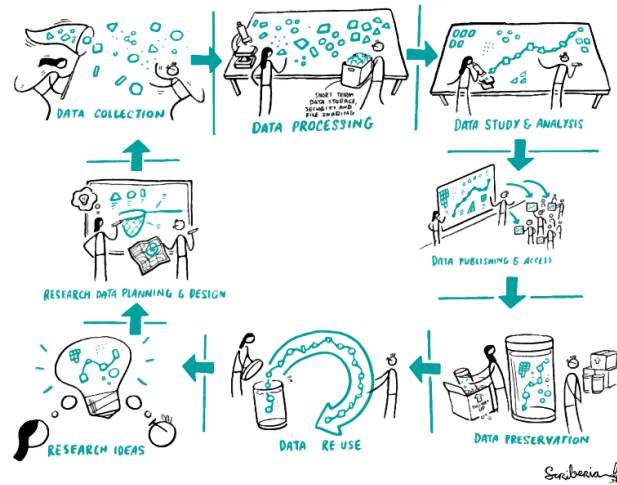


Figure 2.1: Research Data Lifecycle [13]

Collect and Create

This second stage primarily involves using existing research data or collecting new data through various means. Research data is commonly categorized as:

- **Observational, Experimental, or Simulation** data (considered "primary" - generated during the current research).
- **Derived, Compiled or Reference** data (from existing databases, literature, or other sources).

This thesis will focus mainly on simulation data, referencing data from prior literature and internal group research where appropriate, with proper citation.

Particular attention during this step must be paid to **data types** and **file organization**. Data types should ideally be open standards widely used within the relevant community (though constraints, such as proprietary instrument formats, may sometimes prevent this). File organization should be structured, consistent, well-documented (including metadata), and follow a naming convention that clearly describes file content and relationships.

As mentioned, **metadata** (i.e., data about data) are fundamental for describing how data were created/collected, analysed, by whom, using what methodology, etc. Metadata constitute any description enabling findability, reusability, interpretation, or assessment of a resource[11].

Metadata must be machine-readable, in order to do that it must be written following:



Figure 2.2: The 4 steps of research data lifecycle [14]

- An *ontology*: A formal description of a knowledge domain’s structure (classes, properties, relationships), serving as a basis for knowledge graphs to ensure data consistency and model understanding.
- A data *taxonomy*: The hierarchical classification of data to create structure, standardize terminology, and promote dataset understanding within an organization.
- The use of *semantic metadata*: Metadata that explicitly defines the meaning of the data or content, enabling machines (not just humans) to interpret information about it.

Metadata are often developed as **metadata standards** – agreed-upon rules defining metadata structure, organization, and usage (e.g., [Dublin Core](#)). While standards establish general guidelines, **schemas** (structured frameworks defining metadata organization and content) represent concrete implementations of these standards. Schemas are typically developed by scientific communities via consensus to best describe resources according to their specific needs.

Processing and analysis

The processing stage involves preparing collected data for use. This typically includes tasks like handling missing values, documenting provenance

(the historical record of data transformations captured via metadata), reducing noise, and integrating data from multiple sources.

Data analysis, conversely, is the stage where processed data are used to generate new insights – the research findings. This usually involves computational tools or software, the use of which should be documented and, ideally, made available.

Share and reuse

To share research data, an appropriate repository must be selected that exposes metadata online for community discovery. Access controls should be implemented as needed. Shared data can then be reused in subsequent research cycles, potentially serving as raw data for further analysis or comparison.

2.2 FAIR principles

As introduced in Chapter 1.1, the FAIR principles serve as a crucial guide for data stewards aiming to enhance the reuse of scholarly data, although they are not a formal specification to follow. According to guidelines published by the Agency for Digital Italy (AGID)², FAIR criteria are defined as:

“FAIR criteria are the basis for the use and re-use of data from different thematic sources and methodology. To ensure that the production of FAIR data becomes the gold standard for the results of the publicly funded research involves substantial innovation in scientific practice both at the national level and at the production of data and at the level of their use to develop new knowledge”

These guidelines imply that research, particularly when publicly funded, should strive to meet these requirements.

Central to implementing FAIR principles is the concept of a [digital object](#), defined as:

Information in binary form and its associated metadata.

Managing and accessing these digital objects necessitates the use of **Persistent Identifiers (PIDs)**. PIDs are globally unique and long-lasting references to digital resources (such as datasets, publications, and software code)

²[Guidelines AGID \(Agency for Digital Italy\) \(agid.gov.it\)](#)

or even physical entities like researchers and institutions [3]. Beyond persistence, these identifiers are crucially actionable; they can be resolved by an agent (human or AI), for instance via a web browser, to directly access the referenced object. Common types of PIDs include [ORCID](#) identifiers for researchers and [DOIs](#) (Digital Object Identifiers) for publications and datasets.

2.2.1 FAIR for data

The FAIR principles, previously introduced in Section 1.1, provide a framework for enhancing the reusability of digital assets. This section delves into the specific concepts and practical implementations of each principle [15].³.

FINDABLE

It should be possible for others to discover your data.

Findability ensures that data can be located by both humans and machines (aligning with the concept of "Machine Ready Data"). Achieving this relies heavily on rich, machine-actionable metadata that facilitates automatic discovery. To meet the Findable criteria, the following principles (F1-F4) must be addressed:

1. (Meta)data are assigned a globally unique and persistent identifier. (F1)
2. Data are described with rich metadata. (F2)
3. Metadata clearly and explicitly include the identifier of the data they describe. (F3)
4. (Meta)data are registered or indexed in a searchable resource. (F4)

This principles can be explained as:

- F1: Data requires a **globally unique** and persistent identifier (PID), like a DOI, which is essential for realizing other FAIR principles.
- F2: Data needs rich metadata to enable discovery, often enhanced by using standard schemas catalogued in resources like FAIRsharing.

³Quotes are taken from [here](#).

- F3: Metadata must explicitly include the PID of the described data [15], linking the two and enabling discovery through models like Data Catalogue Vocabulary DCAT.
- F4: Data and metadata should be indexed in a searchable resource (e.g., repository, catalogue). While challenging due to diversity, platforms like Google Dataset Search aim to facilitate this.

ACCESSIBLE

It should be possible for humans and machines to gain access to your data.

Accessibility means that once found, the data (or at least its metadata) should be retrievable using a well-defined mechanism. This retrieval process can be automated (e.g., via HTTP) or manual (e.g. verbal exchange). The core idea is to provide a predictable access pathway, irrespective of whether the data itself is open or restricted. The following principles (A1-A2) guide accessibility:

1. (Meta)data are retrievable by their identifier using a standardized communication protocol. (A1)
2. The protocol is open, free and universally implementable. (A1.1)
3. The protocol allows for an authentication and authorization procedure where necessary. (A1.2)
4. Metadata should be accessible even when the data is no longer available. (A2)

A *communication protocol* is a set of rules and standards enabling exchange of data between two or more entities in a consistent and predictable manner[3]. This principles can be explained as:

- A1/A1.1: Data retrieval requires standard, open, free, and universally implementable protocols (like HTTP), avoiding proprietary methods.
- A1.2: Crucially, FAIR does not necessarily mean open. Access may be restricted, so the protocol must support authentication and authorization (often via HTTPS or dedicated AAs like Authentik used by OFED).

- A2: Metadata must persist long-term, even if the data itself becomes unavailable, ensuring a lasting descriptive record. This is typically addressed through data management policies, with the help of a Data Management Plan (DMP).

INTEROPERABLE

It should be possible to combine and exchange data from different sources.

Interoperability allows machines to automatically combine and process data from disparate sources. If multiple digital resources relate to the same entity, machines should ideally merge this information. Similarly, machines should be able to determine if data is compatible with specific tools or services [15]. This requires clear, machine-readable semantics for both data and metadata, which is particularly important in federated environments where data resides across multiple platforms. The principles for interoperability (I1-I3) are:

1. (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation. (I1)
2. (Meta)data use vocabularies that follow the FAIR principles. (I2)
3. (Meta)data include qualified references to other (meta)data. (I3)

This principles can be explained as:

- I1: Data and metadata should use a formal, accessible, shared, and broadly applicable language for knowledge representation, enabling machine processing (e.g., using Resource Description Framework(RDF)).
- I2: Vocabularies (i.e., terms used to describe data and metadata) used in data and metadata must also be FAIR (findable, accessible, interoperable, reusable), often achieved by using standard ontologies or controlled vocabularies (e.g., defined with Web Ontology Language (OWL)) hosted in accessible registries.
- I3: Include qualified references—links that explicitly define the relationship between resources (e.g., 'derived from')—often leveraging technologies like RDF and OWL to create interconnected knowledge.

REUSABLE

Data should be sufficiently documented to support their interpretation and reuse.

Reusability focuses on ensuring that data are described well enough for both humans and machines to understand and reuse them effectively, including clarity on usage conditions and attribution requirements. This necessitates providing clear data usage licenses and detailed provenance information. The principles for reusability (R1-R1.3) are: Principles of reusability are:

1. (meta)data are richly described with a plurality of accurate and relevant attributes. (R1)
2. (meta)data are released with a clear and accessible data usage license. (R1.1)
3. (meta)data are associated with detailed provenance. (R1.2)
4. : (meta)data meet domain relevant community standards. (R1.3)

This principles can be explained as:

- R1: Data requires rich metadata specifically enabling users (human or machine) to judge its suitability for diverse reuse scenarios, going beyond F2's discovery focus.
- R1.1: A clear usage license is mandatory (e.g., applying CC0), as ambiguity hinders reuse.
- R1.2: Detailed provenance (origin, creation process, context) must be provided, often using models like PROV or tools like CEDAR, to help users assess suitability and trustworthiness.
- R1.3: Adherence to relevant domain or community standards (e.g., NOMAD community practices for this thesis) is crucial for maximizing reusability within that context.

2.2.2 The value of FAIR

Implementing the FAIR principles offers direct and indirect benefits to various research stakeholders, positively impacting research quality and organizations. Key advantages include[3]:

- **Efficiency:** Reducing duplication of effort, time, and funding in research.
- **Compliance and Stewardship:** Facilitating rigorous management of digital resources helps researchers meet funder expectations.
- **Enhanced Research:** Enabling the scaling up of findings by integrating and analysing existing data from diverse sources.
- **Focus:** Allowing researchers to concentrate on value-adding activities like interpretation, rather than searching for or recreating data.
- **Infrastructure:** Improving the scientific infrastructure to better support knowledge discovery and innovation.

2.2.3 FAIR for code

The FAIR principles are also applicable to research software [16]. However, because software and data differ significantly in how they are produced, shared, archived, and published—with software being particularly dynamic and subject to continuous updates—the principles originally developed for data require adaptation. Recognizing this, the research community has developed specific recommendations for applying FAIR principles to research software, known as [FAIR4RS](#), which will be discussed next.

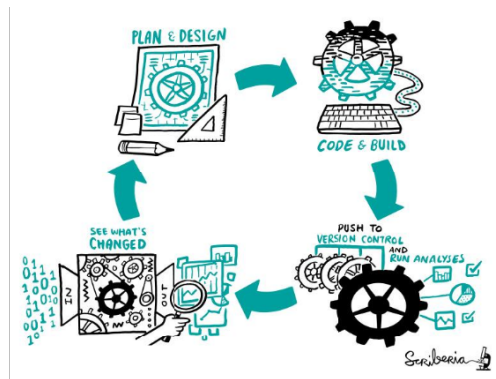


Figure 2.3: Picture from [The Turing way](#)

Repository

Selecting a **publicly accessible repository that incorporates version control** is of utmost importance. Version control systems facilitate tracking changes made by authors and collaborators; [Git](#) is the most prominent

example and will be utilized in this thesis. Several repository hosting platforms support Git integration; [GitHub](#) is a popular choice often used for this purpose.

License

Creative works, including software, are automatically protected by copyright upon creation. Applying a **software license** explicitly grants others permission to use, modify, or share the code under specified conditions. Numerous licenses exist, often drafted by legal professionals and employing complex legal terminology. Resources like [choosealicense.com](#) offer guidance for selecting an appropriate license. For this work, the [MIT](#) license has been chosen, where feasible⁴, due to its permissive nature.

Community Registry

Community registries enhance the discoverability of research software through search engines. These registries utilize metadata to improve software findability, enabling search engines to better understand the software's purpose, the problems it addresses, and its relevant domain(s). A curated list of various community registries is available [here](#). Due to time constraints no suitable community registry was set up for this project.

Citation

While **citation** is crucial for scientific accountability and reproducibility, accurately citing software presents unique challenges compared to citing papers. Identifying authorship, for example, can be difficult for external users. It is therefore highly beneficial when software developers provide clear citation information themselves.

Standards like [CodeMeta](#) are designed precisely for this purpose, allowing developers to package the necessary metadata to facilitate proper citation. Beyond citation metadata, ensuring long-term access and unambiguous identification requires archiving specific software versions. Archiving services address this by storing immutable snapshots of the software and assigning persistent identifiers (PIDs) like DOIs to these versions. This ensures that the PID reliably resolves to the exact archived version indefinitely. Ideally, creating these archived snapshots should be straightforward, potentially automated upon new software releases.

⁴Institutional policies may preclude its use in some cases.

Services such as [Zenodo](#) fulfil these archiving requirements, providing PIDs for software snapshots, and will be utilized where appropriate in this work.

Checklist

Employing **checklists** is a valuable practice for improving software quality. While the definition of 'good quality' varies with the software's application, it commonly encompasses aspects such as comprehensive documentation, continuous testing integration, and adherence to standardized coding practices. Numerous checklists are available, and it is often recommended to include the chosen checklist within the project's README file. One notable example is the [Badge Program developed by the Core Infrastructure Initiative](#). Due to time constraint no checklist was made for this project.

2.2.4 Interplay between FAIR and Data Management

General data management provides the essential framework for handling the research data lifecycle—planning, collection, storage, documentation, preservation, and sharing—to ensure integrity, compliance, and long-term accessibility. The FAIR principles complement this framework by specifically focusing on embedding Findability, Accessibility, Interoperability, and Reusability throughout this lifecycle. Critically, FAIR emphasizes the use of machine-actionable metadata, persistent identifiers, and interoperable formats right from project start.

Consequently, General data management and the FAIR principles are mutually reinforcing. Together, they create a synergistic approach that safeguards data quality and compliance while maximizing visibility, potential for automation, and reuse potential. This combination enhances research integrity, reproducibility, and cross-disciplinary collaboration, ultimately extending the impact and value of research outputs.

By integrating FAIR-by-design principles into standard data management practices, research projects can build robust, future-proof infrastructures. This approach not only meets compliance requirements but also unlocks the full potential for data reuse and collaboration. Embedding FAIR considerations systematically—from metadata strategy to repository selection—streamlines workflows, boosts machine-actionability, and ultimately accelerates scientific discovery.

2.3 FAIRNESS in Material Science: NOMAD and FAIRmat

To illustrate how FAIR data management can be applied field-wide, we will examine the FAIRmat community and the NOMAD platform. These initiatives are dedicated to **material science**, a discipline strongly associated with quantum technologies and information. Analysing their established methods is beneficial because it clarifies the rationale, detailed in subsequent chapters, for leveraging the NOMAD platform and developing a specific plugin, thereby building upon instruments they have already created⁵.

The [FAIRmat](#) consortium, part of the German National Research Data Infrastructure (NFDI), is dedicated to building a FAIR federated data infrastructure for solid state physics[18]. According to its official description⁶, the FAIRmat mission includes:

- Develop and maintain a federated FAIR data infrastructure for materials data with built-in tools and standards to support scientific collaboration and proper research data management (RDM) practices.
- Support the scientific community to introduce and maintain high standards of reproducibility, research integrity, and compliance with ethical and legal requirements by adopting proper RDM practices based on the FAIR data principles.

FAIRmat is a key contributor to the development of the Novel Materials Discovery (NOMAD) Laboratory [19]. [NOMAD](#) serves as a web-based, free, and open-source data management platform enabling researchers to manage and share materials science data. Initially designed for managing raw and processed simulation data according to FAIR principles, NOMAD has recently expanded its scope to include experimental data. The platform facilitates the structured publication of scientific data compliant with FAIR principles; specifically, it enforces the use of **schemas**, assigns **DOI PIDs** to published datasets, and allows machine-actionable data extraction.

A locally installable version, [NOMAD Oasis](#), is also available. Oasis is useful for localized FAIR data management (e.g., at institution level) and serves as a platform for developing NOMAD plugins, which extend NOMAD's capabilities with additional schemas and parsers.

⁵There's also the **Quantum Data Hub**[17], which is a USA focused project largely based on NOMAD.

⁶[FAIRmat](#)

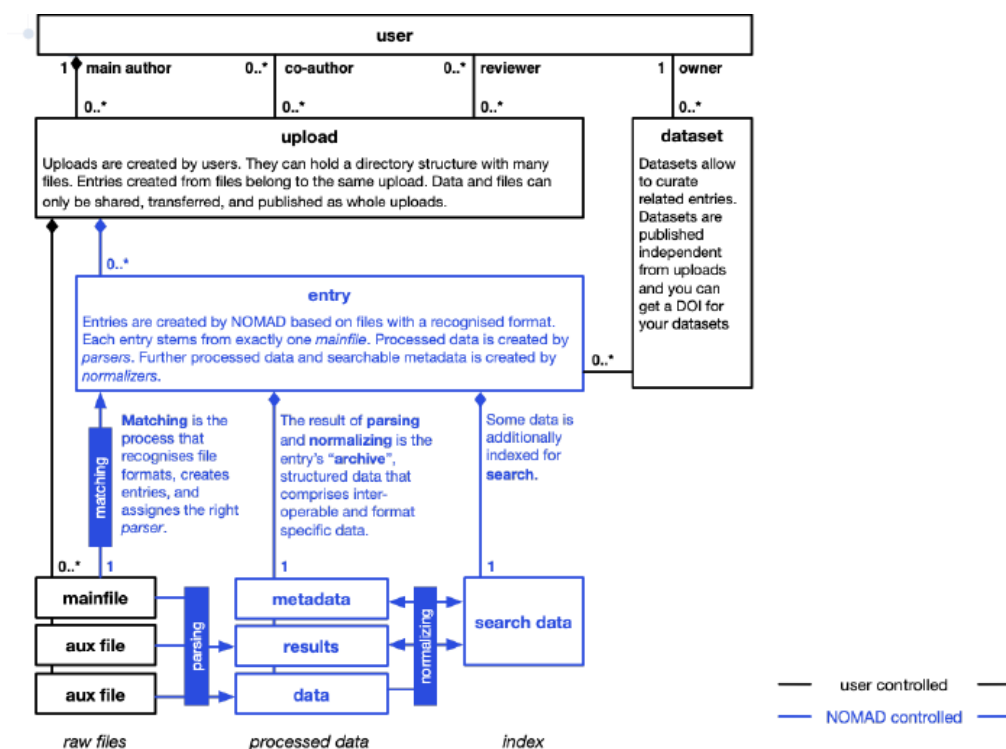


Figure 2.4: Diagram illustrating how NOMAD creates entries and datasets. Rather than supporting only predefined formats, NOMAD processes uploaded files to extract data and metadata from an extendable variety of formats [19].

2.3.1 NOMAD schemas

In NOMAD, schemas are collectively referred to as NOMAD Metainfo⁷. All data within NOMAD adheres to a schema defining its hierarchical structure and cross-references—this is the essence of Metainfo. Upon file upload, NOMAD automatically **processes** the data using available parsers and structures it within a **processed data** entry according to the predefined Metainfo schemas. This entry can subsequently be retrieved as a JSON file or via an API. Multiple entries can be organized into datasets, which are assigned DOIs upon publication.

NOMAD schemas are defined using a dedicated **schema language**, the **NOMAD Metainfo language**, which specifies the structure of packages and

⁷Metainfo: The sum of all schemas; specifically associated with the pre-defined schemas used to represent all processed data in a standardized way [18].

their Sections⁸. Sections can inherit properties from other sections, enabling

nomad.metainfo

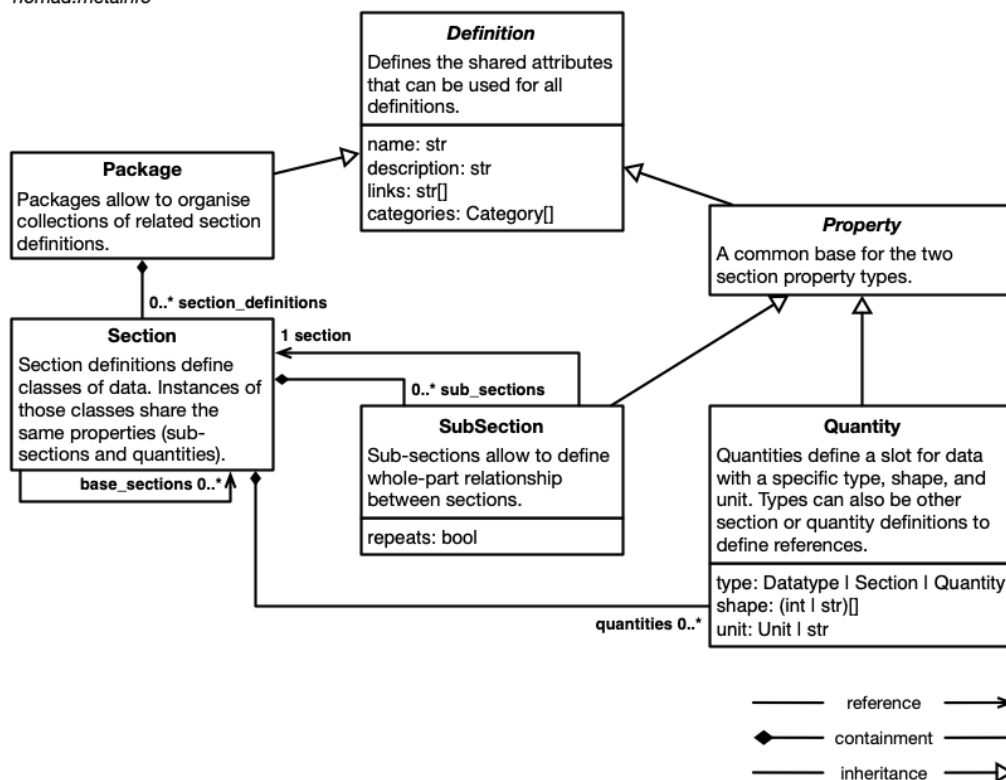


Figure 2.5: The NOMAD Metainfo schema language for structured data definitions

a hierarchical structure.

Schemas generally fall into three categories:

1. **Shared Entry Structure:** Defines a common structure for all entries, regardless of the specific data type.
2. **Reusable Base Sections:** Define common concepts (e.g., Workflow, Material, Method) that specific schemas can utilize and extend. These provide fixed interfaces for tools, maximizing reusability.
3. **Specific Schemas:** Define detailed structures for particular data types, often reusing base sections. Data conforming to specific schemas frequently requires normalization

⁸[All images taken from here](#)

(translation) into the more general 'results' section structure to enhance interoperability.

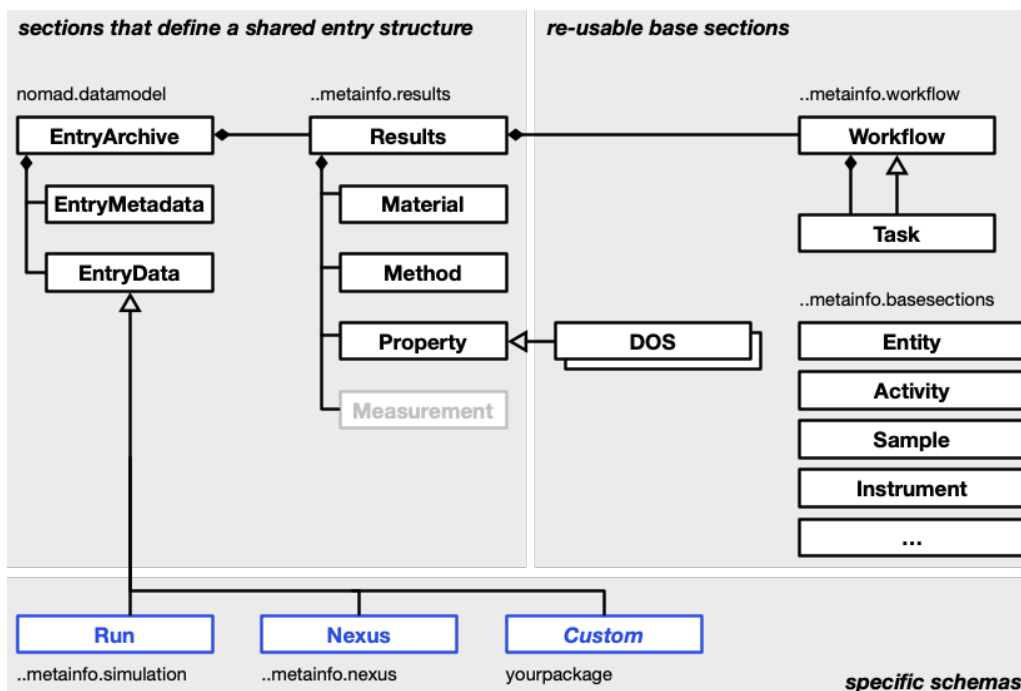


Figure 2.6: From specific data to more general interoperable data.

After processing, all data represents instances of Metainfo definitions. Each entry's archive is an instance of the *EntryArchive* root section. Both the schemas and the processed data are stored within archive files (e.g., archive.json, archive.yaml). These files instantiate *EntryArchive* and typically contain standard sections like metadata (containing NOMAD-added information such as IDs, authors) and results (holding key data in a consistent, source-independent structure), alongside a data section (content varies, e.g., run for simulations). The data section holds the detailed entry-specific information, potentially adhering to various schemas but ideally reusing base sections.

A relevant package example is [NOMAD-Simulations](#). Within this context, the *Simulation* section is treated analogously to other materials science activities, like experiments. This section inherits from the more abstract *BaseSimulation* section, which in turn inherits from *Activity*. *BaseSimulation* contains general program and simulation information, such as start

datetime, while *Simulation* adds details about specific inputs and outputs. Their relationship is shown here⁹:

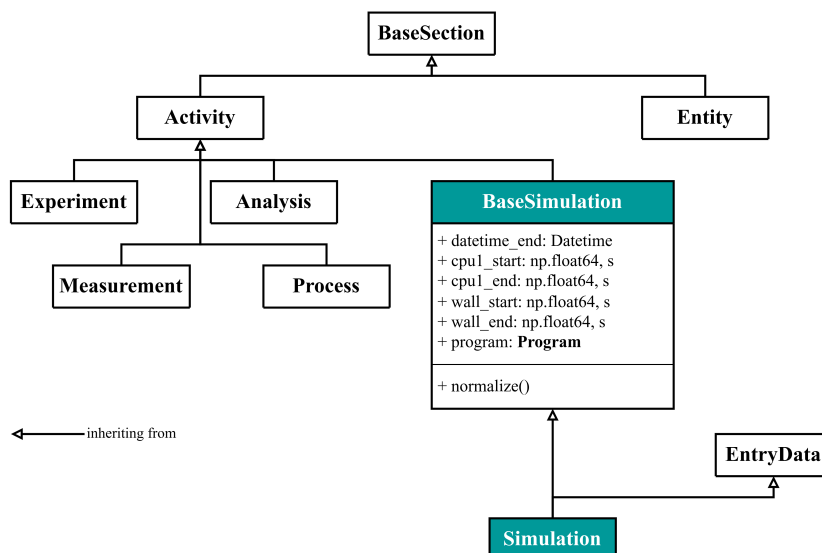


Figure 2.7: Relationship tree.

The *Simulation* section typically contains four main sub-sections:

1. *Program*: Contains program metadata, such as name and version.
2. *ModelSystem*: This input section describes the physical system being simulated. It contains system metadata, such as chemical composition (*ChemicalFormula*) or symmetry information (*Symmetry*).
3. *ModelMethod*: This input section details the mathematical model and methodology applied in the simulation. It often describes the Hamiltonian used, potentially broken down into contributions, and includes a sub-section for *NumericalSettings* (like basis set parameters or convergence thresholds). Specific methods (like DFT) inherit from base method sections.
4. *Outputs*: This section contains the results computed by the simulation. Properties (like *FermiLevel*) are stored here, often inheriting from a base *PhysicalProperty* section that includes the value and potentially variables the property depends on (like temperature).

⁹Reference for the subsequent figures.

2.4 Summary

This chapter provides a background on research data management (RDM) and the FAIR principles. It begins by defining research data and RDM, emphasizing the importance of handling data throughout its lifecycle, before discussing the stages of the research data lifecycle. The chapter then delves into the FAIR principles, positioning them as essential guidelines for enhancing data reuse, with each principle explained in detail. The value of adopting FAIR principles is highlighted, and the chapter also discusses the adaptation of FAIR principles for research software (FAIR4RS). Furthermore, the text emphasizes the synergistic relationship between general RDM and FAIR principles. Finally, the chapter presents FAIRmat and the NOMAD platform as a case study of FAIR implementation in material science, a field closely related to this project's quantum information context.

Having established this background in data management, FAIR principles, and relevant platforms like NOMAD, the subsequent chapters can now detail how these elements will be applied to implement the proposed FAIR data project.

Chapter 3

Scientific Background

This chapter covers the scientific background for the project, starting with key concepts in quantum mechanics, quantum information, and ab initio methods. These fundamentals are crucial because they define the data to be managed, and understanding the physics is vital for interpreting the data schemas discussed later.

The chapter then reviews the literature on quantum technologies possibilities based on Silicon Carbide (SiC), assessing the reasons for selecting this material. Nitrogen vacancies in diamond, a widely used system, are also discussed, as this thesis includes a computational experiment designed to simulate analogous behaviour to a result obtained with this material ([20]).

3.1 Theoretical Minimum

This section provides a concise introduction to the theory underpinning quantum information and computation. It also includes a brief overview of the ab initio methods employed in foundational research relevant to this thesis project. The topics covered are as follows:

- Definitions of fundamental quantum concepts: states, operators, and wavefunctions, including their physical interpretations [21].
- Key concepts in quantum information: the Bloch Sphere representation, quantum entanglement, and entangled states [22].
- Quantum dynamics and computational approaches: time evolution, the Schrödinger equation, and an introduction to how Density Functional Theory (DFT) addresses challenges in solving complex quantum mechanical problems [21][23].

Providing a fully detailed explanation of the requisite mathematics and physics to introduce the theory underlying this research project would significantly exceed the scope of this thesis. For a comprehensive background, readers are advised to consult introductory textbooks on quantum mechanics and quantum information, such as those by Shankar [21] and Nielsen & Chuang [22]. The following pages offer a brief introduction to the core concepts employed in this project¹

3.1.1 Quantum mechanics & Information

In Dirac notation, a quantum state is represented by a 'ket' vector, denoted $|\psi\rangle$, which resides in a Hilbert space $\mathcal{H}$. The corresponding 'bra', denoted $\langle\psi|$, is the associated covector in the dual space $\mathcal{H}^*$. The inner product (scalar product) between a bra and a ket is defined as:

$$\langle\psi|\psi\rangle = |\psi|^2 \quad (3.1)$$

The quantity $|\langle\phi|\psi\rangle|^2$ gives the probability of the system to be in state $|\phi\rangle$. If the state evolves in time, $|\psi(t)\rangle$, the probabilities may also depend on time. For a normalized state vector that contains **all** the basis states, this inner product is equal to 1, $\langle\psi|\psi\rangle = 1$.

As an example, consider a simple two-level system (analogous to the paths in a double-slit experiment), where the basis states can be represented as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (3.2)$$

These basis vectors are orthonormal, meaning $\langle 0|1\rangle = 0$ and $\langle 0|0\rangle = \langle 1|1\rangle = 1$. A general state $|\psi\rangle$ can be written as a **superposition**² of these basis states:

$$|\psi\rangle = \sqrt{p}|0\rangle + \sqrt{1-p}|1\rangle \quad \text{or} \quad |\psi\rangle = \begin{pmatrix} \sqrt{p} \\ \sqrt{1-p} \end{pmatrix} \quad (3.3)$$

p here represents the probability of finding the particle in the slit $|0\rangle$, analogous for the second slit and $1-p$. Its easy to see that $|\psi|^2 = p+1-p = 1$. The state of a single qubit (a two-level quantum system) can be visualized geometrically using the **Bloch sphere** (see Figure 3.1³). This representation

¹Familiarity with Linear Algebra and Functional Analysis is assumed.

²The second one is called "vector column notation", their bra are thus line vectors

³Smite-Meister, CC BY-SA 3.0 <<https://creativecommons.org/licenses/by-sa/3.0/>>, via Wikimedia Commons

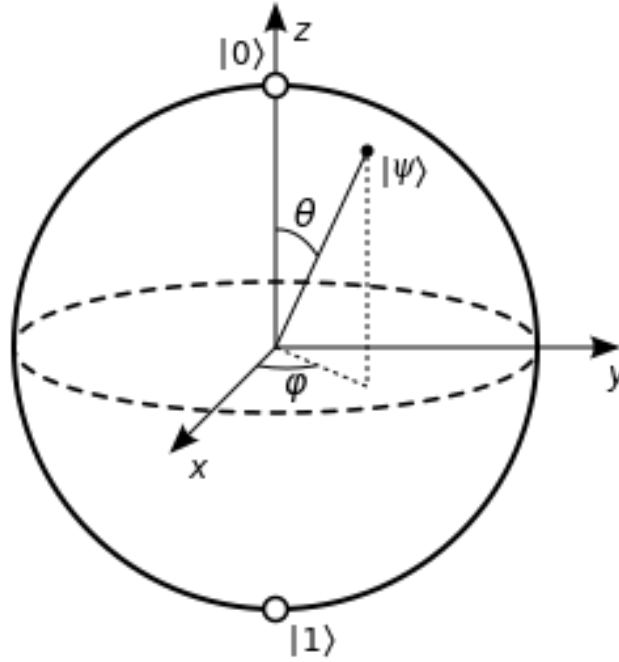


Figure 3.1: The Bloch Sphere representation of a single qubit state.

allows the probability amplitudes to be expressed using spherical coordinates θ and ϕ . A common parametrization is:

$$|\psi\rangle = \begin{pmatrix} \cos \frac{\theta}{2} \\ e^{i\phi} \sin \frac{\theta}{2} \end{pmatrix} \quad (3.4)$$

Physical operations or measurements are represented by applying matrices to these state vectors. For example, consider the [Pauli Matrices](#):

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (3.5)$$

$$\sigma_x |0\rangle = |1\rangle \quad (3.6)$$

$$\sigma_x |1\rangle = |0\rangle \quad (3.7)$$

Matrices acting on state vectors are called **operators**; they map one quantum state to another. Hermitian operators correspond physically to **observables**, and their eigenvalues represent the possible results of measuring that observable. A crucial observable is the **Hamiltonian** operator ($\hat{H}$), whose eigenvalues correspond to the possible energy levels of the system.

The expectation value of an operator $\hat{O}$ for a system in state $|\psi\rangle$ is given by $\langle\psi|\hat{O}|\psi\rangle$. As an example, using $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ and $|\psi\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle$ (with $\phi = 0$):

$$\begin{aligned}\sigma_z |\psi\rangle &= \cos \frac{\theta}{2} |0\rangle - \sin \frac{\theta}{2} |1\rangle \\ \langle\psi| \sigma_z |\psi\rangle &= \cos^2 \frac{\theta}{2} \langle 0| \sigma_z |0\rangle + \sin^2 \frac{\theta}{2} \langle 1| \sigma_z |1\rangle \\ &\quad + \cos \frac{\theta}{2} \sin \frac{\theta}{2} (\langle 0| \sigma_z |1\rangle + \langle 1| \sigma_z |0\rangle) \\ &= \cos^2 \frac{\theta}{2} - \sin^2 \frac{\theta}{2}\end{aligned}$$

A single **qubit** can be represented by a vector on the Bloch Sphere. For systems with multiple qubits, the state space is described by the tensor product of the individual qubit Hilbert spaces, $\mathcal{H}_{total} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_n$. The computational basis for a two-qubit system (Q1 and Q2) is formed by the tensor products of the single-qubit basis states:

Q1	Q2	STATE
$ 0\rangle$	$ 0\rangle$	$ 00\rangle = 0\rangle \otimes 0\rangle$
$ 0\rangle$	$ 1\rangle$	$ 01\rangle = 0\rangle \otimes 1\rangle$
$ 1\rangle$	$ 0\rangle$	$ 10\rangle = 1\rangle \otimes 0\rangle$
$ 1\rangle$	$ 1\rangle$	$ 11\rangle = 1\rangle \otimes 1\rangle$

A general state of two qubits can be written as a superposition:

$$|\psi\rangle = p_{00}|00\rangle + p_{01}|01\rangle + p_{10}|10\rangle + p_{11}|11\rangle \quad (3.8)$$

where $|p_{ij}|^2$ is the probability of measuring the system in the state $|ij\rangle$, and $\sum_{i,j} |p_{ij}|^2 = 1$. The state vector $|\psi\rangle$ now resides in a $2^2 = 4$ dimensional Hilbert space $\mathcal{H}$. In general, a system of n qubits has a state space dimension of 2^n . Operators acting on multi-qubit systems are formed by tensor products of single-qubit operators, e.g., $(\sigma_x \otimes \sigma_x)|10\rangle = (\sigma_x|1\rangle) \otimes (\sigma_x|0\rangle) = |0\rangle \otimes |1\rangle = |01\rangle$.

Entanglement is a key quantum phenomenon where a multi-qubit state **cannot** be written as a simple tensor product of individual qubit states (i.e., it is not separable). For example:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle) = |0\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad (3.9)$$

This state is separable and thus not entangled. In contrast, the state:

$$|\psi\rangle = \frac{1}{\sqrt{2}} (|01\rangle + |10\rangle) \quad (3.10)$$

is entangled because there is no way to express it as $|a\rangle \otimes |b\rangle$ for any single-qubit states $|a\rangle$ and $|b\rangle$.⁴

An alternative way to describe a quantum state is using a **density matrix**, ρ . For a pure state $|\psi\rangle$, the density matrix is defined as the outer product $\rho = |\psi\rangle \langle\psi|$. Using the previous example $|\psi\rangle = \sqrt{p}|0\rangle + \sqrt{1-p}|1\rangle$:

$$\rho = (\sqrt{p}|0\rangle + \sqrt{1-p}|1\rangle)(\sqrt{p}\langle 0| + \sqrt{1-p}\langle 1|) \quad (3.11)$$

$$\begin{aligned} &= p|0\rangle\langle 0| + \sqrt{p(1-p)}|0\rangle\langle 1| \\ &\quad + \sqrt{p(1-p)}|1\rangle\langle 0| + (1-p)|1\rangle\langle 1| \end{aligned} \quad (3.12)$$

In matrix form, relative to the $\{|0\rangle, |1\rangle\}$ basis, this is:

$$\rho = \begin{pmatrix} p & \sqrt{p(1-p)} \\ \sqrt{p(1-p)} & 1-p \end{pmatrix} \quad (3.13)$$

Density matrices are particularly useful for describing mixed states, which are statistical ensembles of pure states.

3.1.2 Ab initio Methods & DFT

Although ab initio and density functional theory (DFT) methods were not directly employed in this research project, a brief overview is warranted as results derived from this theoretical framework are utilized later. The foundation of quantum mechanics describes the time evolution of a physical state $|\psi\rangle$ via the **Schrödinger equation**, which connects the state's temporal development to the action of the Hamiltonian operator $\hat{H}$:

$$i\hbar \frac{d}{dt} |\psi(t)\rangle = \hat{H} |\psi(t)\rangle \quad (3.14)$$

Here, i is the imaginary unit and $\hbar$ is the reduced Planck constant. This equation is fundamentally an **eigenvalue equation**. For the time-independent version ($\hat{H}|\psi\rangle = E|\psi\rangle$), the solutions yield a set of energy eigenvalues E and their corresponding stationary states or eigenfunctions $|\psi\rangle$ for the system under consideration.

It is possible to represent the abstract state vector $|\psi\rangle$ as a complex-valued

⁴This specific state is also maximally entangled.

wavefunction of the spatial coordinates $\mathbf{r}$ by *projecting* it onto a chosen basis. For instance, projecting onto the position basis $\langle \mathbf{r} |$ yields the wavefunction in real space:

$$\langle \mathbf{r} | \psi \rangle = \psi(\mathbf{r}) \quad (3.15)$$

For a condensed matter system typically composed of interacting electrons and nuclei, the time-independent Schrödinger equation (3.14) can often be written explicitly for the electronic wavefunction ψ as:

$$\left[-\frac{\hbar^2}{2m} \sum_{i=1}^N \nabla_i^2 + \sum_{i=1}^N V(\mathbf{r}_i) + \sum_{i=1}^N \sum_{j < i} U(\mathbf{r}_i, \mathbf{r}_j) \right] \psi = E\psi \quad (3.16)$$

Solving Equation (3.16) directly is generally intractable for systems with many interacting particles, as there's no general solving method that gives a closed form analytic solution [21]. An approximation can be made, under the right conditions, in the following way.

$$\psi(\mathbf{r}_1, \dots, \mathbf{r}_N) = \psi(\mathbf{r}_1)\psi(\mathbf{r}_2)\dots\psi(\mathbf{r}_N) \quad (3.17)$$

The **electron density** at a position $\mathbf{r}$, denoted $n(\mathbf{r})$, is a fundamental quantity. For a system described by single-particle orbitals $\psi_i(\mathbf{r})$, the density can be defined as⁵:

$$n(\mathbf{r}) = 2 \sum_i \psi_i^*(\mathbf{r})\psi_i(\mathbf{r}) \quad (3.18)$$

where the sum runs over the occupied spatial orbitals. Significantly, $n(\mathbf{r})$ depends only on the three spatial coordinates, regardless of the number of electrons N , offering a considerable simplification compared to the $3N$ -dimensional wavefunction ψ . Two foundational theorems underpin Density functional theory [23]:

Theorem 1 (1° Hohenberg-Kohn theorem) *The ground-state energy from Schrödinger's equation is a unique functional of the electron density.*

This theorem implies a one-to-one mapping between the external potential $V(\mathbf{r})$, the ground-state density $n_0(\mathbf{r})$, and the ground-state wavefunction ψ_0 . Consequently, the ground-state energy can be expressed as an energy functional $E[n(\mathbf{r})]$.

Theorem 2 (2° Hohenberg-Kohn theorem) *The electron density that minimizes the energy of the overall functional is the true electron density corresponding to the full solution of the Schrödinger equation*

⁵The factor of 2 arises from the Pauli exclusion principle.

This second theorem establishes a **variational principle** for the density functional: any trial density $n(\mathbf{r})$ (satisfying necessary physical constraints) will yield an energy $E[n(\mathbf{r})] \geq E_0$. Minimizing the functional $E[n(\mathbf{r})]$ with respect to $n(\mathbf{r})$ therefore yields the ground-state density and energy, providing a practical route to solve Equation (3.16).

The Kohn-Sham approach facilitates the practical application of DFT. It replaces the difficult interacting many-body problem with a fictitious system of non-interacting electrons that generate the same ground-state density as the real system. These electrons move in an effective potential, and their behaviour is described by single-particle equations known as the **Kohn-Sham equations**:

$$\left[-\frac{\hbar^2}{2m} + V(\mathbf{r}) + V_H(\mathbf{r}) + V_{XC}(\mathbf{r}) \right] \psi_i(\mathbf{r}) = \epsilon_i \psi_i(\mathbf{r}) \quad (3.19)$$

Here, $V(\mathbf{r})$ is the external potential (usually from the nuclei), $V_H(\mathbf{r}) = e^2 \int \frac{n(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d^3r'$ is the Hartree potential representing the classical electrostatic repulsion due to the electron density, and $V_{XC}(\mathbf{r}) = \frac{\delta E_{XC}[n(\mathbf{r})]}{\delta n(\mathbf{r})}$ is the exchange-correlation potential derived from the exchange-correlation energy functional $E_{XC}[n]$. This latter term encapsulates all the complex many-body quantum mechanical effects beyond the Hartree potential [23].

Computationally, the Kohn-Sham equations (3.19) are typically solved iteratively with a procedure:

- Define an initial trial electron density $n(\mathbf{r})$.
- Construct the Kohn-Sham effective potential $V_{\text{eff}}(\mathbf{r})$ using the current density.
- Solve the Kohn-Sham equations to obtain the single-particle orbitals $\psi_i(\mathbf{r})$ and energies ϵ_i .
- Calculate the new electron density $n_{\text{new}}(\mathbf{r})$ from the occupied Kohn-Sham orbitals.
- Compare $n_{\text{new}}(\mathbf{r})$ with the input density $n(\mathbf{r})$. If they are sufficiently similar (i.e., convergence is reached), the process stops. Otherwise, update the density by using $n_{\text{new}}(\mathbf{r})$ and repeat from the second step.

Once convergence is achieved, the ground-state density, energy, and derived properties can be obtained. While practical implementations involve further approximations and numerical techniques, this cycle is central to DFT calculations for determining ground-state properties.

3.2 Previous literature

Understanding the rationale behind the choice of material for this project necessitates a review of the pertinent literature. The discussion begins with Nitrogen Vacancies (NVs) in diamond, a system [already utilized](#) in commercial high-precision magnetometers and possessing considerable potential for broader quantum technology applications. A specific, noteworthy example demonstrating its use as a quantum-enhanced sensor will be presented. Following this, Silicon Carbide (SiC) is introduced; its key quantum properties are outlined to justify its selection as a target material for investigation in this work.

3.2.1 Diamond Nitrogen Vacancy

The use of systems based on Nitrogen-Vacancy (NV) centers in diamond is a recognized approach within quantum technology. This paper[24] proposes utilizing such a system as a quantum simulator capable of operating at ambient temperature and pressure. The simulator's operation relies on leveraging strongly interacting nuclear spins chemically attached to the diamond surface.

The complete Hamiltonian of the systems is:

$$H = H_{NV-gs} + H_F + H_{NV-F} \quad (3.20)$$

Where H_{NV-gs} is the ground state for the NV, H_F is the interaction between the nuclear spins and H_{NV-F} is the interaction between the NV and the nuclear spins. Positioned a few nanometers beneath the diamond surface, the NV center serves to initialize, control, and read out the state of this nuclear spin quantum simulator. Control and readout schemes often involve applying microwave fields to the NV center and radio-frequency (RF) fields to the nuclear spins, utilizing the Hartmann-Hahn resonance condition to selectively transfer polarization or information between the NV electron spin and the target nuclear spins. Readout is typically achieved by optically measuring the spin-dependent fluorescence of the NV center after it has interacted with the nuclear spin system, allowing inference of nuclear spin properties like magnetization and correlations. RF pulses can map different nuclear spin correlations onto measurable NV observables.

The fundamental interaction between the surface nuclear spins is the magnetic dipole-dipole interaction H_F . Under a strong external magnetic field, this interaction can be approximated by an effective XXZ model Hamiltonian, which the system is engineered to simulate. The surface nuclear spin lattice can be constructed from atoms like Fluorine (^{19}F), potentially arranged in

triangular or rectangular lattice configurations. This proposed simulator architecture is designed to investigate diverse strongly correlated spin models, including phenomena such as frustrated magnetism and supersolids. Furthermore, the paper details numerical studies that support the feasibility of realizing this type of quantum simulator.

As of today, no physical realization of this kind of simulator has been made. Recent results [25]⁶ shows that the hyperfine interactions near the surface might be greater than what expected.

The utility of Nitrogen-Vacancy (NV) centers in diamond extends beyond quantum simulation to quantum spin sensing applications[26]. Notably, the NV centre's associated ^{14}N nuclear spin can function as a quantum memory to enhance the sensitivity of magnetic field measurements, exceeding limitations imposed by the electron spin's coherence time. This approach has been demonstrated, for instance, in measuring the weak magnetic field generated by a coupled ^{13}C nuclear spin[20].

While leveraging the same fundamental platform of NV centers in diamond and quantum control techniques, this application differs from quantum simulation. Instead of modeling a many-body system, the focus shifts to using the hybrid electron-nuclear spin system to improve magnetometer sensitivity, particularly for fields with long correlation times. Also the NV is far from the surface ($d \approx 15\mu\text{m}$)

Here, the NV electron spin acts as the primary sensor due to its strong magnetic moment, while the ^{14}N nuclear spin, with its much longer coherence time (approaching the nuclear $T_1 \approx 5\text{ms}$, compared to the electron $T_2 \approx 400\mu\text{s}$), serves as the quantum memory.

The relevant Hamiltonian for the NV electron spin ($S = 1$) interacting with nearby nuclear spins ^{14}N and ^{13}C in an external magnetic field is:

$$H = DS_z^2 - \gamma_e B_z S_z - \sum_i \gamma_{n,i} B_z I_{z,i} + \sum_i A_{zz} S_z I_{z,i} \quad (3.21)$$

Where D is the zero field splitting, B_z is the z-component of the magnetic field, γ_e, γ_n are the gyromagnetic ratios of the electron and the ^{14}N and the ^{13}C nuclear spin and A_{zz} is the zz component of the hyperfine tensor, as the other parts are zero due to symmetry reasons[27].

The core idea involves a phase estimation protocol that utilizes entanglement between the electron spin sensor (e) and the ^{14}N nuclear spin memory (n). The protocol proceeds as follows:

⁶Also not published research from CNR-IMM@CT

1. **Initialization:** Prepare the system in the state

$$|\psi_i\rangle = |0_e\rangle \otimes \frac{1}{\sqrt{2}}(|0_n\rangle + |1_n\rangle)$$

via optical pumping and an RF $\pi/2$ -pulse on the memory spin.

2. **Entanglement:** Apply a controlled-rotation gate ($C_n ROT_e$ [28] [29], specifically a CNOT-like gate controlled by the nuclear $|1_n\rangle$ state) to create a maximally entangled Bell state:

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}(|0_e 0_n\rangle + |1_e 1_n\rangle)$$

3. **First Sensing Interval:** During a free evolution time τ , the sensor accumulates a phase $\phi_1 = -\gamma_e B_z \tau$. The state becomes

$$\frac{1}{\sqrt{2}}(|0_e 0_n\rangle + e^{i\phi_1} |1_e 1_n\rangle)$$

4. **Store Phase:** Apply a second $C_n ROT_e$ gate (controlled by the nuclear $|0_n\rangle$ state) to disentangle the sensor and memory, transferring the phase information entirely to the memory qubit:

$$|\psi_1\rangle = |1_e\rangle \otimes \frac{1}{\sqrt{2}}(|0_n\rangle + e^{i\phi_1} |1_n\rangle)$$

The sensor spin ($|1_e\rangle$) now holds no phase information and can decohere without affecting the stored phase.

5. **Correlation Time (T_c):** Allow the system to evolve for a correlation time T_c . During this time, the stored phase is preserved on the memory (up to a possible detuning $\Delta\omega$), while the sensor qubit is effectively reset. The state becomes

$$|1_e\rangle \otimes \frac{1}{\sqrt{2}}(|0_n\rangle + e^{i(\phi_1 + \Delta\omega T_c)} |1_n\rangle)$$

6. **Second Sensing Interval:** Repeat steps 2-4 (Entanglement, Second Sensing τ , Store Phase). This accumulates a second phase ϕ_2 and stores the phase *difference* $\Delta\phi = \phi_1 - \phi_2$ onto the memory. The sequence is designed like a spin echo, refocusing static fields, so $\Delta\phi$ primarily measures changes in B_z that occurred during T_c . The final state is:

$$|\psi_f\rangle = |0_e\rangle \otimes \frac{1}{\sqrt{2}}(|0_n\rangle + e^{i(\Delta\phi + \Delta\omega T_c)} |1_n\rangle)$$

7. **Readout:** Measure the final state of the memory qubit (e.g., probability of being in $|1_n\rangle$) to determine the accumulated phase difference $\Delta\phi$.

This quantum memory protocol enhances sensing sensitivity in two ways:

- **Extended Interrogation Time:** It allows for the correlation of phase measurements (ϕ_1 and ϕ_2) separated by a time T_c that can be much longer than the sensor's coherence time. The ultimate limit is set by the memory's coherence time, which is significantly longer.
- **Improved Signal Contrast:** By using entanglement to store the full quantum phase ($e^{i\phi_1}$) on the memory (Step 4), the protocol ("enhanced sequence") retains the maximum possible signal contrast (A) in the correlation measurement. This contrasts with classical correlation methods where the sensor state collapses upon measurement or non-entangled storage, typically reducing the final signal contrast by half. A higher contrast leads to a better signal-to-noise ratio, reducing the averaging time needed to reach a given sensitivity.[20]

The benchmarks confirms that the enhanced (entangled) protocol has a 2 times improvement in signal contrast against the conventional (non-entangled) one.

3.2.2 Silicon Carbide

In addition to well-studied Nitrogen-Vacancy (NV) centers in diamond [27], Silicon Carbide (SiC) has emerged as a promising material hosting various optically active defects, or color centers, relevant for quantum applications [30].

SiC is already widely utilized in power electronics [31] owing to its advantageous material properties compared to silicon. Its wide bandgap (e.g., ≈ 3.26 eV for 4H-SiC, compared to silicon's 1.12 eV) imparts several key benefits:

- **High-Temperature Operation:** Electrons require significantly more energy to transition across the bandgap, enabling SiC devices to operate at higher temperatures (up to $\approx 300^\circ\text{C}$) compared to silicon ($\approx 150^\circ\text{C}$). SiC also exhibits enhanced **thermal conductivity**.
- **High Power and Frequency:** The wide bandgap contributes to a high critical electric field strength, allowing SiC devices to withstand **higher breakdown voltages** and support **higher frequency operation**.

- **Robustness:** Strong covalent bonds within the SiC crystal lattice confer exceptional **chemical inertness** and superior **resistance to radiation-induced** damage.

SiC’s potential suitability as a quantum technology has been already explored in different occasions, even as a quantum magnetic field sensor [32]. As an example, first-principles calculations have been instrumental in characterizing the material parameters of a Silicon Vacancy defect in bulk SiC interacting with nearby ^{29}Si and ^{13}C nuclear spins, establishing it as a defect centre of interest for quantum applications.[33].

Another particularly interesting system is the divacancy ($V_{\text{Si}}V_{\text{C}}$) defect near a hydrogen-terminated surface. First-principles investigations ([34]) confirm that the divacancy near $(2 \times 1)\text{:H}$ terminated 3C-SiC surfaces is a robust spin defect featuring a triplet ground state ($S = 1$) with minimal perturbation from the surface. Its electronic structure remains largely intact compared to the bulk, lacking detrimental surface states within the band gap, thus making it a promising candidate for quantum technologies. A recent comprehensive study employed a Quantum-TCAD approach for this system, simulating fabrication processes like epitaxial growth to control defect densities and performing detailed quantum simulations to analyse the coupled defect-hydrogen spin dynamics and the feasibility of quantum protocols[35]. Density Functional Theory (DFT) calculations have been employed to determine the zero-field splitting (ZFS) components, D and E , for divacancies at varying depths beneath the H-terminated surface [34]. These studies show that while the axial component D remains close to the bulk value, the transverse component E becomes non-zero due to the symmetry breaking induced by the surface, with its magnitude sensitive to the defect’s proximity to the surface. Different results for the values of the ZFS, instead, have been found with successive calculations⁷. Furthermore, DFT allows for the calculation of hyperfine tensor components describing the interaction between the divacancy electron spin and nearby nuclear spins, such as those of the surface hydrogen atoms⁸. This electron-nuclear **hyperfine interaction** is the crucial physical mechanism that might enable coupling between the divacancy electron spin (acting as a sensor) and the surface hydrogen nuclear spins (acting as potential memory qubits). To facilitate reader comprehension, mathematical details pertaining to this system, along with diagrams of its crystal lattice, will be presented in the subsequent in conjunction with the description of the research.

⁷Internal research at CNR-IMM@CT, still to appear.

⁸Internal research at CNR-IMM@CT, still to appear.

Building upon these properties, it is compelling to explore the application of a near-surface divacancy in SiC as a quantum sensor, potentially enhanced by using the surface hydrogen nuclear spins as a quantum memory. This concept follows the precedent set by Zaiser et al using the NV center and its native ^{14}N nuclear spin in diamond[20]. Investigating this possibility, leveraging the divacancy as the sensor and surface hydrogen spins as the memory, will be the scope of the next chapter.

Chapter 4

Research Work

This chapter presents the research performed for this thesis, outlining the system studied and the data produced, which forms the basis for the FAIR data management approach discussed later. To provide context, the chapter first briefly reviews the programming language and libraries used in the code-base, outlining the specific tools and class structures involved. It then delves into the core research, describing the selected physical system, its mathematical model (Hamiltonians), and summarizing relevant previous ab initio results. Finally, the simulations executed and their outcomes are presented. This comprehensive description aims to clearly identify all data quantities requiring FAIR implementation, as detailed in the next chapter.

4.1 Python and QuTiP

The [Python](#) programming language was selected as the primary environment for the simulations conducted in this thesis, owing to its open-source nature and widespread adoption within the scientific community.

Python is a high-level, interpreted, object-oriented programming language featuring a vast selection of libraries that facilitate the coding process and enhance code readability for a broad user base. Among these, the libraries most heavily utilized (both generally in scientific computing and specifically in this project) are [NumPy](#), [Matplotlib](#), and [SciPy](#). These are foundational libraries employed extensively in scientific calculations and simulations.

Another library fundamental to this project is [QuTiP](#) (Quantum Toolbox in Python)[36], an open-source library specifically designed for simulating quantum systems. According to the official QuTiP documentation:

QuTiP is open-source software for simulating the dynamics of open quantum systems. [...] QuTiP aims to provide user-friendly

and efficient numerical simulations of a wide variety of Hamiltonians, including those with arbitrary time-dependence, commonly found in a wide range of physics applications such as quantum optics, trapped ions, superconducting circuits, and quantum nanomechanical resonators. QuTiP is freely available for use and/or modification [...]. Being free of any licensing fees, QuTiP is ideal for exploring quantum mechanics [...].

The cornerstone class within QuTiP, serving as the basis for nearly all other components, is the **Quantum Object**(Qobj). This fundamental class essentially wraps a NumPy array with additional attributes that define its properties within a quantum mechanical context. While a comprehensive list of all attributes is unnecessary here, some of the most important include:

- **data**: The underlying numerical data (vector, matrix, or tensor) representing the object.
- **dtype**: The data type used for storing the numerical data.
- **dims**: A list describing the dimensions of the composite Hilbert spaces for the object.
- **shape**: The shape of the data array (e.g., rows and columns for a matrix).
- **type**: A string indicating the object's type, such as *oper* (operator), *ket* (ket vector), *bra* (bra covector), or *super* (superoperator).
- **isherm**: A boolean value indicating whether the object is Hermitian.
- **isunitary**: A boolean value indicating whether the object is a unitary operator.

As an example:

Listing 4.1: Quantum objects examples

```
1 #Bra covector
2 a = np.array([[1, 2, 3, 4, 5]])
3 print(qt.Qobj(a))
4
5 #Ket vector
6 b=[1, 2, 3, 4, 5]
7 print(qt.Qobj(b))
8
```

```

9  #Operator
10 c = np.array([[1,2],[3,4]])
11 print(qt.Qobj(c))

```

Gives as an output:

Listing 4.2: Quantum object outputs

```

Quantum object: dims=[[1], [5]], shape=(1, 5),
type='bra', dtype=Dense
Qobj data =
[[1.  2.  3.  4.  5.]]

Quantum object: dims=[[5], [1]],
shape=(5, 1),
type='ket', dtype=Dense
Qobj data =
[[1.]
 [2.]
 [3.]
 [4.]
 [5.]]

Quantum object: dims=[[2], [2]],
shape=(2, 2),
type='oper', dtype=Dense,
isherm=False
Qobj data =
[[1.  2.]
 [3.  4.]]

```

The Qobj class provides numerous methods for quantum mechanical operations. For example, the **eigenenergies** and **eigenstates** methods return the eigenvalues and eigenstates, respectively, of a given operator Qobj¹. Common operations between quantum objects include **tensor products**, which are essential for constructing the Hilbert spaces of composite systems, such as multi-qubit systems. To simulate quantum dynamics, QuTiP integrates several **solvers**. Among the most important are **mesolve** and **sesolve**.

- **sesolve** calculates the unitary time evolution of a closed quantum system according to the Schrödinger equation (see Equation 3.14). It requires the system's Hamiltonian (H) and the initial state ($|\psi\rangle$) as primary inputs.

¹The eigenstates() method returns both eigenvalues and eigenstates.

- `mesolve` solves the Lindblad master equation[22], which describes the evolution of an open quantum system interacting with its environment. In addition to the Hamiltonian and initial state (which can be a state vector or density matrix), `mesolve` requires a list of collapse operators (C_{ops}) that model the system-environment interaction.

Both solvers return an instance of the `Result` class, which stores the simulation output and metadata. Key attributes of the `Result` class include:

- **Solver:** A string indicating the name of the solver used (e.g., 'mesolve', 'sesolve').
- **Solver stats:** A dictionary containing various statistics about the solver run, such as the specific ODE solver employed, setup time, number of collapse operators and method-specific details.
- **Time interval:** A list or array of the time points at which the simulation results were stored.
- **Number of e_{ops} :** The number of expectation value operators provided (relevant if expectation values were calculated).
- The `Qobj` representing the state of the system at the final time point, if saved.

As an example let's take a simple time evolution:

Listing 4.3: Time evolution example

```

1  #Hamiltonian
2  H = 2*np.pi * 0.1 * qt.sigmax()
3  #Starting state
4  psi0 = qt.basis(2, 0)
5  #Time span
6  times = np.linspace(0.0, 10.0, 100)
7  #Results
8  result = qt.sesolve(H, psi0, times, e_ops=[qt.sigmaz()])
9  print(result)
10 #Plotting
11 fig, ax = plt.subplots()
12 ax.plot(result.times, result.expect[0])
13 ax.set_xlabel('Time')
14 ax.set_ylabel('Expectation values')
15 ax.legend("SZ")
16 plt.show()

```

Gives as an output:

Listing 4.4: Time evolution outputs

```
<Result
  Solver: sesolve
  Solver stats:
    method: 'scipy zvode adams'
    init time: 6.556510925292969e-05
    preparation time: 0.00013065338134765625
    run time: 0.009765148162841797
    solver: 'Schrodinger Evolution'
  Time interval: [0.0, 10.0] (100 steps)
  Number of e_ops: 1
  State not saved.
>
```

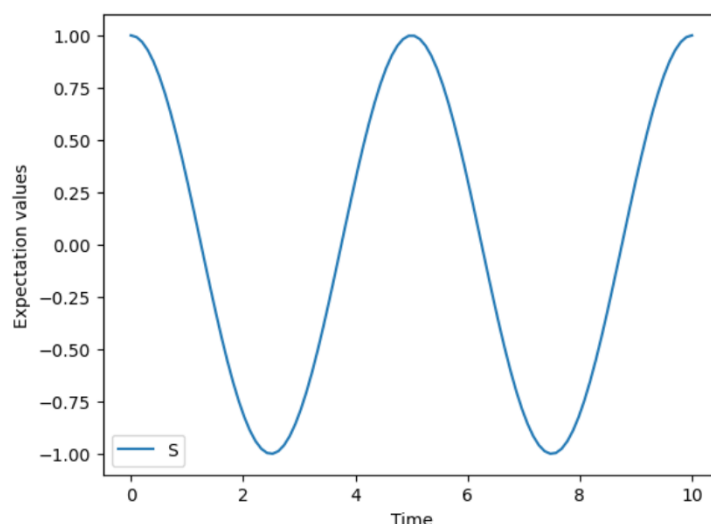


Figure 4.1: Time evolution figure.

Although a more exhaustive treatment of the QuTiP library is possible, the fundamental elements presented are adequate for comprehending the research detailed subsequently. More importantly, this overview establishes the context for identifying the specific data and metadata components that necessitate the FAIR-by-design management approach outlined in the following chapter.

4.2 System Description

As introduced in Section 3.2.2, the system under investigation is the cubic polymorph of Silicon Carbide (3C-SiC). The central defect interacts via hyperfine coupling with the $S=1/2$ nuclear spins of nearby Hydrogen (H) atoms adsorbed on the surface. These H atoms are arranged either in a triangular lattice on the (1 1 1) surface or a square lattice on the (0 0 1) surface, as illustrated in Figures 4.2 and 4.3. However, in the specific scenario consid-

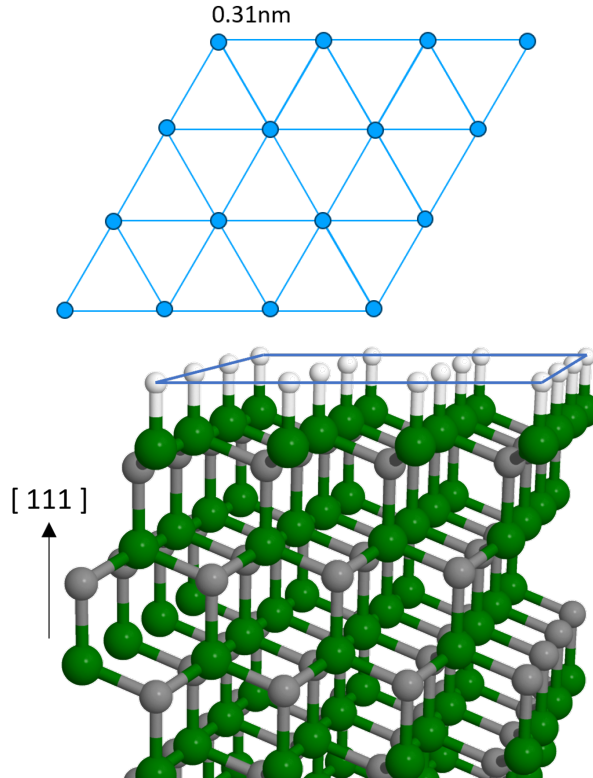


Figure 4.2: (upper) H nuclear spins 2D lattice in the SiC (1 1 1) triangular lattice (filled blue circles). (lower) Stable H distribution in (1 1 1) surface as obtained by ab-initio calculation (white spheres H atoms, gray spheres C atoms, light brown spheres Si atoms). Lattice constants are indicated. (Courtesy of Dr. G. Calogero)

ered in this work, following [20], only a single hydrogen atom is assumed to be present and interacting on the surface.

A divacancy defect ($V_C V_{Si}$) is positioned beneath the SiC surface at varying depths (see Figure 4.4 for an example). Regardless of its proximity to the surface, this defect possesses a triplet ground state ($S=1$). This spin

triplet state is intended to serve as the basis for implementing quantum gate operations enabling a quantum memory mechanism.

4.2.1 Description of Hamiltonians

Let's give a description of the Hamiltonian of this system. The Hamiltonian is divided in 3 parts:

$$H = H_{elect} + H_{zeeman} + H_{int} \quad (4.1)$$

Where H_{elect} represents the electronic spin Hamiltonian of the divacancy, H_{zeeman} includes the Zeeman interactions for both electron and nuclear spins, and H_{int} describes the hyperfine interaction between them. The electronic spin Hamiltonian term is given by:

$$H_{elect} = -\gamma_e \mathbf{B} \cdot \mathbf{S} + D \left(S_z^2 - \frac{2S(S+1)}{3} \mathcal{I} \right) + E(S_x^2 - S_y^2) \quad (4.2)$$

Here, $\mathbf{S} = (S_x, S_y, S_z)$ is the spin-S operator for the divacancy ($S=1$), $\mathbf{B}$ is the external magnetic field vector, $\mathcal{I}$ is the identity operator, D is the axial zero-field splitting (ZFS) parameter, and E is the rhombic ZFS parameter, which arises due to broken symmetry near the surface. Note that the $(-\gamma_e \mathbf{B} \cdot \mathbf{S})$ represents the electronic Zeeman term. The second part is the Nuclear Zeeman term of the Hydrogen atom ($I=1/2$):

$$H_{zeeman} = -\gamma_H \mathbf{B} \cdot \mathbf{I} \quad (4.3)$$

Where γ_H is the gyromagnetic ratio of the Hydrogen nuclear spin, $\mathbf{B}$ is the external magnetic field vector, and $\mathbf{I}$ is the nuclear spin operator. The final term describes the magnetic dipole-dipole interaction (hyperfine interaction) between the electron and nuclear spins:

$$H_{int} = A_{zz}(\mathbf{S} \cdot \mathbf{I} - 3((\mathbf{S} \cdot \hat{\mathbf{r}})(\mathbf{I} \cdot \hat{\mathbf{r}})) \quad (4.4)$$

Where A_{zz} is the z component of the hyperfine tensor (imagined in this case also as the only contribution[20]) and $\hat{\mathbf{r}}$ is unit vector that defines the orientation between spins. The component A_{zz} can be calculated as

$$-\frac{\mu_0}{4\pi\hbar} \frac{\gamma_e \gamma_H \hbar^2}{r^3} [\text{Hz}] \quad (4.5)$$

In the simulations performed in this project, the external magnetic field $\mathbf{B}$ is applied solely along the z-direction, i.e., $\mathbf{B} = (0, 0, B_z)$. Consequently, the quantization axis for both spins aligns with the z-axis, thus the vector $\hat{\mathbf{r}} = (0, 0, 1)$.

4.2.2 Previous ab-initio calculations

As mentioned in Section 3.2.2, first-principles methods were previously used to obtain the Zero-Field Splitting (ZFS) parameters (D and E) for the divacancy defect. However, discrepancies exist between the values reported in the literature [34] and those calculated more recently within our research group². To assess the impact of these different parameter sets, the simulations presented later in this chapter were performed using both sets of values. These ZFS parameters, corresponding to the bulk defect and defects at different distances from a hydrogenated surface (labeled H L6 and H L3, representing specific layer depths or configurations), are summarized in Tables 4.2.2 and 4.2.2.

Structure	Distance (Å)	D (MHz)	E (MHz)-Paper
bulk	0	1425	0
H L6	6	1425	−151
H L3	12	1431	−15

Structure	Distance (Å)	D (MHz)	E (MHz)-Institute
bulk	0	1425	0
H L6	6	1425	−180
H L3	12	1431	−108

Furthermore, while [34] did not report the hyperfine coupling parameter A_{zz} , calculations performed within our group yielded values of 122.458 kHz at 6Å and 189.785 kHz at 12Å. These values will also be used in the subsequent simulations.

4.3 Simulations

This section discusses the objectives, methods, and results of the quantum simulations performed. Due to pre-publication embargo restrictions, the complete simulation code and raw data cannot be presented. However, key results, relevant methodology and code segments are included where necessary to contextualize the data generated and discussed in the subsequent chapter on FAIR data management.

The system under investigation consists of two interacting spins: a divacancy qutrit ($S=1$) and a hydrogen nucleus qubit ($I = \frac{1}{2}$). The Hilbert space for

²Manuscript in preparation.

this composite system is defined by the tensor product of the individual spin bases and their corresponding operators.

The analysis begins by calculating the eigenvalues of the divacancy's electronic Hamiltonian as a function of an external magnetic field, B_z , in order to understand how the states $|m_s = +1\rangle$ and $|m_s = -1\rangle$ are mixed and how the energies involved in the non-interacting system. Following this, the expectation value $\langle S_z \rangle$ and the coefficients of the $|+\rangle$ eigenstate (projected onto the $m_s = \{\pm 1\}$ basis states) are computed and plotted against B_z .

The primary goal of the simulations is to generate entanglement between the electron and nuclear spins. This is attempted by implementing a controlled-NOT gate acting on the nuclear spin conditioned on the electron spin state $C_n ROT_e$, similar to protocols described in [Reference 3.2.1].

The gate is synthesized using a microwave (MW) pulse applied for a duration T_g with a Rabi frequency Ω_{rb} . The gate time duration is typically chosen based on the desired pulse angle, for example, $T_g = \frac{n_\pi \pi}{2\Omega_{rb}}$, where n_π determines the net rotation of $\frac{\pi}{2}$ excitations in the driving sequence. The time-dependent driving Hamiltonian is modelled as:

$$H_t = \frac{1}{2} \Omega_{rb} \cos(\omega_{ex} t) S_{x,eff} \quad (4.6)$$

Here $S_{x,eff}$ represents the effective spin-X operator that drives transitions specifically within the $|m_s = 0\rangle$ and $|m_s = -1\rangle$ subspace of the divacancy spin:

$$S_{x,eff} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The driving frequency ω_{ex} is set to the resonance frequency for the desired transition. This frequency is determined numerically by diagonalizing the full Hamiltonian and finding the energy difference (difference in eigenvalues) between the eigenstates corresponding primarily to $|1\rangle_n \otimes |+\rangle_e$ and $|1\rangle_n \otimes |0\rangle_e$. The algorithm to find the correct eigenstates is referred in A.1.

The simulation follows these steps:

- System parameters are initialized, including the zero-field splitting terms D and E , the hyperfine coupling A_{zz} , and the Rabi frequency Ω_{rb} .
A static external magnetic field of maximum $B_z = 0.03$ T is consistently applied.
- The first time evolution simulates the $C_n ROT_e$ gate, starting from the

initial product state $|\psi_0\rangle = |0_e\rangle \otimes \frac{1}{\sqrt{2}}(|0_n\rangle + |1_n\rangle)$. The aim is to transform this into the target entangled state $|\psi_1\rangle = \frac{1}{\sqrt{2}}(|0_e 0_n\rangle + |1_e 1_n\rangle)$.

- During the evolution, expectation values such as $\langle S_{z,d} \rangle, \langle S_{z,H} \rangle, \langle S_{x,d} \rangle$ are monitored. The fidelity, measuring the similarity between the evolving state and the target state $|\psi_1\rangle$ ([22]), is tracked by calculating the expectation value of the target state projector $\hat{\rho}_{target} = |\psi_1\rangle \langle \psi_1|$.
- The fidelity between the final state of the evolution (at $t = T_g$) and the target state $|\psi_1\rangle$ is explicitly calculated using QuTiP's fidelity function (see [QuTiP documentation](#)).

Following the first gate, the system evolves freely for a duration $\tau = \frac{1}{2A_{zz}}$, as proposed in [20]. During this free evolution, the $|1_e 1_n\rangle$ component acquires a relative phase $\phi = -\gamma_e B_z \tau$. Subsequently, a second $C_n ROT_e$ gate is applied. This gate is analogous to the first, but the driving frequency ω_{ex} is now calculated based on the energy difference between eigenstates corresponding to $|0\rangle_n \otimes |+\rangle_e$ and $|0\rangle_n \otimes |0\rangle_e$. The success of this second stage is evaluated by checking the fidelity with the final target state $|\psi_2\rangle = |1_e\rangle \otimes \frac{1}{\sqrt{2}}(|0_n\rangle + e^{i\phi} |1_n\rangle)$.

4.3.1 Results

This section presents and discusses the simulation results. We first analyze the static properties of the divacancy electron spin system before examining the time evolution during the gate processes.

The fundamental energy level structure and spin characteristics of the divacancy electron spin in response to a static magnetic field are shown in Figure 4.5. These properties depend only on the electronic Hamiltonian and are independent of the electron-nuclear coupling parameters chosen later. The top panel of Fig. 4.5 displays the energy eigenvalues versus the external magnetic field B . The $|0\rangle$ state (blue line) exhibits minimal energy change, consistent with its $m_s = 0$ character and weak dependence on the B field in this range. Its energy near -950MHz is primarily set by the zero-field splitting (ZFS) parameters D and E . Conversely, the $|+\rangle$ (orange) and $|-\rangle$ (green) states show significant Zeeman splitting, with their energy separation increasing approximately linearly with $|B|$. At $B = 0$, these states are split only by the rhombic ZFS term (E), which causes mixing between the $m_s = \pm 1$ basis states.

The middle panel confirms this behavior by plotting the expectation value $\langle S_z \rangle$. The $|0\rangle$ state has $\langle S_z \rangle \approx 0$. The other two states show $\langle S_z \rangle$ approaching ± 1 at high fields, indicating spin polarization, while exhibiting values between -1 and +1 near $B = 0$ due to state mixing.

The bottom panel vividly illustrates this state mixing for the $|+\rangle$ eigenstate near $B = 0$. It plots the coefficients (real part) of the $|m_s = +1\rangle$ (blue, coeff $|0\rangle$) and $|m_s = -1\rangle$ (orange, coeff $|2\rangle$) basis states within the $|+\rangle$ eigenstate. At $B = 0$, the $|+\rangle$ eigenstate is clearly an almost equal superposition of $|m_s = +1\rangle$ and $|m_s = -1\rangle$ (coefficients $\approx 1/\sqrt{2}$), driven by the rhombic ZFS (E). As $|B|$ increases, the eigenstate aligns towards one of the pure basis states, indicated by one coefficient approaching 1 and the other approaching 0.

Next, we examine the time evolution simulations designed to implement the C_nROT_e gate and generate entanglement. By varying the divacancy-hydrogen distance (r), the ZFS parameters (E and D), the Rabi frequency (Ω_{rb}) and the source of the hyperfine coupling (A_{zz}).

For a separation $r=6\text{\AA}$, the gate sequence generally achieves high fidelity, as shown in Figures 4.6 and 4.7. Using the formula-derived A_{zz} (Eq.4.5) and E values from 4.2.24.2.2, the first gate yields fidelities of 0.89–0.90, and the state after the second gate sequence (including free evolution) reaches fidelities of 0.95–0.97. All for $\Omega/2\pi = 0.2\text{MHz}$.

A notable exception occurs when using the DFT-computed A_{zz} value at $r=6\text{\AA}$ (Figure 4.8). While the first gate achieves high fidelity (0.94), the fidelity after the second gate sequence drops significantly to 0.74. This discrepancy might arise from the interaction Hamiltonian using only the A_{zz} component and neglecting other terms in the full hyperfine tensor, which may become more relevant near the surface.

When the separation is increased to $r = 12\text{\AA}$ (Figures 4.9 and 4.10), the gate performance using the formula-derived A_{zz} decreases considerably. The final state after the second gate fails to achieve an expectation value of 1 for $\langle S_{z,d} \rangle$, and the fidelities are generally lower (0.51–0.75). However, using the DFT-computed A_{zz} (Figure 4.11) yields significantly better results, with fidelities of 0.81–0.83 and $\langle S_{z,d} \rangle \approx 1$. This suggests that at larger distances from the surface (or the H atom), approximating the hyperfine interaction by only its A_{zz} component is a solid approximation and the DFT-computed value yield better results this time.

Considering the overall fidelity results across various parameters, the first gate generally performs well, often achieving high fidelity for different choices of $\Omega/2\pi$ (selected through trial and error), particularly when using institute-computed parameters (Table 4.2.2). Results using parameters from [34], however, remained low. In contrast, achieving high fidelity after the full sequence including the second gate and free evolution appears consistently more challenging across most parameter sets investigated.

In conclusion, these results suggest that enhancing SiC magnetometer sensitivity using an adjacent quantum memory is plausible. While promis-

ing, this hypothesis requires further theoretical investigation and ultimately experimental verification to be confirmed.

4.4 Future research directions

This research opens several avenues for future investigation. For example, the model could be extended by increasing the number of hydrogen atoms interacting with the divacancy. Simulating such multi-nuclear spin systems would represent a step towards greater physical realism and would explore more complex many-body dynamics.

An ancillary benefit of this approach would be the generation of substantially larger datasets, thereby enabling a rigorous evaluation of the scalability and robustness of the FAIR data management procedures (detailed in the subsequent chapter).

A second potential extension involves substituting the hydrogen atom ($S=1/2$) on the surface with deuterium, a spin-1 nucleus ($S=1$). Examining the impact of this isotopic substitution on the system's quantum behaviour presents another interesting research direction.

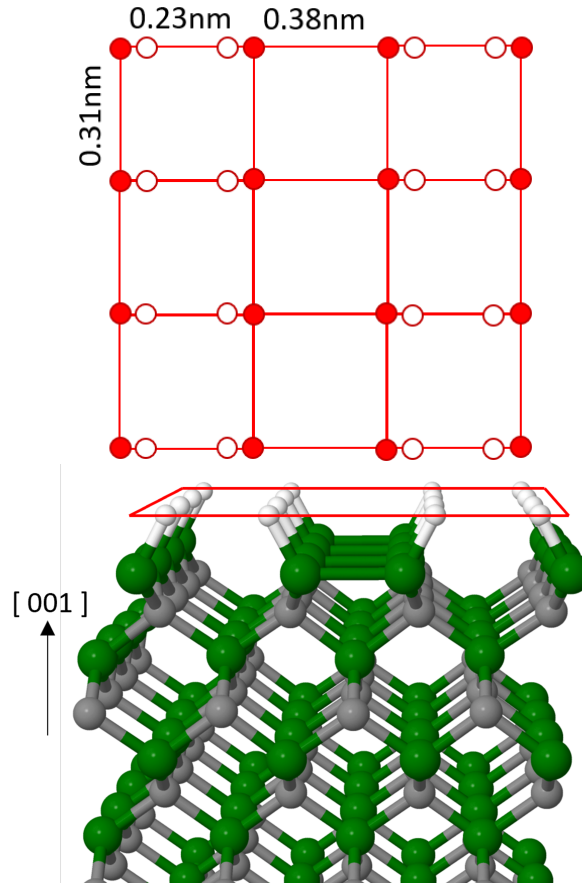


Figure 4.3: (upper) H nuclear spins 2D lattice in the ideal (filled red circles) and 2×1 reconstructed SiC (0 0 1) surfaces (empty red circles). (lower) Stable H distribution in a 2×1 reconstructed (0 0 1) 3C-SiC surface as obtained by ab-initio calculation. Lattice constants are indicated. (Courtesy of Dr. G. Calogero)

Distance from the surface
[~0 ; ~ 5nm]

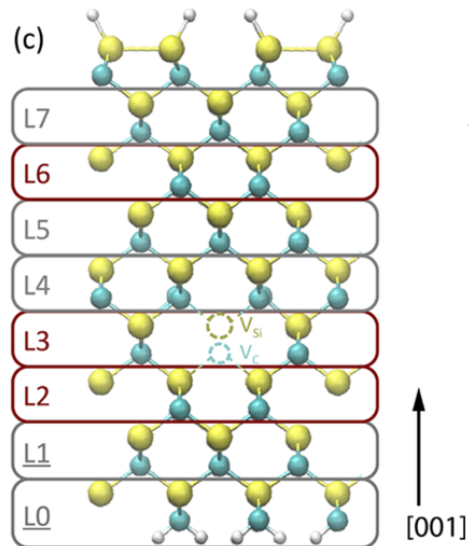


Figure 4.4: Example position of a divacancy in the middle slab. (Courtesy of [34])

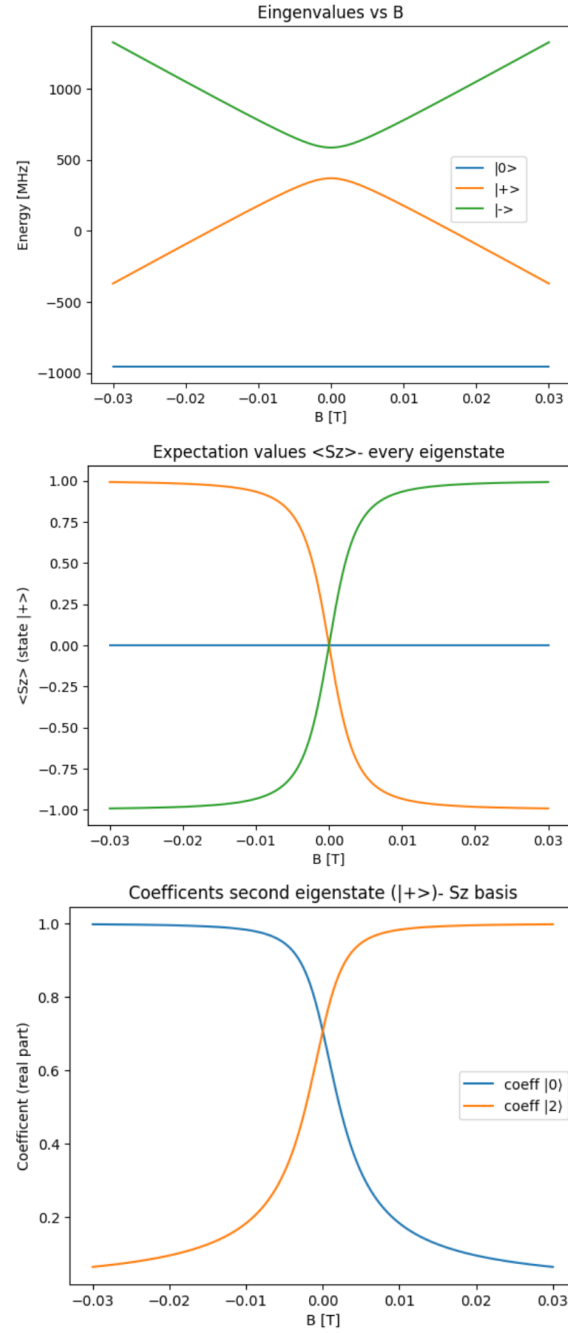


Figure 4.5: (upper) Eigenvalues of the model as a function of the external magnetic field B . (middle) Expectation values of the S_z operator for the same states. (lower) Coefficient of the first excited state of the model in Eq. 1 as a function of the external magnetic field B .

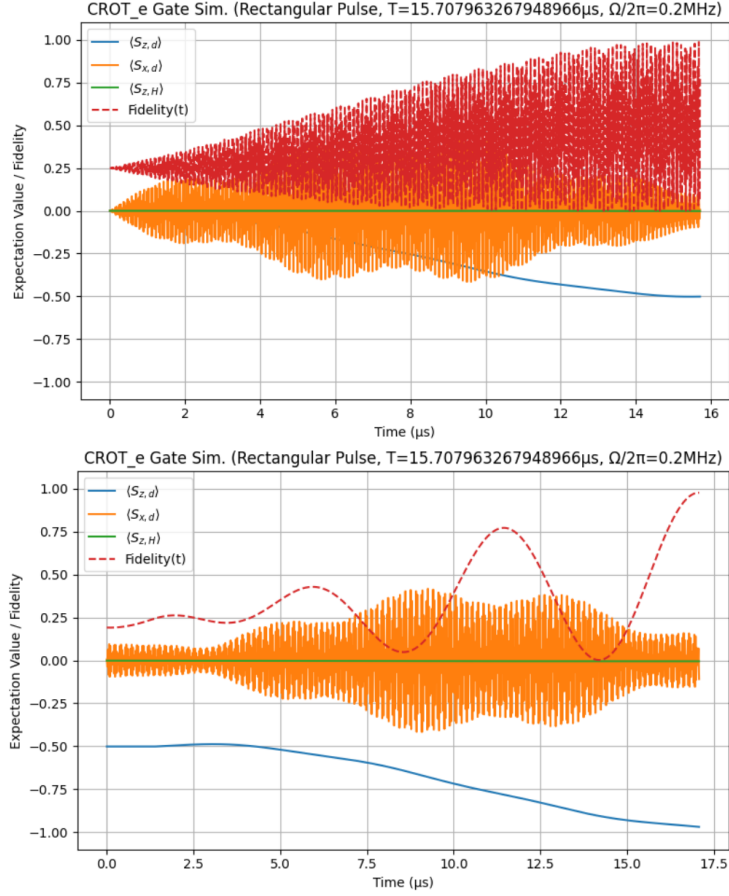


Figure 4.6: (upper) Time evolution for the first gate, final fidelity= 0.89. (lower) Time evolution for the second gate with initial free time evolution, final fidelity= 0.97. Parameters: $r=6\text{\AA}$, $D=1425\text{MHz}$, $E=-151\text{MHz}$, A_{zz} given by the formula 4.5

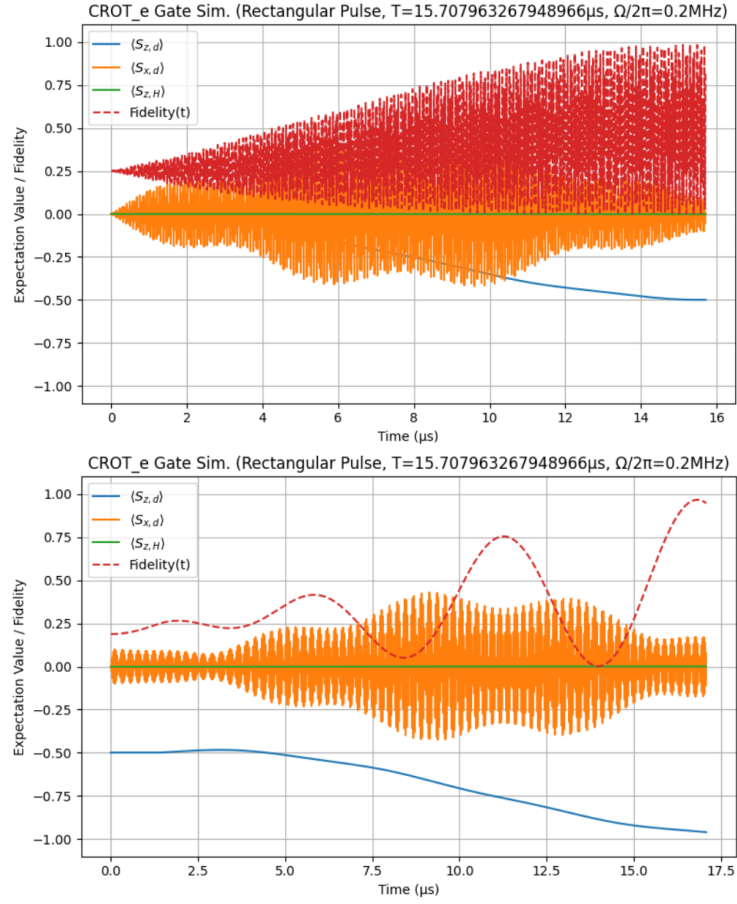


Figure 4.7: (upper) Time evolution for the first gate, final fidelity= 0.90. (lower) Time evolution for the second gate with initial free time evolution, final fidelity= 0.95. Parameters: $r=6\text{\AA}$, $D=1425\text{MHz}$, $E=-180\text{MHz}$, A_{zz} given by the formula 4.5

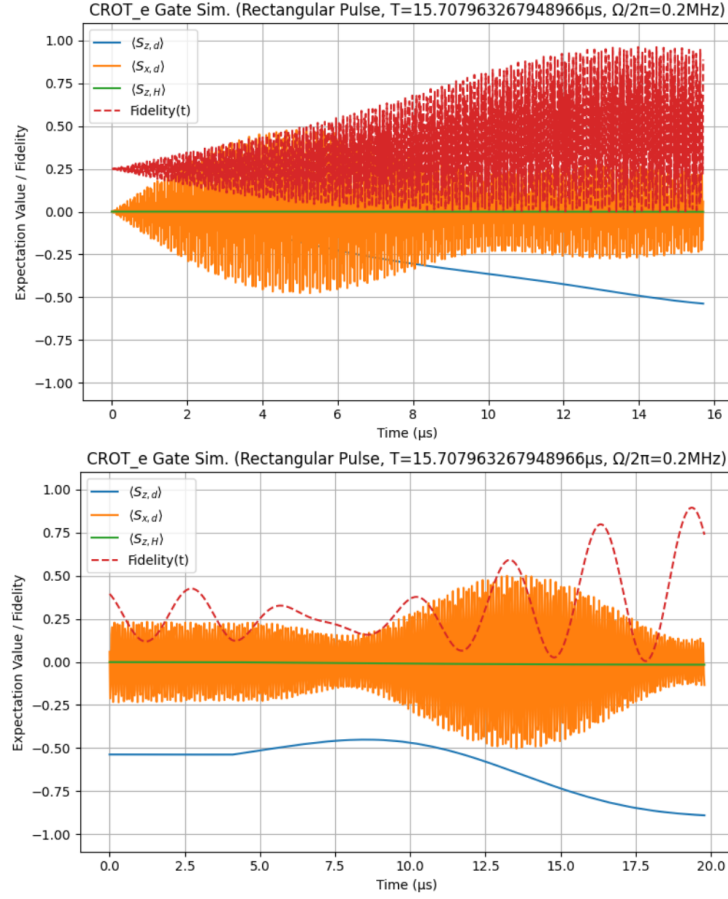


Figure 4.8: (upper) Time evolution for the first gate, final fidelity= 0.94. (lower) Time evolution for the second gate with initial free time evolution, final fidelity= 0.74. Parameters: $r=6\text{\AA}$, $D=1425\text{MHz}$, $E=-180\text{MHz}$, A_{zz} given by the DFT-computed value

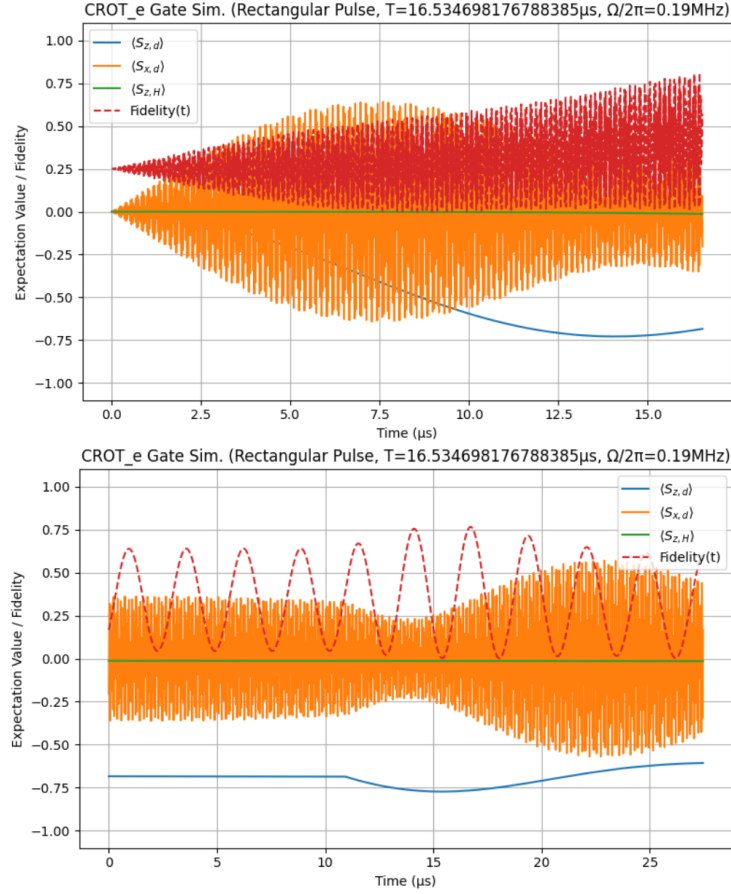


Figure 4.9: (upper) Time evolution for the first gate, final fidelity = 0.75. (lower) Time evolution for the second gate with initial free time evolution, final fidelity = 0.60. Parameters: $r=12\text{\AA}$, $D=1431\text{MHz}$, $E=-15\text{MHz}$, A_{zz} given by the formula 4.5

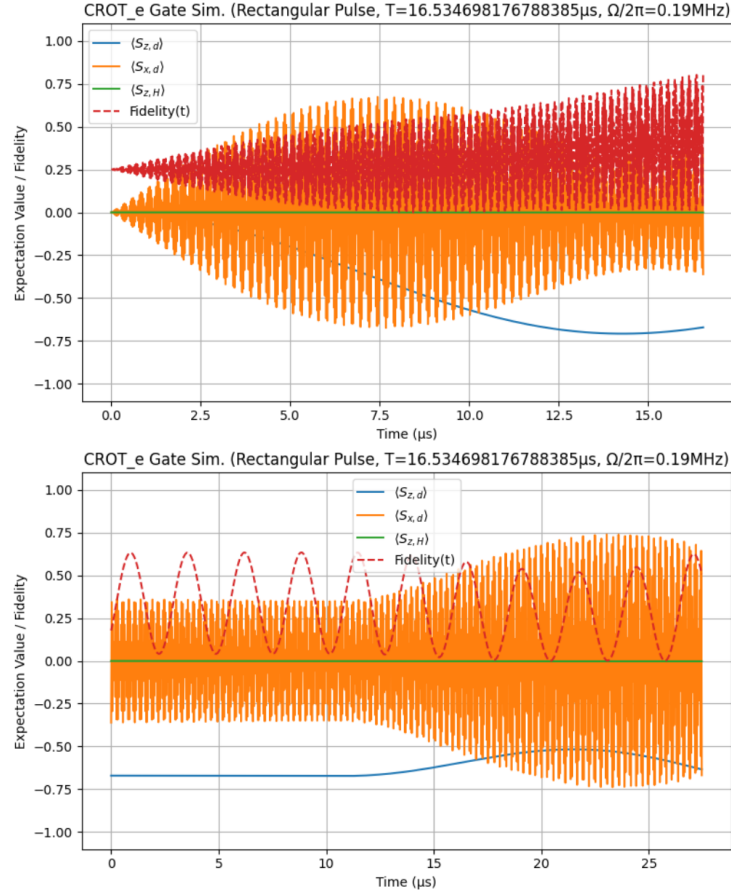


Figure 4.10: (upper) Time evolution for the first gate, final fidelity = 0.90. (lower) Time evolution for the second gate with initial free time evolution, final fidelity = 0.51. Parameters: $r=12\text{\AA}$, $D=1431\text{MHz}$, $E=-108\text{MHz}$, A_{zz} given by the formula 4.5

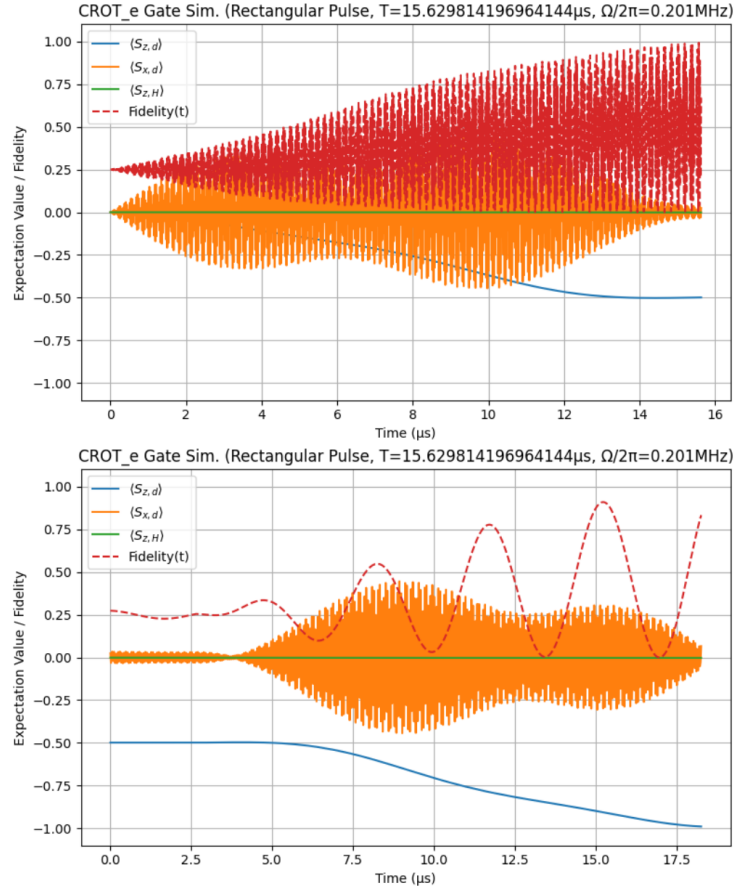


Figure 4.11: (upper) Time evolution for the first gate, final fidelity= 0.81. (lower) Time evolution for the second gate with initial free time evolution, final fidelity= 0.83. Parameters: $r=6\text{\AA}$, $D=1425\text{MHz}$, $E=-180\text{MHz}$, A_{zz} given by the DFT-computed value

Chapter 5

Data Management Process

Having established the nature of the data produced in this project, its scientific context, and the methods used for its generation in the previous chapters, this chapter details the specific data management processes implemented.

5.1 Data Management Plan

The foundation of the data management strategy for this project is the **Data Management Plan** (DMP). This document outlines the approach to handling research data throughout its lifecycle. The DMP was structured following the template provided by the NFFA-DI project for its instruments (available at [this link](#)), adapting it for the computational "instrument" used here (primarily involving QuTiP simulations). Key decisions documented in the DMP include:

- All relevant data and metadata produced during the project will be systematically collected.
- Raw and processed simulation data will be curated and published using the NOMAD platform.
- Data files will be .txt,.dat or .json.
- Any formal publications resulting from this data will, following peer review, be archived on a suitable repository like Zenodo, ensuring clear links to the underlying data (provenance).
- Other scientific outputs (e.g., presentations, code) will be disseminated using permissive open licenses and assigned Persistent Identifiers (PIDs) whenever feasible, subject to the specific publication venue and process.

- Long-term data storage and backup procedures will adhere to the policies established by OFED (details available in their [data policy](#)).

The **implementation of FAIR principles** within this project relies heavily on leveraging the NOMAD platform. As described previously in Section 2.3, NOMAD provides built-in functionalities that facilitate adherence to these principles.

5.2 Developing of a NOMAD Plug-in

As previously discussed, the NOMAD platform primarily caters to the material science community, and prior to this thesis, no dedicated plugin existed for quantum information simulation data, particularly for tools like QuTiP. Consequently, the development of such a plugin was initiated and is available on [GitHub](#)¹. This development effort significantly benefited from the support of the FAIRmat community². The plugin's structure is largely based on the existing *NOMAD-simulations* package, which was described in Section 2.3.

5.2.1 Metadata and Schemas

A key requirement was the development of schemas to capture metadata specific to quantum simulations, particularly concerning the Hamiltonian. In typical first-principles calculations within material science, determining the Hamiltonian or its parameters is often the primary goal. Conversely, in many quantum technology simulations, the Hamiltonian serves as a fundamental input defining the model. To address this, two new schema classes were implemented (see Listing A.2): *HamiltonianParameter* and *ModelHamiltonian*. The *HamiltonianParameter* class represents a specific parameter within the Hamiltonian definition (e.g. γ_e), while *ModelHamiltonian* class inherits from the standard *ModelMethod* simulation class and extends it by adding attributes to store the mathematical formula of the Hamiltonian (as a [LaTeX](#) string) and references to its constituent *HamiltonianParameter* instances.

Furthermore, as highlighted in the discussion of QuTiP (Section 4.1), dedicated schemas are necessary to represent the core QuTiP objects: *Qobj* (quantum states and operators) and *Result* (simulation outputs).

For the quantum object (*Qobj*), the developed schema (see Listing A.3) captures relevant metadata defining the object's quantum mechanical properties. It includes logic to redirect processing based on the specific type of

¹This plugin is distributed under the MIT open-source license.

²With particular thanks to the assistance provided by **Dr. E. B. Boydas** and **Dr. A. Albino**.

quantum object (e.g., state vector, density matrix, operator). From *Model-Method* it inherits the description attribute, as this class was designed for ab initio methods and no other attribute was fitting. Additionally, a normalizer was implemented for this schema to perform consistency checks on the data format and to infer missing metadata from the provided data where possible.

Within the **results schema** [A.4](#) solver statistics are handled by a dedicated *SolverStats* section. The *TimeEvolutionProperty* section, which inherits from *PhysicalProperty* and *PlotSection*, is designed to utilize the visualization capabilities of the [Plotly](#) library. It stores relevant quantities such as the expectation values, the names of the operators used to calculate them, and solver statistics from the *SolverStats* section. Based on this data, it automatically generates a plot displaying the expectation values against time.

Similarly, the *EigenvaluesInVariable* section is responsible for handling and plotting eigenvalues as a function of another variable [A.5](#). It stores the data for the independent variable and the corresponding calculated eigenvalues, automatically generating a plot that shows each set of eigenvalues as a distinct curve against the variable.

Parser

The accompanying parser plays a crucial role in processing the simulation data [A.6](#). It is designed to read the metadata and results stored in the JSON output file and map this information into the structured NOMAD archive sections defined by the schema. It is important for parser compatibility and file name coherence that the JSON file adheres to the naming convention ending in *.qutip.json*.

When processing quantum states and operators from the JSON file, the parser reads their structural metadata (such as dimensions and shape). If dimensions are provided, it often calculates the shape; if dimensions are missing, it uses the shape information present in the file.

Furthermore, the parser addresses a specific characteristic of QuTiP's data representation. Since QuTiP uses the generic type '*oper*' for density matrices, the parser identifies quantum objects intended as states but labelled as '*oper*' in the input file and correctly assigns them the '*dm*' (density matrix) type within the NOMAD archive.

5.3 Data management pipeline

The pipeline is just to use the developed NOMAD plugin within a local NOMAD Oasis installation, it can be added as a Git submodule using the following command in the Oasis root directory:

```
git submodule add https://github.com/GiovanniCastorina/nomad-  
plugin-qutip packages/nomad-plugin-qutip
```

Once added, data can be uploaded in JSON format (remembering *.qutip.json*), adhering to the schemas defined within the plugin.

To facilitate the creation of compliant JSON files, a simplified helper Python library, along with an example usage case, is provided in the following repository: [Thesis Code Repository](#).

Both the plugin and the helper library/example code are released under the MIT license. The helper library includes functions designed to generate JSON structures corresponding to the schemas defined in the nomad-plugin-qutip package. The provided example demonstrates the calculation and formatting of eigenvalues and time-evolution expectation values for a simple spin system described by the Hamiltonian:

$$H = -\gamma_e B_0 S_z - \gamma B_1 \cos(\omega t) S_x \quad (5.1)$$

5.4 Summary: challenges, solutions and future improvements

Implementing a robust data management framework for quantum simulation data, particularly in the absence of established community-wide standards or tools analogous to those in material science, presented significant challenges. To make the task tractable within the scope of this thesis, the decision was made to focus on defining schemas and processes primarily for the specific (meta)data relevant to the project's QuTiP-based simulations, rather than attempting a comprehensive, community-level solution. While this narrowed the scope and allowed for tailored schema development, it inherently resulted in a solution customized to this specific use case.

Despite these constraints, the combination of the NOMAD platform and the developed plugin successfully enabled a FAIR-by-design treatment for the (meta)data generated within this project.

However, several challenges remain, highlighting areas for future improvement:

A major limitation is the current reliance on manual processes. Researchers wishing to adopt this FAIR approach must manually construct *JSON* files conforming to the defined schemas. This is particularly cumbersome for elements like the Hamiltonian description, which needs to be manually specified for each new simulation setup. While the provided helper library aims to partially alleviate this by abstracting some JSON creation details, users still need to understand and interact with its defined functions. Developing more automated tools for metadata extraction and JSON generation would be highly beneficial. This could involve creating graphical user interfaces (GUIs) or more sophisticated scripts that parse simulation input/output files. Integrating functionality to directly upload data and metadata to NOMAD from computational environments like Jupyter Notebooks using libraries such as *nomad-lab* would also significantly enhance automation and usability.

Currently, the plugin's visibility is limited, relying primarily on direct communication ("word of mouth"). Wider recognition and easier adoption are needed for broader impact. Seeking wider dissemination within relevant research communities and potentially pursuing integration or official recognition within the main NOMAD ecosystem (perhaps as part of, or linked to, NOMAD-simulations) could dramatically improve accessibility and adoption.

The current plugin focuses on core QuTiP objects and essential Hamiltonian metadata. It could be significantly expanded to cover more functionalities and data types.

The plugin's schemas could be extended to capture more detailed metadata, such as incorporating checks and metadata fields for operator properties (e.g., unitarity, currently omitted) or supporting a wider range of QuTiP's time evolution solvers and analysis routines. Furthermore, extending support beyond QuTiP to other quantum simulation tools could broaden the plugin's applicability. Implementing these extensions would be feasible and would add considerable value, especially if done with a community minded approach by employing in the effort different researchers in the quantum technology and information field.

Chapter 6

Conclusions

This thesis implemented a FAIR-by-design strategy for managing quantum simulation data, specifically focusing on QuTiP simulations of divacancy defects in Silicon Carbide (SiC). The computational research phase produced substantial datasets encompassing quantum dynamics results and associated metadata generated under diverse simulation parameters.

A novel data management solution was developed through the creation of a dedicated NOMAD plugin. This represented the first formalized FAIR data initiative within the research group, utilizing defined schemas for the standardization, organization, and archiving of QuTiP simulation metadata.

Furthermore, this plugin, coupled with a Python helper library, enables partial automation of the research process by assisting in the generation of schema-compliant JSON files suitable for platforms like NOMAD, thereby enhancing the overall FAIR workflow.

However, critical limitations remain. The developed workflow is only partially automated. Furthermore, within the CNR-IMM group, spin dynamics simulations using QuTiP represent just one component of the diverse research techniques applied across quantum technologies, materials science, and other fields.

Consequently, this work constitutes an initial step in transforming the group's practices. It serves to encourage a shift in mindset across the entire group, moving towards research methodologies fully aligned with open science principles at the institute.

This thesis demonstrates the importance of open science and FAIR data management not as procedural burdens, but as opportunities to enhance research quality and output, improve the recognition of the scientific process,

and strengthen engagement with the international community.

Appendix A

Code

A.1 Eigenvalues finding

Listing A.1: Correct Eigenstates finding

```
1 B = 0.03 #T
2 r_vs = np.array([0, 0, 1]) # Spin directions vector
3
4 # === TOTAL interaction Hamiltonian, [1] is the divacancy
   and [0] is the nuclear spin.
5 H_electron = -gammaeMHz * B * opZSz[1] \
6               + D * (opZSz[1]**2 - cost*Id_total) \
7               + E * (opXSx[1]**2 - opYSy[1]**2)
8
9 H_zeeman = -gammaHMHz * B * opZSz[0]
10
11 #Interaction term
12 S_dot_r = (opXSx[1]*r_vs[0] + opYSy[1]*r_vs[1] + opZSz
13            [1]*r_vs[2])
14 I_dot_r = (opXSx[0]*r_vs[0] + opYSy[0]*r_vs[1] + opZSz
15            [0]*r_vs[2])
16 # Termine Sz*Iz (o S_z*I_z)
17 SzIz_term = opZSz[1] * opZSz[0]
18 H_interaction = cV2H5nm * (SzIz_term - 3.0 * S_dot_r *
19                             I_dot_r)
20
21 #TOTAL
22 Hc= H_electron + H_zeeman + H_interaction
23
24 eigenvalues, eigenstates = Hc.eigenstates()
```

```

22 #Target states for the Sx_d_eff transition (electronic
    part |e_0> <-> |e_m1>)
23 #We want the transition for |n1>
24 #and vice-versa
25 state_e0_n1 = qt.tensor(n1,e_0)
26 state_em1_n1 = qt.tensor(n1,e_m1)
27
28 state_e0_n0 = qt.tensor(n0,e_0)
29 state_em1_n0 = qt.tensor(n0,e_m1)
30
31 # Overlaps between complete eigenstates and target states
32 overlaps_e0_n1 = np.array([abs(eigenstate.dag() *
    state_e0_n1)**2 for eigenstate in eigenstates])
33 overlaps_em1_n1 = np.array([abs(eigenstate.dag() *
    state_em1_n1)**2 for eigenstate in eigenstates])
34
35 overlaps_e0_n0 = np.array([abs(eigenstate.dag() *
    state_e0_n0)**2 for eigenstate in eigenstates])
36 overlaps_em1_n0 = np.array([abs(eigenstate.dag() *
    state_em1_n0)**2 for eigenstate in eigenstates])
37 # Indexs with the higher overlaps
38 idx_e0_n1 = np.argmax(overlaps_e0_n1)
39 idx_em1_n1 = np.argmax(overlaps_em1_n1)
40
41 idx_e0_n0 = np.argmax(overlaps_e0_n0)
42 idx_em1_n0 = np.argmax(overlaps_em1_n0)
43
44 # Eigenvalues
45 E_level_01 = eigenvalues[idx_e0_n1] # Energy of the most
    similar to |e_0, n1>
46 E_level_m11 = eigenvalues[idx_em1_n1] # Energy of the
    most similar to |e_m1, n1>
47
48 E_level_00 = eigenvalues[idx_e0_n0] # Energy of the most
    similar to |e_0, n0>
49 E_level_m10 = eigenvalues[idx_em1_n0] # Energy of the
    most similar to |e_m1, n0>
50
51 # Rabi Excitations
52 RabiEx = np.abs(E_level_m11 - E_level_01) # Resonance
    frequency in MHz
53 RabiEx2 = np.abs(E_level_m10 - E_level_00) # Resonance
    frequency in MHz

```

A.2 Plugin Schemas

Listing A.2: Model Hamiltonian

```

1  class SpinHamiltonianParameter(ArchiveSection):
2      """Stores a single named parameter used in the
3      Spin Hamiltonian formula ."""
4      m_def = Section(
5          a_eln={'properties': SectionProperties(
6              label='Spin_Hamiltonian_Parameter',
7              order=['value', 'unit'])}
8      )
9
10     value = Quantity(
11         type=np.float64,
12         description="Numerical_value_of_the_parameter.",
13         a_eln=ELNAnnotation(
14             component=ELNComponentEnum.NumberEditQuantity
15             ,
16             label='Value'
17         )
18     )
19     unit = Quantity(
20         type=str,
21         description="Unit_of_the_parameter(e.g., 'MHz', 'Hz/T', 'J*s').",
22         a_eln=ELNAnnotation(
23             component=ELNComponentEnum.StringEditQuantity
24             ,
25             label='Unit'
26         )
27     )
28
29 class SpinHamiltonian(ModelMethod):
30     """
31     Describes the Spin Model Hamiltonian using a formula
32     and
33     parameters .
34     """
35     m_def = Section(
36         a_eln={'properties': SectionProperties(
37             label='Spin_Hamiltonian'
38         )}
39     )

```

```

37     # Human-readable name for the model
38     name = Quantity(
39         type=str,
40         description="Name of the Hamiltonian model (e.g.,
           'Heisenberg Model', 'Transverse Field Ising
           Model')."
41     )
42
43     formula = Quantity(
44         type=str,
45         description="""
46             The mathematical formula representing the
47             Hamiltonian, preferably using
48             LaTeX syntax.
49             """
49         a_eln=ELNAnnotation(component=ELNComponentEnum.
           StringEditQuantity, props={'render_value': '
           markdown'})
50     )
51
52     parameters = SubSection(
53         section_def=SpinHamiltonianParameter,
54         repeats=True,
55         description="List of parameters used in the
           Hamiltonian formula with their values.",
56     )

```

Listing A.3: Quantum object

```

1  class QuantumObject(ArchiveSection):
2      """
3      A framework-agnostic data container for storing
4      states/operators/superoperators.
5
6      The key fields track:
7      - dimensional structure ('dims', 'shape'),
8      - type of object ('ket', 'bra', 'oper', 'super', '
9      dm', etc.),
10     - numerical storage format (e.g., 'Dense', 'csr'),
11     - Hermiticity flag,
12     - the underlying data (complex matrix or vector).
13     """
14
15     # !!! dims might be a list of lists describing the

```

```

14         Hilbert space.
15         dims = Quantity(
16             type=np.int32,
17             shape=['*', '*'],
18             description=(
19                 """Tensor-product dimensions.
20                 For example, a single qubit ket might be
21                 [[2], [1]],
22                 or a 2-qubit system [[2,2],[1,1]]."""
23             ),
24         )
25
26         shape = Quantity(
27             type=np.int32,
28             shape=['*'],
29             description=(
30                 """Shape of the object's matrix or vector
31                 representation.
32                 For a 2-level ket, this might be [2,1].
33                 For a 2x2 operator, [2,2]."""
34             ),
35         )
36
37         type = Quantity(
38             type=MEnum('ket', 'bra', 'oper', 'super', 'dm'),
39             description=(
40                 """Logical type of the quantum object:
41                 - 'ket' for state vectors,
42                 - 'bra' for dual vectors,
43                 - 'oper' for operators,
44                 - 'super' for superoperators,
45                 - 'dm' for density matrices."""
46             ),
47         )
48
49         storage_format = Quantity(
50             type=str,
51             description=(
52                 """String label for the numerical storage
53                 format
54                 (e.g., 'Dense', 'csr', 'ELL')."""
55             ),
56         )

```

```

53
54     is_hermitian = Quantity(
55         type=bool,
56         description="""True if this object is Hermitian (
                    self-adjoint).
                    (only relevant for 'oper'/'dm').""",
57     )
58
59
60     data = Quantity(
61         type=np.complex128,
62         shape=['*', '*'],
63         description=(
64             """The underlying array or matrix
                    representing this quantum object.
                    TODO: sparse matrix representations."""
65         ),
66     ),
67 )
68
69 def normalize(self, archive, logger) -> None:
70     super().normalize(archive, logger)
71
72     # If data is present, do shape checks
73     if self.data is not None:
74         # Convert data.shape to a np.array of type
75         # int
76         data_shape = np.array(self.data.shape, dtype=
77             np.int32) # e.g., [2, 2]
78
79         # If shape is None or has zero elements, fill
80         # from data
81         if self.shape is None or self.shape.size ==
82         0:
83             self.shape = data_shape
84             logger.info(f"Filling 'shape' from data:
85                 {self.shape}")
86
87         # Compare with data_shape using np.
88         # array_equal
89         elif not np.array_equal(self.shape,
90             data_shape):
91             logger.warning(
92                 f'Inconsistent shape {self.shape.
93                     tolist()} vs data {data_shape.
94                     tolist()};'

```

```

85         'using_data_shape.')
86         self.shape = data_shape
87
88         # If shape is now set but dims is not, try to
89         infer
90         # (Check shape.size and shape[0], shape[1], etc.
91         carefully)
92         EXPECTED_SHAPE_SIZE = 2
93
94         if (
95             self.shape is not None
96             and self.shape.size == EXPECTED_SHAPE_SIZE # e.g
97             . shape=[m, n] and (self.dims is None or self.
98             dims.size == 0)
99         ):
100             m, n = self.shape
101             if self.type == 'ket':
102                 self.dims = np.array([[m], [1]], dtype=np.
103                     int32)
104                 logger.info(f"Inferred 'dims' for ket: {self.
105                     dims}")
106             elif self.type == 'bra':
107                 self.dims = np.array([[1], [n]], dtype=np.
108                     int32)
109             elif self.type in ('oper', 'dm'):
110                 self.dims = np.array([[m], [n]], dtype=np.
111                     int32)
112             elif self.type == 'super':
113                 # Could guess dims differently, but here's a
114                 simple approach
115                 self.dims = np.array([[m], [n]], dtype=np.
116                     int32)
117
118         # If type is 'oper' or 'dm' and is_hermitian not
119         set, check from data
120         if (
121             self.type in ('oper', 'dm')
122             and self.data is not None
123             and self.is_hermitian is None
124         ):
125             # Compare data and data.conjugate().T
126             if np.allclose(self.data, self.data.conjugate
127                 ()).T):

```

```

116         self.is_hermitian = True
117     else:
118         self.is_hermitian = False
119         logger.info(f"Set 'is_hermitian' to {self.
120                     is_hermitian} based on data.")
121
122 class QuantumSystem(ModelSystem):
123     """
124     A specialized 'model system' for quantum information
125     simulations,
126     e.g., representing a set of qubits, spins, or multi-
127     level systems.
128     Inherits from ModelSystem to remain compatible with
129     existing archiving.
130     """
131
132     name = Quantity(type=str, description='Optional label
133                     for this quantum system.')
134
135 class QuantumOperator(ArchiveSection):
136     """
137     A container for quantum operators, referencing a Qobj
138     or storing
139     additional operator info. Not strictly required to
140     inherit from ModelMethod.
141     """
142
143     name = Quantity(
144         type=str, description="Label for the operator (e.
145                               g. 'Hamiltonian', 'sigmaz')."
146     )
147
148     quantum_object = SubSection(
149         sub_section=QuantumObject.m_def,
150         repeats=True,
151         description='The underlying quantum object (
152                     operator form).',
153     )
154
155 class QuantumState(ArchiveSection):

```

```

150     """
151     A container for wavefunction or density-matrix states
152     .
153     Maybe add matrix-states
154     """
155     label = Quantity(type=str, description='Optional_
156                     label_for_this_quantum_state.')
157
158     quantum_object = SubSection(
159         sub_section=QuantumObject.m_def,
160         repeats=False,
161         description='The_underlying_quantum_object_
162                     representing_this_state.',
163     )

```

Listing A.4: Result schema

```

1 class SolverStats(ArchiveSection):
2     m_def = Section(
3         a_eln=dict(lane_width='400px'),
4         description="Statistics_related_to_the_numerical_
5                     solver_execution."
6     )
7     solver_name = Quantity(
8         type=str,
9         description="Name_of_the_solver_used(e.g.,_
10                     sesolve',_'mesolve')."
11     )
12     method = Quantity(
13         type=str,
14         description="Specific_algorithm_or_method_used_by_
15                     the_solver(e.g.,_'scipy_zvode_adams')."
16     )
17     solver_description = Quantity(
18         type=str,
19         description="Additional_description_provided_by_
20                     the_solver(e.g.,_'Schrodinger_Evolution')."
21     )
22     initialization_time = Quantity(
23         type=np.float64,
24         unit='second',
25         description="Time_taken_for_solver_initialization_
26                     ."

```

```

22     )
23     preparation_time = Quantity(
24         type=np.float64,
25         unit='second',
26         description="Time taken for solver preparation steps."
27     )
28     run_time = Quantity(
29         type=np.float64,
30         unit='second',
31         description="Time taken for the main solver execution run."
32     )
33     time_interval_start = Quantity(
34         type=np.float64,
35         description="Start of the simulation time interval."
36     )
37     time_interval_end = Quantity(
38         type=np.float64,
39         description="End of the simulation time interval."
40     )
41     n_steps = Quantity(
42         type=np.int32,
43         description="Number of time steps in the simulation interval."
44     )
45     n_expectation_operators = Quantity(
46         type=np.int32,
47         description="Number of expectation value operators (e_ops) tracked."
48     )
49     final_state_saved = Quantity(
50         type=bool,
51         description="Flag indicating if the final state of the evolution was saved in the result object."
52     )
53
54 class TimeEvolutionProperty(PhysicalProperty, PlotSection):
55     """
  
```

```

56     A physical property representing the time evolution
      of a quantum state.
57     Expected 'value' shape: [n_time_steps,
      state_dimension]
58     """
59     m_def = Section(a_eln=dict(properties=
      SectionProperties(label='Time_Evolution_Results'))
60 )
61
62     value = Quantity(
63         type=np.float64,
64         shape=['*', '*'],
65         description=(
66             "Expectation values vs time. Rows correspond
              to time steps, "
67             "columns correspond to different expectation
              operators (e_ops). "
68         )
69     )
70
71     e_ops_names = Quantity(
72         type=str,
73         shape=['*'], # Lista of strings
74         description="Names/labels of the expectation
              value operators (e_ops). "
75     )
76
77     solver_statistics = SubSection(
78         section_def=SolverStats,
79         description="Statistics related to the solver
              execution for this time evolution. "
80     )
81
82     def plot_expectation_values(self):
83         if self.value is None or self.value.size == 0:
84             self.logger.warning("No expectation value
              data available to plot.")
85             return None
86         if self.value.ndim != 2:
87             # Could be one expectation value
88             if self.value.ndim == 1:
89                 self.value = self.value.reshape(-1, 1)

```

```

90         else:
91             raise ValueError("Value must be a 1D or 2D array (n_time_steps, [n_e_ops]) for plotting.")
92
93     time = None
94     t_label = 'Index'
95     if self.variables:
96         var_time = self.variables[0] # Assume time is first variable
97         if hasattr(var_time, 'get_values') and var_time.value is not None:
98             time = var_time.get_values()
99             t_label = f"{var_time.name_or_ 'Time'}"
100             if var_time.unit:
101                 t_label += f"_{var_time.unit}"
102         elif isinstance(var_time.value, (np.ndarray, list)) and len(var_time.value) == self.value.shape[0]:
103             time = var_time.value
104             t_label = f"{var_time.name_or_ 'Time'}"
105             if var_time.unit:
106                 t_label += f"_{var_time.unit}"
107
108     if time is None:
109         time = np.arange(self.value.shape[0])
110
111     y_label = 'Expectation Value'
112     plot_title = f"{self.name_or_ 'Expectation Values'} vs. {t_label.split('(')[0].strip()}"
113
114     # fig = px.line(x=time, y=self.value, labels={'x': t_label, 'y': y_label},
115     #               title=plot_title)
116
117     fig = px.line(
118         x=time,
119         y=self.value.T, # transpose so each expectation operator is a separate trace
120         labels={'x': t_label, 'y': y_label},
121         title=plot_title
122     )
123

```

```

124     num_operators = self.value.shape[1]
125     operator_names_available = (
126         self.e_ops_names is not None and
127         len(self.e_ops_names) == num_operators
128     )
129
130     if num_operators > 1:
131         fig.update_layout(showlegend=True)
132         for i, trace in enumerate(fig.data):
133             if operator_names_available:
134                 # Use name when provided
135                 trace.name = self.e_ops_names[i]
136             else:
137                 # Fallback
138                 trace.name = f'&lt;Op_{i}&gt;';
139     else:
140         fig.update_layout(showlegend=False)
141
142     self.figures.append(PlotlyFigure(label='
Expectation_Value_Plot', figure=fig.
to_plotly_json()))
143     return fig
144
145     def normalize(self, archive, logger):
146         super().normalize(archive, logger)
147         try:
148             self.plot_expectation_values()
149         except Exception as e:
150             logger.error(f"Failed to plot expectation
values for '{self.name}': {e}")

```

Listing A.5: Eigenvalues plot schema

```

1 class EigenvaluesInVariable(PhysicalProperty, PlotSection
2 ):
3     """
4     A physical property representing energy eigenvalues
5     as a function of an external variable.
6     Expected 'value' shape: [n_points, n_eigenvalues]
7     """
8     m_def = Section(
9         a_eln=dict(properties=SectionProperties(label='
Eigenvalues_vs_Variable')),)
10     value = Quantity(

```

```

9         type=np.float64,
10        shape=['*', '*'],
11        description="Matrix of eigenvalues. Rows
           correspond to variable points and columns to
           eigenvalues."
12    )
13
14    def plot_eigenvalues(self):
15        # Ensure value is 2D
16        if self.value is None or self.value.size == 0:
17            self.logger.warning("No eigenvalue data
           available to plot.")
18            return None
19        if self.value.ndim != 2:
20            raise ValueError("Value must be a 2D array (
           n_points, n_eigenvalues) for eigenvalue
           plotting.")
21        # Retrieve the x-axis from the first variable if
           available
22        x = None
23        x_label = 'Index'
24        if self.variables:
25            var1 = self.variables[0]
26            if hasattr(var1, 'get_values') and var1.value
           is not None:
27                x = var1.get_values()
28                x_label = f"{var1.name or 'Variable'}"
29                if var1.unit:
30                    x_label += f" ({var1.unit})"
31            elif isinstance(var1.value, (np.ndarray, list
           )) and len(var1.value) == self.value.shape
           [0]:
32                x = var1.value # Fallback
33                x_label = f"{var1.name or 'Variable'}"
34                if var1.unit:
35                    x_label += f" ({var1.unit})"
36
37        # Generate a line plot where each eigenvalue is a
           separate curve
38        plot_title = f"{self.name or 'Eigenvalues'} vs. {
           x_label.split('(')[0].strip()}"
39        # fig = px.line(x=x, y=self.value, labels={'x':
           x_label, 'y': 'Eigenvalue'},

```

```

40         #                                     title=plot_title)
41     fig = px.line(
42         x=x,
43         y=self.value.T, # transpose so each column
                        # becomes its own trace
44         labels={'x': x_label, 'y': 'Eigenvalue'},
45         title=plot_title
46     )
47     # More eigenvalues labels
48     if self.value.shape[1] > 1:
49         fig.update_layout(showlegend=True)
50         for i, trace in enumerate(fig.data):
51             trace.name = f'E_{i}'
52
53
54     # Append the figure to the PlotSection's figures
                        # list
55     self.figures.append(PlotlyFigure(label='
                        Eigenvalues_Plot', figure=fig.to_plotly_json()
56     ))
57     return fig
58
59     def normalize(self, archive, logger):
60         super().normalize(archive, logger)
61         try:
62             self.plot_eigenvalues()
63         except Exception as e:
64             logger.error(f"Failed to plot eigenvalues for
                        '{self.name}': {e}")

```

A.3 Parsers

Listing A.6: Parsers

```

1 class JSONParser(MappingParser):
2     """
3     A minimal JSON-based MappingParser, analogous to
                        XMLParser but for JSON.
4     """
5
6     value_key = '__value'
7     attribute_prefix = '@'

```

```

8
9     def load_file(self):
10         with open(self.filepath) as f:
11             self.data_object = json.load(f)
12         return self.data_object
13
14     def to_dict(self, **kwargs) -> dict:
15         """
16         Return a dictionary representation of data_object
17         . If the file is
18         already JSON, data_object may simply be a dict,
19         so just return it.
20         """
21         if isinstance(self.data_object, dict):
22             return self.data_object
23         return {}
24
25     def from_dict(self, data: dict, **kwargs):
26         """
27         Set the parser's internal data_object from a
28         given dictionary.
29         """
30         self.data_object = data
31
32     def get_program(self, source: dict[str, Any], **
33         kwargs) -> dict[str, Any]:
34         return source.get('program', {'name': '', '
35             version': ''})
36
37     def get_system(self, source: dict[str, Any], **kwargs
38         ) -> dict[str, Any]:
39         return {'name': source.get('simulation_name', '
40             default')}
41
42     def get_operators(self, source: dict, **kwargs) ->
43         dict:
44         """
45         Process operator data from the JSON.
46
47         For each operator, it returns a dictionary with
48         two keys:
49         - "name": the operator name (string)
50         - "quantum_object": a dictionary with fields

```

```

42         required by QuantumObject:
43             "dims", "shape", "type", "storage_format", "
44             is_hermitian", and "data".
45
46     In particular, if the operator data contains a "
47     matrix" key
48     (with "re" and "im"),
49     these are combined into a single complex matrix
50     stored under "data".
51
52     The function returns a dictionary with a single
53     key "quantum_operators"
54     whose value is the list.
55     """
56
57     ops = source.get('operators', {})
58     processed = []
59     for op_name, op in ops.items():
60         qobj = {}
61         if 'dims' in op:
62             qobj['dims'] = op['dims']
63             try:
64                 # Assume dims is of the form [[n],[m]] and derive shape as [n, m]
65                 dimension = 2
66                 if len(op['dims']) == dimension:
67                     qobj['shape'] = [int(op['dims']
68                                         ][0][0]), int(op['dims']
69                                         ][1][0])]
70                 else:
71                     qobj['shape'] = []
72             except Exception:
73                 qobj['shape'] = []
74             qobj['type'] = op.get('type', 'oper')
75             qobj['storage_format'] = op.get('
76             storage_format', 'Dense')
77             qobj['is_hermitian'] = op.get('is_hermitian',
78             True)
79             if 'matrix' in op:
80                 matrix_dict = op['matrix']
81                 re = np.array(matrix_dict.get('re', []))
82                 im = np.array(matrix_dict.get('im', []))
83                 qobj['data'] = (re + 1j * im).tolist()
84             else:

```

```

75         qobj['data'] = op.get('data', None)
76         processed.append({'name': op_name, '
           quantum_object': qobj})
77     return {'quantum_operators': processed}
78
79
80     def get_states(self, source: dict, **kwargs) -> dict:
81         """
82         Process state data from the JSON source.
83         Looks for a 'states' key containing a dictionary
84           of states.
85         Filters states explicitly marked with type 'ket'
86           or 'bra'.
87         If they are marked as 'oper' then counts it as a
88           density matrix ('dm').
89         Returns a dictionary {'quantum_states': [list of
90           filtered state dicts]}.
91         """
92         # Get the dictionary of states from JSON
93         states_source = source.get('states', {})
94         processed_states = []
95
96         # Iterate through each state found JSON
97         for state_label, state_data in states_source.
98           items():
99             # Extract the quantum object data
100
101             qobj_source = state_data.get('quantum_object',
102               , state_data)
103
104             # Process the quantum object data
105             qobj_target = self.
106               _process_quantum_object_data(qobj_source)
107
108             # Check the 'type' and go if is ket or bra or
109               oper
110             state_type = qobj_target.get('type')
111             # the throws away all data populated by
112               _process_quantum_objec_data method.
113             # hence commented out
114             #qobj_target = {}
115             if state_type == 'ket' or state_type == 'bra',

```

```

108         :
109         if 'dims' in qobj_source:
110             qobj_target['dims'] = qobj_source['
111                 dims']
112             try:
113                 dim0 = np.prod(qobj_source['dims'
114                     ][0]) if qobj_source['dims'
115                         ][0] else 1
116                 dim1 = np.prod(qobj_source['dims'
117                     ][1]) if qobj_source['dims'
118                         ][1] else 1
119                 qobj_target['shape'] = [int(dim0)
120                     , int(dim1)]
121             except Exception as e:
122                 if self.logger:
123                     self.logger.warning(f"State_
124                         '{state_label}'_(ket/bra):
125                         _Could_not_determine_shape
126                         _from_dims_{qobj_source['
127                             dims']}:{e}")
128                 qobj_target['shape'] = []
129             else:
130                 qobj_target['shape'] = qobj_source.
131                     get('shape', [])
132
133             #qobj_target['type'] = source_type # Keep
134             #original type for ket and bra
135             qobj_target['type'] = state_type
136
137             qobj_target['storage_format'] =
138                 qobj_source.get('storage_format', '
139                     Dense')
140
141             qobj_target['is_hermitian'] = qobj_source
142                 .get('is_hermitian', None)
143             processed_states.append({
144                 'label': state_label,
145                 'quantum_object': qobj_target
146             })
147
148         elif state_type == 'oper':
149             if 'dims' in qobj_source:
150                 qobj_target['dims'] = qobj_source['

```

```

135         dims']
136     try:
137         dim0 = np.prod(qobj_source['dims']
138                        ][0]) if qobj_source['dims']
139                        ][0] else 1
140         dim1 = np.prod(qobj_source['dims']
141                        ][1]) if qobj_source['dims']
142                        ][1] else 1
143         qobj_target['shape'] = [int(dim0)
144                                , int(dim1)]
145     except Exception as e:
146         if self.logger:
147             self.logger.warning(f"State_
148                                '{state_label}'_
149                                (oper):_
150                                Could_not_determine_shape_
151                                from_dims_{qobj_source['
152                                dims']}:{e}")
153         qobj_target['shape'] = []
154     else:
155         qobj_target['shape'] = qobj_source.
156         get('shape', [])
157         #Changing to 'dm'
158         qobj_target['type'] = 'dm'
159
160         qobj_target['storage_format'] =
161         qobj_source.get('storage_format', '
162         Dense')
163         qobj_target['is_hermitian'] = qobj_source
164         .get('is_hermitian', None)
165         if 'matrix' in state_data:
166             matrix_dict = state_data['matrix']
167             re = np.array(matrix_dict.get('re',
168             []))
169             im = np.array(matrix_dict.get('im',
170             []))
171             qobj_target['data'] = (re + 1j * im).
172             tolist()
173         else:
174             qobj_target['data'] = state_data.get(
175             'data', None)
176         processed_states.append({
177             'label': state_label,
178             'quantum_object': qobj_target

```

```

160         })
161
162
163     else:
164         # If the type is not 'ket', 'bra' or 'oper
165         ' we skip this entry
166         if self.logger:
167             self.logger.debug(
168                 f"Skipping state '{state_label}'
169                 during parsing because its
170                 type"
171                 f"is '{state_type}' (expected '
172                 ket', 'bra' or 'oper')."
173             )
174
175     return {'quantum_states': processed_states}
176
177 def get_outputs(self, source: dict, **kwargs) -> list
178 [dict]:
179     processed_outputs = []
180     if not isinstance(source, dict):
181         return processed_outputs
182
183     for result_key, result_data in source.items():
184         if not isinstance(result_data, dict):
185             continue
186
187         calc_type = result_data.get('calculation_type
188         ')
189
190         # Dict for EigenvaluesInVariable
191         if calc_type == 'eigenvalues':
192             output_dict = {}
193             output_dict['__section_def'] =
194                 EigenvaluesInVariable.m_def
195             output_dict.update(result_data)
196             processed_outputs.append(output_dict)
197
198         # Dict for TimeEvolutionProperty
199         elif calc_type == 'time_evolution':
200             output_dict = {}
201             output_dict['__section_def'] =
202                 TimeEvolutionProperty.m_def
203             output_dict.update(result_data)

```

```
195         processed_outputs.append(output_dict)
196
197     else:
198         if self.logger:
199             self.logger.warning(f"Skipping result
200                                '{result_key}' with unknown "
201                                f"calculation_type: '{
202                                    calc_type}'")
201
202     return processed_outputs
```

Bibliography

- [1] UNESCO, “UNESCO OPEN SCIENCE TOOLKIT,” 2023.
<https://www.unesco.org/open-science>. Collection of resources (guides, policy briefs, factsheets and indexes) designed to support implementation of the UNESCO Recommendation on Open Science.
- [2] E. Giglia, “Introduction to open science - open science why, what, how,” 2024. <https://zenodo.org/doi/10.5281/zenodo.13831946>.
- [3] M. de Luca, “Introduction to open science – fair data management and principles,” 2024.
<https://zenodo.org/doi/10.5281/zenodo.13839535>.
- [4] W. et al., “The fair guiding principles for scientific data management and stewardship,” *Scientific Data* **3** no. 1, (Mar., 2016) 9.
<http://dx.doi.org/10.1038/sdata.2016.18>.
- [5] S. et al., “Open hardware solutions in quantum technology,” 2023.
<https://arxiv.org/abs/2309.17233>.
- [6] G. Milburn, *Schrödinger’s Machines: The Quantum Technology Reshaping Everyday Life*. W.H. Freeman, 1997.
<https://books.google.it/books?id=ONECzgEACAAJ>.
- [7] J. P. Dowling and G. J. Milburn, “Quantum technology: The second quantum revolution,” 2002.
<https://arxiv.org/abs/quant-ph/0206091>.
- [8] S. et al., “Developing silicon carbide for quantum spintronics,” May, 2020. <http://dx.doi.org/10.1063/5.0004454>.
- [9] H. Ou, “Silicon carbide, the next-generation integrated platform for quantum technology,” Aug., 2024.
<http://dx.doi.org/10.1038/s41377-024-01515-0>.

- [10] M. J. Everitt and S. Ji, “Model cards for quantum technologies reporting,” 2024. <https://arxiv.org/abs/2412.13151>.
- [11] M. de Luca, “Introduction to open science - research data management,” 2024. <https://zenodo.org/doi/10.5281/zenodo.13839500>.
- [12] M. de Luca and S. COZZINI, “Pilot training course "master in data management and curation" mdmc,” 2024. <https://zenodo.org/doi/10.5281/zenodo.13781458>.
- [13] T. T. W. Community, “The turing way: A handbook for reproducible, ethical and collaborative research,” 2025. <https://zenodo.org/doi/10.5281/zenodo.15213042>.
- [14] N. Bongaerts and S. Della Chiesa, “Research data management lifecycle,” 2022. <https://zenodo.org/record/6602006>.
- [15] J. et al., “Fair principles: Interpretations and implementation considerations,” *Data Intelligence* **2** no. 1–2, (Jan., 2020) 10–29. http://dx.doi.org/10.1162/dint_r_00024.
- [16] C. H. et al., “Fair principles for research software (fair4rs principles),” 2021. <https://zenodo.org/record/6623556#.YqCJTJNBwlw>.
- [17] G. et al., “Bridging eresearch infrastructure and experimental materials science process in the quantum data hub,” 2024. <https://arxiv.org/abs/2405.19706>.
- [18] A. ALBINO and A. Ilyas, “On-boarding of new users from the overarching fair ecosystem for data (ofed) project for the customization of nomad oasis,” 2025. <https://zenodo.org/doi/10.5281/zenodo.14677026>.
- [19] F. Bazzocchi and C. Zagni, “Tools for data management and curation - using and interacting with nomad,” 2024. <https://zenodo.org/doi/10.5281/zenodo.13955620>.
- [20] Z. et al., “Enhancing quantum sensing sensitivity by a quantum memory,” *Nature Communications* **7** no. 1, (Aug., 2016) 11. <http://dx.doi.org/10.1038/ncomms12279>.
- [21] R. Shankar, *Principles of quantum mechanics*. Springer, New York, NY, Dec., 2013.

- [22] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, June, 2012. <http://dx.doi.org/10.1017/CB09780511976667>.
- [23] K. Capelle, “A bird’s-eye view of density-functional theory,” 2002. <https://arxiv.org/abs/cond-mat/0211443>.
- [24] J. Cai, A. Retzker, F. Jelezko, and M. B. Plenio, “A large-scale quantum simulator on a diamond surface at room temperature,” *Nature Physics* **9** no. 3, (Jan., 2013) 168–173. <http://dx.doi.org/10.1038/NPHYS2519>.
- [25] I. Takács and V. Ivády, “Accurate hyperfine tensors for solid state quantum applications: case of the nv center in diamond,” *Communications Physics* **7** no. 1, (June, 2024) 6. <http://dx.doi.org/10.1038/s42005-024-01668-9>.
- [26] E. et al., “Evolution of quantum spin sensing: From bench-scale odmr to compact integrations,” *APL Materials* **12** no. 4, (Apr., 2024) 13. <http://dx.doi.org/10.1063/5.0193350>.
- [27] D. et al., “The nitrogen-vacancy colour centre in diamond,” *Physics Reports* **528** no. 1, (July, 2013) 1–45. <http://dx.doi.org/10.1016/j.physrep.2013.02.001>.
- [28] P. et al., “Design and integration of single-qubit rotations and two-qubit gates in silicon above one kelvin,” *Communications Materials* **3** no. 1, (Nov., 2022) 7. <http://dx.doi.org/10.1038/s43246-022-00304-9>.
- [29] J. et al., “Observation of coherent oscillation of a single nuclear spin and realization of a two-qubit conditional quantum gate,” *Physical Review Letters* **93** no. 13, (Sept., 2004) 4. <http://dx.doi.org/10.1103/PhysRevLett.93.130501>.
- [30] S. Castelletto and A. Boretti, “Silicon carbide color centers for quantum applications,” *Journal of Physics: Photonics* **2** no. 2, (Mar, 2020) 022001. <https://dx.doi.org/10.1088/2515-7647/ab77a2>.
- [31] T. Kimoto and J. A. Cooper, *Fundamentals of silicon carbide technology*. Wiley - IEEE. John Wiley & Sons, Nashville, TN, Sept., 2014.

- [32] D. Simin, F. Fuchs, H. Kraus, A. Sperlich, P. G. Baranov, G. V. Astakhov, and V. Dyakonov, “High-precision angle-resolved magnetometry with uniaxial quantum centers in silicon carbide,” *Physical Review Applied* **4** no. 1, (July, 2015) 8.
<http://dx.doi.org/10.1103/PhysRevApplied.4.014009>.
- [33] T. Fazio, I. Deretzis, G. Fisicaro, E. Paladino, and A. La Magna, “Stability and decoherence analysis of the silicon vacancy in 3c-SiC,” *Phys. Rev. A* **109** (Feb, 2024) 022603.
<https://link.aps.org/doi/10.1103/PhysRevA.109.022603>.
- [34] Z. et al., “First-principles investigation of near-surface divacancies in silicon carbide,” *Nano Letters* **23** no. 24, (Dec., 2023) 11453–11460.
<http://dx.doi.org/10.1021/acs.nanolett.3c02880>.
- [35] G. Calogero, I. Deretzis, G. Fisicaro, D. Ricciarelli, R. G. Viglione, and A. La Magna, “Tailoring nuclear spins order with defects: a quantum-tcad study,” 2025. <https://arxiv.org/abs/2503.19675>.
- [36] J. R. Johansson, P. D. Nation, and F. Nori, “Qutip 2: A python framework for the dynamics of open quantum systems,” 2012.
<https://arxiv.org/abs/1211.6518>.