Doctorate of Philosophy

Schrödinger's Catwalk

Machine learning methods to distill models of quantum systems

Brian Flynn

University of Bristol

August, 2021

ABSTRACT

Quantum technologies exploit quantum mechanical processes to achieve outcomes beyond the reach of classical machinery. One of their most promising applications is quantum simulation, whereby particles, atoms and molecules can be examined thoroughly for the first time, having been beyond the scope of even the most powerful supercomputers.

Models have been useful tools in understanding physical systems: these are mathematical structures encoding physical interactions, which allow us to predict how the system will behave under various conditions. Models of quantum systems are particularly difficult to design and test, owing to the huge computational resources required to represent them accurately. In this thesis, we introduce and develop an algorithm to characterise quantum systems efficiently, by inferring a model consistent with their observed dynamics. The *Quantum Model Learning Agent* (QMLA) is an extensible framework which permits the study of any quantum system of interest, by combining quantum simulation with state of the art machine learning. QMLA iteratively proposes candidate models and trains them against the target system, finally declaring a single model as the best representation for the system of interest.

We describe QMLA and its implementation through open source software, before testing it under a series of physical scenarios. First, we consider idealised theoretical systems in simulation, verifying the core principles of QMLA. Next, we incorporate strategies for generating candidate models by exploiting the information QMLA has gathered to date; by incorporating a genetic algorithm within QMLA, we explore vast spaces of valid candidate models, with QMLA reliably identifying the precise target model. Finally, we apply QMLA to *realistic* quantum systems, including operating on experimental data measured from an electron spin in a nitrogen vacancy centre.

QMLA is shown to be effective in all cases studied in this thesis; however, of greater interest is the platform it provides for examining quantum systems. QMLA can aid engineers in configuring experimental setups, facilitate calibration of near term quantum devices, and ultimately enable complete characterisation of natural quantum structures. This thesis marks the beginning of a new line of research, into automating the understanding of quantum mechanical systems.

DECLARATION OF AUTHORSHIP

This dissertation is submitted to the University of Bristol in accordance with the requirements for award of the degree of Doctorate of Philosophy in the Faculty of Science.

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's *Regulations and Code of Practice for Research Degree Programmes* and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

Signed:

_____

Date:

_____

Word count: $\sim 56,000$

ACKNOWLEDGEMENTS

I was never one of those people destined for a PhD. I don't think I knew what a PhD was until halfway through my undergraduate studies, by which time I was resigned to being tapped on the shoulder one day and asked to leave the campus, once they realised I had been faking it all along. So, at the risk of bragging, I am proud of this thesis, and the work it took to get here.

When you set out to write a thesis, you are constantly reminded that no one will ever *actually* read it, and you must therefore decide how much time and effort to spend on a document to gather dust on the electronic shelves of the university's digital library. I spent more time than most – not for the sake of examiners, supervisors or hypothetical future students, but for myself, as evidence that I can produce good work when I really try.

In the following 200 pages, I list in (sometimes boring) detail almost everything I've learned in the last four and a half years. I've learned something else important: the people around matter, and you need to tell them that once in a while. Here I just want to mention some people that have helped me along the way, or have been kind enough to share their time with me in some capacity. If the world is lucky, I will never again write anything this long, so I will have to write a lifetime worth of acknowledgements here. I will start in Bristol and work my way home.

The first person to thank has to be Raf, since this entire thesis builds on his initial idea. Beyond giving me the base material to work on for my PhD, he has also given me a huge amount of help and support over the last few years, along with friendship. His passion for machine learning is matched only by his passion for everything else; he is endlessly interested in all facets of science. I hope to live up to his example.

Andreas was equally crucial to any achievements I can claim. Through countless late night messages, he guided everything about this work that succeeded. Several times I thought I had outwitted him, and understood a concept more than him – until a few hours later when I realised I was wrong and he had explained it to me days before. Thanks for sticking with it.

To Anthony and the rest of the group, thanks for the wisdom, stories and beers. The wider *QETLabs* group is a real privilege to have been part of – it's been inspiring and terrifying to be surrounded by such talented people. Thanks to Sebastian for carefully reading my chapter on his beloved NV centres. Thanks to everyone involved for putting up with me.

The *Quantum Engineering Centre for Doctoral Training* gave me an opportunity I never thought I'd have, as well as a community to enjoy it with. The idea that I can call myself a scientist means a lot to me, and simply would not have happened if the QECDT hadn't taken a chance on me – I will always appreciate it. Of course, the CDT peaked with cohort 3: thanks to Dave for being willing to share a house in our first year, to Will for teaching me how to play squash (and beating me every time), and Max for a recent phone call where we agreed that thesis acknowledgements should read like the acceptance speech to an *Oscar* you've awarded yourself,

and to fully embrace it. I am grateful to Jorge and Holly for thinking of me to run *Quantum in the Summer* together, and to Konstantina for working most closely with me along the way.

I moved to Bristol for a PhD, but I surely wouldn't have stayed without building a good life here. I've enjoyed all the time spent cycling; playing Gaelic football, soccer and squash; learning mind games like Avalon, and trips to cinemas and restaurants with colleagues and friends. I always say – you can easily fill six nights per week with wholesome, healthy activities, without drinking. For the seventh night, however, you should hope to make friends like I did: Alex, Dom, Jake, Joel, Frazer, Reece, and all those mentioned already. Patrick had the unique punishment of thesis-reader, office-mate and pub-goer (I am not sure what past-life indiscretion he is atoning for, but I'm grateful for it as well). My happiest memories of this stage of my life will be sunny beer gardens on Friday afternoons with all of the above.

While the last year of my PhD (and thesis writing) coincided with Covid-19 lockdowns, one silver lining has been catching up online with my friends from Dublin more regularly. It is a pleasure to lose money to you all in poker, and I look forward to losing in person. Niall read my attempt to explain quantum mechanics, and was gracious enough not to berate me for it. James and John Mark put up with more than their fair share over a number of weekends. Evan has read every CV, paper, and now thesis, that I've written in the last eight years. If Roy Keane had for football the passion Evan has for physics, Ireland would have won the 2002 World Cup.

Weekends spent with old friends are high on my list of favourite things. I am always pleased when someone goes out of their way to visit – thanks to anyone who has gotten on a plane or train for my sake. Special mention to *Big Red* – a lifelong friend with a talent for meandering, all-encompassing conversation, and a unique outlook: I've heard many people say *Would this make me rich/famous/fulfilled*, he's the only one to ever ask *Would this make me happy?*

Along with all of the great people above, I am lucky to have a supportive family. I hope this thesis goes some way to answering your question, *what do you actually do?* To the next generation: I hope one day in ten or fifteen years, you will find this thesis some rainy afternoon. If you ask about it, don't believe them if they tell you I was smart. Believe them if they say I worked hard, but that I thoroughly enjoyed it – it would not have been worth doing otherwise.

More important than any thesis, these years will be remembered for the short time we got to spend with Lola and Luca. I still don't have words elegant enough to lessen their loss – I just wanted to acknowledge them here.

To Mam and Dad, sincere thanks for everything. We know you're proud of us; we're proud of you too.

Finally, Emma has been constant across all aspects of my life. I don't deserve the patience you've shown, especially this last year when I've been even more stressful, but I will always appreciate it, and do my best to make it worth your while.

Go raibh míle maith agaibh,
Brian

For
Lola & Luca

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LISTINGS

ACRONYMS

C	carbon
^{14}N	nitrogen-14
AI	artificial intelligence
AIC	Akaike information criterion, Eq. (8.5)
AICC	Akaike information criterion corrected, Eq. (8.6)
BF	Bayes factor, Section 5.2
BFEER	Bayes factor enhanced Elo ratings, Section 8.2.7
BIC	Bayesian information criterion, Eq. (8.9)
CLE	classical likelihood estimation, Section 4.3
CPU	central processing unit
DAG	directed acyclic graph
EDH	experiment design heuristic, Section 4.6
ES	exploration strategy, Section 5.4
ET	exploration tree, Section 5.3
FH	Fermi-Hubbard
FN	false negatives, Table 3.1
FP	false positives, Table 3.1
GA	genetic algorithm, Section 3.3
GES	genetic exploration strategy
GPU	graphics processing unit
IQLE	interactive quantum likelihood estimation, Section 4.3.1
JWT	Jordan Wigner transformation, Section 7.4.1
LE	Loschmidt echo, Section 4.3.1
LTL	log total likelihood, Section 4.4

ML machine learning, Chapter 3
MW microwave

NV nitrogen-vacancy
NVC nitrogen-vacancy centre, Section 9.1

OF objective function, Section 3.1.1.1

PBS portable batch system
PGH particle guess heuristic, Section 4.6.1
PL photoluminescence

QC quantum computer, Section 2.3
QHL quantum Hamiltonian learning, Chapter 4.
QL quadratic loss, Eq. (4.19)
QLE quantum likelihood estimation, Section 4.3
QM quantum mechanics, Chapter 2
QML quantum machine learning, Section 3.2
QMLA Quantum Model Learning Agent, Chapter 5.

SMC sequential Monte Carlo, Section 4.2

TL total likelihood, Eq. (4.15)
TLTL total log total likelihood, Eq. (4.18)
TN true negatives, Table 3.1
TP true positives, Table 3.1

VQE variational quantum eigensolver

N_E	Number of experiments to perform during model training through QHL.
N_P	Number of particles used when training a model through QHL.
Q	Quantum system which is the target of QMLA, i.e. the system to be characterised.
$\hat{H}_0$	True model for the target system, Q; i.e. the Hamiltonian model form which QMLA is attempting to retrieve in a given instance.
F_1**-score**	F_1-score, measure of model quality with respect to $\hat{H}_0$, $f \in [0,1]$, Section 8.1.2.
champion	See champion model.
champion model	The model deemed by QMLA as the most suitable for describing the target system.
chromosome	A single candidate, in the space of valid solutions to the posed problem in a genetic algorithm, Section 3.3.
Elo rating	Points transfer system for quantifying relative ability of candidates in a competitive domain, modified for use as an objective function within the genetic algorithm of QMLA, Section 8.2.7
expectation value	Average outcome expected by measuring an observable of a quantum system many times, Section 2.2.1.
experiment	Experiment performed upon Q when training a model through QHL, Section 4.2.
gene	Individual element within a chromosome.
Hamiltonian	Mathematical structure which captures all interactions to which a quantum system is subject, Section 2.1.1.
hyperparameter	Variable within an algorithm that determines how the algorithm itself proceeds.
instance	A single implementation of the QMLA algorithm, resulting in a nominated champion model.

likelihood Value that represents how likely a hypothesis is; usually used in the context of likelihood estiamation, Section 4.3.

model The mathematical description of some quantum system, Section 5.1.

model search Exploration through the model space performed by QMLA, where the progression of the search is determined by the ES Section 5.3.

model space Abstract space containing all descriptions (within defined constraints such as dimension) of the system as models, Section 5.4.1.

particle Sampled from prior distribution during model training through QHL, each particle gives a parameterisation corresponding to a unique hypothesis about Q, Section 4.2.

probe Input state, $|\psi\rangle$, into which the target system is initialised, before unitary evolution, Section 4.7.

results directory Directory to which the data and analysis for a given run of QMLA are stored.

run Collection of QMLA instances, usually targeting the same system with the same initial conditions.

spawn Process by which new models are generated within an ES, ususally by combining previously considered models.

success rate Fraction of instances within a run where QMLA nominates the true model as champion.

term Individual constituent of a model, e.g. a single operator within a sum of operators, which in total describe a Hamiltonian, Section 5.1.

true model The model which correctly describes the target system. $\hat{H}_0$ is known for simulated Q but not known precisely for experimental systems.

volume Volume of a parameter distribution's credible region, Section 4.5.1.

win rate The fraction of instances for which a given candidate model was nominated as champion within a run of QMLA.

LIST OF PUBLICATIONS

The work presented in this thesis has appeared[1] publicly in several formats.

Papers

1. *Learning models of quantum systems from experiments.* A.A. Gentile, Brian Flynn, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, R. Santagati and A. Laing. arXiv preprint arXiv:2002.06169 (2020); accepted *Nature Physics* (2021); referred to throughout as [1].
2. *Quantum Model Learning Agent: quantum systems' characterisation through machine learning.* Brian Flynn, A.A. Gentile, R. Santagati, N. Wiebe and A. Laing. Reporting outcomes of studies on theoretical systems as described in this thesis. In Preparation (2021); referred to throughout as [2].
3. *Quantum Model Learning Agent: Python framework for characterising quantum systems.* Brian Flynn, A.A. Gentile, R. Santagati, N. Wiebe and A. Laing. Technical manuscript detailing software implementation. In Preparation (2021).

Software

4. *QMLA: Python framework for the reverse engineering of Hamiltonian models of quantum systems through machine learning.* Brian Flynn, A.A. Gentile, R. Santagati, N. Wiebe, S. Paesani, C. E. Granade, and A. Laing. Codebase; open sourced via Github repository; referrred to throughout as [3].
5. *Quantum Model Learning Agent*. Brian Flynn, A.A. Gentile, R. Santagati, N. Wiebe, S. Paesani, C. E. Granade, and A. Laing. Documentation for codebase; referrred to throughout as [4].

Conference Proceedings - Talks [2]

6. *Quantum Model Learning Agent.* Brian Flynn, A.A. Gentile, R. Santagati, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, and A. Laing. Quantum Techniques in Machine Learning, Online, 2020.

1 Or will appear in the near future.
2 Note: only conferences proceedings presented by the author are included.

7. *Learning models of quantum systems from experiments.* Brian Flynn, A.A. Gentile, R. Santagati, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, and A. Laing. Bristol Quantum Information Technologies, Online, 2020.
8. *Quantum Model Learning: characterizing quantum systems through machine learning*. Brian Flynn, A.A. Gentile, R. Santagati, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, and A. Laing. Quantum Techniques in Machine Learning, Daejeon, South Korea, 2019.
9. *Quantum Model Learning Agent.* Brian Flynn, A.A. Gentile, R. Santagati, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, and A. Laing. Quantum Engineering Centre for Doctoral Training Conference, Bristol, UK, 2019. Awarded *Talk prize*.

Conference Proceedings - Posters

10. *Quantum Model Learning Agent.* Brian Flynn, A.A. Gentile, R. Santagati, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, and A. Laing. Machine Learning for Quantum, IOP Conference, Online, 2021. Awarded *Poster prize*.
11. *Quantum Model Learning.* Brian Flynn, A.A. Gentile, R. Santagati, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, and A. Laing. Machine Learning for Quantum Technologies, Erlangen, Germany, 2019.
12. *Quantum Model Learning.* Brian Flynn, A.A. Gentile, R. Santagati, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, and A. Laing. Bristol Quantum Information Technologies, Bristol, UK, 2019.
13. *Quantum Model Learning: characterizing quantum systems through machine learning.* Brian Flynn, A.A. Gentile, R. Santagati, S, Knauer, N. Wiebe, S. Paesani, C. E. Granade, J. G. Rarity, and A. Laing. Quantum Engineering Centre for Doctoral Training Conference, Bristol, UK, 2018. Awarded *Poster prize*.

INTRODUCTION

1 INTRODUCTION

Quantum mechanics (QM) is the study of nature's fundamental processes, manifest in its smallest particles. QM has been at the forefront of physics since the early 20th century [5]. Advances in theoretical understanding of quantum mechanical systems in the first half of the century [6–10] came to underpin many modern information and communications technologies[1] in the second half [11, 12]. The 21st century, on the other hand, is poised to see the development of technologies which *deliberately* exploit the most intricate quantum processes [13]. These *quantum technologies* share the promise of super-classical outcomes, i.e. that exploiting quantum phenomena can yield results which could never be achieved through non-quantum (or *classical*) means. The enduring motivation for the development of quantum technologies is *quantum simulation*: controlling quantum systems to represent other quantum systems, enabling the study of such structures and interactions for the first time [13, 14].

Significant advances in the design and construction of quantum hardware in recent years promise reliable, large scale quantum infrastructure in the near future [15, 16]. Alongside improvements in quantum systems' control, progress in quantum algorithms and software foreshadow impactful applications for quantum technologies, from database search [17] to quantum chemistry [18] and drug design [19]. Automated methodologies for characterising quantum systems are among the applications becoming feasible, with the development of quantum devices capable of simulating nature at the quantum level [20]. There is a large and growing interest in automatically identifying the *models* of quantum systems, i.e. the mathematical structure representing a system's interactions [21–25].

In parallel to the rise of quantum technologies over the past several decades, machine learning (ML) and artificial intelligence have enjoyed increasing interest and resources. Landmark outcomes, for example in facial recognition [26] and complex strategy games [27, 28], were bolstered by dramatic gains in the design of information processing machinery such as supercomputing facilities [29] and graphics processing units [30]. ML has been widely adopted to accelerate the impact of quantum technologies, from error correction [31, 32] to metrology [33] and device calibration [34].

In this thesis we report progress in the domain of quantum system characterisation, through novel quantum algorithms empowered by the promise of quantum simulators, leveraging state-of-the-art machine learning techniques. Namely, we introduce and develop the Quantum Model Learning Agent (QMLA) as a powerful platform for the study of quantum systems, ranging from controlled quantum simulators to experimental setups. QMLA distills an approximate model for a given quantum system, by constructing a series of candidate models and testing them against data from the system of interest. In providing a robust software framework for

1 Colloquially referred to as *Quantum 1.0*.

QMLA, we initiate an exciting field of research at the overlap of machine learning and quantum simulation, with proposed applications in calibrating new quantum technologies as well as understanding quantum processes in nature.

1.1 THESIS OUTLINE

The works presented in this thesis are closely related, all stemming from the QMLA protocol and framework. The thesis is organised into *parts*, which group together related bodies of work; within each part, individual studies are presented in self contained chapters. The main results and novel research is contained in Parts II to IV. At the outset of each part, we summarise its chapters and contributions. The contents of each part are as follows.

PART I CONTEXTUAL REVIEW

We introduce the concepts upon which the thesis will build. Chapter 2 establishes the vocabulary of quantum mechanics, followed by a summary of machine learning in Chapter 3. In both cases, we seek to introduce the minimal nomenclature required to contextualise the work in the following chapters; that is, neither topic is described exhaustively.

PART II ALGORITHMS

We provide a thorough explanation of the algorithms underlying this thesis. We start by summarising quantum Hamiltonian learning, which serves as a key subroutine within later studies and should therefore be understood, in order that the contributions of later chapters may be fully appreciated. The first major result is the QMLA algorithm itself, detailed in Chapter 5. All subsequent chapters assume knowledge of the terminology and concepts related to QMLA, so unfamiliar readers will find Chapter 5 essential. Our next contribution is an open source software platform for the implementation of QMLA, for the study of arbitrary quantum systems. In Chapter 6 we list the implementation details of this framework, but do not further any physical or algorithmic concepts. The QMLA software is available at [3] with documentation at [4]. By the end of Part II, we are armed with QMLA as a tool for the inspection of target quantum systems of interest, which will serve as a platform for the remaining chapters.

PART III THEORETICAL STUDY

We perform tests of the QMLA framework under idealised simulated conditions, corresponding directly to [2]. In Chapter 7 we demonstrate that QMLA is trustworthy in the most straightforward scenario, where a number of candidate models are proposed in advance. We then move to more difficult conditions in Chapter 8, exploring spaces of 10^5 valid candidate models, by incorporating a genetic algorithm within QMLA. Together, the cases studied here verify that QMLA shows promise in characterising quantum systems, in particular suggesting a compelling application in the calibration and verification of quantum simulators.

PART IV EXPERIMENTAL STUDY

The final contribution reflects work published in [1]. We extend the QMLA protocol to *realistic* quantum systems, namely targeting the decoherence processes dominating the dynamics of an electron spin in a nitrogen-vacancy centre. In Chapter 9 we operate on data extracted from an experimental system, from which QMLA distills models with high predictive power – i.e. which can reproduce the dynamics of the target system – and which are in agreement with theoretical predictions. Chapter 9 relied on several constraints to facilitate the model search; in Chapter 10 we relax some of those constraints by simulating a similar system, and again exploit a genetic algorithm to explore the model space. The results of this part indicate that QMLA may be helpful in the study of *black-box* quantum systems.

PART V CONCLUSION

We close the thesis with a brief summary of its main contributions, and offer an outlook for model learning methodologies in the context of quantum technologies.

Part I

CONTEXTUAL REVIEW

2

QUANTUM THEORY

This thesis focuses on the application of machine learning to the characterisation of quantum mechanical systems through use of quantum simulators, so it is pertinent first to introduce the vocabulary of quantum mechanics (QM). It is impossible, however, to succinctly capture the entire discipline; in this chapter we will only introduce concepts utilised throughout. For completeness, we elucidate some fundamental topics of linear algebra and quantum theory in Appendix B, but consider them too cumbersome to include in the main text. For a more complete and general introduction to QM, the reader is referred to [35, 36]. Likewise, in this chapter we quickly summarise the key aspects of quantum computation, but for further details, we recommend unfamiliar readers to consult [37], while a more complete discussion is presented in [38].

2.1 QUANTUM MECHANICS

At any time, t, a quantum system, Q, can be described by its *wavefunction*, $\Psi(t)$, which contains all information about Q. In analogy with Newton's second law of motion, which allows for the determination of a particle's position at any time, $\vec{r}(t)$, given its conditions – its initial position, $\vec{r}(t_0)$ and momentum – quantum *equations of motion* can describe the evolution of Q through its wavefunction [39]. One proposal[1] for the equation of motion to describe the evolution of the wavefunction under known conditions, i.e. determining $\Psi(t)$ from $\Psi(t_0)$ $\forall t > t_0$, is the *Schrödinger equation* [35, 40, 41].

Although the Schrödinger equation is a *postulate* of QM, i.e. can act as a starting point for QM (see Appendix B.2), we will sketch a brief, informal derivation to elucidate its meaning following [36]. We have yet to describe the structure of the wavefunction, which we will do in Section 2.2, but here we will represent wavefunctions using *Dirac notation* (Appendix B.6), and can think of them generically as vectors, i.e. $\Psi(t) \to |\psi(t)\rangle$. Suppose we have two such wavefunctions, $|\phi(t)\rangle, |\psi(t)\rangle$ which are functions of time $t > t_0$. We start with the assumption that *similarity* is conserved between two wavefunctions, if they undergo the same transformation (Susskind's *minus first* law of classical mechanics [36]),

$$\langle\phi(t)|\psi(t)\rangle = \langle\phi(t_0)|\psi(t_0)\rangle\,. \tag{2.1}$$

Then, assuming some equations of motion capture the dynamics of Q, there exists some evolution operator, $\hat{U}(t)$, which deterministically maps $|\psi(t_o)\rangle$ to $|\psi(t)\rangle$. That is,

$$|\psi(t)\rangle = \hat{U}(t)\,|\psi(t_0)\rangle\,, \tag{2.2}$$

1 The most noteworthy alternative formalism, due to Heisenberg [9], was shown equivalent to the Schrödinger picture described here.

where we have not yet imposed any restrictions on $\hat{U}$. We also have the *dual vector*[2] of Eq. (2.2),

$$\langle\psi(t)| = \langle\psi(t_o)|\,\hat{U}^\dagger. \tag{2.3}$$

Combining Eqs. (2.1) to (2.3),

$$\begin{aligned} \langle\phi(t)|\psi(t)\rangle &= \langle\phi(t_0)|\hat{U}^\dagger\hat{U}|\psi(t_o)\rangle \\ \Rightarrow \langle\phi(t_0)|\hat{U}^\dagger(t)\hat{U}(t)|\psi(t_o)\rangle &= \langle\phi(t_0)|\psi(t_0)\rangle \\ \Rightarrow \hat{U}^\dagger(t)\hat{U}(t) &= \hat{\mathbb{1}} \quad \forall t, \end{aligned} \tag{2.4}$$

where the result $\hat{U}^\dagger(t)\hat{U}(t) = \hat{\mathbb{1}}$ is the condition for *unitarity* of $\hat{U}(t)$ (Appendix B.5), so we can claim the quantum wavefunction evolves unitarily.

By construction, we require that after zero time, i.e. $t = t_0$, the wavefunction has not changed:

$$\begin{aligned} |\psi(t = t_0)\rangle = \hat{U}(t = t_0)\,|\psi(t_0)\rangle &= |\psi(t_0)\rangle \\ \Rightarrow \hat{U}(t = t_0) &= \hat{\mathbb{1}}. \end{aligned} \tag{2.5}$$

Without loss of generality we can set $t_0 = 0$, giving $\hat{U}(0) = \hat{\mathbb{1}}$. Then, let us consider an infinitesimally small time increment $t_0 + \epsilon$: again, take $t_0 = 0$ so $t = \epsilon$, where $\epsilon \gg \epsilon^2$. We can say

$$\hat{U}(\epsilon) = \hat{\mathbb{1}} + \mathcal{O}(\epsilon), \tag{2.6}$$

which merely suggests that the time evolution operator at very small time is very close to the identity, with some small displacement proportional to the time, which must be an operator to act on the wavefunction (vector). We suppose the form of the offset, so we can write

$$\hat{U}(\epsilon) = \hat{\mathbb{1}} - \epsilon\left(\frac{i}{\hbar}\hat{H}_0\right), \tag{2.7}$$

where the inclusion of the phase $-i$ is arbitrary, and we have named as $\hat{H}_0$ the operator by which the time evolution differs from the identity, scaled by the reduced Planck constant, $\hbar = 1.054 \times 10^{-34}$. In other words, the operator $\hat{H}_0$ is, generically, the generator of the evolution/dynamics

2 see Appendix B.1.

of Q: any difference between $|\psi(t_0)\rangle$ and $|\psi(t)\rangle$ arises solely due to $\hat{H}_0$. So far there is no restriction[3] on $\hat{H}_0$. Recalling the unitarity condition, and that $\lim_{\epsilon\to 0}\mathcal{O}(\epsilon^2)\approx 0$, we have:

$$\begin{aligned}
\hat{U}^\dagger(\epsilon)\hat{U}(\epsilon) &= \hat{\mathbb{1}} \\
\Rightarrow \left(\hat{\mathbb{1}} + \frac{i}{\hbar}\epsilon\hat{H}_0^\dagger\right)\left(\hat{\mathbb{1}} - \frac{i}{\hbar}\epsilon\hat{H}_0\right) &= \hat{\mathbb{1}} \\
\Rightarrow \hat{\mathbb{1}} + \frac{i}{\hbar}\epsilon(\hat{H}_0^\dagger - \hat{H}_0) + \mathcal{O}(\epsilon^2) &= \hat{\mathbb{1}} \\
\Rightarrow (\hat{H}_0^\dagger - \hat{H}_0) &= 0 \\
\Rightarrow \hat{H}_0^\dagger &= \hat{H}_0.
\end{aligned} \tag{2.8}$$

Eq. (2.8) results in the condition for *Hermiticity*, meaning that $\hat{H}_0$ is an observable of Q. In fact, this is the *Hamiltonian* of the system, described in the next section.

We can also use the infinitesimal evolution to see

$$\begin{aligned}
|\psi(t)\rangle &= \hat{U}(t)\,|\psi(t_0)\rangle \\
\Rightarrow |\psi(\epsilon)\rangle &= \hat{U}(\epsilon)\,|\psi(t_0)\rangle \\
\Rightarrow |\psi(\epsilon)\rangle &= \left(\hat{\mathbb{1}} - \epsilon\frac{i}{\hbar}\hat{H}_0\right)|\psi(t_0)\rangle \\
\Rightarrow |\psi(\epsilon)\rangle &= |\psi(t_0)\rangle - \epsilon\frac{i}{\hbar}\hat{H}_0\,|\psi(t_0)\rangle \\
\Rightarrow \frac{|\psi(\epsilon)\rangle - |\psi(t_0)\rangle}{\epsilon} &= -\frac{i}{\hbar}\hat{H}_0\,|\psi(t_0)\rangle\,.
\end{aligned} \tag{2.9}$$

Taking the limit as $\epsilon \to 0$, the left hand side of the final line of Eq. (2.9) is the definition of the derivative of the wavefunction, $\frac{d|\psi(t)\rangle}{dt}$. Taken together, we have

$$\frac{\partial}{\partial t}\,|\psi(t)\rangle = \frac{-i}{\hbar}\hat{H}_0\,|\psi(t_0)\rangle\,, \tag{2.10}$$

where $|\psi(t)\rangle$ is the wavefunction at time t, $|\psi(t_0)\rangle$ is the wavefunction at t_0, such that $t > t_0$, $\hbar$ is the reduced Planck constant and $\hat{H}_0$ is the *Hamiltonian* of Q. For brevity we generally refer to $t_0 = 0$, and absorb $\hbar$ into $\hat{H}_0$, which will later manifest in the Hamiltonian scalar parameters. Eq. (2.10) is the most general form of *Schrödinger equation*, otherwise known as the *time-dependent* Schrödinger equation; we include it as Postulate 6 when describing the fundamentals of QM (Appendix B.2), since it can be seen as an irreducible equation of motion which is essential to the description of quantum systems.

As mentioned, we presented this argument in a nonstandard order: we started with Eq. (2.2), which we can now consider the *solution* to the Schrödinger equation, specifically

$$\begin{aligned}
|\psi(t)\rangle &= \hat{U}(t)\,|\psi(0)\rangle \\
\implies \hat{U}(t) &= e^{-i\hat{H}_0 t}.
\end{aligned} \tag{2.11}$$

3 We do restrict $\hat{H}_0$ to the same dimension as the Hilbert space in question; the concept of Hilbert space will be defined in Section 2.2, but is not needed in this discussion.

$\hat{U}(t)$ then describes the unitary evolution of the wavefunction of a quantum system according to its Hamiltonian, $\hat{H}_0$.

2.1.1 *Hamiltonians*

In the previous section we introduced the Hamiltonian of Q as the generator of its time evolution dynamics; Hamiltonians are of primary importance in this thesis, so it is worth pausing to consider their physical meaning. We saw in Eq. (2.8) that $\hat{H}_0$ is Hermitian; the Hamiltonian is the observable operator (see Postulates 2 to 3 in Appendix B.2) associated with the system's total energy, i.e. the permitted energy levels of the system are given by the eigenfunctions of $\hat{H}_0$.

The quantum Hamiltonian, $\hat{H}_0$, is analogous to the *classical* Hamiltonian, which captures all the interactions of a given classical system which contribute to its time evolution. Knowing the classical Hamiltonian and the initial conditions – position and momentum – Hamilton's equations of motion allow for the calculation of those quantities for the particle in question an infinitesimal time later [42]. Likewise, knowledge of the initial wavefunction, $|\psi(t_0)\rangle$, and the system's quantum Hamiltonian, $\hat{H}_0$, the quantum equation of motion – namely the Schrödinger equation, Eq. (2.10) – permits the calculation of the wavefunction at later times. As such the Hamiltonian must consist of all processes which influence the evolution of Q; we will later break the Hamiltonian into independent *terms* which each correspond to unique physical interactions Q is subject to, in Section 5.1. We can think that each process/interaction Q undergoes contributes to its total energy, giving intuition as to why its eigenvalues are the energy levels.

Hamiltonians describe *closed* quantum systems, i.e. where *all* processes and interactions which influence Q are accounted for. Realistic quantum systems are influenced by a myriad of proximal systems, and it is therefore usually infeasible to analytically account for them all. Instead, *open* quantum systems' dynamics can, in some cases, be described by Lindbladian operators, which encompass the Hamiltonian form. The Lindblad master equation is a generalisation of the Schrödinger equation, providing the equation of motion for open quantum systems [43, 44]. In this thesis we only consider closed models for quantum systems; for meaningful impact of the techniques presented here, it will be necessary to expand them to account for the open system dynamics of realistic experiments. We do, however, show initial progress towards this endeavour by modelling a physical system through a closed Hamiltonian in Chapter 9.

2.2 QUANTUM INFORMATION

The wavefunction for a physical system, Q, is also known as its *state*, a complete mathematical description of the system [45]. States are vectors[4] of complex numbers; the valid state space

3 Aside: the author shares a hometown with the mathematician for whom it is named, William Rowan Hamilton. It is hoped that, after another 150 years, the next physicist from Trim, Co. Meath, Ireland might profitably use knowledge of Hamiltonians on a functional quantum computer.

for Q is its *Hilbert space*, $\mathcal{H}$, which is a generalisation of Euclidean vector space, i.e. $|\psi\rangle \in \mathcal{H}$. The Hilbert space defines the overlap between any two vectors as the *inner product*, $\langle\psi|\phi\rangle$ (see Appendix B.1). In general[5], a state can be seen as a *superposition* across a set of orthonormal states, often represented over the system's eigenstates, $\{|v_i\rangle\}$:

$$|\psi\rangle = \sum_i \alpha_i |v_i\rangle \tag{2.12a}$$

$$\text{subject to} \quad \sum_i |\alpha_i|^2 = 1, \quad \alpha_i \in \mathbb{C}. \tag{2.12b}$$

The cornerstone of QM is the effect of *measurement* on quantum systems: in general Q can be seen as occupying a multitude of eigenstates as in Eq. (2.12a); observing the system forces $|\psi\rangle$ into definite occupation of a measurement basis state, where the *probability* that it is measured in each eigenstate $|v_i\rangle$ is given by $|\alpha_i|^2$, according to Born's rule [7]. α_i are hence named *probability amplitudes* since they inform the probability of measuring the corresponding eigenstate.

For a single particle, when the state, Eq. (2.12a), has two available eigenstates, e.g. the horizontal (H) and vertical (V) polarisation of a single photon, we can designate Q as a two-level computational platform, called a *qubit*[6], analogous to the workhorse of classical computation, the bit. A qubit's state vector can then be written as a sum over the two available eigenstates, where we assign vectors to the eigenstates as

$$\begin{aligned} |H\rangle = |v_1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} =: |0\rangle\,; \\ |V\rangle = |v_2\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} =: |1\rangle\,. \end{aligned} \tag{2.13}$$

The state of a qubit is then given by

$$|\psi\rangle = \alpha_1 |v_1\rangle + \alpha_2 |v_2\rangle\,, \tag{2.14}$$

where $\alpha_i \in \mathbb{C}$ and $|\alpha_1|^2 + |\alpha_2|^2 = 1$.

In general, a qubit requires two orthogonal state vectors to define a *basis*; we list a number of the usual special cases:

$$X\text{-basis} = \begin{cases} |+\rangle = \dfrac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \\ |-\rangle = \dfrac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) \end{cases} \tag{2.15a}$$

4 We immediately use Dirac notation to represent the state; it is defined in Appendix B.6.

5 We expand on this brief description in Appendix B.3.

6 Here we describe ideal, *logical* qubits as the foundation of computation. In reality, physical qubits are beset by errors, demanding error correction routines such that multiple particles are needed in order to attain a single logical qubit.

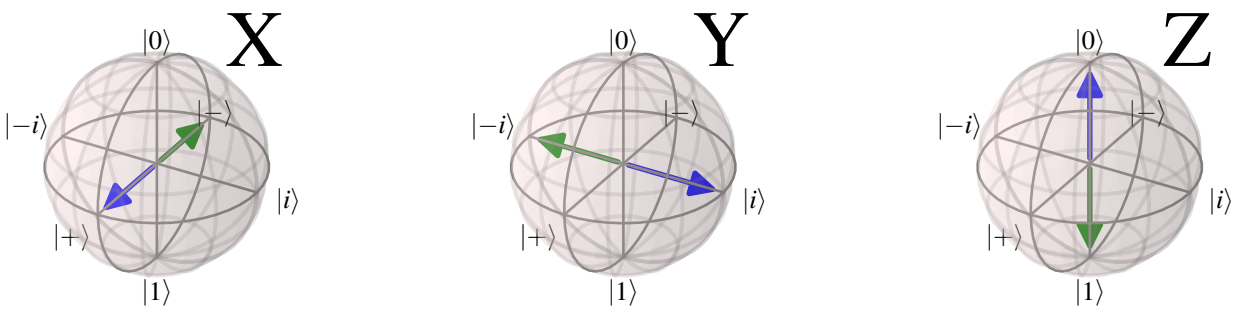


Figure 2.1: Bloch sphere representation of bases, where each pair of basis states are shown by blue and green vectors. The X-basis has basis vectors $\{|+\rangle, |-\rangle\}$; Y-basis has $\{|i\rangle, |-i\rangle\}$ and Z-basis has $\{|0\rangle, |1\rangle\}$.

$$Y\text{-basis} = \begin{cases} |i\rangle = \dfrac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle) \\ |-i\rangle = \dfrac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle) \end{cases} \tag{2.15b}$$

$$Z\text{-basis} = \begin{cases} |0\rangle \\ |1\rangle \end{cases} \tag{2.15c}$$

A visual tool for representing qubits is the *Bloch sphere*, which presents orthogonal basis states as parallel unit vectors of opposite direction: we show each of the bases of Eq. (2.15) in Fig. 2.1.

We can make two remarks about basis states for a single qubit:

- Basis states from one basis can be seen as superpositions with respect to alternative bases
  - e.g. in the X-basis, $|+\rangle$ is a basis vector, but in the Z-basis, $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ is a superposition over basis vectors.
- Bases are local rotations of each other
  - rotating the X-basis through an angle $\pi/2$ about the Y-axis results in the Z-axis.

As we alluded to in Section 2.1, by imposing mathematical structure on quantum systems' states, i.e. representing Q as a state vector at any time, then operations which alter the state of the system must be matrices, which we will call *operators*. In general an n-dimensional vector is rotated by an $n \times n$ matrix; therefore to rotate the one-qubit state, given by a two-dimensional vector, we require a 2×2 operator. One-qubit operators have the effect of rotating the state vector, which we can again visualise on the Bloch sphere. By thinking of qubits generically with respect to any basis, we can encode *information* in the qubit's amplitudes, by performing

operations (or *gates*) upon the qubit, we change the information, i.e. we can design information processing techniques leveraging the infrastructure – states, operators and measurement – of QM.

We introduce a set of special one-qubit operators, the *Pauli matrices*,

$$\hat{\sigma}_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; \tag{2.16a}$$

$$\hat{\sigma}_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}; \tag{2.16b}$$

$$\hat{\sigma}_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.16c}$$

The Pauli matrices are used to define rotation operators about their respective axes, and hence are very useful: we can break *any* rotation of a qubit into rotations of various angles, θ, about the three axes of the Bloch sphere. Any single qubit operation can therefore be expressed as a product of the *rotation operators*, $\hat{R}_x, \hat{R}_y, \hat{R}_z$, exemplified in Fig. 2.2 and defined for $w \in \{x, y, z\}$ as

$$\hat{R}_w(\theta) = e^{-i\frac{\theta}{2}\hat{\sigma}_w} = \cos(\theta/2)\,\hat{\mathbb{1}} - i\sin(\theta/2)\,\hat{\sigma}_w. \tag{2.17}$$

The Pauli matrices are observable; in particular, the eigenstates of $\hat{\sigma}_z$ are the Z-basis states: $\hat{\sigma}_z|0\rangle = |0\rangle; \hat{\sigma}_z|1\rangle = -|1\rangle$. Recalling the earlier claim that the two-level quantum system (e.g. H and V polarisation of a photon) can be mapped to eigenstates of an obserable operator to form a qubit, we term the Z-basis the *computational* basis. By defining the computational basis, we ground abstract computational reasoning in the physical realisation: anywhere throughout this thesis where the basis states $\{|0\rangle, |1\rangle\}$ are referenced, we mean the eigenstates of the physical axis which is defined as the Z-axis for the system in question. In the computational basis, then, a qubit can be specified as

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle. \tag{2.18}$$

The concepts of qubits representing quantum systems, as well as operators altering their states and measurement collapsing those states, extend straightforwardly to multipartite systems by merging Hilbert spaces through tensor products, as we show in Appendix B.3.1. While single qubit states are spanned by the Pauli operators, multi-qubit states are spanned by the Pauli group, $\mathbb{G}$: n-qubit states are spanned by $\mathbb{G}_n = \left(\mathbb{C}^2\right)^{\otimes n}$. Multipartite systems can exhibit the strictly non-classical phenomenon of *entanglement*, where the constituent particles can not be described independently, which we briefly detail in Appendix B.4.

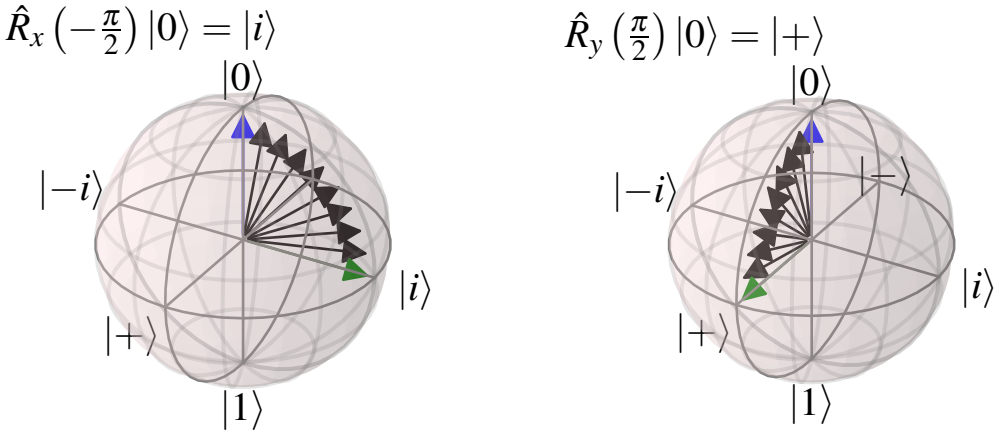


Figure 2.2: Rotations on Bloch sphere. The initial and final states are shown in blue and green respectively, while intermediate states are shown in black. **Left**, The Z-basis unit vector, $|0\rangle$, is rotated about the X-axis, resulting in the unit vector along the Y-axis. **Right**, The Z-basis unit vector, $|0\rangle$, is rotated about the Y-axis, resulting in the unit vector along the X-axis.

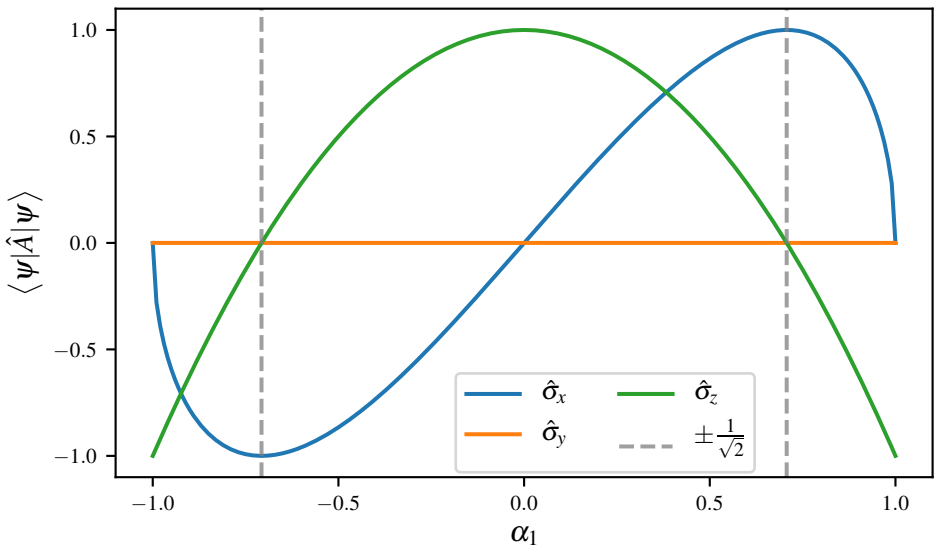


Figure 2.3: Expectation values of the observable $\hat{A} \in \{\hat{\sigma}_x, \hat{\sigma}_y, \hat{\sigma}_z\}$ for $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$. Coefficients are real, varying $\alpha_1 \in (-1, 1)$, such that $\alpha_0 = \sqrt{1 - \alpha_1^2}$.

2.2.1 *Expectation values*

Upon measurement, the state vector of Q has amplitude associated with only one eigenstate. On average, however, the eigenstate to which it would collapse encodes statistical insight on the state prior to measurement. In other words, if we prepared $|\psi\rangle$ and measured it – via some observable, $\hat{A}$ – and repeat the procedure N times, then as $N \to \infty$, the average outcome is the *expectation value* for the system.

$$\langle \hat{A} \rangle = \langle \psi | \hat{A} | \psi \rangle = \sum_i \alpha_i \langle v_i | \hat{A} | v_i \rangle \,, \tag{2.19}$$

where $\langle \hat{A} \rangle$ is the expectation value (average) for the observable $\hat{A}$; $|v_i\rangle$ are the eigenstates of $\hat{A}$, and $\alpha_i \in \mathbb{C}$ are the probability amplitudes associated with each $|v_i\rangle$ when the state $|\psi\rangle$ is represented as in Eq. (2.12a). We show some examples of expectation values for the observable Pauli matrices in Fig. 2.3.

An underlying theme of this thesis is to flip the usual logic: instead of using knowledge of the system to derive the expectation value, per Eq. (2.19), we will *estimate* expectation values, either through experiment or simulation, and use them to infer the structure of the observable. This trick enables machine learning routines to reverse engineer the processes Q is subject to, as we will describe in Part II.

2.3 QUANTUM SIMULATION AND COMPUTATION

Relying on the premise and language of quantum information processing – states, qubits, operators, measurements and expectation values – the growing field of *quantum technology* aims to exploit the non-classical statistics yielded by quantum systems in order to retrieve outcomes beyond the capability of their classical counterparts [46]. Applications range from enhanced sensing and metrology [47, 48], to highly-secure communication and cryptography protocols [49–51]. The initial motivation for the development of quantum technologies, however, was the observation that simulating nature at a quantum level would require exponential resources on a classical device, and is therefore only feasible given controllable quantum systems, which can accurately emulate their true dynamics [13, 52–54].

The notion of controlling quantum systems to mimic the dynamics of natural quantum systems is tantamount to *quantum simulation* [14, 55]. In particular, simulating quantum systems is believed to be of interest for quantum chemistry [18, 56, 57], for example leading to advances in the simulation of molecular dynamics [58, 59]. More generally, however, this led to research into a wider domain of calculations called *quantum computation* which considers the information processing capability of controllable quantum systems beyond merely simulating quantum systems. Then, *universal quantum computers* (or, *quantum Turing machines*), assume access to *logical* qubits and operations (or *gates*) for the implementation of quantum circuits [60]. This ignited interest in *quantum algorithms*, which aim to provide some provable advantage [17, 61–64].

Indeed, it was found that the space of problems addressible by such devices goes beyond the classical counterpart, suggesting there exists a class of quatum algorithms which can offer significant advantage over any feasible algorithm on classical hardware [65].

Of course, while the advances in algorithmic quantum computation promise huge impact, they are tempered by contemporary experimental constraints, which must deal with the reality that construction and control of quantum devices is a significant challenge. In constructing quantum computers (QCs) and dealing with their output, we must account for physical effects which lead to errors, requiring expensive error mitigation schemes in order to be reliable [66, 67]. Furthermore, there are a number of criteria a QC must meet before it can be deemed reliable [68].

Any two-level quantum system can be used as a qubit, so a range of platforms have emerged in attempts to fulfil the potential of quantum computation [69]; here we provide an incomplete list of quantum architectures together with their primary advantages and limitations.

PHOTONIC QUBITS (linear optical QCs) [70].

- existing infrastructure for commercial production of photonics-based technologies suggests the relatively straightforward fabrication of integrated photonic devices at the scale of millions of degrees of freedom [71];
- photons do not decohere so are useful for encoding information [72];
- photons do not interfere [39, 73], making multi-qubit gates difficult to achieve, so information processing must be mediated by non-trivial measurement schemes [74];
- they are liable to a unique error mechanism – photon loss – necessitating novel quantum error correcting codes [75];
- on-demand single photon generation has not yet been demonstrated, although there is significant progress in the area of photon generation [76];
- alternative resource generation is possible through multiplexing, i.e. combining numerous probabilistic photon sources, but this imposes stringent hardware requirements [77].

SUPERCONDUCTING QUBITS [78, 79]

- relatively straightforward to control and couple with each other, enabling high-fidelity two-qubit gates, e.g. 99.7% reached in [80];
- difficult to engineer substantial coherence times, although there has been significant recent progress [81, 82], e.g. $T_1 \approx \mathcal{O}(\text{ms})$ in [83];
- require cryogenic temperatures for operation, demanding expensive and cumbersome infrastructure [78];
- arbitrary qubit connectivity at scale is yet to be demonstrated [79];

  - methods for the fabrication of medium-to-large scale devices required for fault tolerant quantum computation are not yet known [84].

ION TRAPS [85,86]

  - full connectivity between pairs of qubits [87];
  - high two qubit gate fidelities, e.g. 99.9% in [88];
  - very high coherence times, e.g. $\mathcal{O}(10\text{ min})$ in [89];
  - straightforward state preparation and readout, e.g. 99.99% readout fidelity in [90];
  - long gate-times, e.g. $\mathcal{O}(\mu\text{s})$ in [91];
  - uncertain scalability [92].

The ever-increasing space of quantum hardware contenders has led to a growing eco-system for quantum software [93], promising a wide range of applications in the era of noisy intermediate scale quantum devices [94]. Following numerous proposals [95], recent efforts have married state-of-the-art hardware with bespoke algorithms in order to achieve quantum advantage [15,16]. Evidently there is vast effort in bringing quantum computational resources to reality; in this thesis, however, we are not concerned with the architecture underlying our presumed quantum simulator – we perform simulations only on classical hardware. In principle, however, any quantum simulator – universal or otherwise – capable of implementing the time evolution operator, Eq. (2.11), can be called upon as a co-processor by the algorithms presented.

Our restriction to classical resources leads to a few remarks:

- given access to a fault-tolerant QC/simulator, the algorithms described would enjoy considerable speedup:
  - the classical bottleneck is the calculation of the time evolution dynamics, Eq. (2.11), according to the matrix exponential, of dimension 2^n, where n is the dimension of the system;
  - it is believed that the same calculation can be performed in polynomial time on a QC [14,96–99].
- the results achieved in this thesis are limited by the capability of classical computers in simulating quantum systems
  - we study only up to 8-qubit systems, whereas it would be of interest to extend these methods to higher dimensions, which is expected to be feasible when reliable quantum simulators/computers are available.

The remit of this thesis – given these limitations – is therefore to robustly test the presented algorithms, and provide benchmarks achieved through classical facilities, against which the same algorithms can be run in conjunction with quantum hardware.

3 MACHINE LEARNING

Machine learning (ML) is the application of statistics, algorithms and computing power to discover meaning and/or devise actions from data. ML has become an umbrella term, encompassing the family of algorithms which aim to leverage computers to learn without being explicitly programmed, as opposed to the more general artificial intelligence (AI), which seeks to make computers behave intelligently, admitting explicit programs to achieve tasks [100]. Its history is therefore imprecise since a number of early, apparently unrelated algorithms were proposed independently, which now constitute ML routines [101, 102]. Nevertheless, the field of ML has been advancing rapidly since the second half of the 20$^{\text{th}}$ century [103], especially recently due to the availability of advanced hardware such as graphics processing units (GPUs), facilitating significant progress through an ever-increasing arsenal of powerful open source software [104–106].

Throughout this thesis, we endeavour to combine known methods from the ML literature with capabilities of quantum computers (QCs)[1]. Typical ML algorithms, which rely on central processing units (CPUs) or GPUs, are deemed *classical* machine learning, in contrast with quantum machine learning (QML), where QCs are central to processing the data. Similarly to the remit of Chapter 2, here we do not provide an exhaustive account of ML algorithms; we describe only the concepts which are used in later chapters, referring readers to standard texts for a wider discussion [103, 107].

3.1 CLASSICAL MACHINE LEARNING

The first step in any ML application is to consider the ensemble of known algorithms, with respect to the available data. Classical ML is usually described in three categories: *supervised learning*, *unsupervised learning* and *reinforcement learning*. These categories are broadly based on the format of data on which the insight can be built; we will briefly describe each to provide context to discussions throughout this thesis. Later in this thesis we will use the word *model* for descriptions of quantum systems, but here *model* refers to the mapping between inputs and outputs, devised by the ML algorithm.

3.1.1 *Supervised machine learning*

Models are trained using *labelled* data, i.e. each training sample has a known label y_i – or, a set of *feature vectors* $\{\vec{x}_i\}$ are associated with the set of corresponding *classes* $\{y_i\}$ [108]. The output

1 Or simulated QCs, in this thesis.

is a predictive tool which aims to reconstruct the classes of unseen feature vectors: in general, we can view the role of ML in this setting as distilling the function f such that

$$f : \vec{x}_i \longmapsto y_i \quad \forall(\vec{x}_i, y_i). \tag{3.1}$$

There are a number of families of algorithms even within the broad category of supervised ML, we define them as follows.

CLASSIFICATION

Algorithms that aim to produce models which can assign unseen instances to the most appropriate label, from a fixed set of available labels [109].

– For example, labels indicate animals' species, and the feature vector for each sample (data point) encodes the animals' height, weight, number of legs, etc.

REGRESSION

Models which capture the formulaic relationship – either linear or polynamial – between numerical features and a target scalar value, by determining coefficients for each feature.

– For example, y is the salary of employees in a company, and the feature vector consists of the individual employees' age, seniority, experience in years, etc.

NEURAL NETWORKS

Universal function approximators[2]. By invoking a set of linear and non-linear transformations on input data, the *network* is a function of some paramterisation w; neural networks aim to find the optimal network, w', such that Eq. (3.1) is satisfied, $f(w') : \vec{x}_i \longmapsto y_i$.

– Usually used for classification.

– For example, input *neurons*[3] encode the pixel values of images. The neural network can be used to classify the objects it detects within the image.

SUPPORT VECTOR MACHINES

Distinguish similar data points by projecting data into higher dimensional space, and therein finding the hypersurface which separates classes [111].

– Usually used for classification tasks.

– For data $\mathcal{D} \in \mathbb{R}^n$, a hypersurface in $\mathbb{R}^{n-1}$, can be drawn arbitrarilily through the space (e.g. a 2D plane in a 3D space).

– Unseen data can then be classified by which partition they reside in when projected into the n-dimensional space.

2 Including deep learning networks [110].

– The task of the support vector machine is to orient the hypersurface in such a way as to separate distinct classes.

Supervised ML algorithms rely on the existence of a body of labelled samples – the dataset $\mathcal{D}$ – upon which the model can be trained. Training is typically performed on a subset of the data, $\mathcal{D}_t$, usually 80% of samples chosen at random. The remaining (20%) of samples, $\mathcal{D}_v$, are retained for the evaluation of the resultant model: $d \in \mathcal{D}_v$ are not trained upon, so do not contribute to the structure of f as returned by the algorithm. The model therefore can not *overfit*, i.e. simply recognise particular samples and label them correctly, without any meaningful inference. The evalutation thus captures how the model can be expected to perform on future, unlabelled data.

3.1.1.1 *Performance metrics*

The fair assessment of supervised methods can be achieved through a number of *performance metrics*. In each supervised ML algorithm, the machine attempts to learn the structure of f that optimises some internal objective function (OF), e.g. minimising the average distance between predicted and target labels for regression, $\sum_{d \in \mathcal{D}_t} y_d - y'_d$. To assess the resultant model, we introduce a number of *performance metrics*, which aim to measure serveral perspectives of the model's efficacy, by considering the model's predictions with respect to $\mathcal{D}_v$.

By definition of the data format, it is relatively straightforward to define metrics for supervised routines: the classes assigned to feature vectors, y'_i, can be quantitatively assessed with respect to their true class, y_i. For example, in *binary classification*, the output of the model is either correct or incorrect, allowing us to meaningfully assess its average performance. Likewise, for numerical targets, the difference $|y_i - y'_i|$ for each sample cumulatively indicate the strength of the model.

There are a large number of quantities and performance metrics against which to judge models' outputs. In binary classification, we care about whether the model predicts that a given feature is present, and whether it predicts features incorrectly. For example, the model aims to classify whether or not a dog is present in an image. These type of binary predictions have four outcomes, which can be summarised in a *confusion matrix* (Fig. 3.1), defined as in Table 3.1.

We can use the concepts of Table 3.1 to define a series of *rates* which characterise the model's predictions.

ACCURACY The overall rate at which the algorithm predicts the correct result.

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}} \tag{3.2a}$$

PRECISION Positive predicitve rate. Of those *predicted to have* the feature of interest, what percentage *actually have* the feature.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2b}$$

3 The term *neuron*, also known as *node* or *unit*, derives from the motivation for this class of algorithm: the cells in the brain used for processing information.

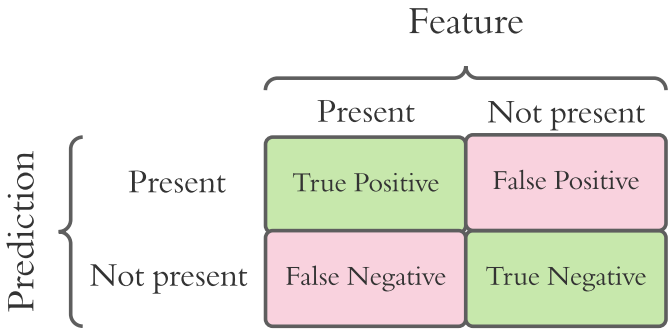


Figure 3.1: Confusion matrix showing the meaning of true positives, true negatives, false positives and false negatives.

	y has feature	y' has feature
True positives (TP)	✓	✓
False positives (FP)	×	✓
True negatives (TN)	×	×
False negatives (FN)	✓	×

Table 3.1: Classification metrics. We define classification outcomes based on whether the considered feature was present and/or predicted.

SENSITIVITY True positive rate (also known as *recall*). Of those which *actually have* the feature, what percentage are *predicted to have* the feature.

$$\text{sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.2c}$$

SPECIFICITY True negative rate. Of those which *do not actually have* the feature of interest, how many are *predicted not to* have the feature.

$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{3.2d}$$

Each metric has clear advantages, but consider also their drawbacks:

- Accuracy can be extremely misleading. For example, consider a dataset of 10,000 samples, of which only 100 contain the feature of interest. A binary model which predicts every instance as False will achieve $\text{TN} = 9,900, \text{FN} = 100$, receiving an overall accuracy $= 99\%$, despite not having found a single positive sample. This is clearly not useful in identifying the minority of cases of actual interest.

- Sensitivity can be inflated by over-fitting to positive cases. That is, by predicting the feature as present in all cases, all true instances of the feature will be found, however all False instances will be labelled as having the feature, so the model has not helped separate the data. The model will yield a high rate of true positives (TP) but also a high rate of false positives (FP).
- Precision can be high for extremely selective models, i.e. those which are conservative in predicting the presence of the feature. By predicting relatively few positive instances, it can ensure that a high proportion of its predictions are correct. The absolute number of instances identified, however, is relatively low as a proportion of the total number in the dataset.
- Specificity, similar to sensitivity, can easily mislead by identifying very few instances as having the target feature. Then, it will correctly predict most non-present instances as False, but will not identify the few instances of interest.

Clearly, the performance metric must be chosen with due consideration for the application; e.g. in testing for a medical condition, rather than incorrectly telling a patient they do not have the condition because the model predicted they were feature-negative, it is preferable to incorrectly identify some patients as feature-positive (since they can be retested). In this case, high accuracy is crucial, at the expense of precision. It is appropriate to blend together these metrics, in order to derive performance metrics which balance the priorities of the outcomes. In general – including in this thesis – the most important aspects of a ML algorithm are precision and sensitivity: a model which performs well with respect to *both* of these is sensitive to the feature, but precise in its predictions. A quantity which captures both of these is the F_β-score,

$$f_\beta = \left(1+\beta^2\right) \frac{\text{precision} \times \text{sensitivity}}{(\beta^2 \times \text{precision}) + \text{sensitivity}}, \tag{3.3}$$

where $\beta \in \mathbb{R}$ is the relative weight of priority of sensitivity with respect to precision. In particular, considering precision and sensitivity as equally important, i.e. $\beta = 1$, we have the F_1-score,

$$f_1 = \frac{2 \times \text{precision} \times \text{sensitivity}}{\text{precision} + \text{sensitivity}}. \tag{3.4}$$

For examples of how F_1-score balances these considerations, see Table 3.2.

3.1.2 *Unsupervised machine learning*

Contrary to supervised algorithms, unsupervised methods operate on *unlabelled* data, $\mathcal{D}$. This is often summarised as finding structure within unstructured data. Although we do not utilise these methods in this thesis, we briefly summarise them here for completeness; again, we can further compartmentalise methods under this umbrella as follows [112].

CLUSTERING

True positives	False negatives	False positives	Precision	Sensitivity	F_1-score
500	500	1000	33	50	$(\frac{2\times33\times50}{33+50}) = 37$
500	500	500	50	50	$(\frac{2\times50\times50}{50+50}) = 50$
1000	0	1000	50	100	$(\frac{2\times50\times100}{50+100}) = 67$
1000	0	0	100	100	$(\frac{2\times100\times100}{100+100}) = 100$

Table 3.2: Examples of how F_1-score behaves for varying true positives, false negatives and false positives.

Finding datapoints which are similar to each other, according to some distance metric.

– For example, online retailers grouping together customers with similar preferences, in order to tailor advertising campaigns.

DIMENSIONALITY REDUCTION

Reducing the feature vector of each sample in a dataset to its essential components, which may be amalgamations of original features, while retaining structure within the data.

– This can be used for visualisation to allow for inspection of complex datasets, e.g. plotting users of a social network as nodes on a 2D map, where distinct social groups are kept distant.

ASSOCIATION LEARNING

Discover correlations among data.

– For instance, a supermarket may find that purchasers of certain products are likely also to buy others, providing actionable insight. For example, purchases involving bread also include butter in 50% of cases, so positioning these nearby may increase sales by reminding consumers of their compatibility.

SEMI-SUPERVISED LEARNING

Combine elements of supervised and unsupervised algorithms, to achieve a task beyond the remit of either alone. This often means classifying data where only occur a small subset of the total dataset is labelled.

– e.g. in facial recognition, a clustering algorithm finds similarities between individual people in photos, and identifies a single person present across a number of photos, and associates those photos together. It combines this with a small set of photos for which people have been tagged, locates the same person and automatically tags them in the wider set of photos.

3.1.3 *Reinforcement learning*

A third category of ML algorithms are reinforcement learning. These are methods where an *agent* interacts with some environment, and refines a *policy* for reacting to different stimuli. As such, the agent can, in principle, deal with a wide array of situations. These methods underly technologies such as self-driving cars, which inspect their surroundings through *sensors*; compute an *action* according to the policy, and implement that action through *actuators*. Following an action, the agent senses whether that action was beneficial or detrimental, and receives a *reward* or *punishment* accordingly. Highly rewarded actions are likely to be repeated in future, allowing the agent to learn what response is appropriate to situational parameters, e.g. a self-driving car braking at a red light is rewarded, while braking at a green light is punished.

The concept of a machine's *agency* is an ongoing discussion in the ML community [113, 114], and will be important in this thesis when we define our model learning protocol as an agent; we will revisit the concept in Section 5.5.1.

3.2 QUANTUM MACHINE LEARNING

A growing domain is the development of ML algorithms which run on quantum hardware, or exploit data from quantum systems; generally referred to as quantum machine learning (QML) [115, 116]. There are a number of methodologies which are referred to as QML; for clarity, we define the main branches here, as shown in Fig. 3.2.

CLASSICAL MACHINE LEARNING

Standard ML as described in Section 3.1, i.e. where the processors are CPUs or GPUs, and the applications are not of specific interest to problems in the quantum domain. Recently, this branch has encompassed quantum *inspired* ML, which still target classical problems, but use subroutines which were orginally found in the context of QML [117].

QUANTUM ENHANCED MACHINE LEARNING

A quantum co-processor is leveraged on classical data for some provable speedup, i.e. data that could otherwise be processed purely classically, is encoded, loaded onto and processed by quantum hardware. The quantum counterparts of classical ML algorithms aim to solve the same problems, e.g. as neural networks [118, 119] and principal component analysis [120].

CLASSICAL LEARNING FOR QUANTUM SYSTEMS

Classical processors are employed to extract insight on problems arising from quantum systems, e.g. data is taken from a quantum system and analysed via purely classical methods. For instance, methods which aim to represent quantum states efficiently by leveraging neural networks [121, 122].

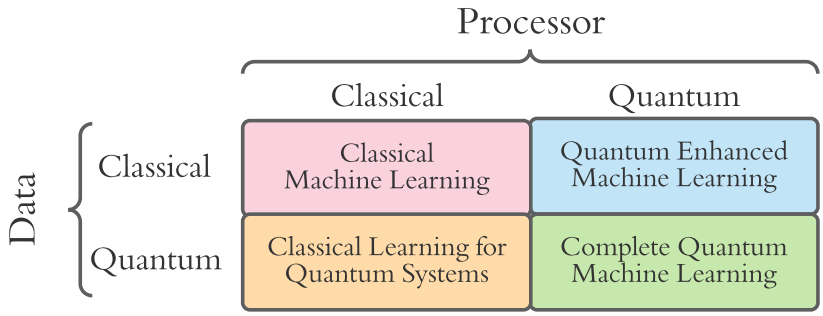


Figure 3.2: Types of quantum machine learning.

COMPLETE QUANTUM MACHINE LEARNING

Data of a quantum nature is processed – at least partially – by quantum processors. The most common technique here is variational quantum eigensolver (VQE), which simulates quantum systems on QCs, in order to retrieve quantum systems' ground states [123]. The algorithm relies on a classical optimisation routine, but was devised explicitly for implementation on quantum hardware.

The algorithms described in Part II and the applications in Parts III to IV can be described as classical learning for quantum systems. This is because the data upon which the applications are built represent quantum systems, but are processed through classical ML algorithms in order to derive insight about those systems. We caveat that it is feasible, and indeed the long term intention of such algorithms, to run in conjunction with a quantum co-processor, which would represent and evolve quantum systems, but all processing presented here are through strictly classical architecture.

3.3 GENETIC ALGORITHMS

In later chapters (Chapter 8 and Chapter 10) we will use a class of optimisation techniques known as *evolutionary algorithms* [124, 125]. In particular, *genetic algorithms (GAs)* are central to our primary applications. Here we describe genetic algorithms (GAs) in general terms for reference throughout.

GAs work by assuming a given problem can be optimised, if not solved, by a single candidate among a fixed, closed space of candidates, called the population, $\mathcal{P}$. A number of candidates are sampled at random from $\mathcal{P}$ into a single *generation*, and evaluated through some objective function (OF), which assesses the fitness of the candidates at solving the problem of interest. Candidates from the generation are then mixed together to produce the next generation's candidates: this *crossover* process aims to combine only relatively strong candidates, such that

the average candidates' fitness improve at each successive generation, mimicking the biological mechanism whereby the genetic makeup of *offspring* is an even mixture of both parents through the philosophy of *survival of the fittest*. The selection of strong candidates as parents for future generations is therefore imperative; in general parents are chosen according to their fitness as determined by the OF. Building on this biological motivation, much of the power of GAs comes from the concept of *mutation*: while offspring retain most of the genetic expressions of their parents, some elements are mutated at random. Mutation is crucial in avoiding local optima of the OF landscape by maintaining diversity in the examined subspace of the population.

GAs are not defined either as supervised or unsupervised methods; this designation depends on the OF. If candidates are evaluated with respect to labelled data, we can consider that GA supervised, otherwise unsupervised. Pseudocode for a generic GA is given in Algorithm 1, but we can informally define the procedure as follows. Given access to the population, $\mathcal{P}$,

1. Sample N_m candidates from $\mathcal{P}$ at random
   (a) call this group of candidates the first generation, μ.
2. Evaluate each candidate $\gamma_j \in \mu$
   (a) each γ_j is assigned a fitness, g_j;
   (b) the fitness is computed through an OF, g acting on the candidate, i.e. $g_j = g(\gamma_j)$.
3. Map the fitnesses of each candidate, $\{g_j\}$, to selection probabilities for each candidate, $\{s_j\}$
   (a) e.g. by normalising the fitnesses, or by removing some poorly-performing candidates and then normalising.
4. Generate the next generation of candidates
   (a) reset $\mu = \{\}$;
   (b) Select pairs of parents, $\{\gamma_{p_1}, \gamma_{p_2}\}$, from μ
      i. each candidate's probability of being chosen is given by their s_j;
   (c) cross over $\{\gamma_{p_1}, \gamma_{p_2}\}$ to produce children candidates, $\{\gamma_{c_1}, \gamma_{c_2}\}$
      i. mutate $\gamma_{c_1}, \gamma_{c_2}$ according to some random probabilistic process;
      ii. keep γ_{c_i} only if it is not already in μ, to ensure N_m unique candidates are tested at each generation;
   (d) until $|\mu| = N_m$, iterate to step (b).
5. Until the N_g^{th} generation is reached, iterate to step 2..
6. The strongest candidate on the final generation is deemed the solution to the posed problem.

Candidates are manifested as *chromosomes*, i.e. strings of fixed length, whose entries, called *genes*, each represent some proposed element of the system. In general, genes can have continuous values, although usually, and for all purposes in this thesis, genes are binary, capturing simply whether or not the gene's corresponding feature is present in the chromosome.

Algorithm 1: Genetic algorithm

Input: $\mathcal{P}$ // population of candidate solutions to given problem
Input: $g()$ // function to compute objective function
Input: map_g_to_s() // function to map fitness to selection probability
Input: select_parents() // function to select parents among generation
Input: crossover() // function to cross over two parents to produce offspring
Input: mutate() // function to mutate offspring probabilistically
Input: N_g // number of generations
Input: N_m // number of candidates per generation

Output: γ' // strongest candidate

$\mu \leftarrow \text{sample}\,(\mathcal{P}, N_m)$
for $i \in 1, ..., N_g$ **do**
  for $\gamma_j \in \mu$ **do**
    $g_j \leftarrow g(\gamma_j)$ // assess fitness of candidate
  end
  $\{s_j\} \leftarrow$ map_g_to_s($\{g_j\}$) // map fitnesses to normalised selection probability
  $\mu_c = \arg\max_{s_j}\{\gamma_j\}$ // record champion of this generation

  $\mu \leftarrow \{\}$ // empty set for next generation
  while $|\mu| < N_m$ **do**
    $p_1, p_2 \leftarrow$ select_parents($\{s_j\}$) // choose parents based on candidates' s_j
    $c_1, c_2 \leftarrow \text{crossover}(p_1, p_2)$ // generate offspring candidates based on parents
    $c_1, c_2 \leftarrow \text{mutate}(c_1, c_2)$ // probabilistically mutate offspring
    for $c \in \{c_1, c_2\}$ **do**
      if $c \notin \mu$ **then**
        $\mu \leftarrow \mu \cup \{c\}$ // keep if child is new
      end
    end
  end
end
$\gamma' \leftarrow \arg\max_{s_j}\{\gamma_j \in \mu\}$ // strongest candidate on final generation

return γ'

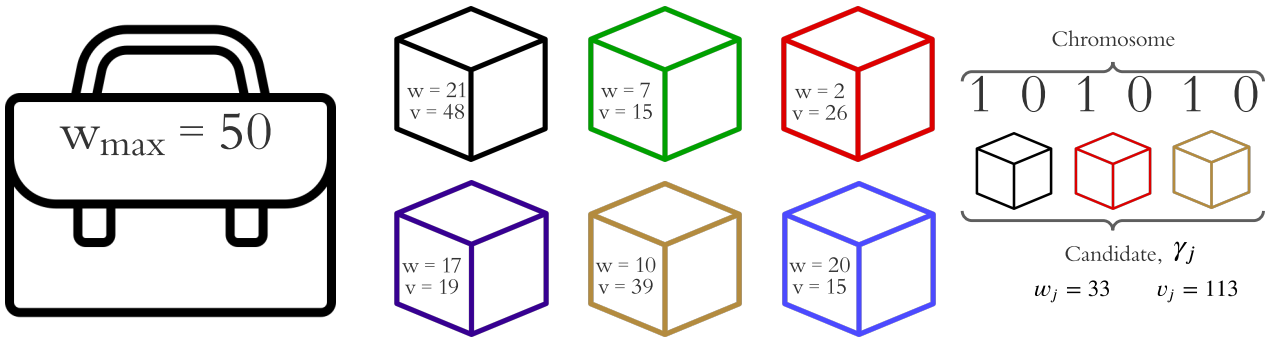


Figure 3.3: Depiction of the knapsack problem. **Left**, A knapsack which can hold any number of objects but is constrained by the total weight it can support, $w_{max} = 50$. **Centre**, A set of objects are available, each with associated weight, w, and value v. The objective is to find the subset of objects which maximise the total value, while not exceeding the capacity of the knapsack. **Right**, An example chromosome, i.e. candidate γ_j, where the bits of the chromosome indicate whether the corresponding object is included, allowing for calculation of the total weight w_j and value v_j of the candidate solution.

3.3.1 *Illustrative example: knapsack problem*

One commonly referenced combinatorial optimisation problem is the *knapsack problem*, which we will use to elucidate the abstract concepts described so far, and introduce further concepts with immediate illustration. The knapsack problem is stated as: given a set of objects, where each object has a defined weight and value, determine the set of objects to pack in a knapsack which can support a limited weight, such that the value of the packed objects is maximised. Say there are n objects; we can write the vector containing the values of those objects as $\vec{v}$, and the vector of their weights as $\vec{w}$. We can then represent configurations of objects – i.e. candidate solutions to the problem – as vectors $\vec{\gamma}_j$, whose elements are binary, and simply indicate whether or not the associated object is included in the set. The candidate vector $\vec{\gamma}_j$ is equivalent to the chromosome γ_j. For example, with $n = 6$,

$$\vec{\gamma}_j = (1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1) \implies \gamma_j = 100001 \tag{3.5}$$

indicates a set of objects consisting only of those indexed first and last, with none of the intermediate objects included.

The fitness of any candidate is then given by the total value of that configuration of objects, $v_j = \vec{v} \cdot \vec{\gamma}_j$, but candidates are only admitted if the weight of the corresponding set of objects is less than the capacity of the knapsack[4] i.e. $w_j = \vec{w} \cdot \vec{\gamma}_j \leq w_{max}$.

4 Note there are alternative strategies to dealing with candidates who violate the weight condition, such as to impose a penalty within the OF, but for our purposes let us assume we simply disregard violators.

Name	Candidate	Value	Weight	Valid
γ_1	110011	117	58	No
γ_2	101010	113	33	Yes
γ_3	011110	99	36	Yes
γ_4	011011	95	39	Yes
γ_5	111000	89	30	Yes
γ_6	010111	88	54	No
γ_7	100010	87	31	Yes
γ_8	110001	78	48	Yes
γ_9	011101	75	46	Yes
γ_{10}	110000	63	28	Yes
γ_{11}	000011	54	30	Yes
γ_{12}	000101	34	37	Yes

Table 3.3: Candidate solutions to the knapsack problem for randomly generated chromosomes.

For example where each individual object has value < 50 and weight < 25 and $w_{max} = 50$, recalling $\gamma_j = 100001$, say,

$$\vec{v} = (48 \quad 15 \quad 26 \quad 19 \quad 39 \quad 15) \implies v_j = \vec{\gamma}_j \cdot \vec{v} = 48 + 15 = 63; \tag{3.6a}$$

$$\vec{w} = (21 \quad 7 \quad 2 \quad 17 \quad 10 \quad 20) \implies w_j = \vec{\gamma}_j \cdot \vec{w} = 21 + 20 = 41. \tag{3.6b}$$

We can hence assess the fitness of γ_j as 63 and deem it a valid candidate since it does not exceed the weight threshold. We can likewise compute the total weight and value of a series of randomly generated candidates, and deem them valid or not. Table 3.3 shows a set of 12 randomly generated candidates, of which ten are valid.

The strongest (valid) candidates from Table 3.3 are 101010, 011110. We can combine these two strong candidates in order to produce further candidates: by merging the first half of the first candidate with the second half of the second candidate[5], we get the *child* candidate $\gamma_c = 101110$, from which we can see $v_{c_1} = 132, w_{c_1} = 50$, i.e. by combining two strong candidates we produce the strongest-yet-seen valid candidate.

By repeating this procedure, it is expected to uncover candidates which optimise v_j while maintining $w_j \leq w_{max}$, or at least to produce near-optimal solutions, using far less time/resources than brute-force evaluation of all candidates, which is usually sufficient. For instance, with $n = 100$ objects, there are $2^{100} \approx 10^{30}$ candidates to consider; the most powerful supercomputers in the world currently claim on the order of Exa-FLOPs, i.e. 10^{18} operations per second, of

5 This process is described as a one-point crossover in Section 3.3.3 with this example shown in Fig. 3.5.

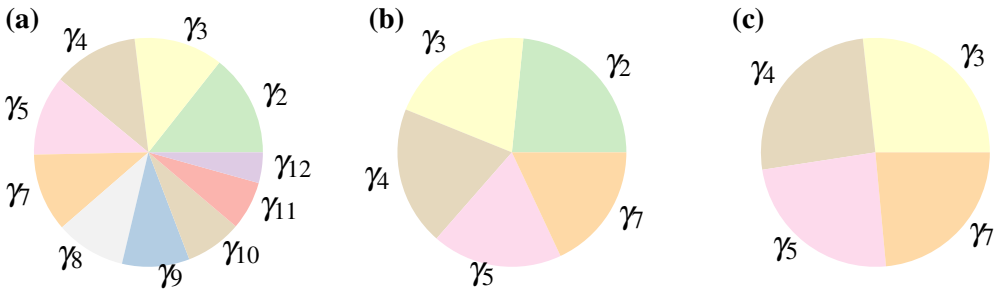


Figure 3.4: Roulette wheels showing selection probability s_j for corresponding candidates γ_j. Colours here only distinguish candidates, they do not encode any information. **a**, All valid candidates are assigned selection probability based on their value in Table 3.3. **b**, The set of potential parents is truncated to include only the strongest five candidates. **c**, After one parent (γ_2) has been chosen, it is removed from the roulette wheel and the remaining candidates' probabilities are renormalised for the selection of the second parent.

which say $\mathcal{O}(1000)$ operations are required to test each candidate, meaning 10^{15} candidates can be checked per second in a generous example. This would still require 10^{12} seconds to solve absolutely, so it is reasonable in cases like this to accept *approximately optimal* solutions[6].

3.3.2 *Selection mechanism*

A key subroutine of every GA is the mechanism through which it nominates candidates from generation μ as parents to offspring candidates in $\mu + 1$ [126]. All mechanisms have in common that they act on a set of candidates from the previous generation, where each candidate, γ_j, has been evaluated and has fitness value, g_j. Among the viable schemes for selecting individual parents from μ are

- Rank selection: candidates are selected with probabilty proportional to their ranking relative to the fitness of contemporary candidates in the same generation;
- Tournament selection: a subset of k candidates are chosen at random from μ, of which the candidate with the highest fitness is taken as the parent;
- Stochastic universal sampling: candidates are sampled proportional to their fitness, but the sampling algorithm is biased to ensure high-fitness candidates are chosen at least once within the generation.

We will only detail the mechanism used in later applications within this thesis: fitness proportional selection, known as *roulette selection* [126]. This is a straightforward strategy where we directly map candidates' fitness, g_j to a selection probability, s_j, simply by normalising $\{g_i\}$,

6 Simply put: in machine learning, *good enough* is good enough. We will adopt this philosophy for the remainder of this thesis and life.

Parent 1	Parent 2	κ	s_{ij}
γ_2	γ_3	2	$11,187\ (= 113 \times 99)$
γ_2	γ_3	3	$11,187$
γ_2	γ_3	4	$11,187$
γ_2	γ_4	2	$10,735\ (= 113 \times 95)$
γ_2	γ_4	3	$10,735$
γ_2	γ_4	4	$10,735$
		$\vdots$	
γ_5	γ_7	2	$7,743\ (= 89 \times 87)$
γ_5	γ_7	3	$7,743$
γ_5	γ_7	4	$7,743$

Table 3.4: Example of parent selection database. Pairs of parents are selected together, with the (unnormalised) selection probability, s_{ij}, given by the product of the individual candidates' fitnesses. Pairs of parents are repeated in the database for differing κ, and all κ are equally likely.

allowing us to visualise a roulette wheel of uneven wedges, each of which correspond to a candidate. Then we need only conceptually spin the roulette wheel to select the first parent, γ_{p_1}. We then remove γ_{p_1} from the set of potential parents, renormalise the remaining $\{s_j\}$, and spin the wheel again to choose the second parent, γ_{p_2}. The roulette selection is shown in Fig. 3.4.

In practice, we repeat the roulette selection process outlined until the next generation is filled, usually we have $|\mu| = N_m$, and desire that every generation should contain N_m candidates, so we repeat the roulette selection $N_m/2$ times per generation, since every pair of parents yields two offspring. It is important that meaningful differences in fitness are reflected by the selection probability, which is difficult to ensure for large N_m, e.g. with 20 candidates, the strongest candidate is only a marginally more probable parent than the worst – this effect is amplified for larger N_m. We therefore wish to reduce the set of potential parents to ensure high quality offspring: we truncate μ with rate τ to retain only the τN_m highest-fitness candidates as selectable parents.

3.3.3 *Reproduction*

When a pair of parents have been nominated by the selection mechanism above, it remains to use those parents to *reproduce*, i.e. to produce offspring which should inherit and improve upon the properties of their parents. Here we use a *one-point crossover*, whereby the two parent chromosomes are mixed together to form two offspring, about a single point, κ. For candidates of n genes, γ_{c_1} is produced when the first κ genes of γ_{p_1} are conjoined with the latter $n - \kappa$ genes of γ_{p_2}; likewise γ_{c_2} consists of the first $n - \kappa$ genes of γ_{p_2} conjoined with the latter κ genes of γ_{p_1}. Often κ is restricted to the midpoint of the chromosomes, although in general it need not

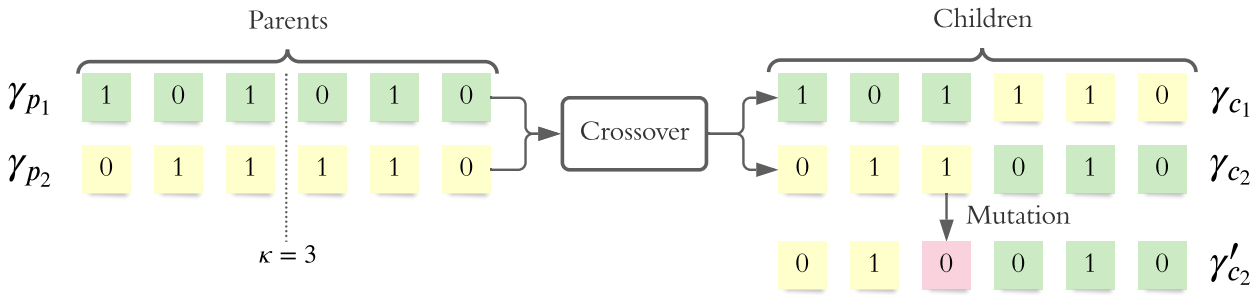


Figure 3.5: Crossover and mutation of chromosomes. Two parents, $\{\gamma_{p_1}, \gamma_{p_2}\}$, are nominated from the process in Fig. 3.4. They are then crossed-over via a one-point crossover with crossing point $\kappa = 3$, resulting in children candidates $\{\gamma_{c_1}, \gamma_{c_2}\}$. One child chromosome is mutated to yield a new candidate, γ'_{c_2}. The candidates added to the next generation are then $\{\gamma_{c_1}, \gamma'_{c_2}\}$.

be: we will instead consider $\kappa \in \left(\frac{n}{4}, \frac{3n}{4}\right)$, e.g. with $n = 12$, $\kappa \in (3, 9)$. The one-point crossover is shown for $n = 6$ with $\kappa = 3$ in Fig. 3.5, recalling the chromosome structure from Fig. 3.5.

By allowing κ other than the midpoint, we drastically increase the number of combinations of parents available for reproduction. Finally, then, parent selection is done by constructing a database of pairs of potential parents with all available crossover points, with selection probability given by the product of their individual fitnesses. This is conceptually equivalent to selection via roulette wheel as above. Recalling the fitnesses (values) of Table 3.3, we generate the parent selection database in Table 3.4.

The GA maintains diversity in the subspace of $\mathcal{P}$ it studies, by *mutating* some of the newly proposed offspring candidates. Again, there are a multitude of approaches for this step [127], but for brevity we only describe the one used in this thesis. For each proposed child candidate, γ_c, we *probabilistically* mutate each gene: with some mutation rate r_m, a mutation occurs and the child is replaced by γ'_c; that is, γ'_c is added to the next generation, and γ_c is discarded. With probability $1 - r_\mathrm{m}$ a mutation does not occur, so γ_c is passed to the subsequent generation. r_m is a *hyperparameter* of the GA: the performance of the algorithm can be optimised by finding the best r_m for a given problem.

3.3.4 *Candidate evaluation*

Within every generation of the GA, each candidate must be evaluated, so that the relative strength of candidates can be exploited in constructing candidates for the next generation. In the example of the knapsack problem, candidate solutions were evaluated by the value of their contents, but also by whether they would fit in the knapsack. Identifying the appropriate method by which to evaluate candidates is arguably the most important aspect of designing a

GA: while the choice of hyperparameters (N_g, N_m, τ, r_m) dictate the efficacy of the search, the lack of an effective metric by which to distinguish candidates would render the procedure pointless. Considerations are hence usually built into the objective function; GA implementations later in this thesis therefore demand we design objective functions with respect to the individual application.

Part II

ALGORITHMS

OVERVIEW AND CONTRIBUTION

This part details the algorithms which form the basis for the research conducted in this thesis. The corresponding software is a primary outcome of this thesis [2–4].

Chapter 4 introduces quantum Hamiltonian learning (QHL), an algorithm for the optimisation of Hamiltonian parameters when the form of the model describing a system of interest is known. This is not presented as new work, but rather as a bedrock for later discussions. The analysis and figures presented in this chapter are unique to this thesis but do not necessarily offer novel insights.

Chapter 5 builds upon QHL by posing the question: without assuming access to the model describing the target system, can we combine model training algorithms, in particular QHL, with model recovery methodologies, to learn the Hamiltonian model governing the system, and hence uncover the physics of quantum systems. This motivation leads to the Quantum Model Learning Agent (QMLA): a machine learning framework for reverse engineering models of quantum systems from their data. This protocol was initially devised by Dr. Raffaele Santagati, and developed together with myself and Drs. Andreas Gentile, Stefano Paesani, Nathan Wiebe and Chris Granade. The protocol has been published in [1], and applied to numerous case studies, which are described in later Parts.

Chapter 6 describes the implementation of QMLA through an open source software package. I was the principal designer and programmer of the codebase described, which constitutes a large portion of the output of my research. The results presented in Part III, Part IV are all achieved through this framework.

4

QUANTUM HAMILTONIAN LEARNING

First suggested in [128] and since developed [129, 130] and implemented [131], quantum Hamiltonian learning (QHL) is a machine learning algorithm for the optimisation of a given Hamiltonian parameterisation against a quantum system whose model is known a priori. Given a target quantum system, Q, known to be described by some Hamiltonian $\hat{H}(\vec{\alpha})$, QHL optimises $\vec{\alpha}$. This is achieved by interrogating Q and comparing its outputs against proposals $\vec{\alpha}_p$. In particular, an experiment is designed, consisting of an input state, $|\psi\rangle$, and an evolution time, t. This experiment is performed on Q, whereupon its measurement yields the datum $d \in \{0, 1\}$ – i.e. the eigenstate $|d\rangle \in \{|0\rangle, |1\rangle\}$ is observed – according to the expectation value $|\langle\psi| e^{-i\hat{H}_0 t} |\psi\rangle|^2$. Then, on a trusted (quantum) simulator, proposed parameters $\vec{\alpha}_p$ are encoded to the known Hamiltonian, and the same probe state is evolved for the chosen t and projected on to $|d\rangle$, i.e. $|\langle d| e^{-i\hat{H}(\vec{\alpha}_p)t} |\psi\rangle|^2$ is computed. The task for QHL is then to find $\vec{\alpha}'$ for which this quantity is close to 1 for all values of $\{|\psi\rangle, t\}$, i.e. the parameters input to the simulation produce dynamics consistent with those measured from Q.

The procedure is as follows. A *prior* probability distribution $\Pr(\vec{\alpha})$ in a parameter space of dimension $|\vec{\alpha}|$ is initialised to represent the constituent parameters of $\vec{\alpha}$. $\Pr(\vec{\alpha})$ is typically a multivariate normal (Gaussian) distribution; it is therefore necessary to pre-suppose some mean and standard deviation for each parameter in $\vec{\alpha}$. This imposes prior knowledge on the algorithm whereby the programmer must decide the range in which parameters are *likely* to fit: although QHL is generally robust and capable of finding parameters outside of this prior, the prior must at least capture the order of magnitude of the target parameters. It is important to understand, then, that QHL removes the prior knowledge of the precise parameter representing an interaction in Q, but does rely on a *ball-park* estimate thereof from which to start.

In short, QHL samples parameter vectors $\vec{\alpha}_p$ from $\Pr(\vec{\alpha})$, simulates experiments by computing the *likelihood* $|\langle d| e^{-i\hat{H}(\vec{\alpha}_p)t} |\psi\rangle|^2$ for experiments $\{|\psi\rangle, t\}$ designed by a QHL heuristic subroutine, and iteratively improves the probability distribution of the parameterisation $\Pr(\vec{\alpha})$ through standard *Bayesian inference*. A given set of $\{|\psi\rangle, t\}$ is called an *experiment*, since it corresponds to preparing, evolving and measuring Q once[1]. QHL iterates for N_E experiments. The parameter vectors sampled are called *particles*: there are N_P particles used per experiment. Each particle used incurs one further calculation of the likelihood function – this calculation, on a classical computer, is exponential in the number of qubits of the model under consideration (because each unitary evolution relies on the exponential of the $2^n \times 2^n$ Hamiltonian matrix of n qubits), Eq. (2.11). Likewise, each additional experiment incurs the cost of calculation of N_P particles, so

1 Experimentally, this may involve repeating a measurement many times to determine a majority result and to mitigate noise.

the total cost of running QHL to train a model is $\propto N_e N_p$. It is therefore preferable to use as few particles and experiments as possible, though it is important to include sufficient resources that the parameter estimates have the opportunity to converge. Access to a fully operational, trusted quantum simulator admits an exponential speedup by simulating the unitary evolution instead of computing the matrix exponential classically, via quantum algorithms for time dynamics simulations [20, 132, 133].

4.1 BAYES' RULE

Bayes' rule is used to update a probability distribution describing hypotheses, Pr(hypothesis), when presented with new information (data). That is, the probability that a hypothesis is true is replaced by the initial probability that it *was* true, Pr(hypothesis), multiplied by the likelihood that the new data would be observed were that hypothesis true, Pr(data|hypothesis), normalised by the probability of observing that data in the first place, Pr(data). It is stated as

$$\Pr(\text{hypothesis}|\text{data}) = \frac{\Pr(\text{data}|\text{hypothesis}) \times \Pr(\text{hypothesis})}{\Pr(\text{data})}. \tag{4.1}$$

We wish to represent our knowledge of Hamiltonian parameters with a distribution, $\Pr(\vec{\alpha})$: in this case hypotheses $\vec{\alpha}$ attempt to describe data, $\mathcal{D}$, measured from the target quantum system, from a set of experiments $\mathcal{E}$, so we can rewrite Bayes' rule as

$$\Pr(\vec{\alpha}|\mathcal{D};\mathcal{E}) = \frac{\Pr(\mathcal{D}|\vec{\alpha};\mathcal{E})\ \Pr(\vec{\alpha})}{\Pr(\mathcal{D}|\mathcal{E})}. \tag{4.2}$$

We can consider Eq. (4.2) at the level of single *particles*, i.e. individual vectors $\vec{\alpha}$ in the parameter space, sampled from $\Pr(\vec{\alpha})$:

$$\Pr(\vec{\alpha}_p|d;e) = \frac{\Pr(d|\ \vec{\alpha}_p;\ e)\ \Pr(\vec{\alpha}_p)}{\Pr(d|e)} \tag{4.3}$$

where

- e are the experimental controls of a single experiment, e.g. evolution time and input probe state;
- d is the datum, i.e. the (usually) binary outcome of measuring Q under conditions e;
- $\vec{\alpha}_p$ is the *hypothesis*, i.e. a single parameter vector, called a particle, sampled from $\Pr(\vec{\alpha})$;
- $\Pr(\vec{\alpha}_p|d;e)$ is the *updated* probability of this particle following the experiment e, i.e. accounting for new datum d, the probability that $\vec{\alpha} = \vec{\alpha}_0$;
- $\Pr(d|\vec{\alpha}_p;e)$ is the likelihood function, i.e how likely it is to have measured the datum d from the system assuming $\vec{\alpha}_p$ are the true parameters and the experiment e was performed;
- $\Pr(\vec{\alpha}_p)$ is the probability that $\vec{\alpha}_p = \vec{\alpha}_0$ according to the prior distribution $\Pr(\vec{\alpha})$, which we can immediately access;

- $\Pr(d|e)$ is a normalisation factor, the chance of observing d from experiment e irrespective of the underlying hypothesis such that $\sum_{\{d\}} \Pr(d|e) = 1$.

In order to compute the updated probability for a given particle, then, all that is required is a value for the likelihood function. This is equivalent to the expectation value of projecting $|\psi\rangle$ onto $|d\rangle$, after evolving $\hat{H}(\vec{\alpha}_p)$ for t, i.e.

$$\Pr(d|\vec{\alpha}; e) = |\langle d| e^{-i\hat{H}(\vec{\alpha}_p)t} |\psi\rangle|^2, \tag{4.4}$$

which can be simulated clasically or using a quantum simulator (see Section 4.3). It is necessary first to know the datum d (either 0 or 1) which was projected by Q under experimental conditions. Therefore we first perform the experiment e on Q (preparing the state $|\psi\rangle$ evolving for t and projecting again onto $\langle\psi|$) to retrieve the datum d. d is then used for the calculation of the likelihood for each particle sampled from $\Pr(\vec{\alpha})$. Each particle's probability can be updated by Eq. (4.3), allowing us to redraw the entire probability distribution. We can hence compute a *posterior* probability distribution by performing this routine on a set of N_P particles: we hypothesise N_P parameterisations $\vec{\alpha}_i$ sampled from $\Pr(\vec{\alpha})$, and update their $\Pr(\vec{\alpha}_i)$ in proportion to their likelihood. In effect, hypotheses (particles) which are found to be highly likely are given increased credence, while those with low likelihood have their credence decreased.

4.2 SEQUENTIAL MONTE CARLO

In practice, QHL samples from and updates $\Pr(\vec{\alpha})$ via sequential Monte Carlo (SMC). SMC samples the N_P particles from $\Pr(\vec{\alpha})$, and assigns each particle a weight, $w_0 = 1/N_p$. Each particle corresponds to a unique position in the parameters' space, i.e. $\vec{\alpha}_p$. Following the calculation of the likelihood, $\Pr(d|\vec{\alpha}_p; e)$, the weight of particle p is updated from its initial value of w_p^{old} by Eq. (4.5).

$$w_p^{\text{new}} = \frac{\Pr(d|\vec{\alpha}_p; e) \times w_p^{\text{old}}}{\sum_p w_p \Pr(\vec{\alpha}_p|d; e)}. \tag{4.5}$$

In this way, strong particles – with high $\Pr(d|\vec{\alpha}_p; e)$ – have their weight increased, while weak particles (low $\Pr(d|\vec{\alpha}_p; e)$) have their weights decreased, and the sum of weights remains normalised. Within a single experiment, the weights of all N_P particles are updated: we *simultaneously* update sampled particles' weights as well as $\Pr(\vec{\alpha})$. The procedure of updating particles' weights iterates for the subsequent experiment, using the *same* particles: we do *not* redraw N_P particles for every experiment. Eventually, the sum of the weights of the particles falls below a threshold, r_t, meaning that only that fraction of particles have reasonable likelihood of being $\vec{\alpha}_0$. At this stage, SMC *resamples*[2], i.e. selects new particles, according to the updated $\Pr(\vec{\alpha})$. Then, the new particles are in the range of parameters which is known to be more likely, while particles in the region of low-weight are effectively discarded. Usually, we set $r_t = 0.5$, although this hyperparameter can have a large impact on the rate of learning, so can be optimised in particular circumstances, see Fig. 4.2.

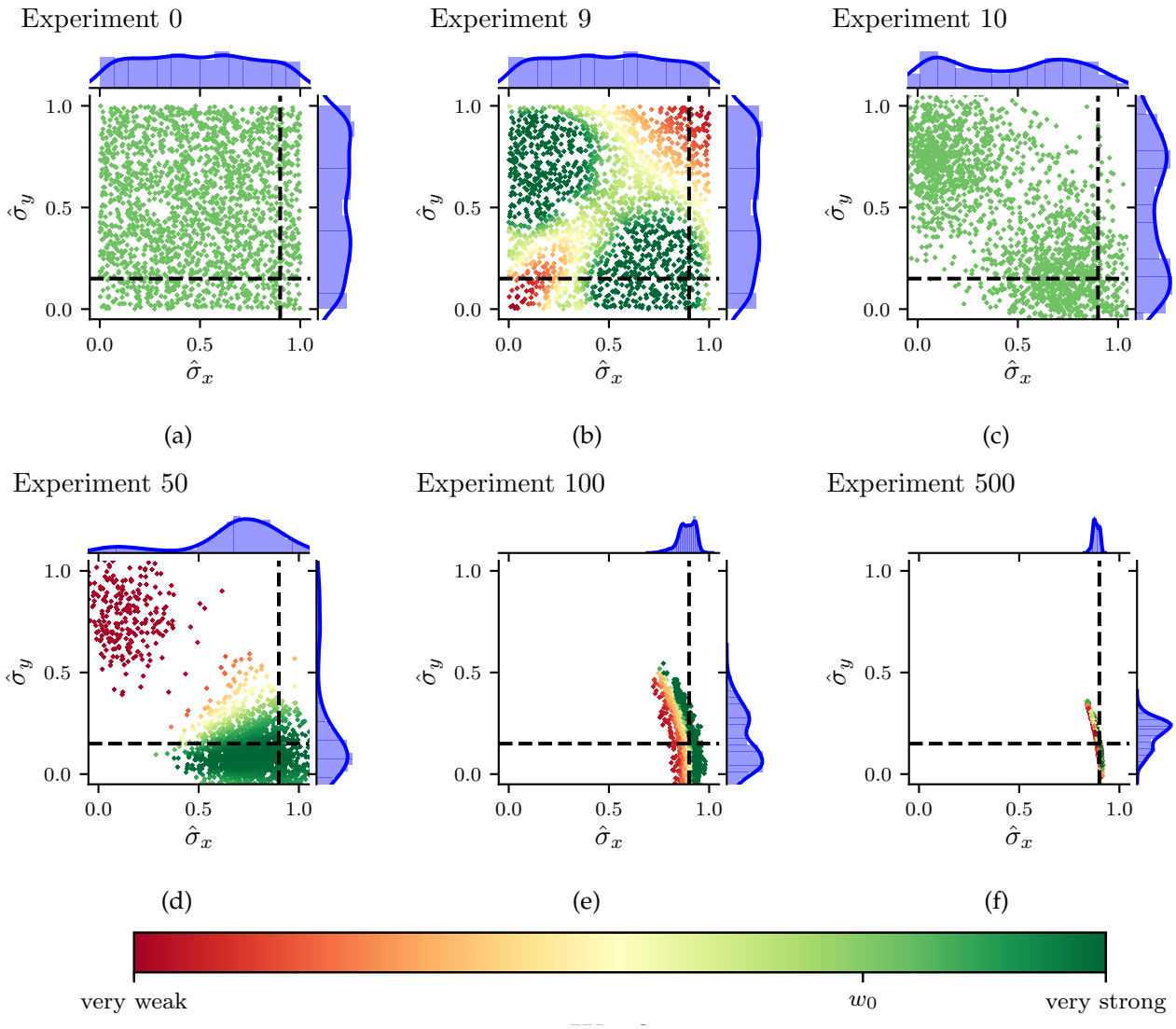


Figure 4.1: Quantum Hamiltonian learning (QHL) via sequential Monte Carlo (SMC). The studied model has two terms, $\{\hat{\sigma}_x, \hat{\sigma}_y\}$ with true parameters $\alpha_x = 0.9, \alpha_y = 0.15$ (dahsed lines), with resources $N_e = 500, N_p = 2000$ for training the model. Crosses represent particles, while the distribution $\Pr(\alpha_p)$ for each parameter can be seen along the top and right-hand-sides of each subplot. Both parameters are assigned a uniform probability distribution $\mathcal{U}(0,1)$, representing our prior knowledge of the system. **a,** SMC samples N_P particles from the initial joint probability distribution, with particles uniformly spread across the unit square, each assigned the starting *weight* w_0. At each experiment e, each of these particles' likelihood is computed according to Eq. (4.3) and its weight is updated by Eq. (4.5). **b,** after 9 experiments, the weights of the sampled particles are sufficiently informative that we know we can discard some particles while most likely retaining the true parameters. **c,** SMC resamples according the current $\Pr(\vec{\alpha})$, i.e. having accounted for the experiments and likelihoods observed to date, a new batch of N_P particles are drawn, and each reassigned weight w_0, irrespective of their weight prior to resampling. **d-e,** After further experiments and resamplings, SMC narrows $\Pr(\vec{\alpha})$ to a region around the true parameters. **f,** The final *posterior* distribution consists of two narrow distributions centred on α_x and α_y. By taking the mean of the posterior distribution, we approximate the parameters of interest as $\vec{\alpha}'$.

This procedure is easiest understood through the example presented in Fig. 4.1, where a two-parameter Hamiltonian is learned starting from a uniform distribution. $N_p = 2000$ particles are used to propose hypotheses distributed evenly throughout the parameter space, each of which are subject to weight updates as outlined above. In this example, after 9 experiments the particles around the diagonal ($x = y$) are deemed unlikely, while clusters form in the opposite corners where the algorithm finds the hypotheses credible. Before the tenth experiment, the algorithm resamples, i.e. reassigns weights based on the present $\Pr(\vec{\alpha})$. The algorithm iteratively reassigns weight to particles based on their likelihoods, redraws $\Pr(\vec{\alpha})$ and resamples. We show the state of the particles after 50, 100 and 500 experiments, with the overall result of a highly peaked parameter distribution, whose mean is near the target parameters.

4.3 LIKELIHOOD

The fundamental step within QHL is the calculation of likelihood, which enables updates of the probability distribution in Eq. (4.3). The key to the learning algorithm is that likelihood can be retrieved from the Born rule, which captures how likely a given a quantum system is to be measured in an eigenstate. When we have retrieved a datum, d, from Q, we can compute the probability that Q would be measured in the corresponding eigenstate $|d\rangle$ – this probability serves as the likelihood, and is given by Eq. (4.4).

In some cases, it is feasible to derive the closed form of the likelihood, for example as a simple expression in terms of the Hamiltonian parameters, which we will exemplify in Section 4.3.2. Closed form likelihoods allow for rapidly testing hypothetical parameters for comparison against the observed data, so QHL can feasibly be run with high N_E, N_P. In general, however, it is not possible to derive the closed form of the likelihood, and instead the likelihood must be computed through Eq. (4.4), which can be done either on a classical or quantum simulator. The case where the likelihood is computed on a quantum simulator is referred to as quantum likelihood estimation (QLE) [130, 131], and can leverage any algorithm for the calculation of Hamiltonian dynamics to achieve *quantum speedup* [14, 20, 98].

In this thesis, we do not implement the presented algorithms on quantum hardware, instead investigating their performance using idealised classical simulations, i.e. classical likelihood estimation (CLE). The reliance on classical resources demands that Eq. (4.4) be computed explicitly, notably involving the matrix exponential $e^{-i\hat{H}(\vec{\alpha}_p)t}$. Since the Hamiltonian matrix scales with the size of the simulated system, running QHL for an n-qubit systems requires exponentiation of its $2^n \times 2^n$ Hamiltonian matrix, in order to compute the exact likelihoods required for learning. This overhead restricts the applicability of CLE: $n = 11$-qubit systems' Hamiltonians exhaust the memory capacity of most conventional classical computers. In practice, QHL is limited by the computation of the total $N_e N_p$ matrix exponentials required for training:

2 Particles are *resampled* according to a resampling algorithm. Throughout this thesis, we always use the Liu-West resampling algorithm [134].

we will only entertain systems which can be represented by Hamiltonians of up to $n = 8$ qubits. In principle, larger systems could be condensed for simulation on available classical resources, or those resources used more efficiently [135], but the remit of this thesis can be fulfilled with demonstrations in the domain $n \leq 8$ qubits, so we do not endeavour to find the most effective classical strategies.

Adopting the notation used by QInfer [136], upon which our software builds, the expectation value for the unitary operator is given by

$$\Pr(0) = |\langle\psi| e^{-i\hat{H}_p t} |\psi\rangle|^2 = l(d = 0|\hat{H}_p; e). \tag{4.6}$$

In Eq. (4.6), the input basis is assigned the measurement label $d = 0$, and this $\Pr(0)$ is the probability of measuring $d = 0$, i.e. measuring the same state as was prepared as input. We assume a binary outcome model[3], i.e. that the system is measured either in $|\psi\rangle$ (labelled $d = 0$), or it is not ($|\psi_\perp\rangle, d = 1$); the likelihood for the latter case is

$$\Pr(1) = l(d = 1|\hat{H}_p; e) = \sum_{\{|\psi_\perp\rangle\}} |\langle\psi_\perp| e^{-i\hat{H}_p t} |\psi\rangle|^2 = 1 - \Pr(0). \tag{4.7}$$

Usually we will refer to the case where Q is projected onto the input state $|\psi\rangle$, so the terms *likelihood*, *expectation value* and $\Pr(0)$ are synonymous, unless otherwise stated.

4.3.1 *Interactive quantum likelihood estimation*

A fundamental result in quantum mechanics (QM) – the Loschmidt echo (LE) – shows that marginally differing Hamiltonians produce exponentially diverging evolutions, undermining the basis of QLE, i.e. that the likelihood function can inform Bayesian updates to a paramater distribution. The LE concerns the result when Q is prepared in some initial state $|\psi\rangle$, evolved forward in time by some $\hat{H}_+$, then evolved *backwards*[4] in time by $\hat{H}_-$, and projected back onto $|\psi\rangle$. The Loschmidt echo – or the *fidelity* – is given by

$$M(t) = \left|\langle\psi|e^{+i\hat{H}_- t} e^{-i\hat{H}_+ t}|\psi\rangle\right|^2. \tag{4.8}$$

$M(t)$ is dictated by the *similarity* between the two Hamiltonians. If $\hat{H}_+ = \hat{H}_-$, then $M(t) = 1$, while $\|\hat{H}_+ - \hat{H}_-\|_2 > 0$ yields $M(t) < 1$, indicating disagreement between the two Hamiltonians. The fidelity is characterised by a number of distinct regions, depending on the evolution time, t:

$$M(t) \sim \begin{cases} 1 - \mathcal{O}(t^2), & t \leq t_c \\ e^{-\mathcal{O}(t)}, & t_c \leq t \leq t_s \\ 1/\|\hat{H}\|, & t \geq t_s \end{cases} \tag{4.9}$$

3 In principle the output does not have to be binary, so we sum over the general set $\{|\psi_\perp\rangle\}$ of eigenstates orthogonal to $|\psi\rangle$ in Eq. (4.7).

4 Equivalently and in practice, evolved forward in time for $(-\hat{H}_-)$.

where $\|\hat{H}\|$ is the dimension of the Hamiltonians, and t_c, t_s are bounds on the evolution time marking the transition between the *parabolic decay*, *asymptotic decay* and *saturation* of the echo [137]. t_c and t_s generally depend on the similarity between $\hat{H}_+$ and $\hat{H}_-$: intuitively, as $\|\hat{H}_+ - \hat{H}_-\|_2$ decreases, the echo does not saturate until higher evolution times.

Recall that the Bayesian updates to the parameter distribution relies on good hypotheses receiving likelihood $l_e \approx 1$, and weak hypotheses receiving $l_e \approx 0$. The LE tells us that there is a small range of evolution times ($t \lesssim t_c$) for which even good particles may expect $l_e \approx 1$. We can exploit this effect, however: by designing experiments with $t \approx t_c$, the likelihood is extremely sensitive to the parameterisation, in that only particles close to the precise parameters will give a high likelihood in this regime. This is the basis of the particle guess heuristic, described in Section 4.6.1.

We can relate the LE to the likelihood, Eq. (4.4), by supposing $\hat{H}_- = \hat{\mathbb{1}}$. It is inescapable that the likelihoods are exponentially small if the evolution times are not short; experimentally, exponentially small expectation values demand an exponential number of measurements to approximate accurately. Furthermore, short-time experiments are known to be uninformative [129, 138]. Together, these problems render QLE unscalable. We overcome these inherent problems by using a modification of QLE: interactive quantum likelihood estimation (IQLE), the key to which is invoking a likelihood function other than Eq. (4.4) [130].

In effect, the LE guarantees that, for most t, if $\hat{H}_- \not\approx \hat{H}_+$, then $M(t) \ll 1$, while $\hat{H}_- \approx \hat{H}_+$ gives $M(t) \approx 1$. This can be exploited for learning: by taking $\hat{H}_+$ as either $\hat{H}_0$ (the true system) or $\hat{H}(\vec{\alpha})$ (particle/hypothesis), and sampling $\hat{H}_-$ from $\Pr(\vec{\alpha})$, we can adopt Eq. (4.8) as the likelihood function. Thus, both $\hat{H}_0$ and $\hat{H}(\vec{\alpha})$ have been evolved for arbitrary t, and unevolved by a common unitary, $e^{i\hat{H}_+ t}$. The likelihood that they are both measured in the same eigenstate is still a function of the overlap between the hypothesis and the true parameters, but here the informative difference between them is not drowned out by the chaotic effects captured by the LE, as it had been in QLE.

Importantly, IQLE can only be used where we can *reliably* reverse the evolution for the system under study. In order that the reverse evolution is reliable, it must be performed on a trusted simulator, restricting IQLE to cases where a coherent quantum channel exists between the target system and a trusted simulator. This automatically excludes any open quantum systems, as well as most realistic experimental setups, although such channels can be achieved [139]. The remaining application for IQLE, and correspondingly QHL in this regime, is in the characterisation of untrusted quantum simulators, which can realise such coherent channels [131]. In order to examine realistic systems, then, we will rely solely on QLE; for example QLE is sufficient for training models in our tests to characterised a real quantum system in Chapter 9.

4.3.2 *Analytical likelihood*

For some Hamiltonians, we can derive an analytical likelihood function to describe their dynamics [140, 141]. For instance, the Hamiltonian for an oscillating electron spin in a nitrogen-vacancy centre is given by

$$\hat{H}(\omega) = \frac{\omega}{2}\hat{\sigma}_z, \tag{4.10}$$

where ω is the Rabi frequency of the spin. Then, recalling that $\hat{\sigma}_z\hat{\sigma}_z = \hat{\mathbb{1}}$, so $\hat{\sigma}_z^{2k} = \hat{\mathbb{1}}$ and $\hat{\sigma}_z^{2k+1} = \hat{\sigma}_z$, using MacLaurin expansion, the unitary evolution of Eq. (4.10) is given by

$$\begin{aligned} U = e^{-i\hat{H}(\omega)t} = e^{-i\frac{\omega t}{2}\hat{\sigma}_z} &= \cos\left(\frac{\omega t\hat{\sigma}_z}{2}\right) - i\sin\left(\frac{\omega t\hat{\sigma}_z}{2}\right) \\ &= \left(\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\left(\frac{\omega t}{2}\right)^{2k}\hat{\sigma}_z^{2k}\right) - i\left(\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}\left(\frac{\omega t}{2}\right)^{2k+1}\hat{\sigma}_z^{2k+1}\right) \\ &= \left(\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\left(\frac{\omega t}{2}\right)^{2k}\right)\hat{\mathbb{1}} - i\left(\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}\left(\frac{\omega t}{2}\right)^{2k+1}\right)\hat{\sigma}_z \\ &= \cos\left(\frac{\omega t}{2}\right)\hat{\mathbb{1}} - i\sin\left(\frac{\omega t}{2}\right)\hat{\sigma}_z \end{aligned} \tag{4.11}$$

Then, evolving a probe $|\psi_0\rangle$ and projecting onto a state $|\psi_1\rangle$ gives

$$\langle\psi_1|\,U\,|\psi_0\rangle = \cos\left(\frac{\omega t}{2}\right)\langle\psi_1|\psi_0\rangle - i\sin\left(\frac{\omega t}{2}\right)\langle\psi_1|\hat{\sigma}_z|\psi_0\rangle\,. \tag{4.12}$$

By initialising and projecting into the same state, say $|\psi_0\rangle = |\psi_1\rangle = |+\rangle$, and recalling $\hat{\sigma}_z\,|+\rangle = |-\rangle$, we have

$$\begin{aligned} \langle\psi_1|\psi_0\rangle &= \langle+|+\rangle = 1 \\ \langle\psi_1|\hat{\sigma}_z|\psi_0\rangle &= \langle+|-\rangle = 0 \\ \implies \langle\psi_1|U|\psi_0\rangle &= \cos\left(\frac{\omega t}{2}\right). \end{aligned} \tag{4.13}$$

If the system measures in $|+\rangle$, we set the datum $d = 0$, otherwise $d = 1$. From Born's rule, and in analogy with Eq. (4.4), we can formulate the likelihood function, where the hypothesis is the single parameter ω, and the sole experimental control is t,

$$\Pr(d = 0|\omega;t) = |\langle\psi_1|U|\psi_0\rangle|^2 = \cos^2\left(\frac{\omega t}{2}\right); \tag{4.14a}$$

$$\Pr(d = 1|\omega;t) = 1 - \cos^2\left(\frac{\omega t}{2}\right) = \sin^2\left(\frac{\omega t}{2}\right). \tag{4.14b}$$

This analytical likelihood will underly the simulations used in the following introductions, except where explicitly mentioned.

4.4 TOTAL LOG TOTAL LIKELIHOOD

We have already used the concept of likelihood to update our parameter distribution during SMC; we can consolidate the likelihoods of all particles with respect to a single datum, d, from a single experiment e, in the *total likelihood (TL)*,

$$l_e = \sum_{p \in \{p\}} \Pr(d|\vec{\alpha}_p; e) \times w_p^{\text{old}}, \tag{4.15}$$

where w_p^{old} are the particle *weights* for the particle with parameterisation $\vec{\alpha}_p$. For each experiment, we use TL as a measure of how well the *distribution* performed overall, i.e. we care about how well all particles, $\{p\}$, perform as a collective, representative of how well $\Pr(\vec{\alpha})$ approximates the system, equivalent to the normalisation factor in Eq. (4.5) [142].

Note, we know the initial weights are normalised,

$$w_p^0 = \frac{1}{N_p} \implies \sum_p^{N_p} w_p^0 = 1, \tag{4.16}$$

so we can see

$$\begin{aligned} \Pr(d|\vec{\alpha}_p; e) \leq 1 &\implies \Pr(d|\vec{\alpha}_p; e) \times w_p^{\text{old}} \leq w_p^{\text{old}} \\ &\implies \sum_{\{p\}} \Pr(d|\vec{\alpha}_p; e) \times w_p^{\text{old}} \leq \sum_{\{p\}} w_p^{\text{old}} \leq \sum_p^{N_p} w_p^0; \\ &\implies l_e \leq 1. \end{aligned} \tag{4.17}$$

Eq. (4.17) essentially says that a good batch of particles, where on average particles perform well, will mean that most w_i are high, so $l_e \approx 1$. Conversely, a poor batch of particles will have low average w_i, so $l_e \approx 0$.

l_e are strictly positive, and because the natural logarithm is a monotonically increasing function, we can equivalently work with $\ln(l_e)$ – the log total likelihood (LTL) – since $\ln(l_a) > \ln(l_b) \iff l_a > l_b$. LTL are also beneficial in simplifying calculations, and are less susceptible to system underflow, i.e. very small values of l will exhaust floating point precision, but $\ln(l)$ will not.

In order to assess the quality of a *model*, $\hat{H}_i$, we can consider the performance of a set of particles throughout a set of experiments $\mathcal{E}$, through its total log total likelihood (TLTL),

$$\mathcal{L}_i = \sum_{e \in \mathcal{E}} \ln(l_e). \tag{4.18}$$

The set of experiments on which $\mathcal{L}_i$ is computed, $\mathcal{E}$, as well as the particles whose sum constitute each l_e, can be the same experiments on which $\hat{H}_i$ is trained, $\mathcal{E}_i$, but in general need not be. That is, $\hat{H}_i$ can be evaluated by considering different experiments than those on which it was trained. For example, $\hat{H}_i$ can be trained with $\mathcal{E}_i$ to optimise $\vec{\alpha}_i'$, and thereafter be evaluated

using a different set of experiments $\mathcal{E}_v$, such that $\mathcal{L}_i$ is computed using particles sampled from the distribution after optimising $\vec{\alpha}$, $\Pr(\vec{\alpha}'_i)$, and may use a different number of particles than the training phase.

Perfect agreement between the model and the system would result in $l_e = 1 \Rightarrow \ln(l_e) = 0$, as opposed to imperfect agreement where $l_e < 1 \Rightarrow \ln(l_e) < 0$. In all cases Eq. (4.18) is negative, and across a series of experiments, strong agreement gives low $|\mathcal{L}_i|$, whereas weak agreement gives large $|\mathcal{L}_i|$.

4.5 PARAMETER ESTIMATION

QHL is a parameter estimation algorithm, so here we introduce some methods to evaluate its performance, which we can reference in later sections of this thesis. The most obvious measure of the progression of parameter estimation is the error between the true parameterisation, $\vec{\alpha}_0$, and the approximation $\vec{\alpha} = \text{mean}\left(\Pr(\vec{\alpha})\right)$, which can be captured by a large family of loss functions. Among others, we use the quadratic loss (QL), which captures this error through the sum of the square difference between each parameters' true and estimated values symetrically. We can record the QL at each experiment of our training regime and hence track its over- or under-estimation. The QL is given by

$$L_Q(\vec{\alpha}) = \|\vec{\alpha}_0 - \vec{\alpha}\|^2 \tag{4.19}$$

where $\vec{\alpha}_0$ is the true parameterisation and $\vec{\alpha}$ a hypothesis distribution. An example of the progression of QL throughout QHL is shown in Fig. 4.2.

4.5.1 *Volume*

We also care about the range of parameters supported by $\Pr(\vec{\alpha})$ at each experiment: the volume of the particle distribution can be seen as a proxy for our certainty that the approximation $\text{mean}\left(\Pr(\vec{\alpha})\right)$ is accurate. For example, for a single parameter ω, our best knowledge of the parameter is $\text{mean}\left(\Pr(\omega)\right)$, and our belief in that approximation is the standard deviation of $\Pr(\omega)$; we can think of volume as an n-dimensional generalisation of this intuition [136, 143].

In general, a confidence region, defined by its confidence level κ, is drawn by grouping particles of *high particle density*, $\mathcal{P}$, such that $\sum_{p\in\mathcal{P}} w_p \geq \kappa$. We use the concept of *minimum volume enclosing ellipsoid* to capture the confidence region [143], calculated as in [144], which are characterised by their covariance matrix, Σ, which allows us to calculate the volume,

$$V(\Sigma) = \frac{\pi^{|\vec{\alpha}|/2}}{\Gamma(1 + \frac{|\vec{\alpha}|}{2})} \det\left(\Sigma^{-\frac{1}{2}}\right), \tag{4.20}$$

where Γ is the Gamma function, and $|\vec{\alpha}|$ is the cardinality of the parameterisation. This quantity allows us to meaningfully compare distributions of different dimension, but we must be cautious

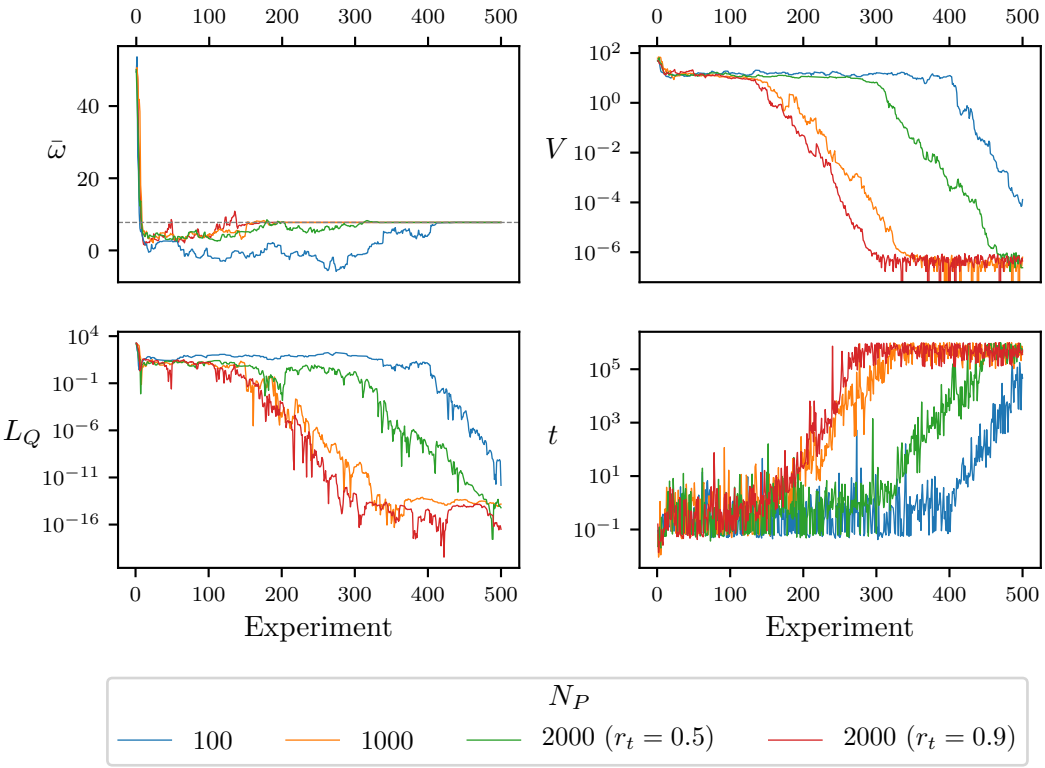


Figure 4.2: Parameter learning for the analytical likelihood, Eq. (4.14), for varying numbers of particles N_P, with $N_e = 500$. For $N_p = 2000$, we show the resampler threshold set to $r = 0.5$ and $r = 0.9$. The parameter estimate, i.e. $\bar{\omega}$, the mean of the posterior distribution after each experiment, approaching $\omega_0 = 7.75$ (dashed line), where the prior is centred on $\omega = 50 \pm 25$. For the same experiments, the volume, V, quadratic loss, L_Q, and evolution time, t, are shown. Implementation details are listed in Table A.1.

of drawing strong comparisons between models based on their volume alone, for instance because they may have started from vastly different prior distributions.

Within SMC, we assume the credible region is simply the posterior distribution, such that we can take $\Sigma = \text{cov}(\Pr(\vec{\alpha}))$ after each experiment, and hence track the uncertainty in our parameters across the training experiments [128]. We use volume as a measure of the learning procedure's progress: slowly decreasing or static volume indicates poor learning, e.g. the blue and green models in Fig. 4.2, possibly highlighting poor experiment design. Exponentially decreasing volume indicates that the parameters' estimation is improving, e.g. the first 300 experiments for the red model in Fig. 4.2, whereas converged volume (the latter 200 experiments) indicate the learning has saturated and there is little benefit to running further experiments in the current regime.

4.6 EXPERIMENT DESIGN HEURISTIC

A key consideration in QHL is the choice of experimental controls implemented in attempt to learn from the system. The experimental controls required are dictated by the choice of likelihood function used within SMC, though typically there are two primary controls we will focus on: the evolution time, t, and the *probe* state evolved, $|\psi\rangle$. The design of experiments is handled by an experiment design heuristic (EDH), whose structure can be altered to suit the user's needs, with respect to the individual target system. Usually, the EDH attempts to exploit the information available, adaptively accounting for some aspects of the inference process performed already. In some cases, however, there may be justification to employ a non-adaptive schedule, for instance to force QHL to train upon a full set of experimental data rather than a subset, as an adaptive method may advise. We can categorise each EDH as either *online* or *offline*, depending on whether it accounts for the current state of the inference procedure, i.e. the posterior. The EDH is modular and can be replaced by any method that returns a valid set of experimental controls, so we can consider numerous approaches, for instance those described in [145, 146].

4.6.1 *Particle guess heuristic*

The default EDH is the particle guess heuristic (PGH) [130], an online method which attempts to design the optimal evolution time based on the posterior at each experiment. Note PGH does not specify the probe, so is coupled with a probe selection routine to comprise a complete EDH.

The principle of PGH is that the uncertainty of the posterior limits how well the Hamiltonian is currently approximated[5], and therefore limits the evolution time for which the posterior can be expected to reasonably mimic $\hat{H}_0$. For example, consider Eq. (4.10) with a single parameter $\omega_0 = 10$, and current $\{\text{mean}\left(\Pr(\omega)\right) = 9, \text{std}\left(\Pr(\omega)\right) = 2\}$: we can expect that the approximation $\omega' = \text{mean}\left(\Pr(\omega)\right)$ is valid up to $t_{max} \approx {}^{1}/_{\text{std}(\Pr(\omega))}$. Fig. 4.3 shows the difference in likelihood between the system and proposed parameters, for parameters within the prior. For $t < t_{\text{max}}$, parameters within the prior all perform reasonably well; while for $t > t_{\text{max}}$, the prior does not provide a reliable estimation of the true likelihood. Experiments using evolution time $t \approx t_{\text{max}}$, however, provide a sensitive test of hypothesis *within* the prior: those values which are closer to ω_0 than ω' perform drastically better than those further away. We can exploit this insight directly by sampling particles throughout the prior: values around the true parameter, i.e. $\omega' < \omega < \omega_0$, will be seen to outperform $\omega < \omega'$, and will therefore be assigned higher weight (see Eq. (4.5)), allowing us to redraw $\Pr(\omega)$ based on meaningful data. In short, we can *learn* effectively from likelihoods based on experiments where $t \approx t_{\text{max}}$.

Then, it is sensible to use $t \approx t_{max}$ as the bespoke design with respect to a given prior, for two main reasons: (i) smaller times are already well explained by the posterior, so offer little

5 The reasoning behind limiting the evolution time according to the posterior distribution is rooted in the effect of the Loschmidt echo, described in Section 4.3.1.

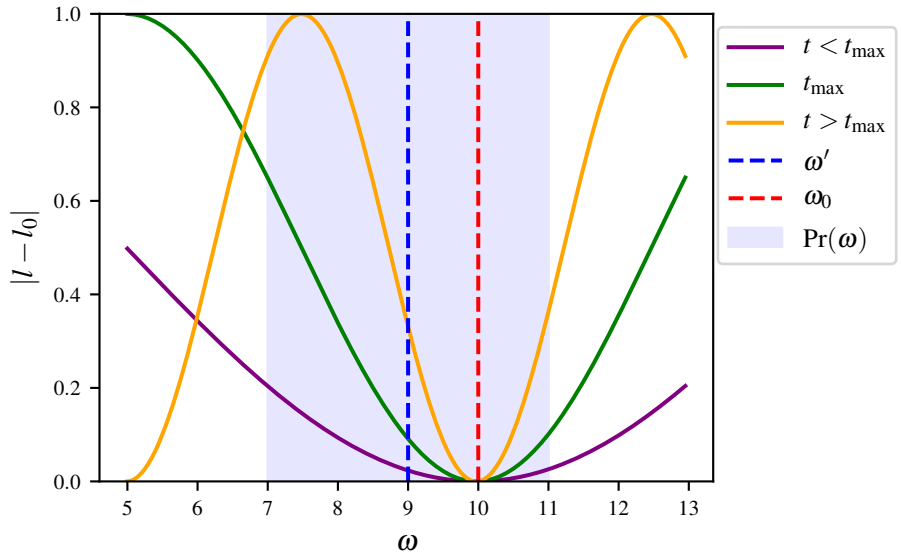


Figure 4.3: Difference in likelihood between true and proposed parameters. The likelihood is given by Eq. (4.14); the true parameter is $\omega_o = 10$ (red). The prior distribution (blue) is characterised by a Gaussian, $\Pr(\omega) = 9 \pm 2$ (arbitrary units). The difference between the true likelihood, l_0, and likelihood for proposed parameters, l, is given for values of ω, for varying proposal experiments, i.e. evolution times. $t_{\max} = {}^{1.26}/_{\text{std}(\Pr(\omega))}$ (green), $t = {}^{t_{\max}}/_{2}$ (purple), $t = 2t_{\max}$ (orange) are shown.

opportunity to learn; (ii) t_{max} is at or near the threshold which particles sampled from the posterior can comfortably explain, so it will expose the relative difference in likelihood between the posterior's better and worse particles, providing a capacity to learn. Informally, as the uncertainty in the posterior shrinks, PGH selects larger times to ensure the training is based on informative experiments while simultaneously increasing certainty about the parameters. In the one-dimensional case, this logic can be used to find an optimal time heuristic, where experiment k is assigned $t_k = {}^{1.26}/_{\text{std}(\Pr(\omega))}$ [141].

For a general multidimensional parameterisation, rather than directly using the inverse of the standard deviation of $\Pr(\vec{\alpha})$, which relies on the expensive calculation of the covariance matrix, PGH uses a proxy whereby two particles are sampled from $\Pr(\vec{\alpha})$. The experimental evolution time for experiment k is then given by

$$t_k = \frac{1}{\|\vec{\alpha}_i - \vec{\alpha}_j\|}, \tag{4.21}$$

where $\vec{\alpha}_i, \vec{\alpha}_j$ are distinct particles sampled from $\mathcal{P}$ where $\mathcal{P}$ is the set of particles under consideration by SMC after experiment $k-1$, which had been recently sampled from $\Pr(\vec{\alpha})$.

4.6.2 *Alternative experiment design heuristics*

The EDH can be specified to the requirements of the target system; we test four examples of customised EDHs against four target Hamiltonians. Here the EDH must only design the evolution time for the experiment, with probe design discussed in the next section. The heuristics tested are:

- Random$(0, t_{max})$: Randomly chosen time up to some arbitrary maximum, we set $t_{max} = 1000$ (arbitrary units). This approach is clearly subobtimal, since it does not account whatsoever for the knowledge of the training so far, and demands the user choose a suitable t_{max}, which can be not guaranteed to be meaningful.
- t list: forcing the training to consider a set of times decided in advance. For instance, when only a small set of experimental measurements are available, it is sensible to train on all of them, perhaps repeatedly. We test uniformly spaced times $t \in (0, t_{max}]$, and cycle through the list twice, aiming first to broadly learn the region of highest likelihood for all times, and then to refine the approximation. Again this EDH fails to account for the performance of the trainer so far, so may use times either far above or below the ability of the parameterisation.
- $(9/8)^k$: An early attempt to match the expected exponential decrease in volume from the training, was to set $t_k = (9/8)^k$ [128]. Note we increment k after 10 experiments in the training regime, rather than after each experiment, which would result in extremely high times which flood CPU memory.
- PGH: as described in Section 4.6.1.

We demonstrate the influence of the EDH on the training procedure by testing models[6] of various complexity and dimension in Fig. 4.4. In particular, we first test a simple 1-qubit model, Eq. (4.22a); followed by more complicated 1-qubit model, Eq. (4.22b); as well as randomly generated 5-qubit Ising, Eq. (4.22c), and 4-qubit Heisenberg models, Eq. (4.22d). Each $\hat{H}_i$ have randomly chosen parameters implicitly assigned to each term.

$$\hat{H}_1 = \hat{\sigma}_1^z \tag{4.22a}$$

$$\hat{H}_2 = \hat{\sigma}_1^x + \hat{\sigma}_1^y + \hat{\sigma}_1^z \tag{4.22b}$$

$$\hat{H}_3 = \hat{\sigma}_1^z\hat{\sigma}_3^z + \hat{\sigma}_1^z\hat{\sigma}_4^z + \hat{\sigma}_1^z\hat{\sigma}_5^z + \hat{\sigma}_2^z\hat{\sigma}_4^z + \hat{\sigma}_2^z\hat{\sigma}_5^z + \hat{\sigma}_3^z + \hat{\sigma}_4^z + \hat{\sigma}_3^z + \hat{\sigma}_5^z \tag{4.22c}$$

$$\hat{H}_4 = \hat{\sigma}_1^z\hat{\sigma}_2^z + \hat{\sigma}_1^z\hat{\sigma}_3^z + \hat{\sigma}_2^x\hat{\sigma}_3^x + \hat{\sigma}_2^z\hat{\sigma}_3^z + \hat{\sigma}_2^x\hat{\sigma}_4^x + \hat{\sigma}_3^z\hat{\sigma}_4^z \tag{4.22d}$$

6 Note the models designed here are not intended to represent physically meaningful situations, but merely to serve as examples of simulatable Hamiltonians.

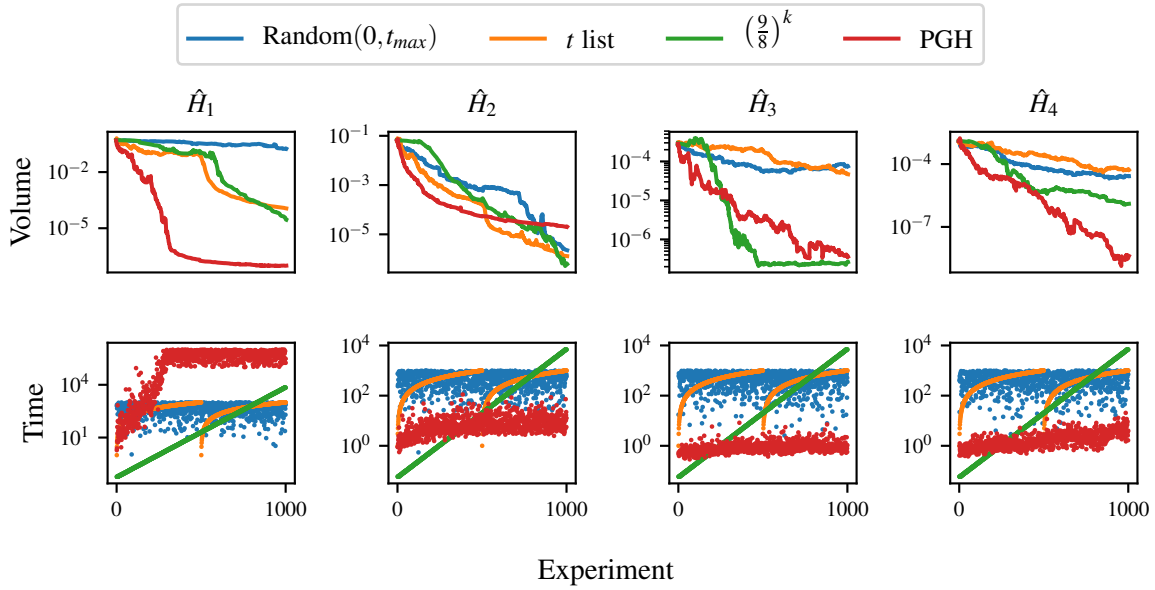


Figure 4.4: The volume (top) and evolution times (bottom) of various models when trained through QHL using different EDHs. We show models of various complexity and dimension, each trained using four heuristics, outlined in the main text. Implementation details are listed in Table A.1.

We show the performance of each of the listed EDHs in Fig. 4.4. The general trend reveals that, although some individual models benefit from bespoke EDHs, the PGH is generically applicable and usually facilitates a reasonable level of training, without providing advantage to any model. We will have cause to use alternative EDHs in particular circumstances, but we adopt PGH as the default EDH throughout this thesis, unless otherwise stated.

4.7 PROBE SELECTION

A final consideration about training experiments within QHL is the choice of input probe state, $|\psi\rangle$, which is evolved in the course of finding the likelihood used during the Bayesian update. We can consider the choice of probe as an output of the EDH, although previous work has usually not considered optimising the probe, instead usually setting $|\psi\rangle = |+\rangle^{\otimes n}$ for n qubits [131, 141]. In principle it is possible for the EDH to design a new probe at each experiment, although a more straightforward approach is to compose a set of probes offline, $\Psi = \{|\psi\rangle\}$, of size $N_\psi = |\Psi|$. Then, a probe is chosen at each experiment from Ψ, allowing for the same $|\psi\rangle$ to be used for multiple experiments within the training, e.g. by iterating over Ψ. Ψ can be generated with respect to the individual learning problem as we will examine later[7], but it is usually sufficient

7 In Section 9.4.

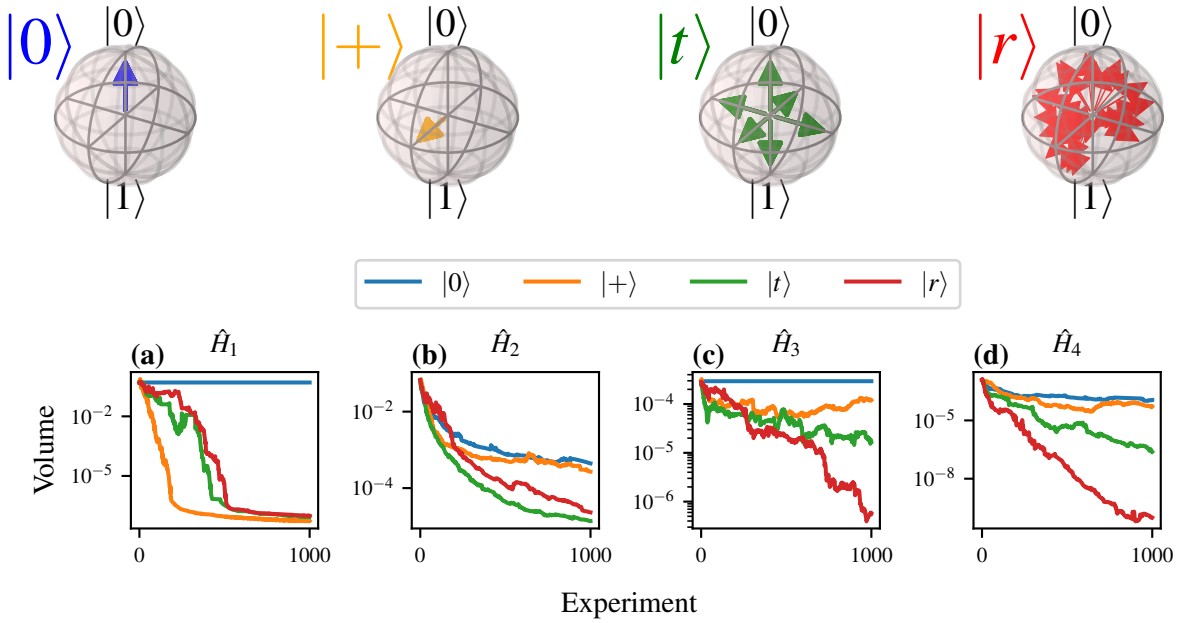


Figure 4.5: Training through QHL with varying probes. **Top,** Probes used, $\Psi = \{|0\rangle^{\otimes n}\}$ (blue); $\Psi = \{|+\rangle^{\otimes n}\}$ (orange); Ψ constructed from tomographic probes (green); Ψ random (red). We show the 1-qubit probes on the Bloch sphere, though probes are constructed up to n-qubits in each case. **Bottom,** Volume of various models, listed in Eq. (4.22), trained through QHL using different initial probe sets. In each case the probes are generated for arbitrary numbers of qubits; for $|0\rangle, |+\rangle$, the number of probes generated is $N_\psi = 1$, and for $|t\rangle, |r\rangle$, $N_\psi = 40$. Implementation details are listed in Table A.1.

to use generic strategies which should work for all models; some straightforward examples are

i. $|0\rangle$: $\Psi = \{|0\rangle^{\otimes n}\}$, $N_\psi = 1$;

ii. $|+\rangle$: $\Psi = \{|+\rangle^{\otimes n}\}$, $N_\psi = 1$;

iii. $|t\rangle$: Ψ is a random subset of probes generated by combining tomographic basis states, $N_\psi = 40$;

iv. $|r\rangle$: $|\psi\rangle$ are random, separable probes, $N_\psi = 40$.

Recalling the set of models from Eq. (4.22), we test each of these probe construction strategies in Fig. 4.5. We can draw a number of useful observations from these simple tests:

- Training on an eigenstate – as in the case for $\hat{H}_1$ and $\hat{H}_3$ using $|0\rangle$ – yields no information gain. This is because all particles give likelihoods $l = 1$, so no weight update can occur, meaning the parameter distribution does not change when presented new evidence.

- Training on an even superposition of the model's eigenstates – e.g. $|+\rangle$ for $\hat{H}_1$ – is maximally informative: any deviations from the true parameterisation are registered most dramatically in this basis, providing the optimal training probe for this case.
- These observations are reinforced by Fig. 4.5**(c)**, where a 5-qubit Ising model also fails to learn from one of its eigenstates, $|0\rangle^{\otimes 5}$. Of note, however, is that $|+\rangle^{\otimes 5}$ is not the strongest probe here: the much larger Hilbert space here can not be scanned sufficiently using a single probe; using a larger number of probes is more effective, even if those are randomly chosen. It is reasonable to presume the optimal set of training probes for any given model would be the set of probes spanned by sums of that model's eigenstates, since these will most radically encode the unitary evolution.
- In general the tomographic and random probe sets perform reliably, even for complex models.

It is an open challenge to identify the optimal probe for training any given model; the design of informative probes could be built into the EDH in principle, e.g. a set of probes could be generated of even superpositions of the candidate's eigenstates. However, for model comparison purposes in general, it is helpful to have a universal set of probes, Ψ, upon which all models are trained. The use of Ψ minimises systematic bias towards particular models, which might arise from probes which serve as favourable bases for a subset of models, for example $|+\rangle$ in Fig. 4.5**(a)**. Careful consideration should be given to N_ψ in the choice of the probe generator, since it is important to ensure probes robustly test the parameterisation across the entire Hilbert space. It is also necessary that SMC has sufficient opportunity to learn within a given subspace before moving to the next, so that slight deviations in $\Pr(\vec{\alpha})$ due to a single probe are not immediately reversed because a distant probe is immediately invoked. We can mitigate this concern by instructing the EDH to repeatedly select a probe from Ψ for a batch of successive experiments, before moving to the next available probe. Unless otherwise stated, for the remainder of this thesis we will adopt the random probe generator as the default mechanism for selecting probes, iterating between probes after batches of 5 experiments.

5 QUANTUM MODEL LEARNING AGENT

A *model* is the mathematical description of a quantum system of interest, Q. In Chapter 4, we discussed a number of systems in terms of their Hamiltonian descriptions, although in general the description of a quantum system need not be Hamiltonian, e.g. Lindbladian models describe open quantum systems, so we will generically refer to the *model* of Q throughout.

Quantum Model Learning Agent (QMLA) is an algorithm that builds upon the concept at the heart of Chapter 4, i.e. applying machine learning (ML) to the characterisation of Hamiltonians. The extension, and central question of QMLA is: if we do not know the structure of the model which describes a target quantum system, can we still learn about the physics of the system? That is, we remove the assumption about the form of the Hamiltonian model, and attempt to uncover which *terms* constitute the Hamiltonian, and in so doing, learn the interactions the system is subject to.

For the remainder of this thesis, our objective is to learn the model underlying a series of target quantum systems. We will first introduce some concepts which will prove useful when discussing QMLA, before describing the protocol in detail in Section 5.3.

5.1 MODELS

Models are simply the mathematical objects which can be used to predict the behaviour of a system. In this thesis, models are synonymous with Hamiltonians, composed of a set of *terms*, $\mathcal{T} = \{\hat{t}\}$, where each $\hat{t}$ is a matrix. Each term is associated with a multiplicative scalar, which may be referred to as that term's *parameter*: we impose order on the terms and parameters such that we can succinctly summarise any model as

$$\hat{H} = \begin{pmatrix} \alpha_0 & \dots & \alpha_n \end{pmatrix} \begin{pmatrix} \hat{t}_1 \\ \vdots \\ \hat{t}_n \end{pmatrix} = \vec{\alpha} \cdot \vec{T}, \tag{5.1}$$

where $\vec{\alpha}, \vec{T}$ are the model's parameters and terms, respectively.

For example, a model which is the sum of the (non-identity) Pauli operators is given by

$$\begin{aligned}\hat{H} &= \begin{pmatrix}\alpha_x & \alpha_y & \alpha_z\end{pmatrix} \cdot \begin{pmatrix}\hat{\sigma}_x \\ \hat{\sigma}_y \\ \hat{\sigma}_z\end{pmatrix} \\ &= \alpha_x\hat{\sigma}_x + \alpha_y\hat{\sigma}_y + \alpha_z\hat{\sigma}_z \\ &= \begin{pmatrix}\alpha_z & \alpha_x - i\alpha_y \\ \alpha_x + i\alpha_y & \alpha_z\end{pmatrix}.\end{aligned} \tag{5.2}$$

Through this formalism, we can say that the sole task of quantum Hamiltonian learning (QHL) was to optimise $\vec{\alpha}$, given $\vec{T}$. The principal task of QMLA is to identify the terms $\vec{T}$ which are supported by the most statistical evidence as describing a target system Q. In short, QMLA proposes *candidate models*, $\hat{H}_i$, as hypotheses to explain Q; we *train* each model independently through a parameter learning routine, and finally nominate the model with the best performance after training. In particular, QMLA uses QHL as the parameter learning *subroutine*, but in principle this step can be performed by any algorithm which learns $\vec{\alpha}$ for given $\vec{T}$, [147–155]. While discussing a model $\hat{H}_i$, their *training* then simply means the implementation of QHL[1], where $\hat{H}_i$ is *assumed* to represent Q, such that $\vec{\alpha}_i$ is optimised as well as it can be, even in the case it is entirely inaccurate, $\hat{H}_i \not\approx \hat{H}_0$.

5.2 BAYES FACTORS

We can use the tools introduced in Section 4.4 to *compare* candidate models. Of course it is first necessary to ensure that each model has been adequately trained: while inaccurate models are unlikely to strongly capture the system dynamics, they should first train on the system to determine their best attempt at doing so, i.e. they should undergo the process in Chapter 4. It is statistically meaningful to compare models via their total log total likelihood (TLTL), $\mathcal{L}_i$, if and only if they have considered the same data, i.e. if models have each attempted to account for the same set of experiments, $\mathcal{E}$ [156].

We can then exploit direct pairwise comparisons between models, by imposing that both models' TLTL are computed based on *any* shared set of experiments $\mathcal{E}$, with corresponding measurements $\mathcal{D} = \{d_e\}_{e\in\mathcal{E}}$. Pairwise comparisons can then be quantified by the Bayes factor (BF),

$$B_{ij} = \frac{\Pr(\mathcal{D}|\hat{H}_i;\mathcal{E})}{\Pr(\mathcal{D}|\hat{H}_j;\mathcal{E})}. \tag{5.3}$$

Intuitively, we see that the BF is the ratio of the likelihood, i.e. the performance, of model $\hat{H}_i$'s attempt to account for the data set $\mathcal{D}$ observed following the experiment set $\mathcal{E}$, against the same likelihood for model $\hat{H}_j$. BFs are known to be statistically significative of the stronger model

1 Or the chosen parameter learning subroutine.

from a pair when both models attempt to explain observed data, while favouring models of low cardinality, thereby suppressing overfitting models.

We have that, for independent experiments, and recalling Eq. (4.15),

$$\begin{aligned}\Pr(\mathcal{D}|\hat{H}_i;\mathcal{E}) &= \Pr(d_n|\hat{H}_i;e_n)\times\Pr(d_{n-1}|\hat{H}_i;e_{n-1})\times\cdots\times\Pr(d_0|\hat{H}_i;e_0)\\ &= \prod_{e\in\mathcal{E}}\Pr(d_e|\hat{H}_i;e)\\ &= \prod_{e\in\mathcal{E}}(l_e)_i.\end{aligned} \tag{5.4}$$

We also have, from Eq. (4.18)

$$\begin{aligned}\mathcal{L}_i &= \sum_{e\in\mathcal{E}}\ln\left((l_e)_i\right)\\ \implies e^{\mathcal{L}_i} &= \exp\left(\sum_{e\in\mathcal{E}}\ln\left[(l_e)_i\right]\right) = \prod_{e\in\mathcal{E}}\exp\left(\ln\left[(l_e)_i\right]\right) = \prod_{e\in\mathcal{E}}(l_e)_i.\end{aligned} \tag{5.5}$$

So we can write

$$B_{ij} = \frac{\Pr(\mathcal{D}|\hat{H}_i;\mathcal{E})}{\Pr(\mathcal{D}|\hat{H}_j;\mathcal{E})} = \frac{\prod_{e\in\mathcal{E}}(l_e)_i}{\prod_{e\in\mathcal{E}}(l_e)_j} = \frac{e^{\mathcal{L}_i}}{e^{\mathcal{L}_j}} \tag{5.6}$$

$$\implies B_{ij} = e^{\mathcal{L}_i-\mathcal{L}_j} \tag{5.7}$$

This is simply the exponential of the difference between two models' TLTLs when presented the same set of experiments. Intuitively, if $\hat{H}_i$ performs well, and therefore has a high TLTL, $\mathcal{L}_i = -10$, and $\hat{H}_j$ performs worse with $\mathcal{L}_j = -100$, then $B_{ij} = e^{-10-(-100)} = e^{90} \gg 1$. Conversely for $\mathcal{L}_i = -100, \mathcal{L}_j = -10$, then $B_{ij} = e^{-90} \ll 1$. Therefore $|B_{ij}|$ is the strength of the statistical evidence in favour of the interpretation

$$\begin{cases} B_{ij} > 1 & \Rightarrow \hat{H}_i \text{ favoured over } \hat{H}_j \\ B_{ij} < 1 & \Rightarrow \hat{H}_j \text{ favoured than } \hat{H}_i \\ B_{ij} = 1 & \Rightarrow \hat{H}_i, \hat{H}_j \text{ equally favoured.} \end{cases} \tag{5.8}$$

Throughout this thesis, Eq. (5.8) will be used to inform algorithmic preferences towards models based on pairwise comparisons. For example, for a fixed set of models, we can compute BFs between all pairs of models, and allocate a single point to the favoured model from each comparison, after which we can deem the model with most points as the best model from the set.

5.2.1 *Experiment sets*

As mentioned, it is necessary for the TLTL of both models in a BF calculation to refer to the same set of experiments, $\mathcal{E}$. There are a number of ways to achieve this, which we briefly summarise here for reference later.

During training (the QHL subroutine), candidate model $\hat{H}_i$ is trained against $\mathcal{E}_i$, designed by an experiment design heuristic (EDH) to optimise parameter learning specifically for $\hat{H}_i$; likewise $\hat{H}_j$ is trained on $\mathcal{E}_j$. The simplest method to compute the BF is to enforce $\mathcal{E} = \mathcal{E}_i \cup \mathcal{E}_j$ in Eq. (5.3), i.e. to cross-train $\hat{H}_i$ using the data designed specifically for training $\hat{H}_j$, and vice versa. This is a valid approach because it challenges each model to attempt to explain experiments designed explicitly for its competitor, at which only truly accurate models are likely to succeed.

A second approach builds on the first, but incorporates *burn–in* time in the training regime: this is a standard technique in the evaluation of ML models whereby its earliest iterations are discounted for evaluation so as not to skew its metrics, ensuring the evaluation reflects the strength of the model. In BF, we achieve this by basing the TLTL only on a subset of the training experiments. For example, the latter half of experiments designed during the training of $\hat{H}_i$, $\mathcal{E}'_i$. This does not result in less predictive BF, since we are merely removing the noisy segments of the training for each model, e.g. the first half of experiments in Fig. 4.2. Moreover it provides a benefit in reducing the computational requirements: updating each model to ensure the TLTL is based on $\mathcal{E}' = \mathcal{E}'_i \cup \mathcal{E}'_j$ requires only half the computation time, which can be further reduced by lowering the number of particles used during the update, N'_p, which will give a similar result as using N_P, assuming the posterior has mostly converged[2].

A final option is to design a set of *evaluation* experiments, $\mathcal{E}_v$, that are valid for a broad variety of models, and so will not favour any particular model. Again, this is a common technique in ML: to use one set of data for training models, and a second, unseen dataset for evaluation. This is a favourable approach: provided for each model we compute Eq. (4.18) using $\mathcal{E}_v$, we can automatically select the strongest model based solely on their TLTLs, meaning we do not have to perform further computationally-expensive updates, as required to cross-train on opponents' experiments during BF calculation. However, it does impose on the user to design a *fair* $\mathcal{E}_v$, requiring unbiased probe states $\{|\psi\rangle\}$ and times $\{t\}$ on a timescale which is meaningful to the system under consideration. For example, experiments with $t > T_2$, where T_2 is the decoherence time of the system, would result in measurements which offer little information, and hence it would be difficult to extract evidence in favour of any model from experiments in this domain. It is difficult to know, or even estimate, such meaningful time scales a priori, so it is difficult for a user to design $\mathcal{E}_v$. Additionally, the training regime each model undergoes during QHL is designed to provide adaptive experiments that take into account the specific model entertained, to choose an optimal set of evolution times, so it is likely that the set of times in $\mathcal{E}_i$ is *reasonable* by default. This approach would be favoured in principle, in the case where such constraints

2 We will verify this claim in Section 8.1.2.1, in the context of real examples.

can be accounted for, e.g. an experiment repeated in a laboratory where the available probe states are limited and the timescale achievable is understood.

5.3 QUANTUM MODEL LEARNING AGENT PROTOCOL

Given a target quantum system, Q, described by some *true model*, i.e. its Hamiltonian $\hat{H}_0$, QMLA distills a model $\hat{H}' \approx \hat{H}_0$. We can think of QMLA as a forest search algorithm[3]: consisting of a number of trees, each of which can have an arbitrary number of branches, where each leaf on each branch is an individual model. QMLA is the search for the leaf in the forest with the strongest statistical evidence of representing Q. Each tree in the QMLA forest corresponds to an independent *model search*, structured according to a bespoke exploration strategy (ES), which we detail in Section 5.4.

In short, the components of the iterative model search for a given ES, depicted in Fig. 5.1**(a-d)**, are

BRANCHES

A set of candidate models, $\{\hat{H}_i\}$, are held together on a branch, μ.

TRAINING

Each model $\hat{H}_i \in \mu$ is trained according to a parameter learning subroutine.

CONSOLIDATION

The performance of candidates in μ are ranked relative to each other, such that some models are favoured over others, for instance through selection of a *branch champion*, $\hat{H}^{\mu}_{C}$. Consolidation can rely on any statistical test, with BFs providing a robust platform to distinguish any pair of candidates.

SPAWN

A set of new models are constructed, accounting for the consolidation stage immediately beforehand, i.e. leveraging the best-yet-known models to construct improved hypotheses.

Multiple model searches can proceed in parallel, and they are each assigned an independent exploration tree (ET), S. Following the iterative model generation procedure, the protocol selects the strongest considered candidate, for instance by consolidating the set of branch champions, $\{\hat{H}^{\mu}_{C}\}$, resulting in the nomination of a single *tree champion*, $\hat{H}'_S$, Fig. 5.1**(e)**. The final step of QMLA is then to consolidate the set of champion models from all ETs, $\{\hat{H}'_S\}$, in order to declare a *global* champion model, $\hat{H}'$, Fig. 5.1**(f)**.

3 Note QMLA is not a random forest, where decision trees are added at random, because in QMLA trees are highly structured and included manually.

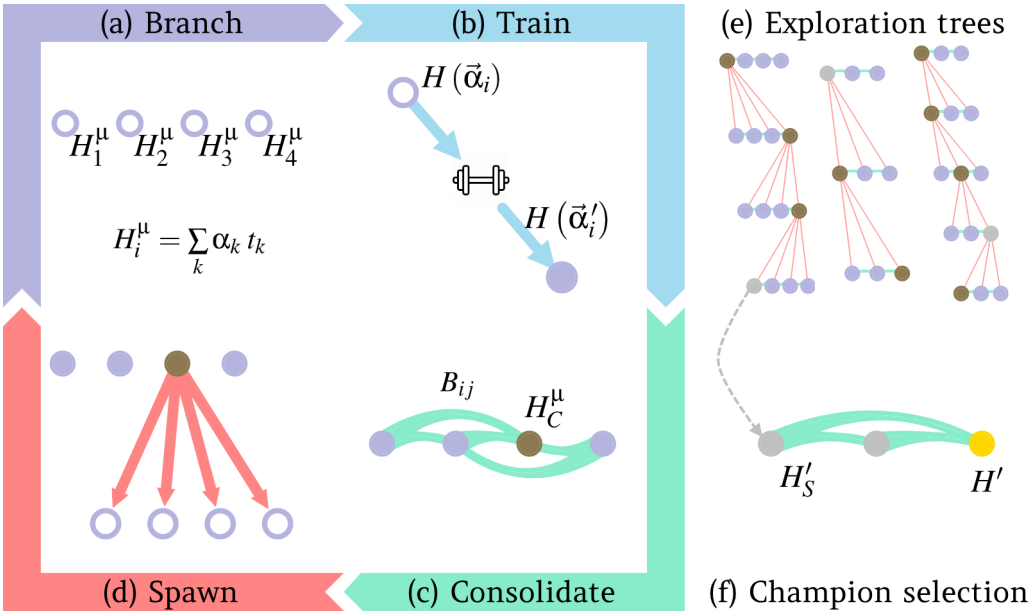


Figure 5.1: Schematic of Quantum Model Learning Agent (QMLA). **a-d,** Model search phase within an exploration strategy (ES). **a,** Models are placed as (empty, purple) nodes on the *active branch* μ, where each model is a sum of terms $\hat{t}_k$ multiplied by corresponding scalar parameters α_k. **b,** Each active model is trained according to a subroutine such as quantum Hamiltonian learning to optimise $\vec{\alpha}_i$, resulting in the trained $\hat{H}(\vec{\alpha}_i')$ (filled purple node). **c,** μ is consolidated, i.e. models are evaluated relative to other models on μ, according to the consolidation mechanism specified by the ES. In this example, pairwise Bayes factors, B_{ij}, between $\hat{H}_i, \hat{H}_j$ are computed, resulting in the election of a single branch champion $\hat{H}_C^\mu$ (bronze). **d,** A new set of models are *spawned* according to the chosen ES's model generation strategy. In this example, models are spawned from a single parent. The newly spawned models are placed on the next branch, $\mu + 1$, iterating back to **(a)**. **e-f,** Higher level of entire QMLA procedure. **e,** The model search phase for a unique ES is presented on an *exploration tree*. Multiple ES can operate in parallel, e.g. assuming different underlying physics, so the overall QMLA procedure involves a *forest search* across multiple ETs. Each ES nominates a champion, $\hat{H}_S'$ (silver), after consolidating its branch champions (bronze). **f,** $\hat{H}_S'$ from each of the above ETs are gathered on a single branch, which is consolidated to give the final champion model, $\hat{H}'$ (gold).

5.4 EXPLORATION STRATEGIES

QMLA is implemented by running N_t ETs concurrently, where each ET corresponds to a unique model search and ultimately nominates a single model as its favoured approximation of $\hat{H}_0$. An exploration strategy (ES) is the set of rules which guide a single ET throughout its model search. We elucidate the responsibilities of ESs in the remainder of this section, but in short they can be summarised as:

i. model generation: combining the knowledge progressively acquired on the ET to construct new candidate models;

ii. decision criteria for the model search phase: instructions for how QMLA should respond at predefined junctions, e.g. whether to cease the model search after a branch has completed;

iii. true model specification: detailing the terms and parameters which constitute $\hat{H}_0$ (in the case where Q is simulated);

iv. modular functionality: subroutines called throughout QMLA are interchangeable such that each ES specifies the set of functions to achieve its goals.

QMLA acts in tandem with one or more ESs, through the process depicted in Fig. 5.2. In summary: QMLA sends a request to the ES for a set of models; the ES designs models and places them as leaves on a new branch on its ET, and returns the set $\mathbb{H}$; QMLA trains the models in $\mathbb{H}$; QMLA consolidates $\mathbb{H}$; QMLA informs the ES of the results of training/consolidation of $\mathbb{H}$; ES decides whether to continue the search, and informs QMLA.

5.4.1 *Model generation*

The main role of any ES is to design candidate models to test against $\hat{H}_0$. This can be done through any means deemed appropriate, although in general it is sensible to exploit the information gleaned so far in the ET, such as the performance of previous candidates and their comparisons, so that successful models are seen to *spawn* new models, e.g. by combining previously successful models, or by building upon them. Conversely, model generation can be completely determined in advance, or entirely random. This alludes to the central design choice in composing an ES: how broad and deep should the searchable *model space* be, considering that adequately training each model is expensive, and that model comparisons are similarly expensive. The size of the model space can usually be easily found by assuming that terms are binary – either the interaction they represent is present or not. If all possible terms are accounted for, and the total set of terms is $\mathcal{T}$, then there are $2^{|\mathcal{T}|}$ available candidates in the model space. The model space encompasses the closed[4] set of models construable by the set of terms considered by an ES. Because training models is slow in general, a central aim of QMLA is to search this space efficiently, i.e. to minimise the number of models considered, while retaining

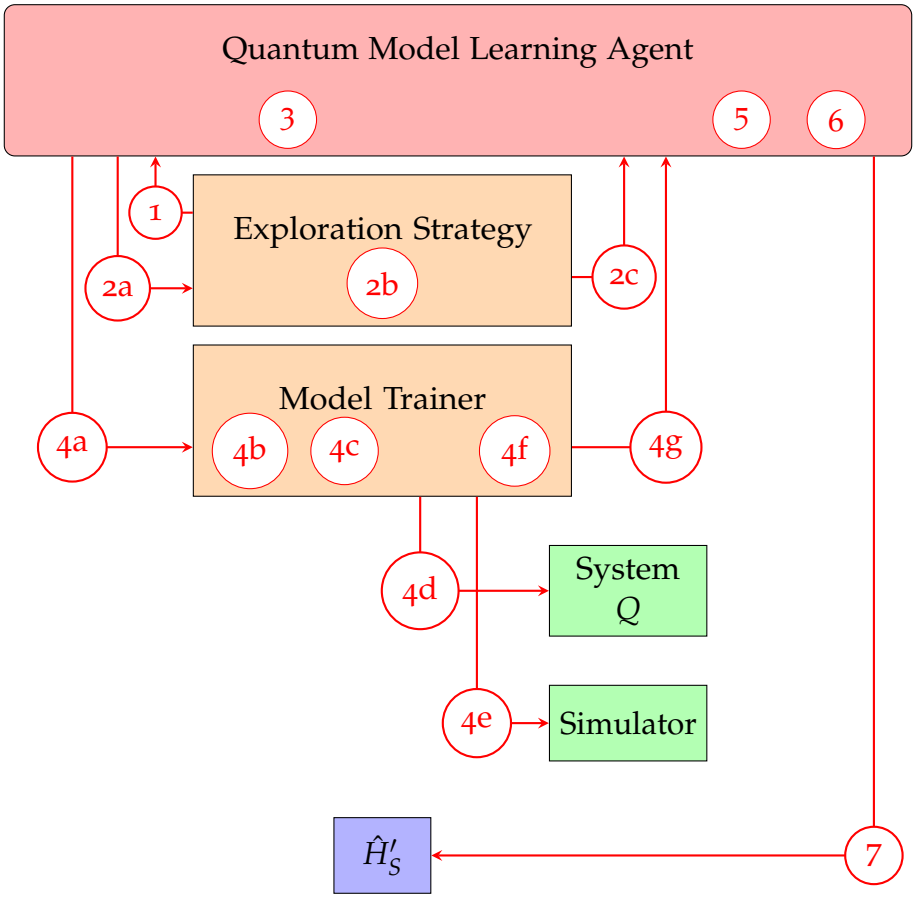


Figure 5.2: Interface between Quantum Model Learning Agent (QMLA) and a single exploration strategy (ES). The main components are the ES, model training subroutine, target quantum system Q, and (quantum) simulator. The main steps of the algorithm, shown in red with arrows denoting data transferred during that step, are as follows. **1**, QMLA retrieves decision infrastructure from ES, such as the consolidation mechanism and termination criteria. **2**, models are designed/spawned; **2a**, QMLA signals to ES requesting a set of models, passing the results of the previous branch's models if appropriate. **2b**, ES spawns new models, $\mathbb{H}$; **2c**, ES passes $\mathbb{H}$ to QMLA. **3**, QMLA assigns a new branch $(\mu \leftarrow \mu + 1)$ and places the newly proposed models upon it. **4**, Model training subroutine (here quantum Hamiltonian learning), performed independently for each model $\hat{H}_i \in \mu$; **4a**, QMLA passes $\hat{H}_i$ to the model trainer; **4b**, construct a prior distribution $\Pr_i(\vec{\alpha})$ describing the model's parameterisation $\vec{\alpha}_i$; **4c**, design experiment e to perform on Q to optimise $\vec{\alpha}_i$; **4d**, perform e on Q to retrieve a datum d; **4e**, simulate e for particles $\{\vec{\alpha}_1, \ldots, \vec{\alpha}_{N_p}\}$ sampled from $\Pr_i(\vec{\alpha})$ to retrieve likelihoods for each particle $\{l_e^j\}_{j\in(1,\ldots N_P)}$; **4e**, update the prior $\Pr_i(\vec{\alpha})$ based on $\{(d, l_e^j)\}_{j\in(1,\ldots,N_P)}$. **5**, Evaluate and rank $\hat{H}_i \in \mu$ according to the ES's consolidation mechanism. **6**, Check ES's termination criteria; if reached, proceed to **(7)**, otherwise return to **(2)**. **7**, Nominate champion model, $\hat{H}'_S$.

high quality models and providing a reasonable prospect of uncovering the true model, or a strong approximation thereof.

5.4.2 *Decision criteria for the model search phase*

Further control parameters, which direct the growth of the ET, are set within the ES. At several junctions within Algorithms 2 to 3, QMLA queries the ES in order to decide what happens next. Here we list the important cases of this behaviour.

PARAMETER-LEARNING SETTINGS

- such as the prior distribution to assign each parameter during QHL, and the parameters needed to run sequential Monte Carlo (SMC);
- the time scale on which to examine Q;
- the input probes to train upon, Ψ, described in Section 4.7.

BRANCH CONSOLIDATION STRATEGY

- How to consolidate models within a branch. Some examples used in this work are:

* a points-ranking, where all candidates are compared via BF, and points are assigned to the favoured model in each case, according to Eq. (5.8);
* ranking reflecting each model's log-likelihood (Eq. (5.5)) after training;
* models are ranked according to some objective function, as in the case of genetic algorithms (GAs) which we detail in Chapter 8.

MODEL SEARCH TERMINATION CRITERIA

- For example, instruction to stop after a fixed number of iterations, or when a certain fitness has been reached.

CHAMPION NOMINATION

- when a single ET is explored, identify a single champion model from the branch champions, $\{\hat{H}_C^\mu\}$;
- if multiple ETs are explored, the mechanism to compare champions across trees, $\{\hat{H}_S'\}$.

4 It is feasible to define an ES which uses an open model space, that is, there is no pre-defined $\mathcal{T}$, but rather the ES determines models through some other heuristic mechanism. In this thesis, we do not propose any such ES, but note that the QMLA framework facilitates the concept, see Chapter 6.

5.4.3 *True model specification*

It is necessary also to specify details about the true model, $\hat{H}_0$, at least in the case where QMLA acts on simulated data. Within the ES, we can set $\vec{T}_0$ as well as $\vec{\alpha}_0$. For example where the target system is an untrusted quantum simulator to be characterised, S_u, by interfacing with a trusted (quantum) simulator S_t, we decide some $\hat{H}_0$ in advance: the model training subroutine calls for likelihoods, those corresponding to $\hat{H}_0$ are computed S_u, while particles' likelihoods are computed on S_t.

5.4.4 *Modular functionality*

Finally, there are a number of fundamental subroutines which are called upon throughout the QMLA algorithm. These are written independently such that each subroutine has a number of available implementations. These can be chosen to match the requirements of the user, and are set via the ES.

MODEL TRAINING PROCEDURE

Subroutine used to optimise parameters for candidate models.

- i.e. whether to use QHL or quantum process tomography, etc.
- In this work we always use QHL.

LIKELIHOOD FUNCTION

The method used to estimate the likelihood for use during quantum likelihood estimation (QLE) within QHL, which ultimately depends on the measurement scheme.

- The role of these functions is to compute the probability of measuring each experimental outcome.
- These functions compute the *expectation value* of the unitary operator, $e^{-i\hat{H}t}$, corresponding to the dynamics of either Q or the hypothesis model.
- By default, we use projective measurement back onto the input probe state, $|\langle\psi| e^{-i\hat{H}t} |\psi\rangle|^2$.
- In the usual case where Q has binary outcomes, we label one outcome – say, measurement in the $|+\rangle$ state – as $d = 0$ and compute $\Pr(0)$ so that the likelihood, expectation value and $\Pr(0)$ refer to the same quantity, see Section 4.3.
- It is possible instead to implement any measurement procedure, for example an experimental procedure where the environment is traced out, as we address in Chapter 9.

PROBE

Defining the input probes to be used during training, Ψ, see Section 4.7.

– In general it is preferable to use numerous probes in order to avoid biasing particular terms.

– In some cases we are restricted to a small number of available input probes, e.g. to match experimental constraints.

EXPERIMENT DESIGN HEURISTIC

The method through which to design bespoke experiments to maximise the information on which models are individually trained, described in Section 4.6.

– In particular, in this work the experimental controls consist solely of $\{|\psi\rangle, t\}$.

– Currently, probes are generated offline, but in principle it is feasible to choose optimal probes based on available or hypothetical information. For example, probes can be chosen as a normalised sum of the candidate model's eigenvectors.

– Choice of t has a large effect on how well the model can train. By default, times are chosen proportional to the inverse of the current uncertainty in $\vec{\alpha}$ to maximise Fischer information, through the multi-particle guess heuristic described in Section 4.6.1 [130].

∗ Alternatively, evolution times may be chosen from a fixed set in order to force QHL to reproduce the dynamics within those times' scale. For instance, if a small amount of experimental data is available offline, it is sensible to train all candidate models against the entire dataset.

MODEL TRAINING PRIOR

Specify the structure of the prior distribution, e.g. Fig. 4.1**(a)**.

– Set the initial mean and standard deviation of each parameter separately to define the prior multi-dimensional $\Pr(\vec{\alpha})$.

5.4.5 *Exploration strategy examples*

To solidify the concept of ESs, and how they affect the overall reach and runtime of a given ET, consider the following examples, where each strategy specifies how models are generated, as well as how trained models are consolidated within a branch. Recall that all of these strategies rely on QHL as the model training subroutine, so that the run time for training, is $t_{\text{QHL}} \sim N_e N_p t_{U(n)}$, where $t_{U(n)}$ is the time to compute the unitary evolution via the matrix exponential for an n-qubit model. All models are trained using the default likelihood in Eq. (4.4). Assume the conditions

- all models considered are represented by 4-qubit models;
  - $t_{U(4)} \approx 10^{-3}$sec.
- each model undergoes a reasonable training regime;
  - $N_e = 1000, N_p = 3000$;
  - $\implies t_{\text{QHL}} = N_e \times N_p \times t_{U(4)} = 3000s \approx 1\text{h}$;
- Bayes factor calculations use
  - $N_e = 500, N_p = 3000$
  - $\implies t_{\text{BF}} \approx 2 \times 500 \times 3000 \times 10^{-3} \approx 1\text{h}$;
- there are 12 available terms
  - allowing any combination of terms, this admits a model space of size $2^{12} = 4096$
- access to 16 computer cores to parallelise calculations over
  - i.e. we can train 16 models or perform 16 BF comparisons in 1h.

Then, consider the following model generation/comparison strategies.

a. Predefined set of 16 models $\mathbb{H}$, with BF comparisons between every pair of models
   (i) Training takes 1h, and there are $\binom{16}{2} = 120$ comparisons spread across 16 processes, requiring 8h
   (ii) total time is 9h.

b. Generative procedure for model design, comparing every pair of models, running for 12 branches
   (i) One branch takes 9h $\implies$ total time is $12 \times 9 = 108$h;
   (ii) total number of models considered is $16 \times 12 = 192$.

c. Generative procedure for model design, where less model comparisons are needed (say one third of all model pairs are compared), running for 12 branches
   (i) Training time is still 1h
   (ii) One third of comparisons, i.e. 40 BF to compute, requires 3h
   (iii) One branch takes $4h$ $\implies$ total time is 36h
   (iv) total number of models considered is also 192.

These examples illustrate some of the design decisions involved in composing an ES, namely whether timing considerations are more important than thoroughly exploring the model space. They also show considerable time–savings in cases where it is acceptable to forego all model comparisons. The approach in (a) is clearly limited in its applicability, mainly in that there is a heavy requirement for prior knowledge, and it is only useful in cases where we either know $\hat{H}_0 \in \mathbb{H}$, or would be satisfied with approximating $\hat{H}_0$ as the closest available $\hat{H}_j \in \mathbb{H}$. On the opposite end of this spectrum, (c) is an excellent approach with respect to minimising prior knowledge required by the algorithm, although at the significant expense of testing a

much larger number of candidate models. There is no optimal strategy: each use case demands particular considerations, and the amount of prior information available informs how wide the model search should reach.

5.5 GENERALITY

Several aspects of QMLA are deliberately vague in order to facilitate generality.

MODEL

Any description of a quantum system which captures the interactions it is subject to.

– Here we exclusively consider Hamiltonian models, but Lindbladian models can also be considered as generators of quantum dynamics.

MODEL TRAINING

Any subroutine which can train a given model, i.e. optimise a given parameterisation under the assumption that it represents the target system.

– Currently only QHL has been implemented, although for example tomography is valid in principle, with its own advantages and disadvantages. Overall QHL is found to fulfil the remit of model training with a balance of efficiency and rigour [1].

– QHL relies on the calculation of a characteristic likelihood function; this too is not restricted to the generic form of Eq. (4.4) and can be replaced by any form which represents the likelihood that experimental conditions e result in measurement datum d. We will see examples of this in Chapter 9 where we trace out part of the system in order to represent open systems.

CONSOLIDATION

As rigorous as desired by the user.

– Consolidation occurs at the branch level of each ET, but also in finding the tree champion, and ultimately the global champion.

– In practice, we use either BF or a related concept such as TLTL which are statistically significative. However, in Chapter 8 we will consider a number of alternative schemes for discerning the strongest models.

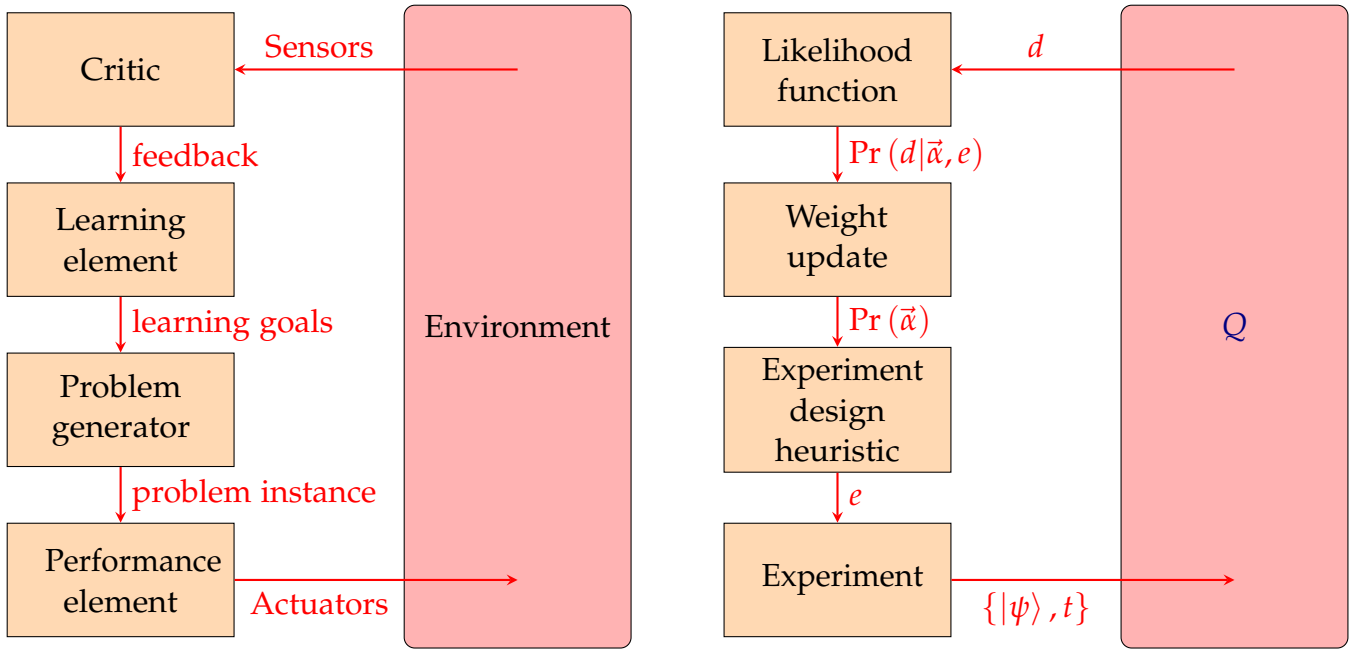


Figure 5.3: Learning agents. **Left,** definition of a learning agent, where an *environment* is affected by *actuators* which realise a *problem instance*, designed by a *problem generator*, through some *performance element*. The result of the agent's action is detected by *sensors*, which the *critic* interprets with respect to the agent's *learning goals*, by providing *feedback* to the *learning element*. **Right,** mapping of the concept of a learning agent on to an individual model. A target quantum system, Q, is queried by performing some experiment e, designed by an experiment design heuristic, and implemented by evolving a probe state $|\psi\rangle$ for time t. The systems is measured, and the datum d is sent to the likelihood function, which sends the likelihood $\Pr(d|\vec{\alpha}, t)$ to the weight update (and the parameter distribution update), before designing another experiment.

5.5.1 *Agency*

While the concept of *agency* is contentious [113], we can view our overall protocol as a multi-agent system [114], or even an agent based evolutionary algorithm [157], because any given ES satisfies the definition, *the population of individuals can be considered as a population of agents*, where we mean the population of models present on a given ET. More precisely, we can view individual models as *learning agents* according to the criteria of [103], i.e. that a learning agent has

- a *problem generator*: designs actions in an attempt to learn about the system – this is precisely the role of the EDH;
- a *performance element*: implements the designed actions and measures the outcome – the measurement of a datum following the experiment designed by the EDH;
- a *critic*: the likelihood function informs whether the designed action (experiment) was successful;
- a *learning element*: the updates to the weights and overall parameter distribution improve the model's performance over time.

We depict this analogy in Fig. 5.3.

Overall, the QMLA model search can be regarded as an unsupervised ML algorithm, since the true model is unknown. However, the flexibility of the ES paradigm permits any model design mechanism: in general we can say the model search – and the model training subroutine – is *guided* by the environment, i.e. it learns whether subspaces of candidates are effective from interaction with the system. We are therefore justified in labelling the entire procedure as the *quantum model learning agent*.

5.6 ALGORITHMS

We conclude this chapter by listing the algorithms used most frequently, in order to clarify each of their roles, and how they interact. Algorithm 2 shows the overall QMLA algorithm, which is simplified greatly to a loop over the model search of each ES. The model search itself is listed in Algorithm 3, which contains calls to subroutines for model learning (QHL, Algorithm 6), branch consolidation (which can be based upon BF, Algorithm 7) and centers on the generation of new models, an example of which – based on a *greedy search* prerogative – is given in Algorithm 5.

Algorithm 2: Quantum Model Learning Agent

Input: Q // some physically measurable or simulatable quantum system
Input: $\mathbb{S}$ // set of exploration strategies
Output: $\hat{H}'$ // champion model

$\mathbb{H}_c \leftarrow \{\}$
for $S \in \mathbb{S}$ **do**
  $\hat{H}'_S \leftarrow$ model_search(Q, S) // run model search for this ES, e.g. Algorithm 3
  $\mathbb{H}_c \leftarrow \mathbb{H}_c \cup \{\hat{H}'_S\}$ // add ES champion to collection
end
$\hat{H}' \leftarrow$ final_champion($\mathbb{H}_c$)
return $\hat{H}'$

Algorithm 3: Exploration strategy subroutine: model_search

Input: Q // some physically measurable or simulatable quantum system
Input: S // exploration strategy: collection of rules/subroutines
Output: $\hat{H}'_S$ // exploration strategy's nominated champion model

$\nu \leftarrow \{\}$
$\mathbb{H}_c \leftarrow \{\}$
while !S.terminate() **do**
  $\mu \leftarrow$ S.generate_models(ν) // spawn new models, e.g. Algorithm 5
  for $\hat{H}_i \in \mu$ **do**
    $\hat{H}'_i \leftarrow$ S.train($\hat{H}_i$) // train candidate model, e.g. Algorithm 6
  end
  $\nu \leftarrow$ S.consolidate(μ) // consolidate set of models, e.g. pairwise via Algorithm 4
  $\hat{H}^{\mu}_c \leftarrow$ S.branch_champion(ν) // use ν to select a branch champion
  $\mathbb{H}_c \leftarrow \mathbb{H}_c \cup \{\hat{H}^{\mu}_c\}$ // add branch champion to collection
end
$\hat{H}'_S \leftarrow$ S.nominate_champion($\mathbb{H}_c$)
return $\hat{H}'_S$

Algorithm 4: Exploration strategy subroutine: consolidate (points per Bayes factor win)

Input: μ // information about models considered to date
Input: b // threshold for sufficient evidence that one model is favoured
Input: BF() // function to compute the BF between $\hat{H}_j$ and $\hat{H}_k$, Algorithm 7
Output: $\hat{H}_{k'}$ // favoured model within μ

$\mathbb{H} \leftarrow$ extract_models(μ)
for $\hat{H}_j \in \mathbb{H}$ **do**
  $s_j = 0$ // initialise score for each model
end
for $\hat{H}_j, \hat{H}_k \in \mathbb{H}$ // pairwise Bayes factor between all models in the set
do
  $B \leftarrow \mathrm{BF}(\hat{H}_j, \hat{H}_k)$
  if $B > b$ // increase score of winning model
  then
    $s_j \leftarrow s_j + 1$
  else if $B < 1/b$ **then**
    $s_k \leftarrow s_k + 1$
end
$k' \leftarrow \max_k\{s_k\}$ // find which model has most points
return $\hat{H}_{k'}$

Algorithm 5: Exploration strategy subroutine: generate_models (greedy spawn)

Input: ν // information about models considered to date
Input: $\mathcal{T}$ // set of terms to search
Output: $\mathbb{H}$ // set of candidate models

$\hat{H}_C^{\mu} \leftarrow$ top_model(ν) // find previous branch champion
$\{\hat{t}_c^{\mu}\} \leftarrow$ get_terms($\hat{H}_C^{\mu}$) // extract terms of branch champion
$\mathcal{T}' \leftarrow \mathcal{T} \setminus \{\hat{t}_c^{\mu}\}$ // remove terms of branch champion

$\mathbb{H} \leftarrow \{\}$
for $\hat{t} \in \mathcal{T}'$ **do**
  $\hat{H}_i \leftarrow \hat{H}_C^{\mu} + \hat{t}$ // add one term to the branch champion
  $\mathbb{H} \leftarrow \mathbb{H} \cup \{\hat{H}_i\}$
end
return $\mathbb{H}$

Algorithm 6: Quantum Hamiltonian Learning

Input: Q // some physically measurable or simulatable quantum system, described by $\hat{H}_0$
Input: $\hat{H}_i$ // Hamiltonian model attempting to reproduce data from $\hat{H}_0$
Input: $\Pr(\vec{\alpha})$ // probability distribution for $\vec{\alpha} = \vec{\alpha}_0$
Input: N_E // number of experiments to iterate learning procedure for
Input: N_P // number of particles to draw from $\Pr(\vec{\alpha})$
Input: $\Lambda(\Pr(\vec{\alpha}))$ // experiment design heuristic
Input: $\mathrm{RS}(\Pr(\vec{\alpha}), N_p)$ // resampling algorithm for redrawing N_p particles from $\Pr(\vec{\alpha})$
Output: $\vec{\alpha}'$ // estimate of Hamiltonian parameters

$\mathcal{P} \leftarrow \mathrm{RS}(\Pr(\vec{\alpha}), N_p)$ // sample particles from prior
for $p \in \mathcal{P}$ **do**
  $w_p \leftarrow 1/N_p$ // set weights for each particle
end

for $e \in \{1 \rightarrow N_e\}$ **do**
  $t, |\psi\rangle \leftarrow \Lambda(\Pr(\vec{\alpha}))$ // design an experiment
  $d \leftarrow$ prepare Q in $|\psi\rangle$, evolve and measure after t // datum
  for $p \in \mathcal{P}$ **do**
    $\vec{\alpha}_p \leftarrow$ parameter vector corresponding to p
    $\Pr(d|\vec{\alpha}_p; t) \leftarrow |\langle d| e^{-iH(\vec{\alpha}_p)t} |\psi\rangle|^2$ // likelihood
    $w_p \leftarrow w_p \times \Pr(d|\vec{\alpha}_p; t)$ // weight update
  end

  if $1/\sum_p w_p^2 < N_p/2$ // check whether to resample (are weights too small?)
  then
    $\mathcal{P} \leftarrow \mathrm{RS}(\Pr(\vec{\alpha}), N_p)$ // redraw particles via resampling algorithm
    for $p \in \mathcal{P}$ **do**
      $w_p \leftarrow 1/N_p$ // set weights for each particle
    end
  end
end

$\vec{\alpha}'_i \leftarrow \mathrm{mean}(\Pr(\vec{\alpha})) \leftarrow \vec{\alpha}'$
return $\vec{\alpha}'$

Algorithm 7: Bayes Factor calculation

Input: Q // some physically measurable or simulatable quantum system.
Input: $\hat{H}_j, \hat{H}_k$ // Hamiltonian models to compare
Input: $\Pr_j(\vec{\alpha}), \Pr_k(\vec{\alpha})$ // posterior distribution following training for $\hat{H}_j, \hat{H}_k$
Input: N'_p // number of particles to draw from posteriors for evaluation
Input: $\mathcal{E}_j, \mathcal{E}_k$ // experiments on which $\hat{H}_j$ and $\hat{H}_k$ were trained during QHL
Output: B_{jk} // Bayes factor between two candidate Hamiltonians

$\mathcal{E} = \{\mathcal{E}_j \cup \mathcal{E}_k\}$ // common experiments for fair comparison

for $\hat{H}_i \in \{\hat{H}_j, \hat{H}_k\}$ **do**
  $\mathcal{L}_i = 0$ // total log total likelihood for $\hat{H}_i$
  for $e \in \mathcal{E}$ **do**
    $e \leftarrow t, |\psi\rangle$ // assign evolution time and probe from experiment control set
    $d \leftarrow$ Prepare Q in $|\psi\rangle$, evolve and measure after t // datum

    $\mathcal{P} \leftarrow \text{sample}(\Pr_i(\vec{\alpha}), N'_p)$ // sample from $\hat{H}_i$'s posterior
    $l_e \leftarrow 0$ // total likelihood for $\hat{H}_i$ on this experiment

    for $\vec{\alpha}_p \in \mathcal{P}$ **do**
      $\Pr(d|\hat{H}_i, t) \leftarrow \left|\langle d| e^{-i\hat{H}_i(\vec{\alpha}_p)t} |\psi\rangle\right|^2$ // likelihood for particle $\vec{\alpha}_p$ on e
      $l_e \leftarrow l_e + \Pr(d|\hat{H}_i, t)$ // add l_e to total likelihood
    end
    $\mathcal{L}_i \leftarrow \mathcal{L}_i + \ln(l_e)$ // add $\ln(l_e)$ to total log total likelihood
  end
end
$B_{jk} \leftarrow \exp\left(\mathcal{L}_j - \mathcal{L}_k\right)$ // Bayes factor between models

return B_{jk}

6

SOFTWARE

All of the details in Chapter 4 and Chapter 5 are implemented in the Quantum Model Learning Agent (QMLA) software framework, a (mostly) Python codebase which underlies all of the arguments, results and figures in this thesis [3]. The codebase is designed to simplify the process of running QMLA or quantum Hamiltonian learning (QHL) on novel systems. In particular, the core QMLA algorithm can support a wide range of exploration strategys (ESs), allowing for the design of bespoke ESs to account for the specific requirements of any given target system, Q. In this chapter we give an overview of the QMLA software, implementation and instructions for its use. We do not introduce new mathematical, physical or algorithmic concepts, so readers interested in applications of the techniques may prefer to skip to Part III.

6.1 IMPLEMENTATION

In this section we describe the technical details of the implementation of the algorithm described in Chapter 5, as well as a number of relevant subroutines. These discussions aim first to familiarise readers with some fundamental programming conventions, and then describe how we can leverage those concepts to construct the QMLA infrastructure.

6.1.1 *Object oriented programming*

We first introduce the concepts of object-oriented programming, and in particular *inheritance* between objects, since this will feature in later discussion about the implementation of QMLA and ESs. Python is a robust object-oriented language [158], meaning that we can frame concepts as *objects*, permitting actions to be performed to/by them. In particular, objects in Python are formulated as *classes*, which can have associated *attributes* and *methods*. For example, we can encode the concept of a Footballer as a class, such that the player object holds attributes such as number of games played and goals scored in a season; the player objects also has methods which achieve specific calculations, e.g. to summarise their record. We can then utilise the Footballer class to store information about an *individual* player, by making an *instance* of the class.

A fundamental concept in object-oriented programming is *inheritance* between objects, such that a *child* object inherits properties of its *parent*. In general, a parent object can be thought of as an abstract concept, which provides basic functionality and reasonable default properties, while a child object can incorporate specific requirements. For example, an Athlete class can act as a parent to the Footballer class, where the Athlete class holds core information such as date of birth. This allows for the Athlete class to be recycled as the *base* class for other child

classes which have the same underlying requirements, e.g RubgyPlayer. We list this example in Listings 6.1 to 6.2.

```
class Athlete():

    def __init__(
        self,
        name,
        birth_day,
        birth_month,
        birth_year,
    ):
        # Use information given
        self.name = name
        self.date_of_birth = datetime.date(
            birth_year, birth_month, birth_day
        )

    def age(self, round_down=True):
        # Method to compute this athlete's age
        days_since_birth = (
            datetime.date.today()
            - self.date_of_birth
        )
        age = days_since_birth.days / 365

        if round_down:
            age = int(age)

        return age

    def summary(self):
        # Method to summarise this athlete
        summary = "{name} is a {age}-year old athlete.".format(
            name = self.name,
            age = self.age()
        )
        print(summary)
```

```
bob = Athlete(
    name='Bob',
    birth_day = 11,
    birth_month = 11,
    birth_year = 1993,
)
bob.summary()
```

Listing 6.1: Parent class, encoding the concept of an athlete. Programmed in Python.

```
class Footballer(Athlete):
    def __init__(
        self,
        footed,
        team,
        size='medium',
        **kwargs
    ):
        # Pass arguments to the parent class
        super().__init__(**kwargs)

        # Use information given
        self.team = team
        self.footed = footed
        self.size = size

        # Default attributes
        self.goals_scored = 0

    def summarise(self):
        # Overwrite parent's summarise method
        # with method specific to Footballers
        summary = (
            "{size} {player} plays for {team} and has scored {
               num_goals} goals.".format(
                size = self.size,
                player = self.name,
                team = self.team,
                num_goals = self.goals_scored
            )
```

```
            )
            print(summary)

        def record_goals(self, num_new_goals):
            # Method to record that the Footballer
            # has scored a number of new goals
            self.goals_scored += num_new_goals

    # Make an instance of Footballer to represent an individual
    mickey = Footballer(
        name = 'Mickey',
        footed = 'left',
        team = 'QECDT-FC',
        birth_day = 11,
        birth_month = 11,
        birth_year = 1993,
        size = 'Big'
    )
    # Call the methods on the instance
    mickey.record_goals(num_new_goals = 10)
    mickey.summarise()
```

Listing 6.2: Child class, encoding the concept of a footballer, which adopts the abstract representation of an athlete. Programmed in Python.

6.2 PYTHON FRAMEWORK

A driving motivation for the development of QMLA is generality: we endeavour to make QMLA applicable to any target quantum system, Q. We provide a framework, where users can tailor the inputs and methodology to their needs. The main components of the framework are depicted in Fig. 6.1, broadly grouping concepts as part of its *infrastructure*, *algorithm* or *application*. In short, users need only specify the elements of the framework in the *application* segment, without concern for the underlying mechanics of QMLA. In particular, users interface with the framework through the design of a bespoke ES, described next.

6.2.1 *Application*

The application of QMLA refers to the choice of target system, Q, and how QMLA searches the model space in attempt to uncover its model. As outlined in Section 5.4, ESs play the role of

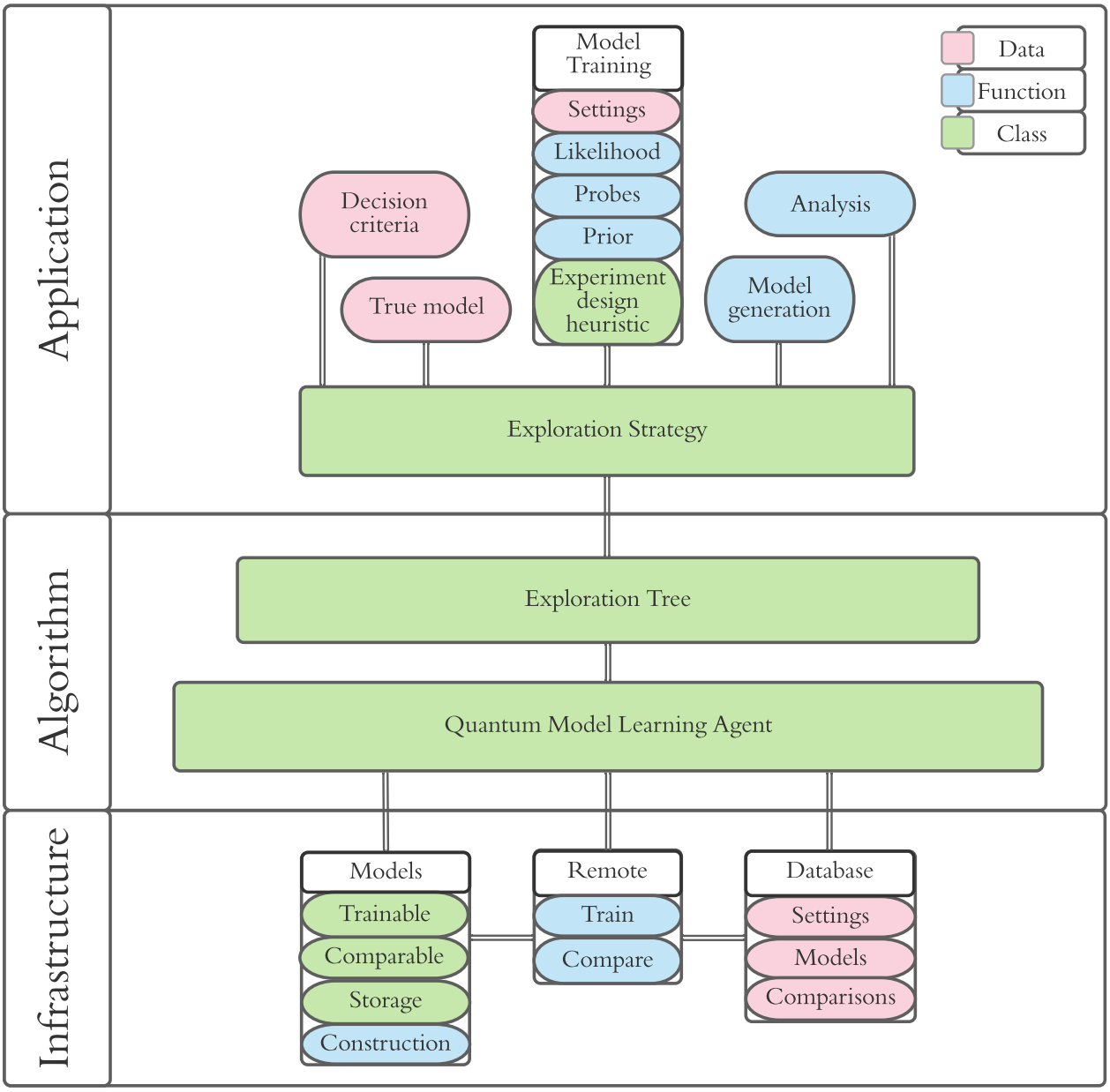


Figure 6.1: Overview of important *objects* in the QMLA framework. The objects' colour encodes its *type*: red objects are *data/properties*, blue are *functions/methods* and green are *classes*. Objects are grouped broadly, with double lines showing communication channels between (groups of) objects. **Infrastructure,** functions for the implementation of model training/comparisons on a remote compute server, Section 6.2.3. **Algorithm,** implementation of the iterative procedures and decision-making laid out in Chapter 5. The algorithmic controls are detailed in Section 6.2.2. **Application,** inter-changeable data/functionality for the unique requirements of a given target system, Section 6.2.1. Users wishing to customise QMLA must choose a valid implementation for each object in this segment but need not alter any of the underlying framework.

defining QMLA's objectives, guiding the steps it takes, and designing the models to be tested. We facilitate the study of any system by providing a robust ExplorationStrategy base class, with all of the functionality expected of a generic ES, allowing users to inherit and build upon it. In particular, ESs allow users to specify the implementation of aspects listed in Section 5.4, as well as further details.

6.2.1.1 *Modular functionality*

The most crucial methods[1] of the ES class are modular, described in Section 5.4.4, meaning that they can be directly replaced, provided the alternative method fulfils the same role. Our base ES class uses sensible defaults for this modular functionality, but this flexible mechanism allows for adapting QMLA by choosing an approach for each of the following subroutines.

- Likelihood function. As described in Section 4.3, the likelihood is the means by which QHL trains candidate models. By default, QHL calls a subroutine to compute Eq. (4.4). This can be replaced by any function which, given a Hamiltonian, evolution time and probe state, returns the likelihood, according to the desired experiment for simulation. For example, in Chapter 9, the data on which models are trained comes from experimental measurements, so we replace the likelihood function with a calculation corresponding to the experimental procedure.
- Probe generation. The training phase requires a set of probes against which to optimise individual models, as examined in Section 4.7. Users may wish to specify the design of such probes, for example to match experimental constraints which restrict the realisable probes in the performance of the experiment. Alternatively, it may be feasible to design probes which increase the information gained per experiment, enabling faster learning.
- Experiment design heuristic (EDH). The choice of EDH greatly influences how the training will perform, see Section 4.6. We provide a base class implementing particle guess heuristic (PGH), as well as child classes for each of the EDHs listed in Section 4.6.2.
- Prior. The method of drawing the prior distribution can be replaced, for example, with a method for constructing a uniform distribution on each parameter. A key input to the procedure is the initial knowledge the user has about the system, which is encoded in the prior, for instance varying orders of magnitude of the viable terms.

Additionally, applications require a series of settings for the model training phase, such as the hyperparameters required by the resampling algorithm, [134], as well as detailing the true model, $\hat{H}_0$, in the case where Q is simulated. We can also direct QMLA to perform ES-specific analyses to examine its internal performance, although this is generally required during development/testing, and less useful thereafter.

1 The words *method* and *function* are mostly interchangeable, although methods are specifically associated with a class, while functions are stand-alone.

6.2.2 *Algorithm*

The algorithm layer of Fig. 6.1 implements the core steps of QMLA, as shown in Fig. 5.1, by running a set of exploration trees (ETs), each of which communicate with a unique ES. The core QMLA class manages the database of models and their comparisons, and decides how to react at certain stages, by consulting the decision criteria set by the ES.

The implementation of QMLA seeks to separate the organisation of the model search from the cumbersome calculations which enable the search. We can offload those calculations to a compute cluster (server) to run *in parallel*, allowing for significant speedup of the entire QMLA procedure, limited by Amdahl's law. Amdahl's law stipulates that the *speedup* available to any program due to parallelisation is limited by the portion of the program which is inherently parallelisable, versus inherently serial [159]. In QMLA, all the model training and model comparison subroutines can be run in parallel, while only the administrative steps of the core QMLA algorithm are inherently serial, so QMLA can benefit greatly from parallelism.

While there are a number of strategies for parallelising code over a cluster of individual *processes*[2], we use the *master-worker* strategy, where one process acts as the *master*, determining which calculations are required at any given moment, then brokering self-contained *tasks* to *workers*, which blindly solve a small problem, without knowledge of the wider context or algorithm [160]. The mapping here is trivial: the master of our algorithm is QMLA, while workers can be used for the tasks of training and comparing models. QMLA distributes tasks to worker processes in a server, i.e. we assume that QMLA is run on a machine with N_c available parallel *processes*[3]. One process is designated for the QMLA class alone, e.g. for the ranking of models and determination of the next models to tests, while the remaining $N_c - 1$ processes lay dormant until QMLA requests that they perform a task. The role of QMLA is to collate the outcome of those calculations in conjunction with the set of exploration trees (ETs), until each ET is deemed complete, and then to consolidate the set of ET champions, ultimately setting the global champion, $\hat{H}'$. Thereafter it can perform some analysis; see Section 6.3.1 for further details.

QMLA and all workers have shared access to a database, through which they communicate data pertaining to individual tasks [161]. We use a simple *task queue* for the distribution of jobs: QMLA adds tasks to the queue and any available worker can take the next job and compute it [162]. The queue can be accessed by only one thread at a time, to ensure tasks are not duplicated. There are two types of task for workers:

- to train a candidate model, $\hat{H}_i$: the worker first requests some essential information about the model from the database, e.g. the name, terms and prior associated with the model, packaged in M_i; following completion, the worker compresses the result, R_i, and sends it to the database for storage.

2 Each process is a single CPU.

3 Note when running in *serial* (e.g. running locally on a personal machine), it is valid to simply set $N_c = 1$.

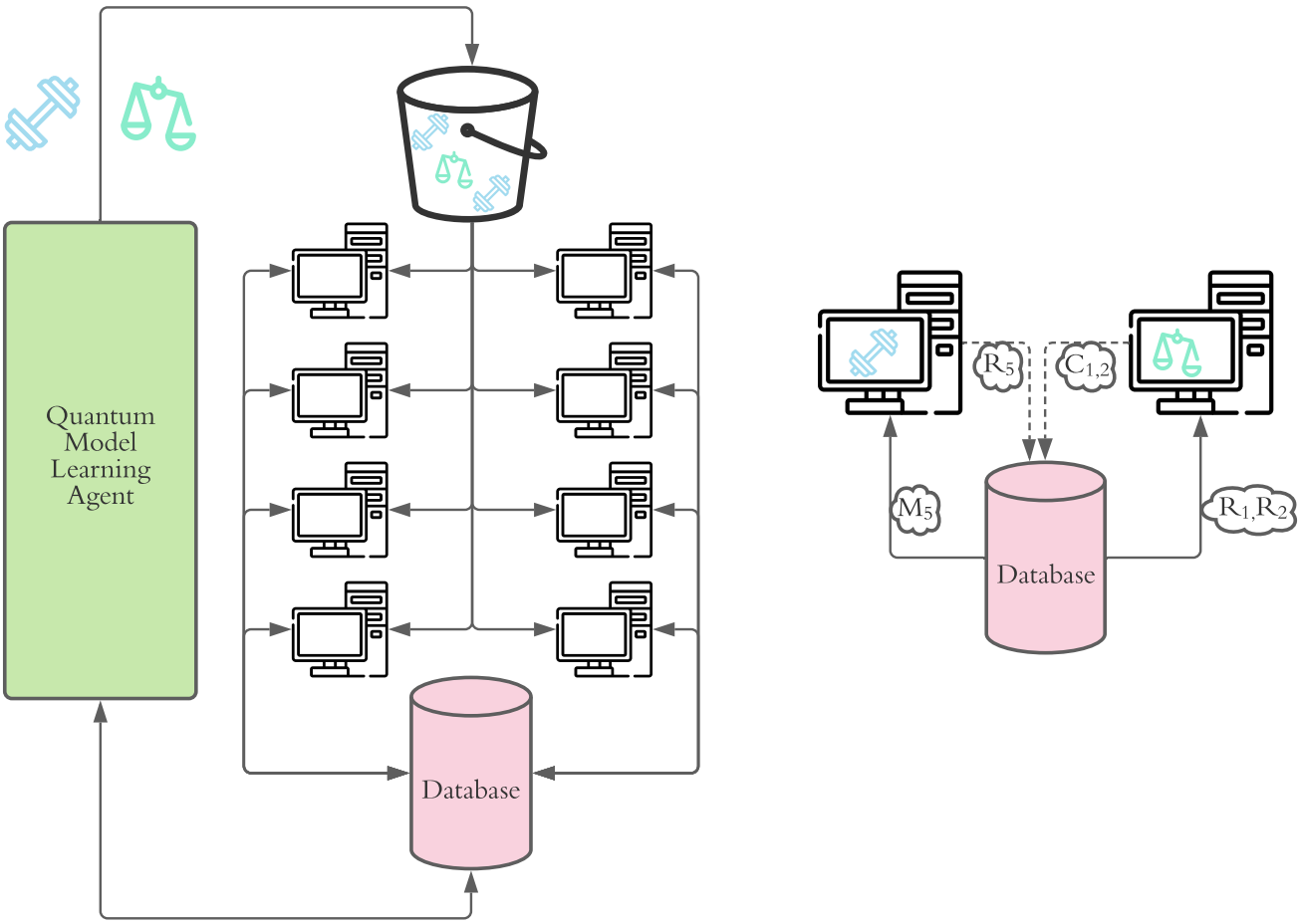


Figure 6.2: Parallel architecture for Quantum Model Learning Agent (QMLA). **Left,** QMLA generates tasks – either to train (blue dumbbells) or compare (green scales) models – and places them in a task queue. Worker processes (depicted as computers) retrieve those tasks and compute them in parallel, and interact with a database. **Right,** Distributed tasks occurring in parallel, with each process communicating with the database. The left-hand process assumes the task of training the model with ID 5, $\hat{H}_5$: it first queries the database for a packet of core information, M_5, which informs the model training procedure, for example the terms and parameters of $\hat{H}_5$. After training, it sends a packet, R_5, summarising the result of $\hat{H}_5$'s training. The right-hand process compares two models with IDs 1 and 2, by first retrieving the results packets R_1, R_2, then storing the comparison $C_{1,2}$ on the database.

- to compare two models, $\hat{H}_i, \hat{H}_j$: the worker retrieves R_i, R_j from the database, performs the calculation, and returns the compressed outcome of the comparison, C_{ij}, to the database.

The QMLA class copies the compressed results packets R_i and C_{ij}, in order to account for the results in its decision-making. It is worth noting that tasks are completely independent, so some worker processes may compute comparisons while others train models simultaneously, although the comparison between $\hat{H}_i, \hat{H}_j$ can not begin until both R_i, R_j are available. To ensure tasks are not launched in advance of their dependencies, we enforce a *blocking* protocol, whereby new batches of jobs are not released until the master receives all the results of jobs on which the new tasks depend: QMLA simply waits until all models on a given branch have been trained before queuing comparisons on that branch.

Models are assigned a unique ID upon creation, and are uniquely described by their name, represented as a string in the QMLA class, such that newly proposed models can be checked against the set of previously considered models before being added to the database. QMLA can hence check whether a proposed model, $\hat{H}_i$, has already been trained, in which case it does not resubmit the model, but instead relies on the existing result, R_i. Likewise QMLA can check for the presence of any comparison result, C_{ij}, before submitting the comparison as a new task, ensuring we do not duplicate expensive calculation. We depict the structure of this parallel architecture, and the master-worker strategy, in Fig. 6.2.

6.2.3 *Infrastructure*

The infrastructure enabling the distribution of QMLA's tasks across a set of worker processes can be summarised as:

- a set of classes representing the objects on which we must perform expensive calculations;
- functions to launch those calculations independently of any other calculation;
- a database which can be accessed by all workers as well as the QMLA master class.

We need a series of distinct classes to represent models, for use in each stage of QMLA: a *trainable* class is used for the parameter optimisation, while *comparable* classes are used for computing Bayes factors (BFs). Crucially, this separation allows us to perform data-heavy calculations independently, e.g. on a remote process within a compute cluster, and discard the class instance used for the calculation and the large amount of data it generates, while only the relatively small *storage* class is retained by QMLA for later use.

The tasks which actually implement the calculations (Section 6.2.2) are captured by standalone *remote* functions. These functions receive instructions such as $\mathrm{train\ model\ 10}$; they then contact the database for the set of shared settings, such as N_E, N_P and the set of probes, before performing the task, and then send the compressed result, R_{10}, to the database for storage. To achieve this separation between calculation and analysis, we use a redis database [161], which holds the core implementation settings, e.g. N_E, N_P and the set of probes to train upon, as well as the compressed summaries of the outcomes of tasks.

6.3 USAGE

Several aspects of QMLA are *probabilistic*. Firstly, the Bayesian updates within model training – i.e. QHL – relies on likelihoods which implicitly depend on the measurement datum of a quantum system. In the case where the projective measurement finds Q in a less-likely eigenstate, i.e. one with lower probability immediately prior to measurement, the likelihoods will indicate poor outcomes from good hypotheses, resulting in misguided posterior distributions. It is thus *possible* that the parameter learning will converge on incorrect values, or not converge at all even given ample resources. Moreover, the model design subroutine is not guaranteed to exploit the aspects of favoured models which are actually informative, e.g. given a favoured model with four correct terms and two incorrect terms, the model generator may opt to build upon the incorrect terms, in the common situation where it can not distinguish between helpful and misleading constituent terms.

Overall then, it is pertinent to run the entire QMLA algorithm repeatedly and gather statistics about its performance and outcomes, rather than making definitive claims about Q based on a single implementation. We say that a single implementation of QMLA is an *instance*, and N_r instances are grouped in a *run*. Instances can be realised in parallel, each relying on the master-worker parallel structure laid out above. We are primarily concerned with the performance of the run instead of any individual instance. For example, for each model, $\hat{H}_i$, in the model space, we can interpret its *win rate* – the fraction of instances for which QMLA finds $\hat{H}' = \hat{H}_i$ – as evidence for that $\hat{H}_0 = \hat{H}_i$. For the sake of evaluating QMLA itself, as in Part III, we can use the win rate of $\hat{H}_0$ as indication of the overall *success rate*, i.e. the fraction of instances within a run where QMLA identifies precisely $\hat{H}' = \hat{H}_0$. Note, however, that neither the win rate nor success rate are singularly informative of QMLA's performance: in some cases, we can deem QMLA successful even if it does not identify $\hat{H}_0$ exactly, e.g. if it finds the majority of terms present in $\mathcal{T}_0$ from a large space, i.e. a high F_1-score, see Section 8.1.2.

The QMLA codebase is available at [3], with complete documentation including a tutorial at [4]. We show how to design a custom ES and incorporate it within QMLA, and deploy the computation on a cluster in Appendix C.

6.3.1 *Outputs and analysis*

When a run is launched, QMLA generates a *results directory* unique to that run, identified by the time and date of its launch, in which all the pertinent information for that run, including raw data and figures, are stored. It includes an `analyse.sh` script to generate analysis after all instances have completed[4]. QMLA provides a large amount of analytics to assess the performance of the protocol. These range from *big picture* perspectives such as the win rate across the entire run, to focusing on internal metrics for training individual models. Some of these analyses are generated by default, while others are optional depending on the level of detail the user requires. A number of sub-directories are produced in the results directory, each containing data/figures

from a different view of the run; these are listed in Appendix A. Results are categorised across the levels of the framework, the most important of which are:

RUN

Results across a number of instances.

- Win rates for all models which are found as champion model at least once.
- Average dynamics reproduced by champion models.

INSTANCE

Performance of a single instance.

- Models generated and the branches on which they reside.

MODEL

Individual model performance within an instance.

- Parameter estimation through QHL.

PAIRWISE COMPARISONS

Direct comparison of models' performance.

- Dynamics of both candidates (with respect to a single basis).

EXPLORATION STRATEGY

Analysis specific to the ES.

- For example, model generation metrics.

Most plots used in this thesis are generated directly by the QMLA framework[5]; complete details for reproducing each figure are listed in Table A.1, with further details for navigating QMLA's outputs in Appendix A. Examples of some of the available analyses, as well as a demonstration for customising the QMLA software is given in Appendix C.

4 Note this script is not run automatically since, on remote servers, instances finish independently without any central process noticing. Therefore this script must be run by the user when the run is complete.

5 Figures presented in this thesis are minor modifications of figures available for automatic analysis in QMLA.

Part III

THEORETICAL STUDY

OVERVIEW AND CONTRIBUTION

In this part, we examine a series of theoretical quantum systems, to assess the usefulness of the Quantum Model Learning Agent (QMLA) protocol described in Part II. The results presented here will be made public in [2].

We begin in Chapter 7 with ideal systems described by lattice models under standard formalisms, i.e. Ising, Heisenberg and Hubbard models. This serves as a first test of the QMLA framework under reasonably straightforward conditions, where a small number of candidate models are proposed in advance, with the true model guaranteed to be among them. We then show that QMLA is also capable of classifying the family to which a target system belongs, i.e. whether it should be considered within Ising, Heisenberg or Hubbard formalisms. The initial idea for this chapter was proposed by Dr. Raffaele Santagati, and refined together with Dr. Andreas Gentile and myself. I modified the QMLA software for this application, ran the instances and analysed the data. The figures presented are my own.

In Chapter 8 we consider more general application of the QMLA protocol, searching through model spaces comprising over 250,000 valid candidate models. We explore these spaces efficiently by incorporating a genetic algorithm within QMLA. I proposed genetic algorithms for the study of large model spaces, and performed the initial hyperparameter tuning. I devised the numerous objective functions considered, and in particular the combination of Bayes factors with Elo ratings for a bespoke objective function which takes advantage of QMLA's core strengths. I built the genetic algorithm infrastructure into the QMLA software, ran the instances presented, and analysed the data. The figures presented are my own.

7 PRESCRIBED MODEL SETS

A sensible first case study for Quantum Model Learning Agent (QMLA) is to search within a closed space of models targeting some typical quantum systems; in particular we consider Ising, Heisenberg and Hubbard formalisms as a test-bed for the QMLA framework. We can prescribe a set of models for QMLA to consider, $\mathbb{H}$, from which it will choose a champion model, $\hat{H}'$. By including the true model, $\hat{H}_0 \in \mathbb{H}$, then QMLA should perform well at retrieving $\hat{H}' = \hat{H}_0$, provided its core assumptions are reliable, i.e. that model training, comparison and selection are feasible on meaningful models.

This application can be useful, for example, for expedited device calibration: suppose we wish to characterise a new, *untrustued* quantum simulator/device, S_u, and we have access to a *trusted*[1] simulator, S_t. In order to perform this calibration, we treat S_u as the system, Q, i.e. we call upon it to retrieve the datum d in Eq. (4.4), where the calculation of the likelihoods for each particle are computed through S_t. If S_u is reliable, the data from its calculations will be consistent with some $\hat{H}_0$ of our choosing. Conversely, miscalibrations can manifest as imperfectly implemented gates/steps in the calculation of the system's likelihood, and so would result in data inconsistent with $\hat{H}_0$. Therefore, if we can prescribe the most likely miscalibrations, it may be feasible to compose a set of models, $\mathbb{H}$, which represent those cases, and search for $\hat{H}'$ only within $\mathbb{H}$, to identify the dominant error mechanism(s). For example, by encoding in $\hat{H}_0$ the connections between every pair of qubits on the device, we can compose candidate models of restricted connectivity, for instance where some pairs of qubits are disconnected, and hence discover whether the device allows arbitrary two-qubit gates, and which pairs are disallowed.

In this chapter we perform such a study of the QMLA framework itself, by manually defining target and candidate models in simulation, given by lattice structures. We test the protocol by varying the physical systems it aims to represent, and finish by demonstrating that QMLA can classify the family of models underlying the target system, when allowed explore several families.

7.1 LATTICES

In each case examined in this chapter, QMLA will aim to identify the true model, $\hat{H}_0$, determined by a series of target systems, Q. Q represents some physical configuration; we specify the configuration of all models through unique *lattices*. The goal of QMLA is then to identify the structure of the true lattice, while the set of viable models are specified by alternative lattices. Due to simulation constraints, because we train models through exact unitary evolution, we are

1 Note: here a classical computer can fulfil the role of the trusted simulator.

restricted to 8-qubit Hamiltonians, so we only consider lattices which can be simulated in this limit. The exploration strategy (ES) in this chapter is then simply to propose a set of models $\mathbb{H}$, with no further model generation, and perform comparisons between all pairs of models through Bayes factors (BFs).

Connectivity between lattice sites is achieved within the specific Hamiltonian formalisms introduced in the following sections, although in general we write $\mathcal{C} = \{\langle k, l\rangle\}$ as the set of connected pairs $\langle k, l\rangle$, such that the Hamiltonian for a given lattice can be thought of as some function of its configuration, $\hat{H}(\vec{\alpha}, \mathcal{C})$, where $\vec{\alpha}$ is the usual vector of multiplicative parameters corresponding to each term in the Hamiltonian. Then, we can specify candidate models only by their $\mathcal{C}$, e.g. a 3-site chain can be summarised by $\mathcal{C} = \{\langle 1, 2\rangle, \langle 2, 3\rangle\}$, whereas a fully connected 3-site lattice (i.e. a triangle) is given by $\mathcal{C} = \{\langle 1, 2\rangle, \langle 1, 3\rangle, \langle 2, 3\rangle\}$. We can then summarise the set of candidate models through the descriptions of lattice configurations, corresponding to those depicted in Fig. 7.1:

a. 2-site chain;
b. 3-site chain;
c. 3-site fully connected (triangle);
d. 4-site fully connected (square);
e. 4-site linearly connected (loop);
f. 4-site chain;
g. 5-site chain;
h. 6-site chain;
i. 5-site fully connected (pentagon);
j. 6-site partially connected (grid).

We will use this set of lattice configurations throughout the remainder of this chapter.

7.2 ISING MODEL

The quantum Ising model – otherwise known as the transverse field Ising model, hereafter simply the Ising model – is one of the most studied concepts in all of physics, representing electrons on a lattice of N sites, where each electron can have *spin* up or down [163–165]. Interactions[2] between spins $\langle k, l\rangle$ have strength J_{kl}, and the transverse magnetic field acts on spin k with strength h_k. The Ising model is usually stated as

$$\hat{H}_I(\mathcal{C}) = \sum_{\langle k,l\rangle \in \mathcal{C}} J_{kl}\, \hat{\sigma}_k^z\, \hat{\sigma}_l^z + \sum_{k=1}^{N} h_k \hat{\sigma}_k^x. \tag{7.1}$$

2 The Ising model usually considers only nearest neighbour interaction. Here we present the more general $\langle k, l\rangle$ which can connect any pair of sites, and specify pairs in the set $\mathcal{C}$.

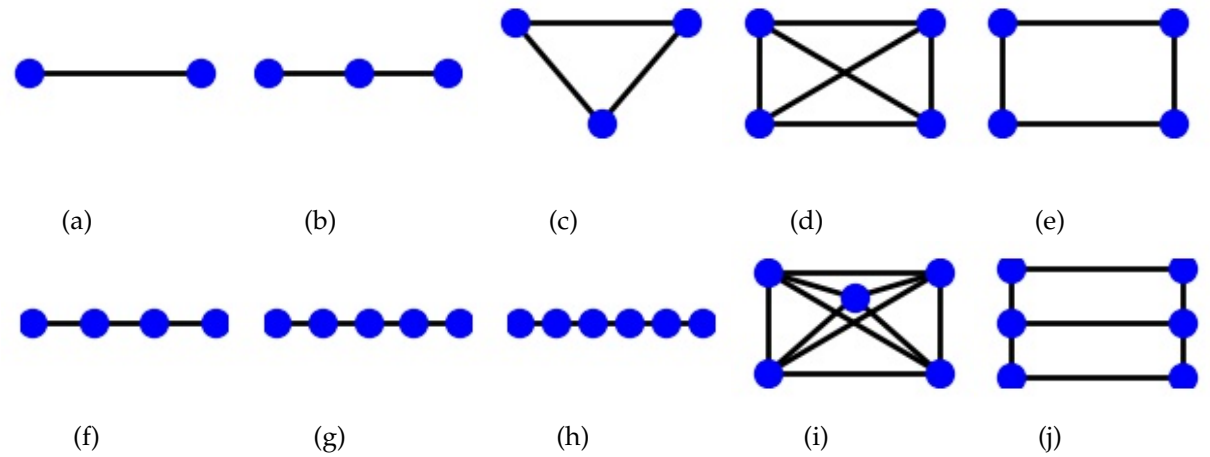


Figure 7.1: Lattices used for prescribed models test for QMLA. Lattices are characterised by the connectivity of their sites; dotted lines show connection between pairs of sites.

The interaction term[3] indicates the class of magnetism of the pair's interaction, i.e.

$$\begin{cases} J_{kl} < 0, & \text{ferromagnetic;} \\ J_{kl} > 0, & \text{antiferromagnetic;} \\ J_{kl} = 0, & \text{noninteracting.} \end{cases} \tag{7.2}$$

If all interaction pairs are described by the same case in Eq. (7.2), the entire system can be seen as belonging to that class of magnetism.

7.2.1 *Note on optimising the Ising model*

Many treatments of the Ising model seek to find the ground state of the system by optimising the configuration of spins in the system. This involves treating the Ising model *classically*, effectively by neglecting the transverse magnetic field term ($h_k \to 0$), such that the ground state is found by minimising the energy function is given by

$$E_I = \langle \psi | H_I | \psi \rangle = \sum_{\langle k,l \rangle \in \mathcal{C}} J_{kl} \ \langle \psi | \hat{\sigma}_k^z \, \hat{\sigma}_l^z | \psi \rangle \, , \tag{7.3}$$

where $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \cdots \otimes |\psi_N\rangle$.

This optimisation relies on the relationship between the Ising model with its eigenvalues and eigenstates: Eq. (7.3) consists only of $\hat{\sigma}_z$ terms, and we have that

$$\hat{\sigma}_z |+\rangle = +1 |+\rangle \quad ; \quad \hat{\sigma}_z |-\rangle = -1 |-\rangle \, . \tag{7.4}$$

3 Note: the terms J_{kl} is often presented as $-J_{kl}$ in Eq. (7.1), such that $J_{kl} > 0$ indicates ferromagnetism. Here we keep the parameter general, and set the sign implicitly when defining and learning the parameter, so $J_{kl} > 0$ corresponds to ferromagnetism. This is a matter of convention and does not impact any of the further discussion.

Then, for a single pair of spins $\langle k,l\rangle$, we have

$$\begin{aligned}
\langle +_k\, +_l|\, \hat{\sigma}_k^z\, \hat{\sigma}_l^z\, |+_k\, +_l\rangle &= \langle +_k\, +_l|\, (+1)(+1)\, |+_k\, +_l\rangle = +1,\\
\langle +_k\, -_l|\, \hat{\sigma}_k^z\, \hat{\sigma}_l^z\, |+_k\, -_l\rangle &= \langle +_k\, -_l|\, (+1)(-1)\, |+_k\, -_l\rangle = -1,\\
\langle -_k\, +_l|\, \hat{\sigma}_k^z\, \hat{\sigma}_l^z\, |-_k\, +_l\rangle &= \langle -_k\, +_l|\, (-1)(+1)\, |-_k\, +_l\rangle = -1,\\
\langle -_k\, -_l|\, \hat{\sigma}_k^z\, \hat{\sigma}_l^z\, |-_k\, -_l\rangle &= \langle -_k\, +_l|\, (-1)(1)\, |-_k\, -_l\rangle = +1.
\end{aligned} \tag{7.5}$$

So, by restricting the individual spins to $|\psi_k\rangle \in \{|+\rangle, |-\rangle\}$, we can equivalently consider every spin s_k in the system as a binary variable $s_k \in \{\pm 1\}$, i.e. $s_k s_l = \pm 1$ in Eq. (7.5), such that the energy function,

$$E_I(\mathcal{S}) = \langle\psi|\hat{H}_I|\psi\rangle = \sum_{\langle k,l\rangle\in\mathcal{C}} J_{kl}\ \ s_k s_l, \tag{7.6}$$

can be minimised by optimising the configuration $\mathcal{S} = s_1, \ldots, s_N$, when the interaction terms $\{J_{\langle k,l\rangle}\}$ are known. The optimal configuration $\mathcal{S}_0$ can then be mapped to a state vector $|\psi_0\rangle$, i.e. the ground state of the system.

While this task can be greatly simplified by the reduction in Eq. (7.5), meaning we do not have to compute any unitary evolution to evaluate Eq. (7.6), it is still an expensive optimisation, because effectively it is a search over $\{|\psi\rangle\}$, so the search space has 2^N candidates [164, 166]. This allows for a straightforward mapping between ground state search and solving combinatorial optimisation algorithms, namely MAX-CUT, known to be NP-complete [167], allowing for proposed advantage in mapping computationally challenging problems to quantum hardware [168]. This mapping underlies ongoing research into quantum annealing as a computational platform capable of providing advantage for a specific family of problems [169–171].

Crucially, our goal is *not* to find the ground state of Q, but instead to find the generator of its dynamics. Therefore, we treat the Ising model *quantum mechanically*: instead of treating Eq. (7.1) as the underlying mechanism for a cost function to be optimised, i.e. Eq. (7.6), we use quantum operators and do not necessarily restrict the probe state $|\psi\rangle$, allowing us to use Eq. (7.1) within the likelihood function Eq. (4.4).

7.2.2 *Ising model cases*

We consider two cases: firstly, where it is assumed that the strength of interactions $\{J_{k,l}, h_k\}$ are uniform (given by J, h); and secondly, where each interaction is assigned a unique parameter (J_{kl}, h_k). In the first case, we can represent the Ising model for a given lattice configuration $\mathcal{C}$ as

$$\hat{H}(\mathcal{C}) = J \sum_{\langle k,l\rangle\in\mathcal{C}} \hat{\sigma}_k^z \hat{\sigma}_l^z + h \sum_{k=1}^{N} \hat{\sigma}_k^x, \tag{7.7}$$

allowing for the compact representation, following Section 5.1,

$$\vec{\alpha}_I = (J \quad h); \tag{7.8a}$$

$$\vec{T}_I = \begin{pmatrix} \sum\limits_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}_k^z \hat{\sigma}_l^z \\ \sum\limits_{k=1}^{N} \hat{\sigma}_k^x \end{pmatrix}. \tag{7.8b}$$

In the more general second case, termed the *fully parameterised* Ising model, we instead have the parameter and term sets

$$\alpha_I = \{J_{k,l},\ h_k\}_{\langle k,l\rangle \in \mathcal{C}}; \tag{7.9a}$$

$$\mathcal{T}_I = \{\hat{\sigma}_k^z \hat{\sigma}_l^z,\ \ \hat{\sigma}_k^x\}_{\langle k,l\rangle \in \mathcal{C}}. \tag{7.9b}$$

with unique parameters J_{kl} associated with each interaction term $\hat{\sigma}_k^z \hat{\sigma}_l^z$, and h_k associated with each field term, $\hat{\sigma}_k$. We summarise these cases in Table 7.1.

	$J_{\langle k,l\rangle}$	h_k
Standard	J	h
Fully parameterised	$J_{\langle k,l\rangle}$	h_k

Table 7.1: Forms of Ising model. Varying whether parameters $J_{\langle k,l\rangle}, h_k$ are shared across terms result in distinct models.

We first construct models under each of these forms to verify quantum Hamiltonian learning (QHL) is capable of learning in this regime: we train using the standard Ising model (Eq. (7.8) as $\hat{H}_0$ in Fig. 7.2), and separately using the fully parameterised model (Eq. (7.9) as $\hat{H}_0$ in Fig. 7.3). Ultimately, these two cases give the same Hamiltonian when we set $J_{\langle k,l\rangle} = J;\ h_k = h\ \forall k,l$. The fully parameterised model will learn the same parameters as the standard Ising model, and we can take the BF between them to determine which parameterisation is favourable. Encouragingly, both models learned the parameters to high precision, although neither model converged, i.e. the volume continues to reduce exponentially in both cases. This outcome is common: training is not guaranteed to converge[4], so it can be impractical to seek saturation in the training phase for every model, since this may require a very large number of experiments and particles. It would be preferable for each model's parameterisation to have converged before comparing them, but this is infeasible due to the indefinite resources required; in practice, model comparisons must rely on limited training schedules which are presumed to reflect the overall ability of the models to capture Q's dynamics. We will see throughout this thesis[5] that the choice of such training resources, namely N_E, N_P, has a large impact on the outcome of QMLA, for example through misleading comparisons between models in the case where the better model's training was under performant. This capacity for error is mitigated by combining many instances together in

4 When the model being trained is $\hat{H}_0$, the volume may not saturate until numerical precision is reached [130].

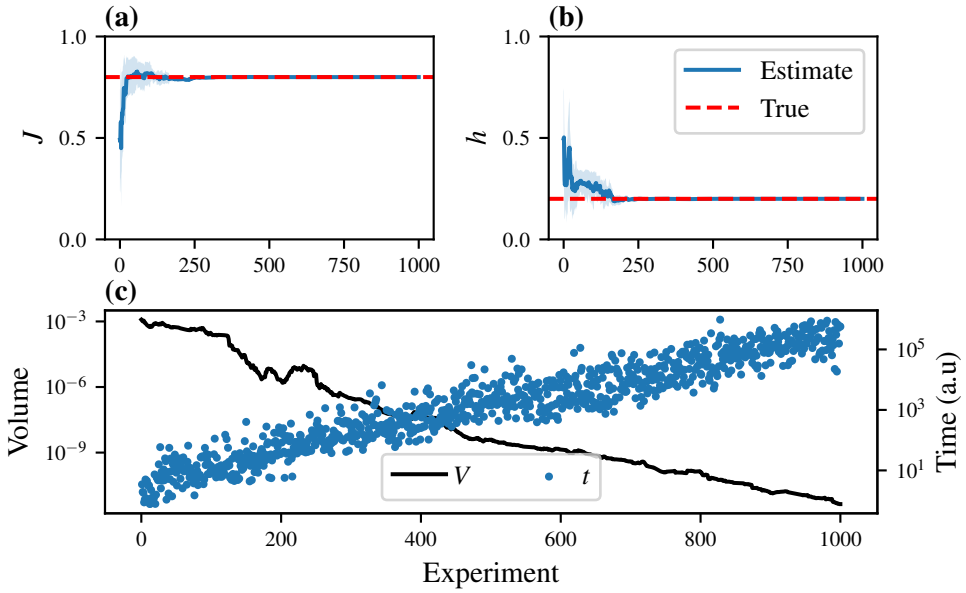


Figure 7.2: Quantum Hamiltonian learning for the standard Ising model, where terms are grouped by their functionality, as in Eq. (7.1). **a-b,** the parameter estimates' progression against training experiments, with the corresponding term labelling the y-axis. The parameters and volume are presented in arbitrary units of energy. **c,** the volume of the parameter distribution at each experiment, as well as the evolution time chosen by the experiment design heuristic. Implementation details are listed in Table A.1.

a QMLA run, such that any conclusions drawn rely on the average performance, where strong models are likely to perform better overall, even given access to limited resources.

The dynamics produced by both models are shown in Fig. 7.4: the dynamics are almost indistinguishable by eye, but the standard Ising model, which in this case is $\hat{H}_0$, outperforms the fully parameterised model, by a BF of $\mathcal{B} = 10^{19}$. This serves as a good *sanity check*, confirming our expectation that the BF will favour the simpler model (i.e. fewer parameters) even when both models are trained to a high precision to very similar parameters, and are difficult to distinguish through human intuition[6].

5 Including Section 7.7 where we study the effect of varying resources on QMLA's outcome in the context of lattice systems described here.

6 Here we have deliberately described the special case where models of more- and fewer-parameters are identical, and the fewer-parameter-model can exactly reproduce the larger parameterisation. Had the true model been composed of unequal $\{J_{kl}\}$, the under-parameterised model should not be preferred.

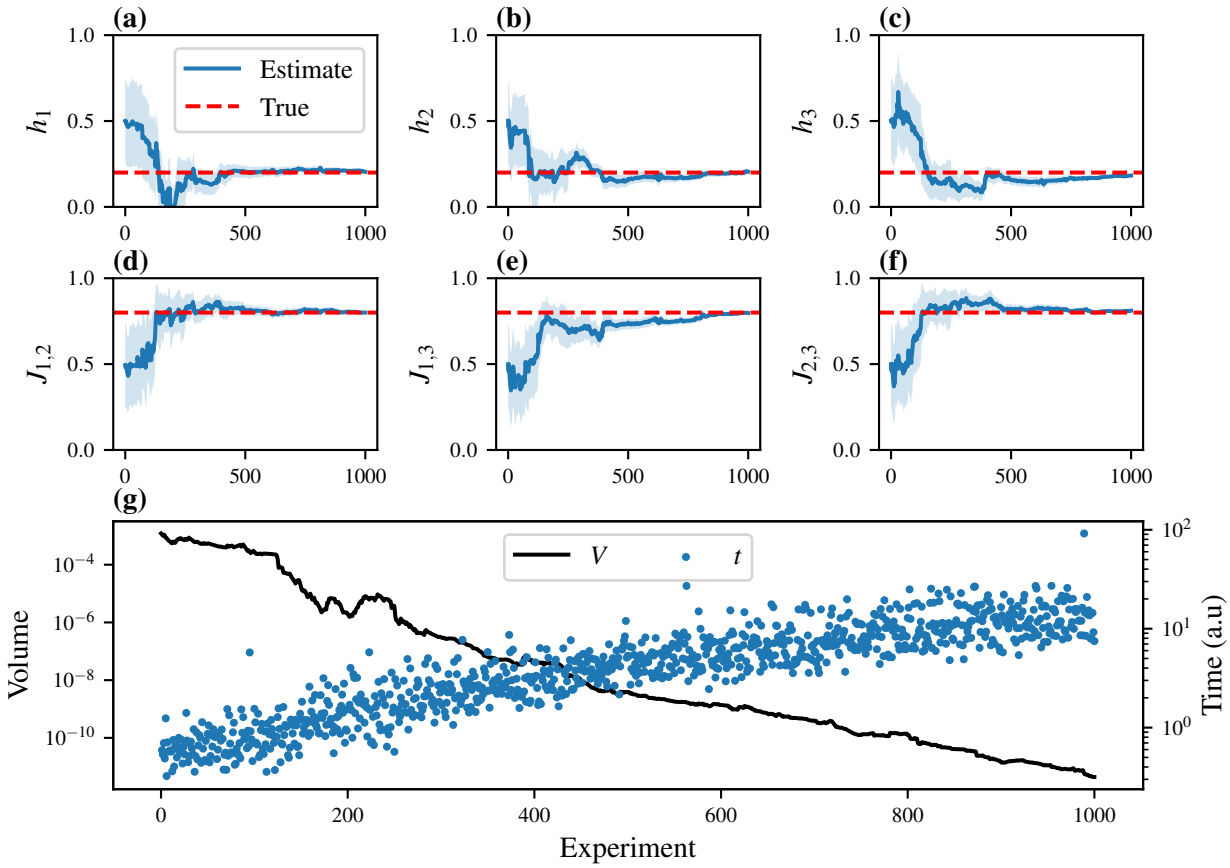


Figure 7.3: Quantum Hamiltonian learning for the fully parameterised Ising model, where every interaction between pairs of sites are assigned unique parameters, as in Eq. (7.9). **a-f**, the parameter estimates' progression against training experiments, with the corresponding term labelling the y-axis. The parameters and volume are presented in arbitrary units of energy. **g,** the volume of the parameter distribution at each experiment, as well as the evolution time chosen by the experiment design heuristic (EDH). Implementation details are listed in Table A.1.

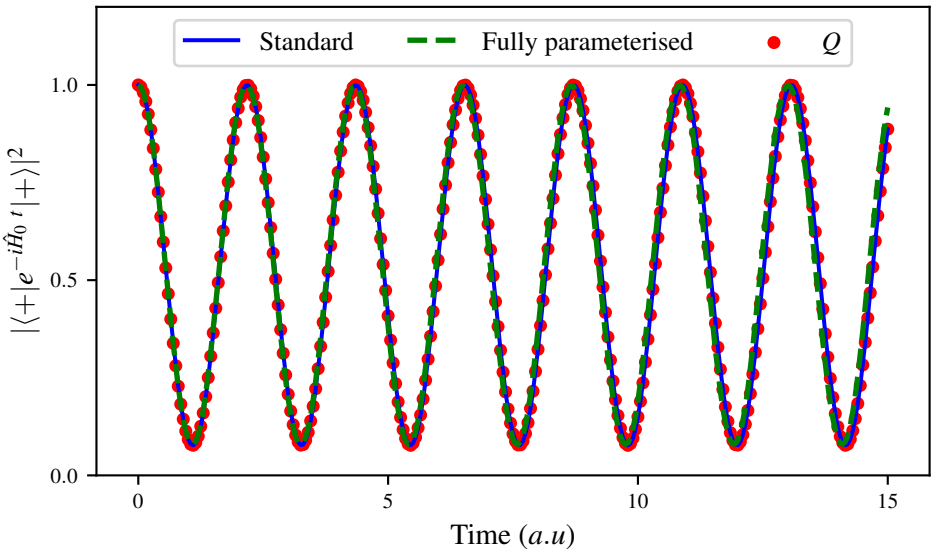


Figure 7.4: Dynamics reproduced by Ising models using the standard form (blue, training shown in Fig. 7.2) and the fully parameterised form (dotted green, training in Fig. 7.3), compared with dynamics for the true system, Q (red dots). $\hat{H}_0$ for Q is given by the standard form, Eq. (7.1), with parameters set arbitrarily to $J = 0.8$ and $h = 0.2$. The Bayes factor favours the standard formalism over the fully parameterised, with a value of $\mathcal{B} = 10^{19}$. Implementation details are listed in Table A.1.

7.3 HEISENBERG MODEL

Generalising the Ising model, the Heisenberg Hamiltonian is another model for magnetic systems consisting of a set of spins on a lattice [172]. It builds on the Ising model by additionally considering the spins' couplings about the x- and y-axes, generally stated as

$$\hat{H}_H(\mathcal{C}) = \sum_{\langle k,l\rangle\in\mathcal{C}} J^x_{kl}\,\hat{\sigma}^x_k\hat{\sigma}^x_l + \sum_{\langle k,l\rangle\in\mathcal{C}} J^y_{kl}\,\hat{\sigma}^y_k\hat{\sigma}^y_l + \sum_{\langle k,l\rangle\in\mathcal{C}} J^z_{kl}\,\hat{\sigma}^z_k\hat{\sigma}^z_l + \sum_{k=1}^{N} h_k\hat{\sigma}^z_k. \tag{7.10}$$

We can consider a number of formulations of the Heisenberg model, by considering whether the interaction parameters are completely unique for each pair of spins in each axis, or are shared by pairs of spins. We list a number of prominent representations within the family of Heisenberg models in Table 7.2.

Again, there are a number of possibile models to test, although we can reasonably expect these to follow the same arguments as for the Ising model cases: increasing generality – at the expense of larger parameter dimension – requires more resources to learn to a reasonable level. We will consider the fully parameterised model in Chapter 8; here however we will focus on the

	J^x_{kl}	J^y_{kl}	J^z_{kl}	h_k
XXX	J^x	J^x	J^x	h
XXZ	J^x	J^x	J^z	h
XYZ (standard)	J^x	J^y	J^z	h
Fully parameterised	J^x_{kl}	J^y_{kl}	J^z_{kl}	h_k

Table 7.2: Heisenberg model forms: varying whether the interaction parameters J^w_{kl} are shared among pairs of spins give distinct descriptions, all of which are within the family of Heisenberg models.

more restrictive Heisenberg-XYZ model, such that the parameters and terms of interest are then captured by Eq. (7.11).

$$\vec{\alpha}_H = \begin{pmatrix} J^x & J^y & J^z & h \end{pmatrix}; \tag{7.11a}$$

$$\vec{T}_H = \begin{pmatrix} \sum_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}^x_k \hat{\sigma}^x_l \\ \sum_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}^y_k \hat{\sigma}^y_l \\ \sum_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}^z_k \hat{\sigma}^z_l \\ \sum_{k=1}^{N} \hat{\sigma}^z_k \end{pmatrix}. \tag{7.11b}$$

7.4 HUBBARD MODEL

Another representation of solid state matter systems is given by the Hubbard model [173–175]. The Hubbard model deals with systems of correlated fermions, allowing spins to *hop* between sites and to localise on sites. Spins are *correlated* in this model because the spin on any site is subject to the Coulomb interaction, i.e. a repulsive force due to the presence of another electron on the same site, so it is energetically favourable for spins to arrange across sites. Note the Hubbard model is synonymous with the Fermi-Hubbard (FH) model, which can be used to distinguish the statistics from a set of fermions from the statistics of a similar set of bosons, given by the Bose-Hubbard model. In this thesis we will not study the Bose-Hubbard model, but will use the subscript FH to distinguish the (Fermi-)Hubbard model from the Heisenberg model $\hat{H}_H$, Eq. (7.10). The Hubbard model is generally stated in second quantisation as

$$\hat{H}_{FH}(\mathcal{C}) = -\sum_{s\in\{\uparrow,\downarrow\}} \sum_{\langle k,l\rangle \in \mathcal{C}} t^s_{\langle k,l\rangle} \left(\hat{c}^\dagger_{ks} c_{ls} + \hat{c}^\dagger_{ls} c_{ks} \right) + \sum_k^N U_k \hat{n}_{k\uparrow} \hat{n}_{k\downarrow} + \sum_k^N \mu_k \left(\hat{n}_{k\uparrow} + \hat{n}_{k\downarrow} \right) \tag{7.12}$$

where

- $\hat{c}_{ks}$ and $\hat{c}^\dagger_{ks}$ are respectively the fermionic annihilation and creation operators for spin $s \in \{\uparrow, \downarrow\}$ on site k;
- $\hat{n}_{ks} = \hat{c}^\dagger_{ks}\hat{c}_{ks}$ is the onsite term, i.e. a counting operator to count the number of spins s on site k;
- $t^s_{\langle k,l\rangle}$ is the kinetic (hopping) term for spin s between sites k and l;
- U_k is the onsite (repulsion) energy for site k;
- μ_k is the chemical potential for k;
- N is the number of sites in the system.

Again, we can achieve differing physics by controlling whether the parameters are shared (e.g. $t^s_{\langle k,l\rangle}$), with similar consequences to the Ising and Heisenberg models, where additional parameterisation comes at the expense of slower/worse performance in training. We list a subset of possible configurations in Table 7.3; again here we will use the standard form for the remainder of this chapter, Eq. (7.13).

	$t^\uparrow_{\langle k,l\rangle}$	$t^\downarrow_{\langle k,l\rangle}$	U_k	μ_k
Standard	t	t	U	μ
Fully parameterised	$t^\uparrow_{\langle k,l\rangle}$	$t^\downarrow_{\langle k,l\rangle}$	U_k	μ_k

Table 7.3: Forms of Hubbard model. Varying whether parameters $t^s_{\langle k,l\rangle}, U_k, \mu_k$ are shared across sites gives distinct models.

$$\vec{\alpha}_{FH} = (t^\uparrow \quad t^\downarrow \quad U \quad \mu); \tag{7.13a}$$

$$\vec{T}_{FH} = \begin{pmatrix} \sum_{\langle k,l\rangle\in\mathcal{C}} (\hat{c}^\dagger_{k,\uparrow}\hat{c}_{l,\uparrow} + \hat{c}^\dagger_{l,\uparrow}\hat{c}_{k,\uparrow}) \\ \sum_{\langle k,l\rangle\in\mathcal{C}} (\hat{c}^\dagger_{k,\downarrow}\hat{c}_{l,\downarrow} + \hat{c}^\dagger_{l,\downarrow}\hat{c}_{k,\downarrow}) \\ \sum_{k=1}^{N} \hat{n}_{k\uparrow}\hat{n}_{k\downarrow} \\ \sum_{k=1}^{N} (\hat{n}_{k\uparrow} + \hat{n}_{k\downarrow}) \end{pmatrix}. \tag{7.13b}$$

7.4.1 *Jordan Wigner transformation*

In order that the Hubbard model is simulatable with qubits[7], it must first undergo a mapping from the fermionic representation to a spin system representation; such a mapping is given

by the Jordan Wigner transformation (JWT) [176, 177]. We implement the JWT within QMLA through OpenFermion's fermilib package [178].

In second quantisation, the fermions on the lattice can occupy one (or a superposition of) *modes*, for example, spin $\uparrow$ on the site indexed 3 is a mode. The system can then be given by a state in the *number basis*,

$$|\psi_f\rangle = |n_{m_1}, n_{m_2}, \dots, n_{m_n}\rangle\,, \tag{7.14}$$

where n_{m_i} is the number of fermions on mode m_i and there are n modes in total.

$\hat{c}^\dagger_{m_i}$ ($\hat{c}_{m_i}$) is the creation (annihilation) operator on the mode m_i: it acts on the system by adding (removing) a fermion to (from) m_i:

$$\hat{c}^\dagger_{m_i}|\psi_f\rangle = |n_{m_1}, \dots, n_{m_i}+1, \dots, n_{m_n}\rangle\,, \tag{7.15a}$$

$$\hat{c}_{m_i}|\psi_f\rangle = |n_{m_1}, \dots, n_{m_i}-1, \dots, n_{m_n}\rangle\,. \tag{7.15b}$$

In the Hubbard model, we assign a mode for each combination of spin $s \in \{\uparrow, \downarrow\}$ with each site k, i.e. the system is in the state

$$|\psi_{FH}\rangle = |n_{1\uparrow}, n_{1\downarrow}, \dots, n_{N\uparrow}, n_{N\downarrow}\rangle\,. \tag{7.16}$$

In particular, since fermions obey the Pauli exclusion principle, i.e. every spin/site can be occupied by at most one electron, and we can view them as two-level systems, so we have $n_{sk} \in \{0, 1\} \forall s, k$. We therefore use a similar system to the number basis: a qubit registered as $|0\rangle$ corresponds to an empty mode, while $|1\rangle$ holds a fermion. Empty lattices are thus given by $|0\rangle^{\otimes 2N}$. Then, in analogue with the annihilation and creation operators, we introduce operators $\hat{\sigma}^+, \hat{\sigma}^-$ such that

$$\hat{\sigma}^+ = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \implies \hat{\sigma}^+|0\rangle = |1\rangle \tag{7.17a}$$

$$\hat{\sigma}^- = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \implies \hat{\sigma}^-|1\rangle = |0\rangle \tag{7.17b}$$

Then, to map the number basis of Eq. (7.16) to a state which can be prepared on qubits, the JWT assigns a single qubit to each mode, where qubits are ordered simply by the site index and spin type, as shown in Table 7.4. The JWT can be summarised by mapping – for the mode m – the creation (annihilation) operator $\hat{c}^\dagger_m$ ($\hat{c}_m$), to an operator which adds (removes) a spin to (from) the corresponding state through the operator $\hat{\sigma}^+_m$ ($\hat{\sigma}^-_m$).

$$\hat{c}_m \to (\hat{\sigma}^z)^{\otimes k-1} \otimes \hat{\sigma}^- \otimes (\hat{\sigma}^z)^{\otimes 2N-1} \tag{7.18a}$$

$$\hat{c}^\dagger_m \to (\hat{\sigma}^z)^{\otimes k-1} \otimes \hat{\sigma}^+ \otimes (\hat{\sigma}^z)^{\otimes 2N-1} \tag{7.18b}$$

7 Or simulations of qubits, as in this thesis.

Mode	Site	Spin	Qubit
1	1	$\uparrow$	1
2	1	$\downarrow$	2
3	2	$\uparrow$	3
4	2	$\downarrow$	4
	$\vdots$		
$2N-1$	N	$\uparrow$	$2N-1$
$2N$	N	$\downarrow$	$2N$

Table 7.4: Jordan Wigner mode/qubit indices.

For example, an empty 2-site lattice $|\psi_0\rangle$ is acted on by a creation operator on mode 3, corresponding to spin $\uparrow$ on site 2:

$$\hat{c}^\dagger_{2\uparrow}|0000\rangle = \hat{c}^\dagger_3|0000\rangle = \hat{\sigma}^z_1\hat{\sigma}^z_2\hat{\sigma}^+_3\hat{\sigma}^z_4|0000\rangle = |0010\rangle\,. \tag{7.19}$$

7.4.2 *Half filled basis*

In principle there can be $2N$ spins on a lattice of N sites, although in general we will restrict to the case where there are N spins in the lattice, known as *half-filling*, such that Eq. (7.16) is effectively projected into the subspace spanned by half-filled basis states. For example, with $N = 2$

$$\{|1100\rangle, |1010\rangle, |1001\rangle, |0101\rangle, |0110\rangle, |0011\rangle\} \tag{7.20}$$

Therefore, in the design of probes for training Hubbard models, we can generate probes in the subspace spanned by half-filled states.

7.5 MODEL LEARNING FOR LATTICES

Finally, then, we can use the model formalisms introduced in Sections 7.1 to 7.4 as first case studies for QMLA, in conjunction with the set of lattices $\mathbb{C}$ shown in Fig. 7.1. Each $\mathcal{C} \in \mathbb{C}$ can specify a unique model under their standard forms: Eq. (7.8) for Ising, Eq. (7.11) for Heisenberg and Eq. (7.13) for Hubbard models. We can then devise a simple ES (exploration strategy) which only tests the models corresponding to lattices in $\mathbb{C}$, with no further model generation, Algorithm 8, and compares every pair of models through BFs, deeming the champion as that which wins the largest number of comparisons, Algorithm 9.

For example, we adopt the fully connected four site lattice (Fig. 7.1**(d)**) as the true lattice specifying $\hat{H}_0$, under the Ising formalism (Eq. (7.7)). We run QMLA by training the ten models

Algorithm 8: Lattice exploration strategy: model generation

Input: $\mathbb{C}$ // set of lattice configurations
Output: $\{\hat{H}_i\}$ // set of models to test

$\mathbb{H} = \{\ \}$
for $\mathcal{C} \in \mathbb{C}$ **do**
  $\hat{H}_i \leftarrow \text{map\_lattice\_to\_model}(\mathcal{C})$
  $\mathbb{H} \leftarrow \mathbb{H} \cup \{\hat{H}_i\}$
end
return $\mathbb{H}$

Algorithm 9: Lattice exploration strategy: consolidation

Input: $\mathbb{H}$ // set of trained models
Output: $\hat{H}'$ // favoured model

for $\hat{H}_i \in \mathbb{H}$ **do**
  $s_i \leftarrow 0$ // score for every model
end
for $\hat{H}_i \in \mathbb{H}$ **do**
  for $\hat{H}_j \in \mathbb{H} \setminus \{\hat{H}_i\}$ **do**
    $B_{ij} \leftarrow BF(\hat{H}_i, \hat{H}_j)$ // compute Bayes factor via Algorithm 7
    if $B_{ij} > 1$ **then**
      $s_i \leftarrow s_i + 1$ // $\hat{H}_i$'s score increases if it is favoured by the BF
    end
  end
end
$\hat{H}' \leftarrow \arg\max_{s_i} (\hat{H}_i)$
return $\hat{H}'$

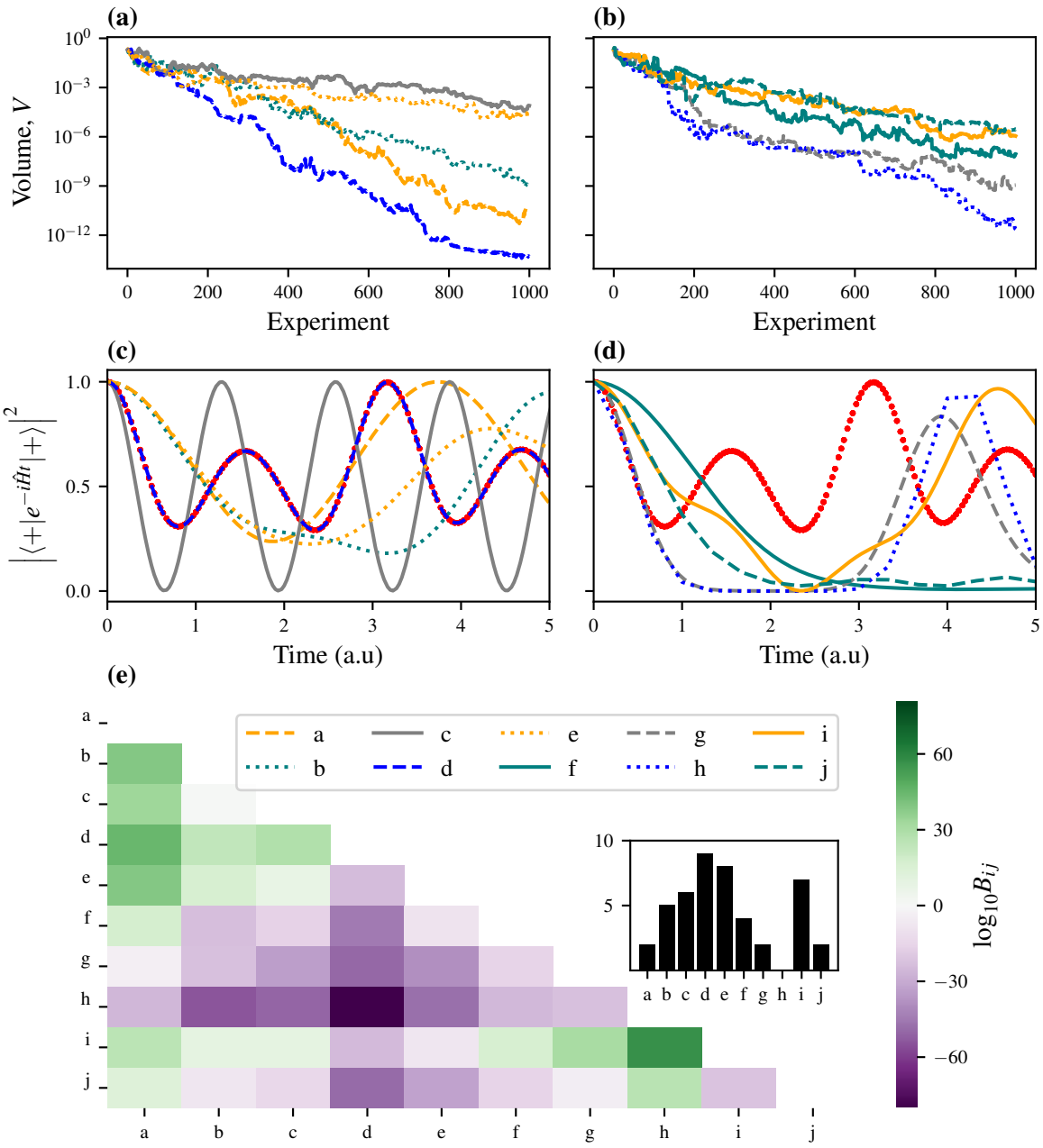


Figure 7.5: QMLA for prescribed set of lattices under Ising formalism. The lattice indices correspond to those in Fig. 7.1, indexed by the legend in **(e)**, and the true system is given by lattice d. **a-b,** the decrease in volume for each model's training phase (in arbitrary energy units, spread over two plots for readibility). **c-d,** trained models are used to reproduce dynamics, compared with the dynamics of the true model (red dots). **e,** Heatmap of $\log_{10} B_{ij}$ between every pair of models. Each tile gives the BF between a pair of models, $(\hat{H}_i, \hat{H}_j)$. The BF is read as $\hat{H}_i$ versus $\hat{H}_j$, where $\hat{H}_i$ is the model on the y-axis and $\hat{H}_j$ is the model on the x-axis. $\log_{10} B_{ij} > 0$ (green) favours the model listed on the y-axis; $\log_{10} B_{ij} < 0$ (purple) favours the model listed on the x-axis. **Inset,** the number of BF comparisons won by each lattice, i.e. the models' scores. Implementation details are listed in Table A.1.

corresponding to the ten lattices, Fig. 7.5**(a-b)**; comparing the models predictive power, Fig. 7.5**(c-d)**, through BF (Fig. 7.5**(e)**), and choosing the model which wins the largest number of BF contests. In this example, $\hat{H}_0$ is stronger than every alternative model according to the BFs, where the lattice d wins all nine of its BF comparisons, and is hence determined as $\hat{H}'$.

7.6 COMPLETE QMLA RUN FOR LATTICE SETS

In order to test QMLA robustly, we can use each of the lattices shown in Fig. 7.1 to specify $\hat{H}_0$, to ensure the algorithm is capable of finding the underlying model of arbitrary complexity, within the constraints of a prescribed model set[8]. Moreover, we can extend this test to the Heisenberg and Hubbard formalisms; note that due to the overhead given by the JWT (Section 7.4.1), i.e. the requirement of two qubits per site, we restrict study of the Hubbard model to lattices $a - e$ for practicality[9]. By running 10 independent QMLA instances for each lattice under each formalism, we can gauge the success rate of the algorithm for distinguishing basic lattices from each other. We present the result of these tests in Fig. 7.6, finding in all cases that QMLA identifies $\hat{H}_0$ with success rates at least 70%. A general trend appears to emerge – especially in the case of the Hubbard model – where target models of higher dimension are identified less often. This can likely be attributed to the training resources provided: here models are trained with $N_e = 1000$, $N_p = 4000$, which are clearly sufficient for training models of few qubits/parameters, but may not allow larger models to train well. In the next section we investigate whether increasing N_E, N_P leads to higher success rates in general, and find a strong correlation between training resources and QMLA success rate. We therefore expect that the results for large target models could be improved by drastically increasing the training resources, although for practicality, we do not perform such a test.

While there is compelling evidence that BFs can be used for model selection in general [179], this straightforward test verifies that the BF is a fair mechanism by which to distinguish between models in the context of candidate descriptions for quantum systems. In general it will not be possible to prescribe the set of models to test, although this might serve as a straightforward mechanism for the calibration of quantum devices. Suspected miscalibrations can be used in the design of such a set of models, along with a target $\hat{H}_0$ which the device should be able to implement. By testing such a prescribed set and determining $\hat{H}'$, we can map the miscalibration between the intended and actual operations. In the ideal case, where it is mostly believed the device works, this application of QMLA may allow for fast, automated *verification* of the device:

8 The remainder of this thesis is dedicated to cases where we do not prescribe the model set, but instead generate models dynamically.

9 The limitation of Hubbard models to 4-site lattices is due to the 6-qubit models required via the JWT. The primary expense of simulation is the complete unitary evolution, requiring calculation of $e^{-i\hat{H}_i t}$ for each particle's likelihood calculation at every training experiment. Training becomes infeasible for systems requiring $\gtrapprox 9$ qubits, whereas moving to 5-site lattices would require 10-qubit models. We include a mixture of lattices up to 4 sites for the Hubbard model, and some further models up to 6 sites for the Ising and Heisenberg models.

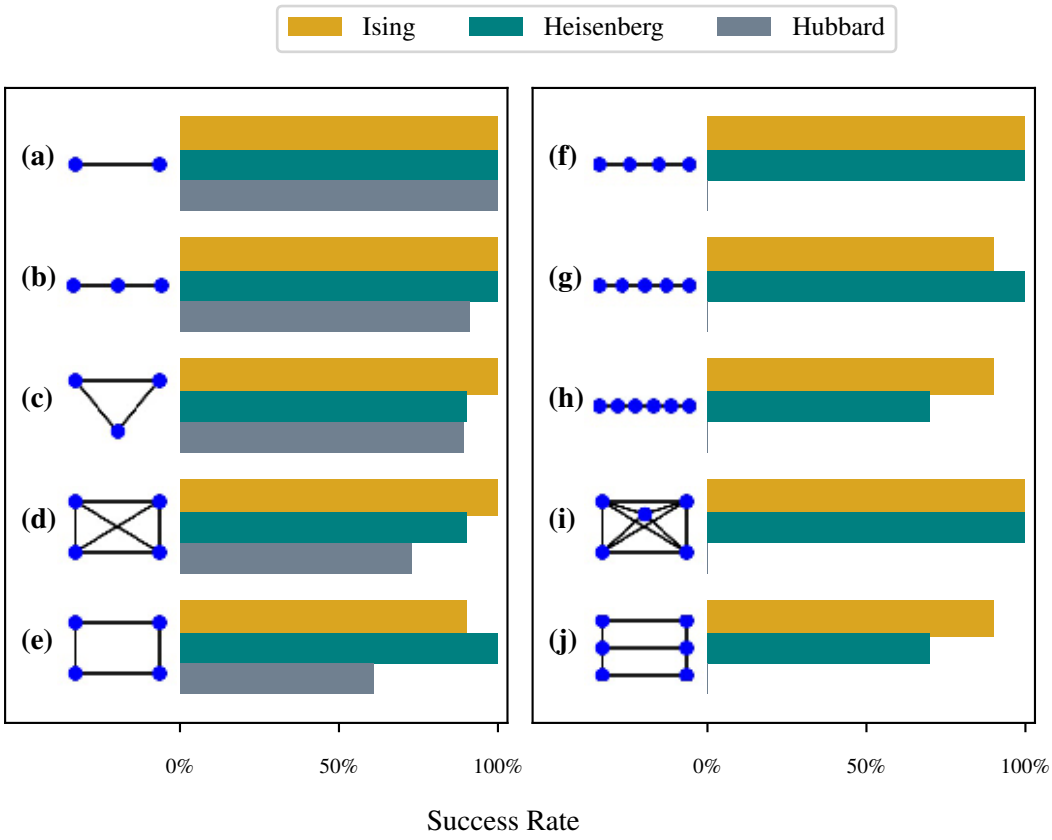


Figure 7.6: Rates of success for QMLA under various conditions. Each lattice is set as the true model $\hat{H}_0$ for ten independent instances. In each instance, the ES considers the available lattices (**a-j** for Ising and Heisenberg cases and **a-e** for the Hubbard case), and selects a champion model $\hat{H}'$ as that most consistent with data generated by $\hat{H}_0$. The figure displays the rate at which each lattice is correctly identified as $\hat{H}_0$ under standard Ising, Heisenberg and Hubbard formalisms. Implementation details are listed in Table A.1.

if QMLA finds $\hat{H}' = \hat{H}_0$ with high success given reasonable opportunity to miscompute, it may be sufficient verification that the device behaves as desired, or at least part thereof.

Recall from Section 5.3 (and Fig. 5.1**(e)**) that QMLA can grow multiple exploration trees (ETs) concurrently, each corresponding to a unique exploration strategy (ES). This functionality permits ETs of different underlying physical assumptions, and can therefore be used to examine alternative formalisms in parallel. For instance, in order to examine Q, we can independently run ESs for each of the Ising, Heisenberg and Hubbard model *families*: QMLA first deems the most appropriate model under each formalism, $\hat{H}'_S$, before consolidating $\{\hat{H}'_S\}$ and declaring the global champion model, $\hat{H}'$. $\hat{H}'$ therefore encodes which family best describes the system of interest: even if the precise model is not found, $\hat{H}' \neq \hat{H}_0$, we can still classify the model family – i.e. the underlying physical mechanism – which is most useful for describing Q.

Earlier in this chapter, we alluded to a fundamental question in the discussion of model training and comparison through Bayesian inference, as underlies QHL: to what extent is a trained model undermined by its *limited* training resources[10], and what resources should be granted in order to retrieve reliable outcomes? As usual in machine learning (ML) methods, this represents a trade-off between the results of the algorithm against training time and computational resources required. This question motivates the next section, where we examine the role played by the numbers of training experiments and particles, in correctly identifying models.

7.7 MODEL FAMILY CLASSIFICATION

We combine the task of family classification with non-exhaustive testing of the trade-off between resources and outcomes. In this case, we vary the target model $\hat{H}_0$ as deriving again from the lattices in Fig. 7.1, however we reduce the number of tested models in each case. ESs corresponding to the Ising and Heisenberg model consider lattices **(a-f)**, while the ES for the Hubbard model only consider (a-c). Fig. 7.7 shows the rate at which the precise $\hat{H}_0$ is identified, as well as the rate with which the family of $\hat{H}_0$ is classified, compared with increasing resources, N_E and N_P. As expected, a clear trend demonstrates that the success rates scale with resources, which we can leverage in practice in two core ways. First, we can ensure that the training regime for models is sufficiently robust that we can reasonably expect models to have trained well, and therefore model comparisons can be trusted. If QMLA does not give clear results, then, we may expect a clearer outcome from increasing N_E, N_P. Secondly, we can mitigate the unpredictable[11] failures of QMLA by running many instances per run: one (or few instances) in a scarce-resource training regime are prone to error, so in cases where we are unsure whether

10 Recall the main resources for model training: the number of experiments performed, N_E, and the number of particles used during QHL, N_P.

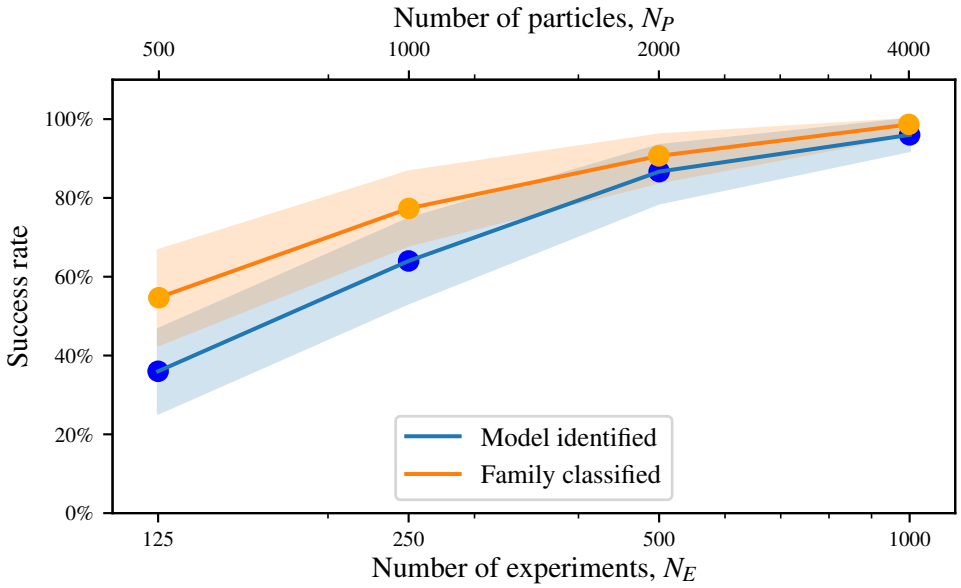


Figure 7.7: QMLA outcomes for varying training resources. Independent ESs are implemented for Ising, Heisenberg and Hubbard families, with $\hat{H}_0$ cycling through lattices (*a-f*) for the Ising and Heisenberg cases, and (*a-c*) for the Hubbard case, where lattices' connectivity are as depicted in Fig. 7.1. Each lattice in each family is tested as $\hat{H}_0$ in five instances, so there are 75 instances per datapoint. The success rates are shown for QMLA identifying the correct model precisely (blue), as well as classifying the correct model family (orange), against the numbers of experiments and particles used to train each candidate model. Implementation details are listed in Table A.1.

the resources provided are sufficient, we must run enough independent instances to overcome these artefacts.

QMLA's capacity to classify the family of model to which Q belongs suggests powerful future applications of the framework, namely to automatically discover the type of physics underlying systems of interest. For example, QMLA could be used to classify whether a sample is ferromagnetic or antiferromagnetic; or further, is best described as a system of bosons or fermions.

11 QMLA depends on several probabilistic processes which can cause misleading outcomes; these are highlighted in Section 6.3.

8

GENETIC EXPLORATION STRATEGY

The Quantum Model Learning Agent (QMLA) framework lends itself easily to the family of metaheuristic optimisation techniques called *evolutionary algorithms*, where individuals, sampled from a population of candidates, are considered as solutions to the given problem. Candidates are batched in *generations*, such that iterative generations aim to efficiently search the available population by mimicking biological evolutionary mechanisms [124]. In particular, we develop an exploration strategy (ES) which incorporates a genetic algorithm (GA) in the construction of models; GAs are a subset of evolutionary algorithms where candidate solutions are expressed as strings of numbers representing some configuration of the system of interest [180]. We describe the concepts of GAs in Section 3.3, so we begin this chapter by describing the adaptations which allow us to build a genetic exploration strategy (GES) within QMLA.

8.1 ADAPTATION TO QMLA FRAMEWORK

Unlike the generic aspects of GAs described in Section 3.3, in the context of QMLA, we must deviate from default mechanisms. The overarching goal of QMLA – to characterise some black box quantum system, Q – proceeds by designing and performing experiments upon Q which enable us to improve the modelling of $\hat{H}_0$. Nothing described so far provides a natural objective function (OF), upon which GAs rely to assess the suitability of candidates relative to contemporary candidates. We can not assume full knowledge of $\hat{H}_0$ while generating candidates models, so we can not simply invoke some loss function with respect to the target model, for example. Instead, we must devise schemes which exploit the knowledge we *do* have about each candidate $\hat{H}_j$, which is the primary challenge in building a GES. We propose and discuss a number of options in Section 8.2. Common to all proposed OFs, however, is that candidates should be trained before evaluation, so that their assessment is based on their actual power in describing Q, rather than some initial parameterisation which may not capture their potential capability. This is a tenet of QMLA: for each candidate $\hat{H}_j(\vec{\alpha}_j)$, we use a subroutine to optimise $\vec{\alpha}_j$; as in earlier applications of QMLA, for this study we rely on quantum Hamiltonian learning (QHL) as the parameter optimisation subroutine.

Ultimately, the conceived role of a GA within QMLA is to generate the sets of models to place on successive branches[1] of an exploration tree (ET) as depicted in Fig. 5.1. The apparatus within QMLA which facilitates novel model generation techniques is the exploration strategy (ES). Here we will design an ES which acts in cooperation with a GA. The ES specifies that consolidation of a generation μ involves evaluating the *fitness*, g_i, of each candidate, $\hat{H}_i$, via the chosen OF.

1 Branches in QMLA and generations of the genetic algorithm are equivalent here.

The GA then maps $\{g_i\}$ of each $\hat{H}_i \in \mu$ to a selection probability, and composes new candidates via crossover, Section 3.3.3. Recall from Section 5.4.1, that we capture the space of available terms as $\mathcal{T}$, i.e. we list – in advance – the feasible terms which may be included in candidate models[2], with $N_t = |\mathcal{T}|$ the number of terms considered. QMLA is then an optimisation algorithm, attempting to find the set $\mathcal{T}'$ which *best* represents the true terms $\mathcal{T}_0$. Note, this does not require identification of the precise true model to be successful, as insight can be gained from approximate models which capture the physics of Q. We introduce metrics for success in Section 8.1.2. We recognise the limitations this structure imposes: we can only identify terms which were conceived in advance; this may restrict QMLA's applicability to entirely unknown systems, where such a primitive set can not even be compiled.

The structure of the overall QMLA algorithm (recall Fig. 5.1) is unchanged. In a GES:

- Branches: models are still grouped in branches, here called generations.
- Training: models are still trained, again through QHL.
- Consolidation: all models are evaluated according to the OF (to be described in Section 8.2), so branches are consolidated by ranking models according to their fitness.
- Spawning: new models are spawned through the GA by selecting pairs of parents for crossover, with the resultant offspring models probabilistically mutated.

The design of any ES centres on the implementation of the generate_models subroutine – we summarise the GES's method in Algorithm 10. We can restate the informal description of GAs[3], now in the context of QMLA, as

1. Sample N_m models from the total population, $\mathcal{P}$, at random
   (a) this is the first generation, μ.
2. Evaluate each model $\hat{H}_j \in \mu$
   (a) train $\hat{H}_j$ through QHL;
   (b) apply the objective function to assign the model's fitness, g_j.
3. Map the fitnesses of each model, $\{g_j\}$, to selection probabilities for each model, $\{s_j\}$
   (a) e.g. by normalising the fitnesses, or by removing some poorly-performing models and then normalising.
4. Generate the next generation of models
   (a) Reset $\mu = \{\}$;
   (b) Select pairs of parents, $\hat{H}_{p_1}, \hat{H}_{p_2}$, from μ
      i. Each model's probability of being chosen is proportional to their s_j;
   (c) Cross over $\hat{H}_{p_1}, \hat{H}_{p_2}$ to produce children models, $\hat{H}_{c_1}, \hat{H}_{c_2}$
      i. mutate $\hat{H}_{c_1}, \hat{H}_{c_2}$ according to some random probabilistic process;

2 Recall that models impose structure on sets of terms: $\hat{H}_j = \vec{\alpha}_j \cdot \vec{T}_j = \sum_{k \in \{j\}} \alpha_k \hat{t}_k$.

3 First stated on Page 25.

ii. $\mu \leftarrow \mu \cup \{\hat{H}_{c_i}\}$, only if $\hat{H}_{c_i}$ is not already in μ, to ensure N_m *unique* models are tested at each generation;

(d) until $|\mu| = N_m$, iterate to step (b).

5. Until the N_g^{th} generation is reached, iterate to step 2..
6. The strongest model on the final generation is deemed the approximation to the system, $\hat{H}'$.

8.1.1 *Models as chromosomes*

We first need a mapping from models to chromosomes; this is straightforward given the description of chromosomes as binary strings, exemplified in Section 3.3.1. We assign a gene to every term in $\mathcal{T}$, so that candidate models are succinctly represented by bit strings of length N_t. We give an example of the mapping between models and chromosomes in Table 8.1. Given that every model is contained in the space of bit strings spanned by N_t bits, we can say that there are a total of 2^{N_t} available models in the model space.

Model		Chromosome					
	$\vec{T}$	$\hat{\sigma}^x_{(1,2)}$	$\hat{\sigma}^z_{(1,2)}$	$\hat{\sigma}^y_{(2,3)}$	$\hat{\sigma}^x_{(2,3)}$	$\hat{\sigma}^y_{(2,3)}$	$\hat{\sigma}^x_{(2,3)}$
γ_{p_1}	$(\hat{\sigma}^x_{(1,2)} \quad \hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^y_{(2,3)})$	1	0	1	0	1	0
γ_{p_2}	$(\hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^y_{(2,3)} \quad \hat{\sigma}^z_{(2,3)})$	0	0	1	0	1	1
γ_{c_1}	$(\hat{\sigma}^x_{(1,2)} \quad \hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^y_{(2,3)} \quad \hat{\sigma}^z_{(2,3)})$	1	0	1	0	1	1
γ_{c_2}	$(\hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^y_{(2,3)})$	0	0	1	0	1	0
γ'_{c_2}	$(\hat{\sigma}^z_{(1,2)} \quad \hat{\sigma}^x_{(2,3)} \quad \hat{\sigma}^y_{(2,3)})$	0	0	1	1	1	0

Table 8.1: Mapping between QMLA's models and chromosomes used by a genetic algorithm. Example shown for a three-qubit system with six possible terms, $\hat{\sigma}^w_{i,j} = \hat{\sigma}^w_i \hat{\sigma}^w_j$, $w \in \{x, y, z\}$. Model terms are mapped to binary genes: if the gene registers 1 (0) then the corresponding term is (not) present in the model. The top two chromosomes are *parents*, $\gamma_{p_1} = 101010$ (blue) and $\gamma_{p_2} = 001011$ (green): they are mixed to spawn new models. We use a one–point cross over about the midpoint: the first half of γ_{p_1} is mixed with the second half of γ_{p_2}, and vice versa, to produce two new offspring chromosomes, $\{\gamma_{c_1}, \gamma_{c_2}\}$. Mutation occurs probabilistically: each gene has a 25% chance of being mutated, e.g. a single gene (red) flipping from $0 \rightarrow 1$ to mutate γ_{c_2} to γ'_{c_2}. The next generation of the genetic algorithm will then include $\{\gamma_{c_1}, \gamma'_{c_2}\}$ (assuming γ_{c_1} does not mutate). To generate N_m models for each generation, $N_m/2$ parent couples are sampled from the previous generation and crossed over.

8.1.2 F_1-score

We need a metric against which to evaluate models, and indeed the entire QMLA procedure. We can gauge the performance of QMLA's model search by the quality of candidate models produced at each generation, so we introduce a metric to act as proxy for model quality: the F_1-score, denoted f. We define F_1-score formally in this section, but in short, $f \in (0,1)$ indicates the degree to which $\hat{H}_i$ captures the physics of the target system: $f = 0$ indicates that $\hat{H}_i$ shares no terms with $\hat{H}_0$, while $f = 1$ is found uniquely for $\hat{H}_i = \hat{H}_0$. We defined F_1-score, as well as a number of metrics in the field of classification in machine learning (ML), in Section 3.1.1.1; here we modify those definitions to align with the nomenclature of QMLA.

We emphasise that the goal of this work is to identify the *model* which best describes quantum systems, and not to improve on parameter-learning when given access to particular models, since those already exist to a high standard [23, 129]. Therefore, in this context we can consider the role of QMLA as a classification routine[4], with the goal of classifying whether individual terms $\hat{t}$ from a set of available terms $\mathcal{T} = \{\hat{t}\}$ are helpful in describing data which is generated by $\hat{H}_0$, whose terms constitute $\mathcal{T}_0$. Candidate models $\hat{H}_i$ then have $\mathcal{T}_i$. We can assess $\hat{H}_i$ using standard metrics used regularly in the ML literature, which simply count the number of terms identified correctly and incorrectly:

- true positives (TP): number of terms in $\mathcal{T}_0$ which are in $\mathcal{T}_i$
- true negatives (TN): number of terms not in $\mathcal{T}_0$ which are also not in $\mathcal{T}_i$
- false positives (FP): number of terms in $\mathcal{T}_i$ which are not in $\mathcal{T}_0$
- false negatives (FN): number of terms in $\mathcal{T}_0$ which are not in $\mathcal{T}_i$.

These concepts – shown in Fig. 8.1 – allow us to define

- *precision*: how precisely does $\hat{H}_i$ capture $\hat{H}_0$, i.e. if a term is included in $\mathcal{T}_i$ how likely it is to actually be in $\mathcal{T}_0$, Eqn 8.1a;
- *sensitivity*: how sensitive is $\hat{H}_i$ to $\hat{H}_0$, i.e. if a term is actually in $\mathcal{T}_0$, how likely $\mathcal{T}_i$ is to include it, Eqn. 8.1b.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{8.1a}$$

$$\text{sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8.1b}$$

Informally, precision prioritises that predicted terms are correct, while sensitivity prioritises that true terms are identified. In practice, it is important to balance these considerations. F_β-

4 The designation of QMLA as supervised, unsupervised, or otherwise depends upon the ES employed, Section 5.5.1. The model search of this GES is unsupervised, but we use metrics from literature about classification, since we can assess performance absolutely.

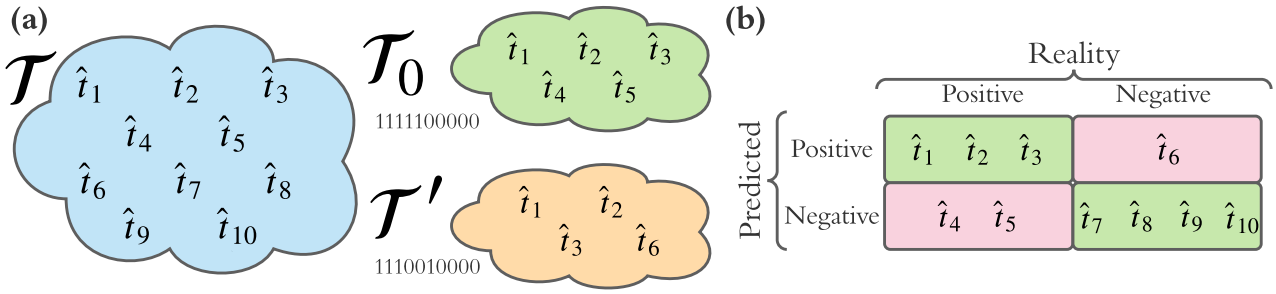


Figure 8.1: Concepts used for classification. **a**, the set of available terms $\mathcal{T}$ containing individual terms $\hat{t}_1$ to $\hat{t}_{10}$. The true model $\hat{H}_0$ is constructed from the set $\mathcal{T}_0$. Suppose a candidate $\hat{H}'$ has the set $\mathcal{T}'$. **b**, the confusion matrix for $\hat{H}'$. Correctly classified terms are true positives and true negatives (green), and incorrectly classified terms are false positives and true negatives (red).

score (Eq. (3.3)) is a measure which balances these, with weighting β in favour of sensitivity. In particular, F_1-score considers precision and sensitivity as equally important:

$$F_1 = \frac{2 \times (\text{precision}) \times (\text{sensitivity})}{(\text{precision} + \text{sensitivity})} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})} =: f. \tag{8.2}$$

We give an example of these quantities in Fig. 8.1, where $\text{TP} = 3, \text{TN} = 4, \text{FP} = 1, \text{FN} = 2$, giving precision $= 3/4$ and sensitivity $= 3/5$, with a final $f = 0.67$, i.e. f is the average of the indicators of model quality we care about.

We adopt F_1-score as an indication of model *quality* because we are concerned both with precision and sensitivity of the models QMLA predicts as representations of Q. We can use F_1-score to measure the success of the algorithm, by recording f for all models in all generations, allowing us to see whether or not the approximation of the system is improving on average. Of course in realistic cases we can not assume knowledge of $\mathcal{T}_o$ and therefore cannot compute F_1-score, but it is a useful tool in the development of the GES itself, or in cases where $\hat{H}_0$ is known, such as when the target system is simulated, e.g. in the case of device calibration. Our search for an effective OF can then be guided by seeking the method which most strongly improves the average F_1-score in test-cases. We will not use F_1-score within the algorithm[5], i.e. to inform any steps taken by QMLA, but simply to assess its performance independently.

5 Except for meta-analysis in Section 8.1.3

Algorithm 10: ES subroutine: generate_models via genetic algorithm

Input: ν // information about models considered to date
Input: τ // truncation rate

Input: $g(\hat{H}_i)$ // objective function that can act on any model $\hat{H}_i$
Input: rank() // function to rank models relative to each other
Input: truncate() // function to truncate set of models
Input: normalise() // function to normalise models' scores
Input: roulette() // function to select models through roulette
Input: crossover() // function to crossover two parents to produce offspring
Input: mutate() // function to mutate offspring probabilistically

Output: $\mathbb{H}$ // set of models

$N_m = |\nu|$ // number of models
for $\hat{H}_i \in \nu$ **do**
  $g_i \leftarrow g(\hat{H}_i)$ // model fitness via objective function
end

$r \leftarrow \text{rank}(\{g_i\})$ // rank models by their fitness
$\mathbb{H}_t \leftarrow \text{truncate}(r, N_m \times \tau)$ // truncate models by rank: only keep $N_m \times \tau$
$s \leftarrow \text{normalise}(\{g_i\})\ \forall \hat{H}_i \in \mathbb{H}_t$ // normalise remaining models' fitnesses
$\mathbb{H} = \{\}$ // new batch of chromosomes/models
while $|\mathbb{H}| < N_m$ **do**
  $p_1, p_2 = \text{roulette}(s)$ // use s to select two parents via roulette selection
  $c_1, c_2 = \text{crossover}(p_1, p_2)$ // produce offspring models
  $c_1, c_2 = \text{mutate}(c_1, c_2)$ // probabilistically mutate
  $\mathbb{H} \leftarrow \mathbb{H} \cup \{c_1, c_2\}$ // add new models to batch
end

return $\mathbb{H}$

8.1.2.1 *Distinguishing F_1-score through Bayes factors*

We have so far relied on Bayes factor (BF) as the means by which to distinguish models' ability to explain data from Q. We conjecture that models of higher F_1-score are usually statistically better at predicting dynamics of Q than those of lower F_1-score, and therefore BFs will favour models of higher F_1-score. Verifying this hypothesis will allow us to incorporate statistical tools into the design of OFs; we can perform straightforward tests training models of equally spaced F_1-score, and computing BF between all pairs.

In Fig. 8.2, we show the relationships between F_1-score and BF for various conditions. Firstly, under a standard training regime with full BF comparisons between all pairs, we see that in most cases, the model with higher F_1-score is favoured by BF. Some comparisons near the diagonals of Fig. 8.2 favour the model with lower F_1-score, although in these cases we argue that the difference in model quality is not overwhelming, since $\left|f_i - f_j\right| \lessapprox 0.1$. In Fig. 8.2b, we run a complete model training subroutine, but compute the BF based on fewer experiments and particles (retaining a fraction $N_p' = 0.2N_p,\ N_e' = 0.2N_e$ for comparisons). This verifies an earlier claim from Section 5.2.1: although the strength of evidence is weaker given reduced BF resources, the direction of the evidence is usually the same, i.e. the insight is indicative of the true physics, so we can save considerable compute time by trusting these restricted BF calculations. On the other extreme, in Fig. 8.2**(c)**, where models are trained with – and BFs based upon – even greater resources than Fig. 8.2**(a)**, we see a similar effect: adding resources strengthens the evidence, but does not fundamentally change the outlook. Finally, in addition to reducing the resources used per BF calculation, we reduce the number of comparisons computed in Fig. 8.2**(d)**, as permitted when rating models according to the OF to be described in Section 8.2.7, or similar measures which can yield fitnesses from reduced data. Essentially we can see that the insight is largely the same from the most and least expensive training/comparison strategies, and by leveraging the available evidence (Fig. 8.2**(d)**), rather than brute-force computing as much evidence as possible (Fig. 8.2**(c)**), we can achieve similar results. Note that the time saving reported between full and partial connectivity between models scales with N_m: here, with $N_m = 10$, the former computes 45 BFs, while the latter computes 17; for $N_m = 60$, as used in full instances/runs presented in this chapter, these rise to 1770 and 600 BFs computations respectively, so the benefit of the latter scheme is amplified.

8.1.3 *Hyperparameter search*

Firstly we will validate our reasoning that F_1-score is a sensible figure of merit, by directly invoking it as the objective function. That is, we first implement a GA, using the mapping between models and chromosomes outlined above, where we fix the numbers of sites $d = 4$, and assume full connectivity between the sites, with $x-$, $y-$ and $z-$ couplings available, such that there are $N_t = 3 \times \binom{4}{2} = 18$ terms in $\mathcal{T}$, so that the total population is of size 2^{18} chromosomes. We can then sweep over the GA hyperparameters to find a suitable configuration: in Fig. 8.3

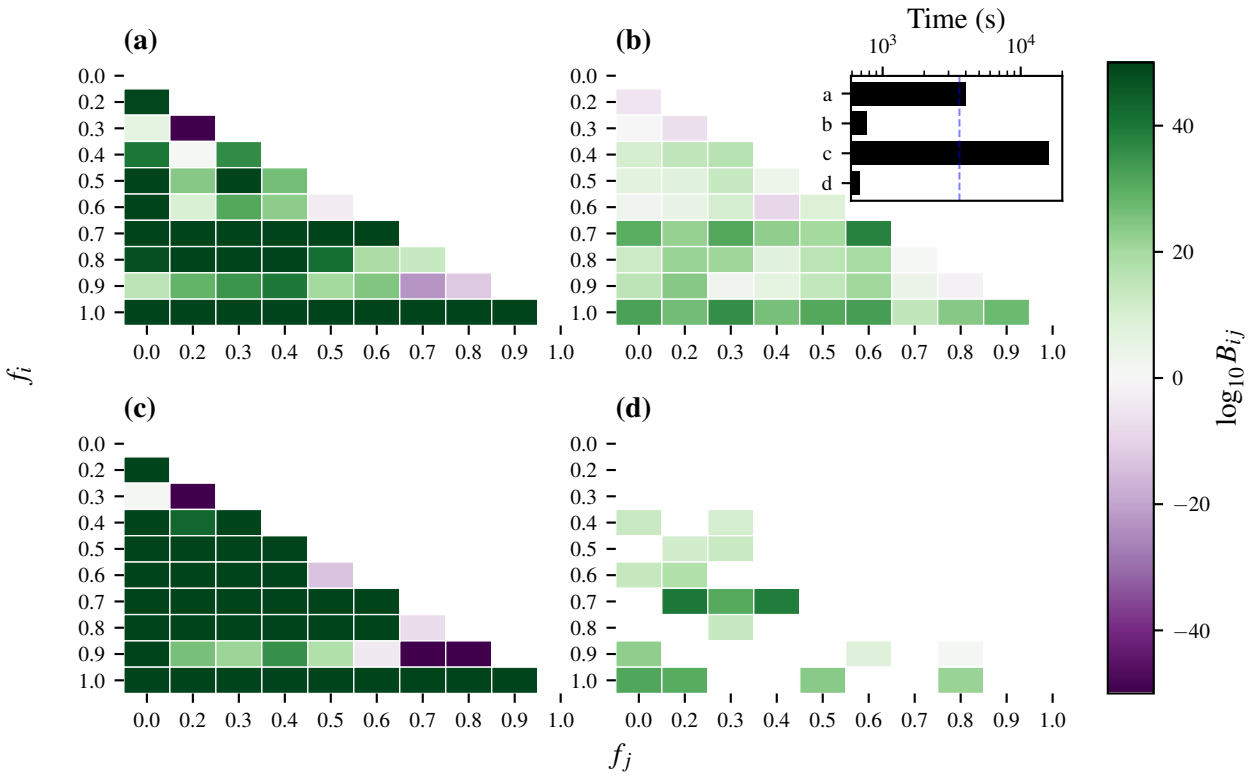


Figure 8.2: Pairwise Bayes factor by F_1-score. Each tile shows $\log_{10} B_{ij}$ against the F_1-score of the two candidates $\hat{H}_i$ (f_i on the y-axis) and $\hat{H}_j$ (f_j on the x-axis). $\log_{10} B_{ij} > 0$ shown in green ($\log_{10} B_{ij} < 0$ shown in purple) indicates statistical evidence that $\hat{H}_i$ ($\hat{H}_j$) is the better model with respect to the observed data. Visualisation is curtailed to $\log_{10} B_{ij} = \pm 50$. **a**, Models are trained with $N_e = 500, N_p = 2500$, and all available data is used in the calculation of BFs. **b**, $N_e = 500, N_p = 2500$ using only a fraction (0.2) of experiments/particles for BF calculations. **c**, $N_e = 1000, N_p = 5000$, using all available data in the calculation of BFs. **d**, $N_e = 500, N_p = 2500$, comparing only a subset of pairs of models through BFs, and using only a fraction (0.2) of experiments/particles for those calculations. This pairwise comparison strategy is used for the OF in Section 8.2.7. **Inset**, timings for each approach in seconds, with $t = 1$hr marked vertically in blue. Implementation details are listed in Table A.1.

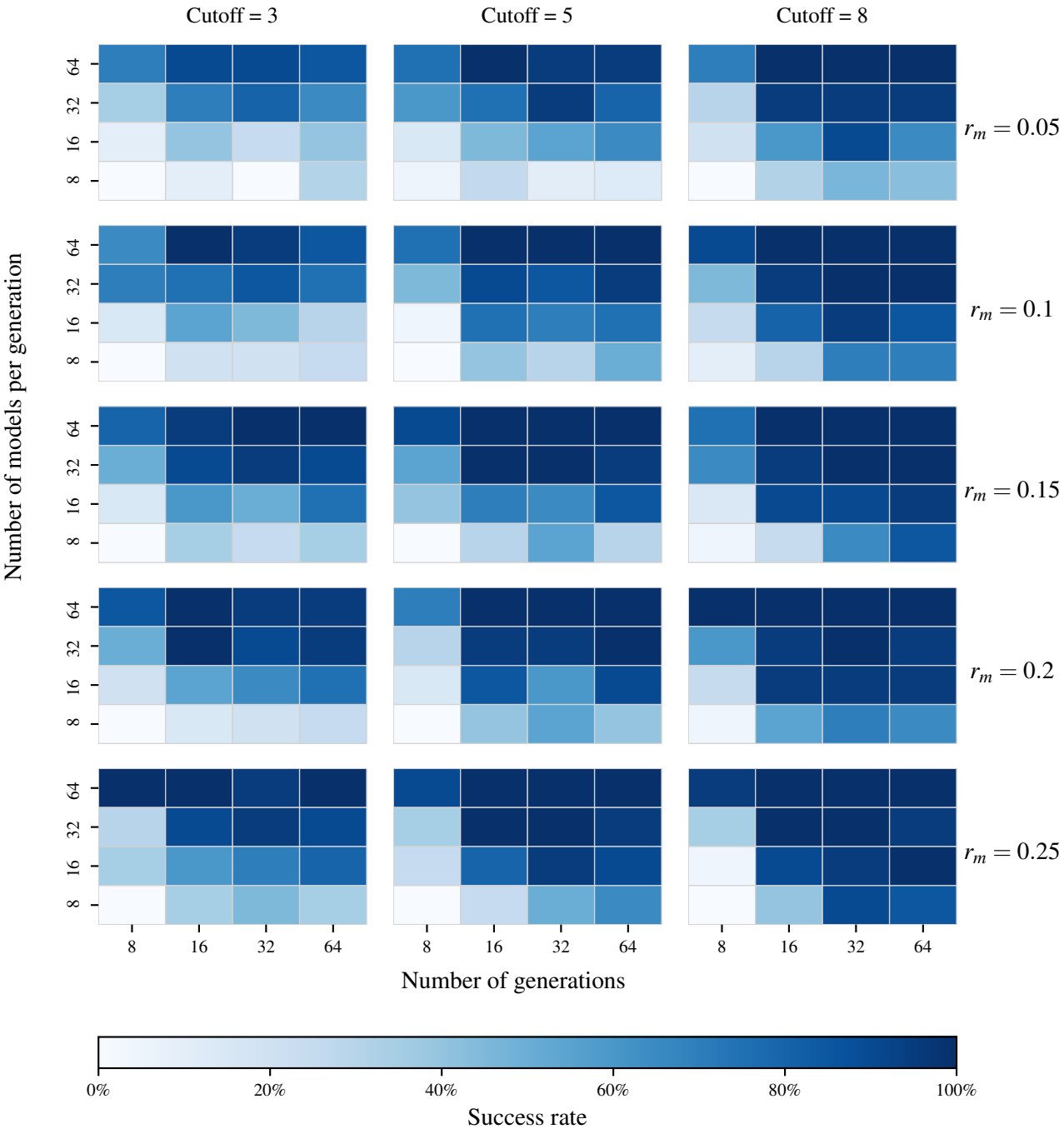


Figure 8.3: Genetic algorithm hyperparameter sweep. Each tile shows the success rate of a run with hyperparameter settings $\{N_m, N_g, r_m, \text{cutoff}\}$, where the success rate is the percentage of 20 instances which found a random $\hat{H}_0$ using F_1-score as objective function. Each subplot shows the success rates for varying numbers of generations, $N_G \in \{8, 16, 32, 64\}$, and numbers of models per generation, $N_m \in \{8, 16, 32, 64\}$. A subplot is generated for ranges of the mutation rate, r_m and the number of generations for which the elite model is unchanged after which the GA is cut off. Implementation details are listed in Table A.1.

we show how the choice of parameters affect the success rate of precisely identifying the target chromosome, which is chosen at random for each instance, and we run 20 instances of each configuration. The studied hyperparameters[6] are

i. number of generations, N_g;
ii. number of models per generation, N_m;
iii. mutation rate, r_m;
iv. number of generations for which a candidate must reign as the strongest observed, before the search terminates, the *cutoff*.

Naturally, we expect that running for more generations with more models per generation will result in a more effective search in the model space, having examined $N_g N_m$ models. We must also consider, however, that – in realistic cases of QMLA – the total computation time scales dramatically with these parameters, since training and comparing models are expensive subroutines. Our goal is therefore to identify the set of hyperparameters which best searches the model space while demanding the lowest $\{N_g, N_m\}$. We see that, unsurprisingly, the GA performs poorly when run with few resources, but broadly the performances are similar provided it is run with sufficient resources. We can bound the parameters $r_m \geq 0.1, \text{cutoff} \geq 5, N_m \geq 16, N_g \geq 16$ to ensure a reasonable search through the model space, without having to consider a prohibitive number of models. We must bear in mind, however, that this parameter sweep refers only to the trivial case where the F_1-score is used as the OF, so we do not expect such high success rates in realistic cases.

8.2 OBJECTIVE FUNCTIONS

We have alluded to the central problem in building a GA into QMLA: how to evaluate trained candidate models in the absence of a natural objective function (OF). In Sections 8.2.1 to 8.2.7 we will propose and analyse a number of potential OFs, some of which will underlie later studies in this thesis. We conclude this study by comparing the proposed OFs and selecting one for consideration in the remainder of this chapter; readers interested in the final application may prefer to skip to Section 8.2.8.

We will show how each OF computes a fitness, g_i, for candidate models, $\hat{H}_i$. For examples of each, we group together some demonstrative values in Table 8.2. For each $\hat{H}_i$, we may refer to

- $\mathcal{L}_i$, total log total likelihood (TLTL), introduced in Section 4.4;
- k_i, the model's cardinality, i.e. number of terms in its parameterisation;
- $\mathcal{E}_i$, the bespoke set of experiments composed by the experiment design heuristic (Section 4.6) solely for training $\hat{H}_i$;
- $n = |\mathcal{E}_i|$, the number of samples (datapoints) used in training $\hat{H}_i$.

6 These and further hyperparameters can be swept using code within the QMLA codebase, in the directory scripts/genetic_alg_param_sweep.

In Table 8.2, we consider six randomly generated exemplary models – of varying quality with respect to the target, $\hat{H}_0$, listed in Eq. (8.3) – to demonstrate each OF's outcomes.

$$\begin{aligned}
\hat{H}_0 &= \hat{\sigma}^z_{(1,2)}\hat{\sigma}^z_{(1,3)}\hat{\sigma}^z_{(2,3)}\hat{\sigma}^z_{(2,5)}\hat{\sigma}^z_{(3,5)};\\
\hat{H}_a &= \hat{\sigma}^z_{(1,5)}\hat{\sigma}^z_{(3,4)}\hat{\sigma}^z_{(4,5)};\\
\hat{H}_b &= \hat{\sigma}^z_{(1,4)}\hat{\sigma}^z_{(1,5)}\hat{\sigma}^z_{(2,5)}\hat{\sigma}^z_{(3,4)};\\
\hat{H}_c &= \hat{\sigma}^z_{(1,2)}\hat{\sigma}^z_{(1,5)}\hat{\sigma}^z_{(2,4)}\hat{\sigma}^z_{(2,5)}\hat{\sigma}^z_{(4,5)};\\
\hat{H}_d &= \hat{\sigma}^z_{(1,3)}\hat{\sigma}^z_{(1,4)}\hat{\sigma}^z_{(1,5)}\hat{\sigma}^z_{(2,4)}\hat{\sigma}^z_{(2,5)}\hat{\sigma}^z_{(3,4)}\hat{\sigma}^z_{(3,5)};\\
\hat{H}_e &= \hat{\sigma}^z_{(1,2)}\hat{\sigma}^z_{(1,3)}\hat{\sigma}^z_{(1,5)}\hat{\sigma}^z_{(2,3)}\hat{\sigma}^z_{(2,5)}\hat{\sigma}^z_{(4,5)};\\
\hat{H}_f &= \hat{\sigma}^z_{(1,2)}\hat{\sigma}^z_{(1,3)}\hat{\sigma}^z_{(2,3)}\hat{\sigma}^z_{(2,4)}\hat{\sigma}^z_{(2,5)}\hat{\sigma}^z_{(3,4)}\hat{\sigma}^z_{(3,5)}.
\end{aligned} \tag{8.3}$$

8.2.1 *Inverse log-likelihood*

$\mathcal{L}_i$, defined in Eq. (4.18), can be thought of as a measure of the ability of a given model to reproduce a given dataset $\mathcal{D}$ from experiments $\mathcal{E}$. This can be immediately interpreted as an OF, provided each candidate model computes a meaningful total log total likelihood (TLTL), requiring that they are all based on the same set of experiments, $\mathcal{E}_v$, which are designed explicitly for the purpose of model evaluation.

TLTL are negative and the strongest model has lowest $|\mathcal{L}_i|$ (or highest $\mathcal{L}_i$ overall), so the corresponding OF for candidate $\hat{H}_i$ is

$$g^L_i = \frac{-1}{\mathcal{L}_i}. \tag{8.4}$$

In our tests, Eqn. 8.4 is found to be too generous to poor models, assigning them non-negligible probability. Its primary flaw, however, is its reliance on $\mathcal{E}_v$: in order that the TLTL is significant, it must be based on meaningful experiments, the design of which can not be guaranteed in advance, or at least risks introducing strong bias towards some models.

8.2.2 *Akaike information criterion*

A common metric in the general field of model selection is Akaike information criterion (AIC) [181]. Incorporating TLTL, AIC objectively quantifies how well a given model accounts for data from the target system, and explicitly punishes models which use extraneous parameters by incurring a penalty on k_i. AIC is given by

$$AIC_i = 2k_i - 2\mathcal{L}_i. \tag{8.5}$$

Method		$\hat{H}_a$	$\hat{H}_b$	$\hat{H}_c$	$\hat{H}_d$	$\hat{H}_e$	$\hat{H}_f$
	F_1	0.0	0.2	0.4	0.5	0.7	0.8
	k	3	4	5	7	6	7
	$\overline{l_e}$	0.86 ± 0.29	0.84 ± 0.29	0.77 ± 0.27	0.78 ± 0.29	0.79 ± 0.26	0.79 ± 0.26
	$\mathcal{L}_i$	-143	-152	-131	-150	-125	-124
Inverse log-likelihood	g_i^L	0.00698	0.00659	0.00766	0.00669	0.00803	0.00804
	%	23	0	25	0	26	26
Akaike Info Criterion	AIC	293	311	271	313	261	263
	AICC	293	312	272	314	262	264
	w_i^A	1.81e-07	1.4e-11	0.00724	4.15e-12	1	0.334
	g_i^A	1.17e-05	1.03e-05	1.35e-05	1.01e-05	1.46e-05	1.43e-05
	%	22	0	25	0	27	26
Bayesian Info Criterion	BIC	301	322	284	331	277	281
	w_i^B	5.49e-66	1.26e-70	1.97e-62	1.11e-72	8.43e-61	8.95e-62
	g_i^B	1.11e-05	9.65e-06	1.24e-05	9.11e-06	1.31e-05	1.27e-05
	%	23	0	25	0	27	26
Bayes factor points	g_i^p	0	2	3	2	3	5
	%	0	13	20	13	20	33
Ranking	Ranking	6	4	3	5	2	1
	g_i^R	0	0.1	0.2	0	0.3	0.4
	%	0	10	20	0	30	40
Elo rating	Rating	909	944	1042	1007	1011	1084
	g_i^E	0	35	133	98	102	175
	%	0	0	26	19	20	34
Residuals	mean$\{\tilde{r_p^e}\}$	0.132	0.146	0.114	0.138	0.0858	0.0715
	g_i^r	0.753	0.729	0.785	0.743	0.836	0.862
	%	23	0	24	0	26	27

Table 8.2: Examples of how each objective function (OF), g as described in Section 8.2.1 to Section 8.2.7, assign selection probability (denoted %) to the same set of candidate models, $\{\hat{H}_i\}$ listed in Eq. (8.3), when attempting to learn data from $\hat{H}_0$. Intermediate quantities, e.g. w_i^A, g_i^p are described in the section of the main text describing the corresponding OF. For each model we first summarise its F_1-score (Eq. (8.2)), number of terms k, median likelihood $\overline{l_e}$ (Eq. (5.5)), and TLTL $\mathcal{L}_i$ (Eq. (4.18)), We use $n = 250$ samples, i.e. $\mathcal{L}_i$ is a sum of n likelihoods . The set of models is truncated so that only the strongest four are assigned selection probability.

In practice we use a slightly modified form of Eqn. 8.5 which corrects for the number of samples $n = |\mathcal{E}_i|$, called the Akaike information criterion corrected (AICC),

$$AICC_i = AIC_i + 2k_i \frac{k_i + 1}{n - k_i - 1}. \tag{8.6}$$

Model selection from a set of candidates occurs simply by selecting the model with lowest $AICC$. Following [181], by using Eqn. 8.6 as a measure of *relative likelihood* we retrieve selection probability via the *Akaike weights*,

$$w_i^A = \exp\left(\frac{AICC_{\min} - AICC_i}{2}\right), \tag{8.7}$$

where $AICC_{\min} = \min_i\{AICC_i\}$.

Akaike weights impose strong penalties on models which do not explain the data well, but also punish models with more parameters, i.e. potentially overfitting models, effectively searching for the strongest and simplest model simultaneously. The level of punishment for poorly performing models is likely too drastic: very few models will be in a range sufficiently close to $AICC_{\min}$ to receive a meaningful Akaike weight, suppressing diversity in the model population. Indeed, we can see from Table 8.2 that this results in most models being assigned negligible weight, which is not useful for parent selection. Instead we compute a straightforward quantity related to AIC,

$$g_i^A = \left(\frac{1}{AICC_i}\right)^2, \tag{8.8}$$

where we square the inverse AICC to amplify the difference in quality between models, such that stronger models are rewarded.

8.2.3 *Bayesian information criterion*

Related to the concept of AIC (Eqn. 8.5), is that of Bayesian information criterion (BIC),

$$BIC_i = k_i \ \ln(n_i) - 2\mathcal{L}_i, \tag{8.9}$$

where k_i, n_i and $\mathcal{L}_i$ are as defined on Page 112. Analogously to Akaike weights, *Bayes weights* as proposed in §7.7 of [182], are given by

$$w_i^B = \exp\left(-\frac{BIC_i}{2}\right). \tag{8.10}$$

BIC is harsher than AIC in its punishment of models' cardinality k_i, demanding substantial statistical justification for the inclusion of more parameters. Again, this may be overly cumbersome for our use case: with such a relatively small number of parameters, the punishment is disproportionate. As with Akaike weights, rather than using Bayes weights directly, we opt for an OF related to them,

$$g_i^B = \left(\frac{1}{BIC_i}\right)^2. \tag{8.11}$$

8.2.4 *Bayes factor points*

A cornerstone of model selection within QMLA is the calculation of BFs (see Section 5.2). We can compute the pairwise BF between two candidate models, B_{ij}, according to Eqn. 5.7. B_{ij} can be based on some evaluation dataset, $\mathcal{E}_v$, but can also be calculated from $\mathcal{E}_i \cup \mathcal{E}_j$: this is a strong advantage since the resulting insight (Eqn. 5.8) is based on experiments which were bespoke to both $\hat{H}_i, \hat{H}_j$. As such we can be confident that this insight accurately points us to the stronger of two candidate models.

We can utilise this facility by computing the BF between all pairs of models in a set of N_m candidates $\{\hat{H}_i\}$, i.e. compute $\binom{N_m}{2}$ BFs. Note that this is computationally expensive: in order to train $\hat{H}_i$ on $\mathcal{E}_j$ requires a further $|\mathcal{E}_j|$ experiments, each requiring N_P particles[7], where each particle corresponds to a unitary evolution and therefore the calculation of a matrix exponential. The size of the model space is then quite a heavy disadvantage: examining N_g generations requires $N_g \times \binom{N_m}{2}$ BF calculations for complete assessment.

In the case where all pairwise BF are performed, we can assign a point to $\hat{H}_i$ for every comparison in which it is deemed superior, according to Eq. (5.8).

$$g_i^p = \sum_{j \in \mu} b_{ij}, \quad b_{ij} = \begin{cases} 1, & B_{ij} > 1 \\ 0, & \text{otherwise.} \end{cases} \tag{8.12}$$

This is a straightforward mechanism, but is overly blunt because it does not account for the *strength* of the evidence in favour of each model. For example, a dominant model will receive only a slightly higher selection probability than the second strongest, even if the difference between them was $B_{ij} = 10^{100}$. Further, the unfavourable scaling make this an expensive method.

8.2.5 *Ranking*

Related to the BF points of the previous section, we can rank models in a generation based on their number of BF points. BF points are assigned as in Eqn. 8.12, but instead of corresponding directly to fitness, we assign models a rank R, i.e. the model with highest g_i^p gets $R = 1$, and the model with n^{th} highest g_i^p gets $R = n$. Note here we truncate μ, meaning we remove the worse-performing models and retain only N'_m models, before calculating R, because computing R using all N_m models results in less distinct selection probabilities.

$$g_i^R = \frac{N'_m - R_i + 1}{\sum_{j=1}^{N'_m} j}, \tag{8.13}$$

where R_i is the ranking of $\hat{H}_i$ and N'_m is the number of models retained after truncation. Eq. (8.13) has a similar effect to Eq. (8.12) but awards higher selection probability to the strongest models.

7 Caveat the reduction in overhead outlined in Section 5.2.1.

However, it too overlooks the nuanced perspective available through the total statistical evidence gathered by the series of BFs.

8.2.6 *Residuals*

Recall at each experiment, N_P particles are compared against a single experimental datum, d. By definition, d is the binary outcome of the measurement on Q under experimental conditions e. That is, d encodes the answer to the question: after time t under Hamiltonian evolution, did Q project onto the basis we have labelled $|d\rangle$ (usually the same as the input probe state $|\psi\rangle$)?

In practice we often have access to the complete likelihood, i.e. rather than a binary value, we have a number representing the probability that Q will project on to $d = 0$ for a given experiment e, $\Pr_Q(0|e)$. The likelihood – in this case equivalent to the expectation value[8] – for Q is usually given by $\left|\langle\psi| e^{-i\hat{H}_0 t} |\psi\rangle\right|^2$. Likewise, we can simulate this quantity for each particle, $\Pr_p(0|e)$. This allows us to calculate the *residual* between the system and individual particles' likelihoods, r_p^e; we can hence compute the mean residual across all particles in a single experiment, r^e:

$$\begin{aligned} r_p^e &= \left|Pr_Q(0|e) - Pr_p(0|e)\right| \\ r^e &= \underset{p}{\text{mean}}\{r_p^e\} \end{aligned} \tag{8.14}$$

Residuals capture how closely the particle distribution reproduced the dynamics from Q: $r_p^e = 0$ indicates perfect prediction, while $r_p^e = 1$ is completely incorrect. We can therefore maximise the quantity $1 - r$ to find the best model, using the OF

$$g_i^r = |1 - \underset{e\in\mathcal{E}}{\text{mean}}\{r^e\}|^2. \tag{8.15}$$

This OF can be thought of in frequentist terms as similar to the residual sum of squares, although instead of summing the residual squares, we take the average to ensure $0 \leq r \leq 1$. g_i^r encapsulates how well the candidate model reproduces a particular set of dynamics from the target system, as a proxy for how well that candidate describes the system. This is not always a safe figure of merit: in most cases, we do not expect parameter learning to perfectly optimise $\vec{\alpha}_i$. Reproduced dynamics alone can not capture the prospect that $\hat{H}_i = \hat{H}_0$, but rather inform statistical measures such as BF, that allow us to make qualified statements about the system.

This OF provides a useful test for QMLA's GA: by simulating the case where parameters *are* learned perfectly, such that we know that g_i^r truly represents the ability of $\hat{H}_i$ to mimic $\hat{H}_0$, then this OF guarantees to promote the strongest models, especially given that $\hat{H}_i = \hat{H}_0 \implies r_p^e = 0 \; \forall \; \{e, p\}$. In realistic cases, however, the non-zero residuals – even for strong $\hat{H}_i$ – may arise from imperfectly learned parameters, rendering the usefulness of this OF uncertain. Finally, it

8 For consistency with QInfer [136] – on which QMLA's code base builds – we call the expectation value for the system $\Pr_Q(0)$; the same quantity can be computed for each particle, called $\Pr_p(0)$.

does not account for the cardinality, k_i, of the candidate models, which all ML protocols aim to avoid in general; this could result in favouring severely overfitting models in order to gain marginal improvement in residuals.

8.2.7 *Bayes factor enhanced Elo ratings*

A popular tool for rating individual competitors in sports and games is the *Elo rating* system, e.g. used to rate chess players and soccer teams [183, 184], also finding application in the study of animal hierarchies [185]. Elo ratings allow for evaluating the relative quality of individuals based on incomplete pairwise competitions, e.g. despite two football teams having never played against each other before, it is possible to quantify the difference in quality between those teams, and therefore to predict a result in advance [186]. There is a direct parallel between these types of competitions and QMLA: we similarly have a pool of individual competitors (models), which we can place in direct competition, and quantify the comparative outcome through BF, in order to determine the preferred candidate.

Elo ratings are transitive: given some inter-connectivity in a generation, we need not compare *every* pair of models in order to make meaningful claims about which are strongest; it is sufficient to perform a subset of comparisons, ensuring each individual undergoes robust competition. We can take advantage of this transitivity to reduce the combinatorial overhead usually associated with computing bespoke BFs between all models (i.e. using their own training data $\mathcal{E}_i$ instead of a generic $\mathcal{E}_v$). In practice, we map N_m models within a generation to vertices on a regular graph of degree $N_m/3$, i.e. each model is connected to $N_m/3$ other models within μ. Models which share an edge then undergo BF comparison. For example, with $N_m = 60$ this leads to 600 BF calculations, compared with 1770 calculations in the fully connected graph. While every pair of models $(\hat{H}_i, \hat{H}_k)$ are not directly connected, there is always a chain of length $l \leq l_{\max}$ edges between them. For $N_m = 60$, we find $l_{\max} = 2$, e.g. for $\hat{H}_i, \hat{H}_k$ disconnected, there are comparisons $\{(\hat{H}_i, \hat{H}_j), (\hat{H}_j \hat{H}_k)\}$.

The Elo rating scheme is a nonlinear points transfer system, as follows: upon creation, $\hat{H}_i$ is assigned a rating R_i; every comparison with a competitor $\hat{H}_j$ results in B_{ij}; R_i is updated according to the known strength of its competitor, R_j, as well as the result B_{ij}. The Elo update ensures that winning models are rewarded for defeating another model, but that the extent of that reward reflects the quality of its opponent. As such, this is a fairer mechanism than BF points, which award a point for every victory irrespective of the opposition: if $\hat{H}_j$ is already known to be a strong or poor model, then ΔR_i changes the credence of $\hat{H}_i$ proportional to the latest evidence. It achieves this by first computing the *expected* result of a given comparison with respect to each model, with an exponential response to the difference in their current ratings,

$$E_i = \frac{1}{1 + 10^{\frac{R_j - R_i}{400}}}; \tag{8.16a}$$

Model		R_i	E_i	S_i	B_{ij}	$\log_{10}(B_{ij})$	ΔR_i	R_i'
$\hat{H}_a > \hat{H}_b$	$\hat{H}_a$	1000	0.76	1	1e+100	100	0.24	1024.0
	$\hat{H}_b$	800	0.24	0	1e-100	100	-0.24	776.0
$\hat{H}_b > \hat{H}_a$	$\hat{H}_a$	1000	0.76	0	1e-100	100	-0.76	924.0
	$\hat{H}_b$	800	0.24	1	1e+100	100	0.76	876.0

Table 8.3: Example of Elo rating updates. We have two models, where $\hat{H}_a$ is initially quantified as a stronger candidate than $\hat{H}_b$, i.e. has a higher starting Elo rating, R_i. We demonstrate the effect when there is strong evidence[9] in favour of either model through BF comparison, $B_{ij} \approx \mathcal{O}(10^{100})$. In the first case, $\hat{H}_a$ defeats $\hat{H}_b$, as firmly expected according to their initial ratings, so the corresponding reward (cost) for $\hat{H}_a$ ($\hat{H}_b$) is relatively small. In the second case, contrary to prediction $\hat{H}_b$ outperforms $\hat{H}_a$, so $\hat{H}_b$ receives a large share of Elo points from $\hat{H}_a$.

$$E_i + E_j = 1, \tag{8.16b}$$

Then, we find the binary *score* from the perspective of each model,

$$\begin{cases} B_{ij} > 1 & \Rightarrow \; S_i = 1; \; S_j = 0 \\ B_{ij} < 1 & \Rightarrow \; S_i = 0; \; S_j = 1 \end{cases} \tag{8.17}$$

which is used to determine the change to each model's rating,

$$\Delta R_i = \eta \times (S_i - E_i)\,. \tag{8.18}$$

An important detail is the choice of η, i.e. the *weight* of the change to the models' ratings. In standard Elo schemes this is a fixed constant, but here – taking inspiration from football ratings where η is the number of goals by which one team beat the other – we weight the change by the strength of our belief in the outcome: $\eta \propto |B_{ij}|$. That is, similarly to the interpretation of Eqn. 5.8, we use the evidence in favour of the winning model to transfer points from the loser to the winner, albeit we temper the effect by instead using $\eta = \log_{10}(B_{ij})$, since BF can give very large numbers. In total, then, following the comparison between models $\hat{H}_i, \hat{H}_j$, we can perform the Elo rating update

$$R_i' = R_i + \log_{10}(B_{ij}) \left(S_i - \frac{1}{1 + 10^{\frac{R_j - R_i}{400}}} \right). \tag{8.19}$$

This procedure is easiest to understand by following the example in Table 8.3.

Finally, it remains to select the starting rating R_i^0 to assign models upon creation. Although this choice is arbitrary, it can have a strong effect on the progression of the algorithm. Here

9 Note to achieve $B_{ij} = 10^{100} = e^{\mathcal{L}_i - \mathcal{L}_j} \implies \mathcal{L}_i - \mathcal{L}_j = \ln(10^{100}) \approx 7$.

we impose details specific to the QMLA GES: at each generation we admit the top two models automatically for consideration in the next generation, such that strongest models can stay alive in the population and ultimately win. These are called *elite* models, $\hat{H}_e^1, \hat{H}_e^2$. This poses the strong possibility for a form of generational wealth: if elite models have already existed for several generations, their Elo ratings will be higher than all alternatives by definition. Instead, we would prefer that newly spawned models can overtake the Elo rating of elite models. To resolve this, at each generation, all models – including $\hat{H}_e^1, \hat{H}_e^2$ – are assigned the same initial rating, $R_i^0 = 1000$.

In order to derive a meaningful selection probability for each candidate, we must first ground the raw Elo rating at each generation μ: we subtract the lowest rating among the entertained models, R_{min}^{μ}. This serves to ensure the range of remaining R_i represent only by the difference between models as assessed within μ: a very strong model might have much higher R_i than its contemporaries, but that difference was earned exclusively by comparison within μ, so it is deserving of its higher fitness and therefore greater selection probability. We perform this step before truncation[10], so that the models remaining post-truncation all have non-zero fitness. Finally, then, we name this OF the *Bayes factor enhanced Elo ratings (BFEER)*: the fitness of each model $\hat{H}_i^{\mu} \in \mu$ is attained directly from its rating R_i after undergoing Elo updates based on BFs in the current generation, minus the minimum rating of any model in the same generation R_{min}^{μ},

$$g_i^E = R_i^{\mu} - R_{\text{min}}^{\mu}. \tag{8.20}$$

The advantage of this OF is that it gives a meaningful value on the absolute quality of every model, allowing us to determine the strongest, and importantly to find the relative strength between models. Further, it exploits bespoke BFs, i.e. based on the considered models' individually designed $\mathcal{E}_i$, removing the impetus to design $\mathcal{E}_v$ which can evaluate models definitively. One disadvantage is that it does not explicitly punish models based on their cardinality, however this feature is partially embedded by adopting BF for the comparisons, which are known to protect against overfitting [156].

8.2.8 *Objective function selection*

Having proposed a series of possible objective functions, we are now in a position to analyse their appropriateness in the context of QMLA. Recall from Section 8.1.2 that we use F_1-score as the figure of merit against which individual models are measured; we can compare OFs on the basis of the F_1-score of models they spawn.

First we can remark on the examples listed in Table 8.2. The OFs which rely on the TLTL, i.e. g^L, g^A, g^B, g^r, are effectively tricked by the log-likelihood, which appears reasonably convincing

10 We truncate the N_m models on μ by the truncation rate τ, i.e. only τN_m models are considered as potential parents in the GA. In this chapter we use $\tau = 1/3$.

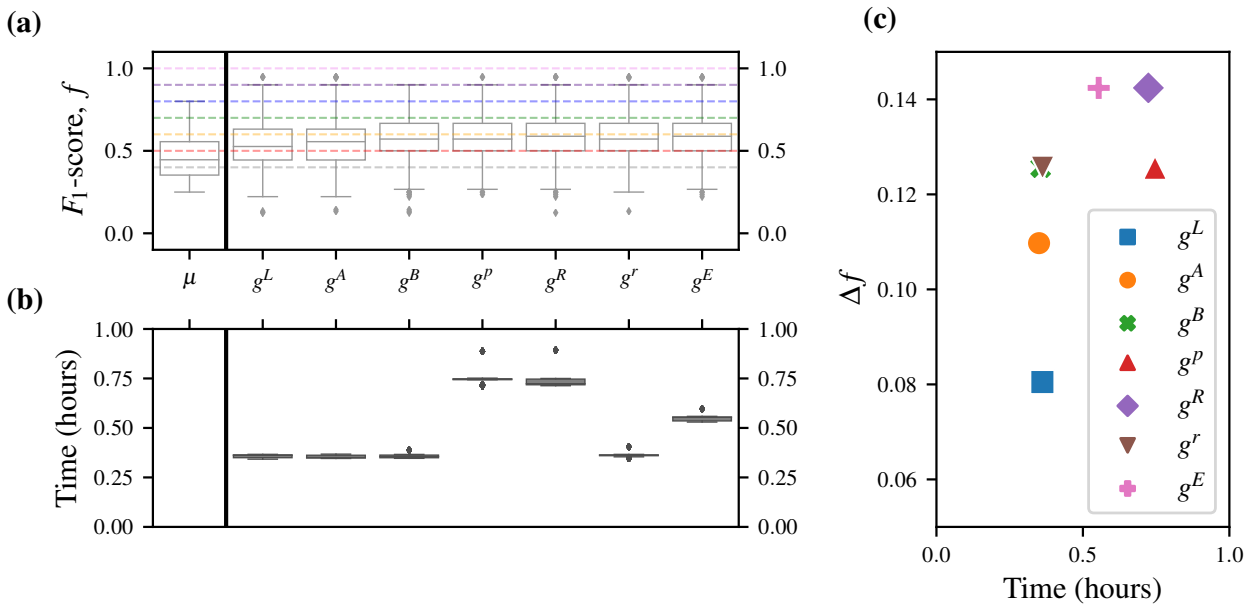


Figure 8.4: Comparison between proposed objective functions (OFs). Each OF trains the same initial generation of $N_m = 28$ models with resources $N_E = 500, N_P = 2000$, and then design a new set of N_m models, μ_h, through a roulette strategy, such that the only difference between OF's output is how they assign selection probability. We run each OF 25 times for the same target system, a 4–qubit Heisenberg–XYZ model. **a**, Box-plot of F_1-score, f, for the models in μ_h in each case, where the median and inter–quartile ranges are indicated by the boxes, as well as those of the initial generation μ centered on $f_\mu = 0.45$. We mark $f = \{0.4, 0.5, ..., 1.0\}$ for ease of interpretation. **b**, Box-plots of the time taken to compute the single generation in each case. **c**, Difference between the median f among the newly proposed models from f_μ, Δf, plotted against the time to achieve the result.

for poor models, e.g. $\hat{H}_a, \hat{H}_c$. This underlines the risk in building $\mathcal{E}_v$, which can be biased towards weak models, for example resulting in high selection probability for $\hat{H}_a$ which has $f = 0$, while $\hat{H}_d$, with $f = 0.4$ is discarded. On the other hand, OFs grounded by the BF (g^p, g^R, g^E) invariably promote models of higher F_1-score, justifying the role of statistical evidence used for those calculations. Overall, however, the insights from this complete example are insufficient to make general claims about the performance of each OF, so here we examine their outputs systematically.

Returning to the task of determining our favoured OF, we choose some random target $\hat{H}_0$, and run a single generation of the GES with each OF, allowing us to assess their performance based on the quality of models the GA produces under their respective guidance. We train the same batch of $N_m = 28$ random models in each case, and allow each OF to compute the selection probabilities for those models, and therefore direct the design of the hypothetical next generation of models, μ_h. Fig. 8.4 shows the distribution of F_1-score for the models within μ_h, as

generated by each proposed OF, also accounting for the time taken in each case, i.e. we report the time to train and evaluate the single generation on a 16–core node.

We can see that a strong balance of outcome with resource considerations are achieved by the BFEER strategy, Section 8.2.7, so we will use it for the case study presented in this chapter. We strongly emphasise, however, that the performance of each objective function can vary under alternative conditions, and therefore similar analysis may be warranted for future applications. For instance, if $t_{\max}$ is known to be small, in smaller model spaces, using g^r results in higher success rates. We retain BFEER, however, for generality and novelty, but it is important to recognise that the results listed do not reflect an upper limit of QMLA's performance, but rather reflect the constraints of the system under study; each Q will bring its own unique considerations which can result in significantly stronger or weaker performance under each OF. In particular, in Chapter 10, we will later use g^r_i – the residual OF described in Section 8.2.6 – to study a larger model space under assumptions of perfect parameter learning.

8.3 APPLICATION

Having introduced all the necessary concepts of GAs, mapped them to the QMLA framework and chosen a suitable OF, we can finally use the GES for model search. In summary of this chapter so far, we use the following settings.

- Models are mapped to a unique bit string (chromosome), where each bit represents whether a given model term (gene) is present; chromosomes are of length N_t genes.
- A maximum of N_g generations are run, each with N_m unique models.
- Candidate models are trained using QHL, specifically by using interactive quantum likelihood estimation (IQLE)[11] for parameter estimation.
- Models' fitness are determined by their BFEER, after having been trained by QHL and compared against some set of competing candidate models.
- For generating models on $\mu + 1$, the models on μ are first truncated with truncation rate τ; the remaining τN_m models are assigned selection probability based on their fitness.
- Pairs of models are selected to become parents sequentially using roulette selection. Highly favoured models can parent many offspring models.
- Selected parent models are crossed over via a one-point cross-over, at crossover location $\kappa \in \left(\frac{N_t}{4}, \frac{3N_t}{4}\right)$, and probabilistically mutated with rate $r_m = 0.25$.
- The top two elite models from μ, $\hat{H}^{\mu}_e$, are included on the subsequent generation $\mu + 1$.
- If, after 5 generations, the highest-fitness (elite) model is unchanged, i.e. $\hat{H}^{\mu}_e = \hat{H}^{\mu-5}_e$, we terminate the search and declare that model as the champion, $\hat{H}' = \hat{H}^{\mu}_e$.

- Otherwise, after N_g generations, the highest-fitness model on the final generation is declared the global champion model, $\hat{H}' = \hat{H}_C^{N_g}$.

We will use a four-qubit model space under the fully parameterised Heisenberg formalism, Eq. (7.10), such that any pair of sites $\langle k, l \rangle$ can be coupled by any of the terms $\{\hat{\sigma}^x_{\langle k,l \rangle}, \hat{\sigma}^y_{\langle k,l \rangle}, \hat{\sigma}^z_{\langle k,l \rangle}\}$, so in total there are $N_t = |\mathcal{T}| = 3 \times \binom{4}{2} = 18$ terms, giving a model space of $2^{18} \approx 250,000$ viable models/chromosomes. For practical reasons[12], we set $N_m = 60$ and $N_g = 16$, although in most cases the elitism clause is triggered so the search terminates long before N_g is reached. The true parameters $\vec{\alpha}_0$ are assigned randomly in the range $(0.25, 0.75)$; within QHL the prior is set as a multivariate normal distribution with mean and standard deviation 0.5 ± 0.125. We choose $\hat{H}_0$ at random to contain half the available terms[13],

$$\hat{H}_0 = \hat{\sigma}^{yz}_{(1,2)} \hat{\sigma}^{z}_{(1,3)} \hat{\sigma}^{y}_{(1,4)} \hat{\sigma}^{xy}_{(2,3)} \hat{\sigma}^{x}_{(2,4)} \hat{\sigma}^{xz}_{(3,4)}. \tag{8.21}$$

8.3.1 *Analysis*

We will analyse the GES from four perspectives: a single model, a single generation, a single QMLA instance, and the overall performance across many instances, i.e. a run.

Recall that BFEER are mediated through random graphs: given N_m models on μ, a given model $\hat{H}_i$ undergoes some $N_i^{BF} < N_m$ BF comparisons. In Fig. 8.5 we show the BF results and effects on the rating of a random model, $\hat{H}_i$, where $N_m = 60$ and $N_i^{BF} = 12$, i.e. $\hat{H}_i$ is directly compared against 20% of contemporary models on μ. We see that $\hat{H}_i$'s rating is effected by whether it wins a given comparison, but also by the strength of evidence provided by the comparison (the BF), and the quality of its opposition $\hat{H}_j$, i.e. the initial rating of $\hat{H}_j$. For example, the sixth comparison finds $\hat{H}_j$ as the superior model, but the evidence is relatively weak and $\hat{H}_i, \hat{H}_j$ began with similar ratings, so R_i is not effected drastically.

We extend the single model analysis of Fig. 8.5 to all N_m models in the first generation in Fig. 8.6. The general trend is that models of higher F_1-score have their ratings increased, at the expense of models of lower F_1-score. After assessing models thus, the set of models is truncated with rate $\tau = 1/3$ to retain only the strongest 20 candidates, which are assigned selection probability, i.e. their chance of being chosen to become a parent during roulette selection, as in Section 3.3.3. N_m models are required to populate the next generation: the two models with highest R_i – the *elite* models – are automatically granted a position; the remaining positions are filled through the crossover procedure outlined above.

11 IQLE assumes complete access to the target system, see Section 4.3.1. This restricts the present analysis to simulatable, rather than physical, use cases, e.g. device calibration.

12 This is to ensure, with 15 available worker nodes, that all N_m models in a generation are trained within $4t_{\text{qhl}}$, where t_{qhl} is the time to train a single model.

13 Note we use a compact model representation, e.g. $\hat{H}_i = \hat{\sigma}^{yz}_{(1,2)} \hat{\sigma}^{z}_{(1,3)} = \hat{\sigma}^{y}_{(1,2)} + \hat{\sigma}^{z}_{(1,2)} + \hat{\sigma}^{z}_{(1,3)}$.

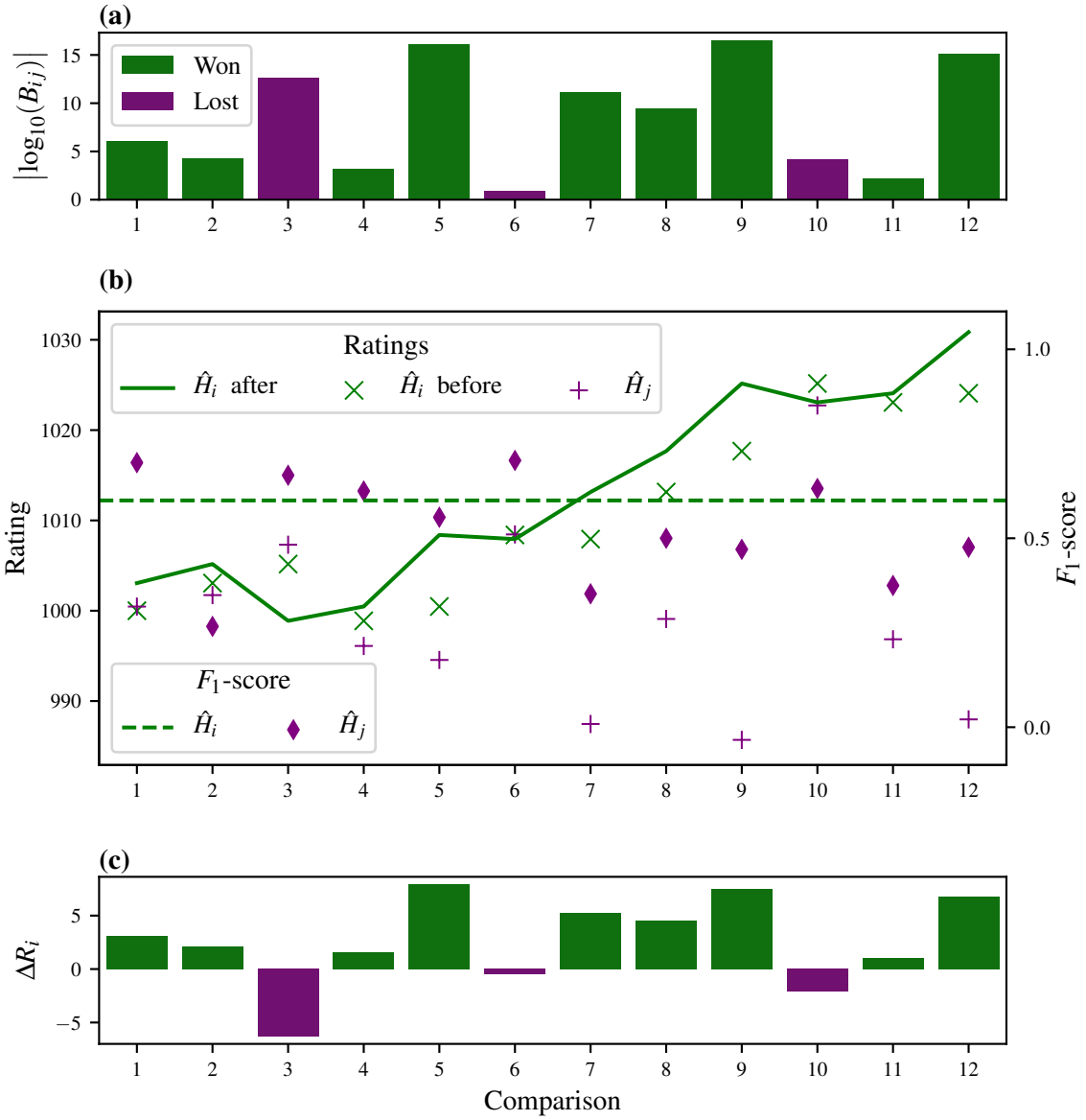


Figure 8.5: Progression of Bayes factor enhanced Elo ratings for a single candidate, $\hat{H}_i$, within a single generation. The x-axis marks successive comparisons for $\hat{H}_i$ against a subset of models from the same generation (see main text). **a**, The BFs between $\hat{H}_i$ and some opponents, $\{\hat{H}_j\}$, from the perspective where $\hat{H}_i$ *wins* given $B_{ij} > 1 \Rightarrow \log_{10} B_{ij} > 0$, and *loses* otherwise. **b**, $\hat{H}_i$'s rating is shown (solid green line) changing according to the BFs comparisons with 12 other models from the same generation. Before each comparison, $\hat{H}_i$'s rating is shown (green cross) as well as the rating of its opponent, $\hat{H}_j$ (purple plus). The F_1-scores are also shown for $\hat{H}_i$ (dashed green line) and $\hat{H}_j$ (purple diamond). **c**, The corresponding change in $\hat{H}_i$'s rating, ΔR_i. Implementation details are listed in Table A.1.

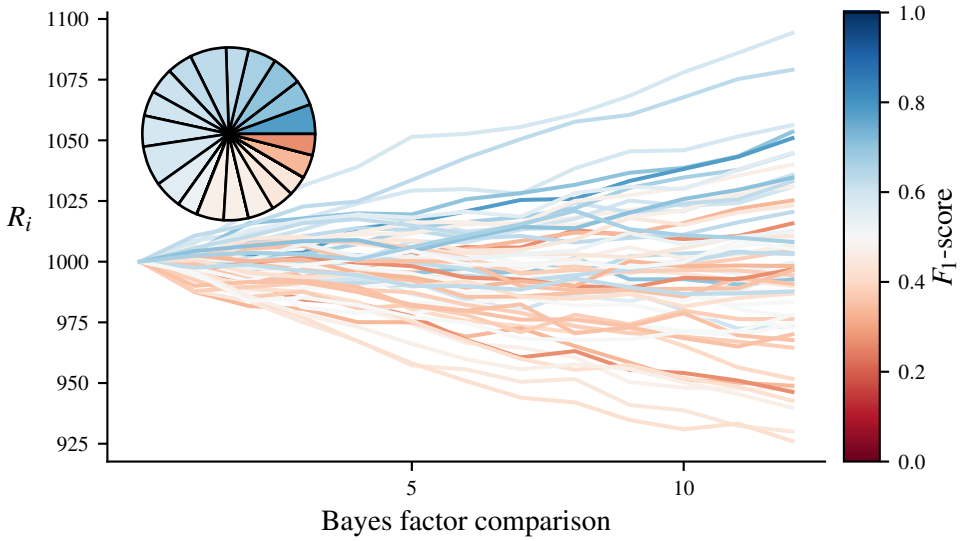


Figure 8.6: Ratings of all $N_m = 60$ models in a single genetic algorithm generation. Each line represents a unique model and is coloured by the F_1-score of that model. The x-axis marks successive comparisons for $\hat{H}_i$ against a subset of models from the same generation (see main text). **Inset**, the selection probabilities resulting from the final ratings of this generation. Only $\tau = 1/3$ of models are assigned selection probability, meaning only those models depicted in the pie chart can become parents to new candidates, while the remaining poorer-performing models are discarded. Implementation details are listed in Table A.1.

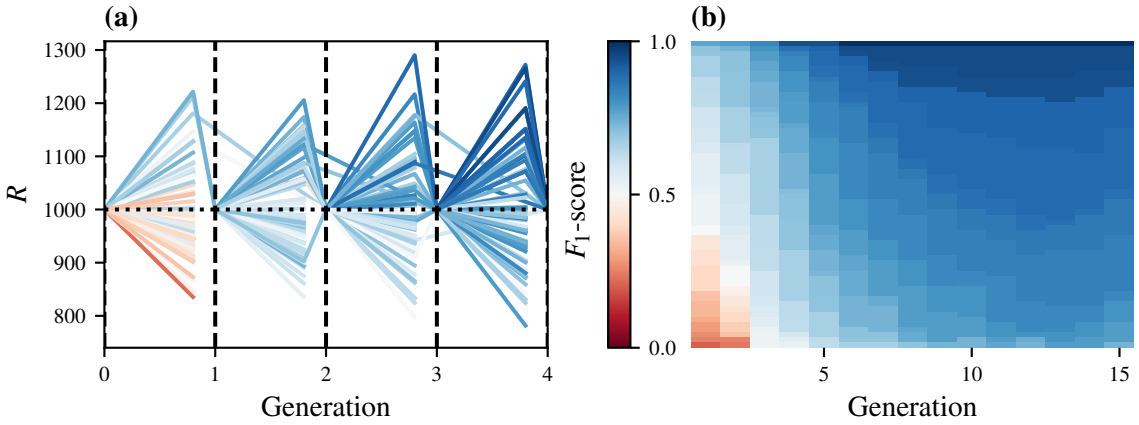


Figure 8.7: A single instance of the QMLA genetic algorithm. **a**, Ratings of all models for the first four generations. Each line in each generation represents a model by its F_1-score through colour. Horizontal dotted lines show the starting rating at that generation. **b**, Gene pool progression for $N_m = 60, N_g = 15$. Each tile at each generation represents a model by its F_1-score through colour. Implementation details are listed in Table A.1.

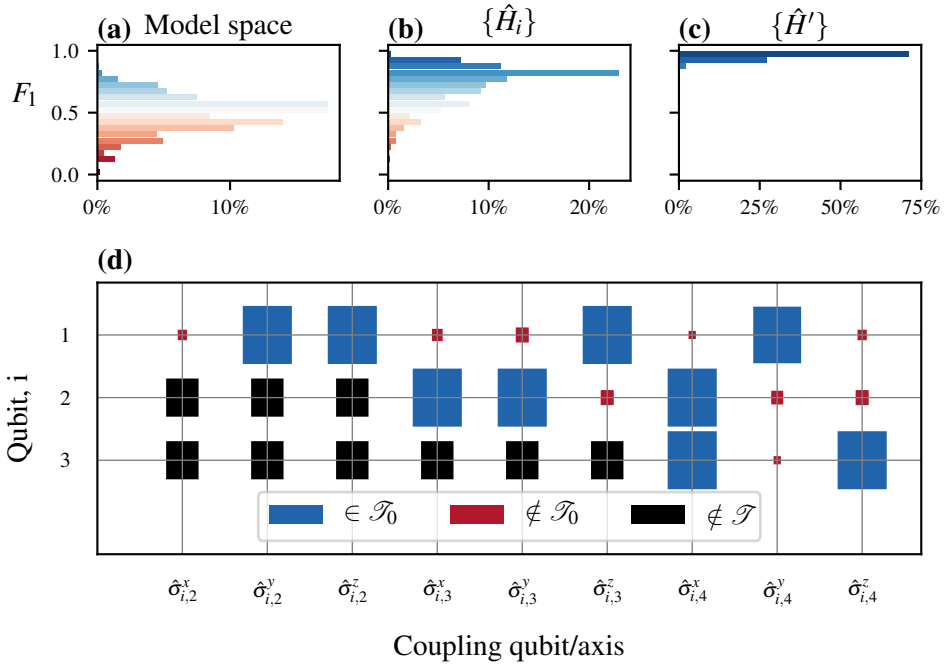


Figure 8.8: A run of the QMLA genetic algorithm (GA), consisting of 100 independent instances. **a**, The model space contains $2^{18} \approx 250,000$ candidate models, normally distributed around $f = 0.5 \pm 0.14$. **b**, The models explored during the model search of all instances combined, $\{\hat{H}_i\}$. QMLA generates $\mathcal{O}(43,000)$ chromosomes in total across the 100 instances, i.e. each instance trains $\mathcal{O}(430)$ distinct models; Models generated by QMLA are described overall by a distribution of $f = 0.76 \pm 0.15$, **c**, Champion models from each instance, showing in general QMLA nominates champion models with $f \geq 0.88$ in all instances, and in particular finds the true model $\hat{H}_0$ (with $f = 1$) in 72%. **d**, Hinton diagram showing the rate at which each term is found in the winning model. The size of the blocks show the frequency with which they are found, while the colour indicates whether that term was (not) in the true model in blue (red). Terms represent couplings between two qubits, e.g $\hat{\sigma}^x_{(1,3)}$ couples the first and third qubits along the x-axis. Available terms involve four qubits with full connectivity, resulting in 18 unique terms (terms with black rectangles are not considered by the GA). Implementation details are listed in Table A.1.

We can likewise consider the quality and ratings of models across generations. In Fig. 8.7**(a)** we see the ratings for models over the first four generations of a QMLA instance: the trend suggested by Fig. 8.6 continues, where models of higher F_1-score tend to achieve higher BFEER. The gene pool as a whole tends towards a homogeneous set of high-quality models: the final generation consists only of models with $f \geq 0.85$, Fig. 8.7**(b)**. Consequently, even in instances where the precise model, $\hat{H}_0$, is not identified, the champion model is highly informative, in that it captures many of the same interactions, therefore most-likely providing meaningful insight on the system's physics.

Finally, to understand the performance of the QMLA algorithm overall, we combine 100 independent instances in a run, Fig. 8.8. We see that, while the overall model space can be characterised by a distribution of models with $\bar{f} = 0.5 \pm 0.14$ (Fig. 8.8**(a)**), QMLA quickly moves to the subspace of high-quality models, i.e. the models explored have median $f = 0.76 \pm 0.15$ (Fig. 8.8**(b)**). This exploration is based on 430 ± 45 chromosomes per instance, i.e. QMLA trains only 0.16% of the 2^{18} permitted models. Ultimately QMLA nominates champion models $\{\hat{H}'\}$ with $f \geq 0.88$ in all instances, and precisely identifies $\hat{H}' = \hat{H}_0$ in 72% of instances, Fig. 8.8**(c)**. Considering the big picture – where the remit of QMLA is to identify the interactions Q undergoes – we show the rate at which each individual term/gene is included in $\hat{H}'$ in Fig. 8.8**(d)**. Crucially, we see that terms which really are within the true Hamiltonian, $\hat{t} \in \mathcal{T}_0$, are found at a higher rate than those without, $\hat{t} \notin \mathcal{T}_0$. This level of analysis can be used to post-validate the outcome of QMLA, i.e. rather than relying on $\hat{H}'$ from a single instance, trusting the terms' individual frequencies as evidence that they are of importance when describing the system of interest.

8.3.2 *Device calibration*

The use-case presented in this chapter is restrictive, so cannot be considered as a solution to characterising any black box quantum system. Firstly, the set of conceivable terms must be prescribed in advance to facilitate a chromosome mapping; this either limits the range of insight QMLA can achieve to interactions envisaged by the user, or requires a vast set of permissible terms, leading to substantially larger model search phases than shown here. Secondly, models were trained using IQLE in order to learn effectively with relatively few resources. IQLE is only available to train models where we can reliably reverse the evolution of the target system (see Section 4.3.1), and as such it is only useful when we have complete control over Q, for example where Q is some quantum simulator.

The adaptive GES presented in this chapter may therefore prove a useful application of QMLA in the domain of device calibration, in particular to characterise some untrusted quantum simulator. That is, by using the simulator to implement some target $\hat{H}_0$, QMLA can identify which operator is *actually* implemented. For instance, implementation of a four–qubit model relies on high-fidelity two-qubit gates between arbitrary qubit pairs. QMLA can effectively reconstruct which operations were and were not faithfully computed, i.e. determine in which

operations the device failed to perform the intended calculations, allowing for the calibration of said device. The extension of QMLA to the characterisation of real quantum devices is one of the most promising applications for future research in the scope of the QMLA framework beyond this thesis.

Part IV

EXPERIMENTAL STUDIES

OVERVIEW AND CONTRIBUTION

This Part details the application of the algorithms described in Part II to the study of experimental and realistic quantum systems. The results here were presented in [1].

In Chapter 9, we describe the physical system considered, the nitrogen-vacancy (NV) centre in diamond. We detail the design of an exploration strategy within the Quantum Model Learning Agent (QMLA) framework targeting the study of such a system. The application was conceived by Dr. Raffaele Santagati; the retrieval of experimental data used throughout this section, as well as the initial model reduction to a set of sensible Hamilontonian terms, were performed by Drs. Sebastian Knauer and Andreas Gentile. The machine learning methodologies presented, such as the greedy search rule, were refined by Drs. Santagati, Gentile and myself. I performed the adaptation of the QMLA software, ran the instances, analysed the data and generated the figures, except where explicitly referenced.

Chapter 10 continues the theme of applying QMLA to data from realistic systems: we extend the analysis to larger systems than those considered in Chapter 9, at the expense of resorting to simulations instead of experimental data. I proposed genetic algorithms for the exploration of large model spaces within QMLA, as examined in Part III, including the study presented in this chapter. Together with Drs. Knauer, Gentile, Santagati and Nathan Wiebe, we devised the target model, including the choice of parameters, to reflect a realistic system interacting with a spin-bath environment. I adapted the QMLA software, ran the instances, performed the analysis and generated the figures shown in this chapter.

9 NITROGEN VACANCY CENTRE

It is of primary interest to apply the Quantum Model Learning Agent (QMLA) algorithm to real-life, experimental systems. In this chapter we devise an exploration strategy (ES) to operate in conjunction with experimental data in order to characterise an electron spin in a nitrogen-vacancy centre (NVC) in diamond. In particular, we model, through Hamiltonian terms, interactions between the spin and the spin bath in which it resides, so that QMLA is finding an effective model for the open system dynamics.

Here we will first introduce a basic picture of NVCs, using basic but nonstandard nomenclature for simplicity; for thorough descriptions of the underlying physics, readers are referred to [187]. We next discuss the target system with respect to its modelling, determining the suitable terms which *might* represent the NVC's interactions, to inform the starting point for QMLA. Finally we describe the implementation of an ES for the examination of the NVC, and the results of the QMLA procedure.

9.1 NITROGEN-VACANCY CENTRE

Nitrogen vacancies are point defects in diamond, occurring intrinsically (naturally) [188] or extrinsically (synthetically) [189, 190]. A substitutional nitrogen-14 (^{14}N) isotope is embedded in a lattice of carbon atoms in diamond, adjacent to a lattice vacancy, such that it is surrounded by three carbons (Cs) (either ^{12}C or ^{13}C) [191]. Of the ^{14}N atom's five valence electrons, three bond with nearby Cs; the remaining two unbonded electrons couple with the lattice vacancy, forming a triplet state, considered as the NVC. We can experimentally drive the outer electron, moving the NVC between energy levels characterised by the triplet. In this section we describe how we can exploit those energy levels in order to define a mechanism by which to prepare and implement gates on the controlled system, a readout procedure and a computational basis.

A *manifold* is a set of states with marginal differences, such as a single differing quantum number. For example, states near the absolute ground state might differ only in their magnetic spin quantum number: together they can be characterised as the *ground state manifold*. We consider two principal manifolds of the system: the ground state and excited manifolds, each consisting of three states, corresponding to the allowed values for magnetic spin m_s, see Fig. 9.1a. For brevity, we denote states with reference to their magnetic spin and manifold, e.g. the state in the ground state manifold with $m_s = 0$ is denoted $|m_s = 0\rangle_g$. In the absence of a magnetic field, the states corresponding to $|m_s = \pm 1\rangle$ are degenerate, but in the presence of a magnetic field, B, they have distinct energy levels, referred to as the Zeeman effect, Fig. 9.1b.

For the purposes of computation, we choose the ground state and one of the excited states as the two levels of a qubit. We designate the states $|m_s = 0\rangle_g$ and $|m_s = -1\rangle_g$ as the computational

basis states $|0\rangle, |1\rangle$ respectively, such that we have defined a qubit and computational basis, Fig. 9.1d. We also require a reliable mechanism through which we can be confident that our qubit is in a definite state, to serve as the starting point of computation: usually qubits are initialised to $|0\rangle$, so here we aim to prepare the NVC in $|m_s = 0\rangle_g$. By shining a laser of 532 nm (green) on the NVC, irrespective of which state within the ground state manifold the spin starts, it is excited in to the excited manifold, from which it decays back to the ground state manifold. The process of this decay can be exploited for the preparation of the NVC in $|m_s = 0\rangle_g$ and therefore enable initialisation for computation. That is, when the NVC is excited to the $|m_s = 0\rangle_e$ level, the dominant decay process is spin-preserving, so after decay it ends in $|m_s = 0\rangle_g$. On the other hand, if the NVC had been excited instead to $|m_s = \pm 1\rangle_e$, the dominant decay process is through a meta-stable/shelving state, which does not preserve spin, so in this case it also ultimately decays to the $|m_s = 0\rangle_g$, Fig. 9.1**(c)**. Therefore, irrespective of the initial state, by shining the green laser on the NVC and exciting it into any of the states in the excited manifold, after decay it is most likely that it has been prepared in $|m_s = 0\rangle_g = |0\rangle$, providing a starting point from which to perform computation.

The difference in energy between our defined computational basis states $|0\rangle$ and $|1\rangle$ is ≈ 2.87GHz, i.e. it is addressable by microwave (MW) radiation. Via antenna, we can deliver a MW pulse upon the NVC, driving the NVC between the two levels providing an implementation of an X-gate. Likewise, having initialised the state to $|0\rangle$, we can perform a $\pi/2$ rotation about the logical z-axis, by running the MW laser for half the time, resulting in the state $|+\rangle$. We can similarly devise MW radiation to achieve quantum gates and operations on our NVC qubit. We depict these cycles in Fig. 9.1c.

We can further exploit the decay mechanism to compose a readout procedure, to infer the population of $\{|0\rangle, |1\rangle\}$ at a given instant, for example following the application of a series of gates (a circuit) to the system. We know that the excitation due to the green laser is spin-preserving, i.e. when the NVC has been excited to $|m_s = 0\rangle_e$, it had originated in $|m_s = 0\rangle_g$. We also know that the decay $|m_s = 0\rangle_e \to |m_s = 0\rangle_g$ is spin preserving, with the emission of a red photon: by simply counting the number of excess[1] photons emitted, we quantify the population of $|0\rangle$ at the time of query. On the contrary, when the $|m_s = -1\rangle_g$ is excited, spin is also preserved, so it goes to $|m_s = -1\rangle_e$, but $|m_s = -1\rangle_e$ decays through the shelving state as outlined earlier, *without* the emission of a red photon (the decay emits out infrared radiation instead). We can hence infer the population of $|m_s = -1\rangle_g$ at the time of query by the fraction of incidents which don't emit a photon [192]. That is, say we first calibrate the system by retaining the green laser for some time: after a few μs, a steady state is achieved where the majority of the time, the triplet is in the computional state $|0\rangle = |m_s = 0\rangle_g$. Then, excitation from the same laser results in the excitation to $|m_s = 0\rangle_e$, which decays back to $|m_s = 0\rangle_g$ and emits a photon in the process; by counting the red photons emitted in a certain time window – equivalently, measuring the photoluminescence (PL) signal – we benchmark the population of $|0\rangle$ when nothing else has happened as p_0. Now, when we apply gates (i.e. MW pulses) to the NVC, we can similarly read out the population of $|0\rangle$ as p_0', and infer that the likelihood that the NVC is found in the

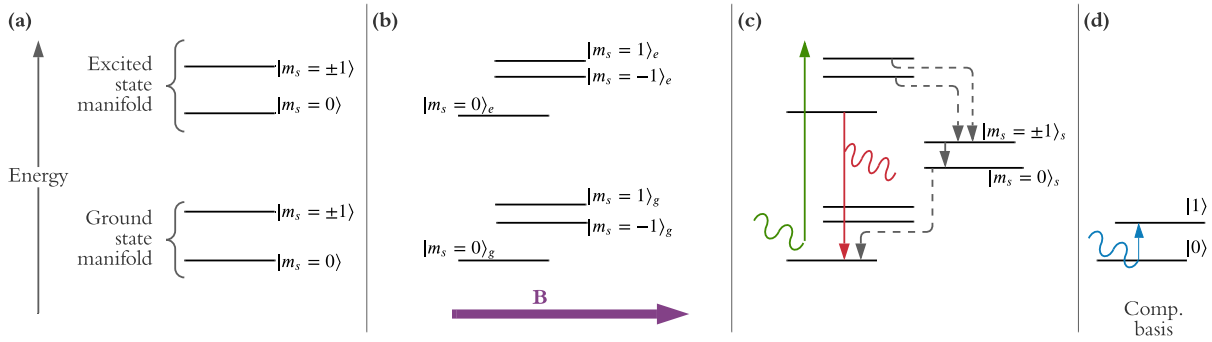


Figure 9.1: Simplified depiction of energy levels of the nitrogen-vacancy centre, corresponding to its triplet state. **a**, With no external magnetic field, the system has excited and ground-state manifolds, each of which consist of two energy levels depending on the magnetic spin, m_s. **b**, In the presence of a magnetic field (purple, B), the magnetic spins have distinct energy levels, i.e. Zeeman splitting giving distinct m_s. States are denoted by their magnetic spin, m_s and subscripted by their manifold (e for excited and g for ground-state). **c**, Application of a 532 nm laser (green arrow) excites the nitrogen-vacancy centre from any of the states in the ground state manifold into the excited manifold. The dominant decay mechanism for the excited states are shown: (i) $|m_s = 0\rangle_e \rightarrow |m_s = 0\rangle_g$ (photoluminescence, red) through the emission of a photon at 637 nm); (ii) $|m_s = \pm 1\rangle_e \rightarrow |m_s = 0\rangle_g$ (dotted grey lines) via the shelving manifold which allows for non-spin-preserving transition, emitting a photon in the infrared (not shown). **d**, Computational basis states $|0\rangle$ and $|1\rangle$ are assigned to the two lowest energy states. The difference in energy between these states is such that a microwave (MW, blue) can drive transition from $|0\rangle \leftrightarrow |1\rangle$. MW pulses can also be used to achieve other states apart from the basis states, allowing for the implementation of quantum logic gates.

initial state $|0\rangle$ is p_0'/p_0. We can use this quantity as the likelihood within quantum likelihood estimation (QLE), allowing us to learn from the NVC, as we will discuss in the next sections.

In summary then, by assigning computational basis states $|0\rangle, |1\rangle$ to energy levels of the ground state manifold, we are able to ensure the preparation of the NVC in $|0\rangle$ by first shining a green laser on the NVC. We can then apply MW radiation to achieve quantum logical gates on the system, and read out the final state of the system, again by shining a green laser and observing the PL (i.e. the emitted photons), and inferring the population level of each basis state. We represent these concepts in a simplified format in Fig. 9.1.

1 A large number of photons are emitted by the NVC when it is excited by a 532 nm laser, which can be profiled by its emission spectrum. At the *zero phonon line* (637 nm), a relatively large number of photons are emitted, compared with nearby wavelengths. This is where decay from the excited to ground state occurs without interacting with proximal phonons, as is the case during the spin-preserving decay $|m_s = 0\rangle_e \rightarrow |m_s = 0\rangle_g$. The excess photons are taken as indication that the electron had been in the state $|m_s = 0\rangle_e$ immediately prior to emission.

9.1.1 *Experimental procedure*

As our primary objective, we aim to model the decoherence and precession of the NVC (described in Section 9.2), and therefore must implement an experimental strategy which highlights the decoherence effects dominating the spin. The *Hahn echo sequence* is a series of operations which decouple a spin from the slowly varying components of its environment, i.e. the nuclear bath [193–197]. For short evolution times, i.e. in the first decay of the NVC, the spin is influenced mostly by *fast* decoherence processes, providing a platform to study the contributions of dominant decoherence effects in isolation.

During the Hahn echo sequence, the NVC spin undergoes a series of evolutions – either according to application of quantum logic gates or the natural evolution of the system interacting with its environment. Intuitively, the stages are as follows:

(a) Prepare spin in $|0\rangle$.

(b) Apply a $\pi/2$ MW pulse. A π-pulse is a MW pulse of sufficient duration to flip the spin completely ($|0\rangle \leftrightarrow |1\rangle$), so a $\pi/2$-pulse generates a superposition, i.e. $\pi/2 : |0\rangle \rightarrow |+\rangle$.

(c) The superposition precesses freely for t. During this time, the $|1\rangle$-component of the superposition picks up a phase proportional to t, such that the spin is now in the state $\frac{1}{\sqrt{2}}\left(|0\rangle + e^{-i\delta t}|1\rangle\right)$.

(d) A MW π-pulse is applied, which inverts the basis states, yielding the state $\frac{1}{\sqrt{2}}\left(|1\rangle + e^{-i\delta t}|0\rangle\right)$.

(e) The spin is again allowed precess freely, here for a duration t'. Again the $|1\rangle$-component gains a phase $e^{-i\delta t'}$, i.e. the spin is in the state $\frac{1}{\sqrt{2}}\left(e^{-i\delta t}|0\rangle + e^{-i\delta t'}|1\rangle\right)$.

(f) A final $\pi/2$ pulse is applied, giving $\frac{1}{\sqrt{2}}\left(e^{-i\delta t}|+\rangle + e^{-i\delta t'}|-\rangle\right)$. Expanding $|+\rangle, |-\rangle$ and factoring $e^{-i\delta t}$, we get the final state

$$|\psi_H(t,t')\rangle = \frac{e^{-i\delta t}}{2}\left\{\left(1+e^{-i\delta(t'-t)}\right)|0\rangle + \left(1-e^{-i\delta(t'-t)}\right)|1\rangle\right\}. \tag{9.1}$$

If the second free evolution is run for $t' = t$, we retrieve

$$|\psi_H(t,t'=t)\rangle = \frac{e^{-i\delta t}}{2}\left\{(1+1)|0\rangle + (1-1)|1\rangle\right\} = |0\rangle, \tag{9.2}$$

i.e. the phase accumulated is effectively removed. This is therefore an excellent scheme when seeking to isolate the spin from its environment, e.g. when the spin is intended to act as a qubit. In practice it is impossible to decouple the environment entirely, since distant nuclei still interact with the spin, causing decoherence on a relatively long time scale[2]. These are what we refer to as

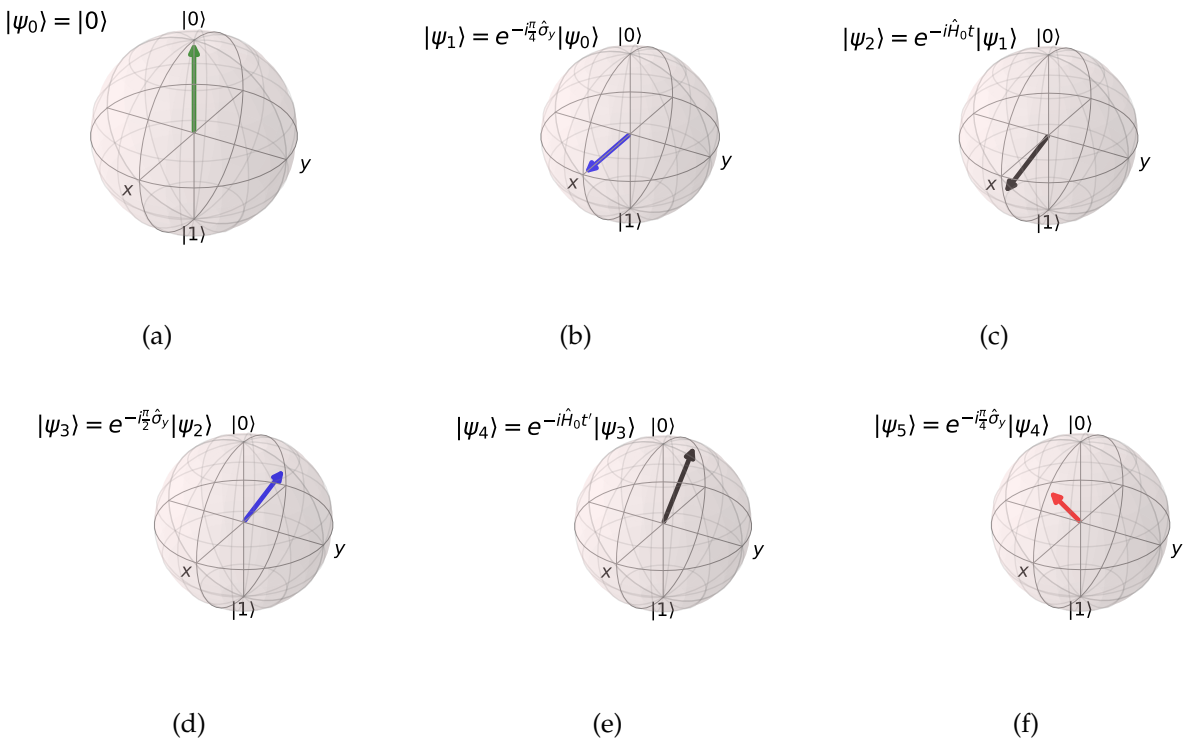


Figure 9.2: States of spin qubit at each stage of Hahn echo sequence shown on the Bloch sphere. **a,** The state of the NVC spin is initialised by a green laser into state $|\psi_0\rangle = |0\rangle$. **b,** We apply a $\pi/2$ rotation about the y-axis (i.e. a MW (microwave) pulse, implemented as $e^{-i\frac{\pi}{4}\hat{\sigma}_y}$), yielding the state $|\psi_1\rangle = |+\rangle$. **c,** The system is allowed to evolve according to its own $\hat{H}_0$ for t, $|\psi_2\rangle = e^{-i\hat{H}_0 t}|+\rangle$. **d,** We apply a second MW pulse, this time for a π-rotation about the y-axis, $|\psi_3\rangle = e^{-i\frac{\pi}{2}\hat{\sigma}_y}e^{-i\hat{H}_0 t}|+\rangle$. **e,** Again the system evolves according to interactions with the environment, this time for t'. **f,** We apply a final $\pi/2$ MW pulse to rotate about the y-axis again, projecting it upon $|0\rangle$. Here $|\psi_5\rangle$ is roughly half way between $|0\rangle$ and $|+\rangle$, i.e. along the z-axis. The spin is read out from $|\psi_5\rangle$ via the NVC's photoluminescence. Here $\hat{H}_0 = 0.25\,\hat{\sigma}_y$ was evolved for $t = 0.5$ (arbitrary units), and the final state overlap with the initial state, i.e. the likelihood of measuring the spin in $|0\rangle$ is $\Pr(0|\hat{H}_0, t) = 0.865$.

slow interactions: this decoupling scheme can therefore be viewed as corresponding to a model where the Hamiltonian consists solely of slow terms, enabling us to study the spin's interactions with its environment beyond the nearest C atom. We will utilise this strategy in Chapter 10.

On the other hand, when $t' \neq t$, the spin is instead found in the state

$$|\psi_H(t, t' \neq t)\rangle = \frac{e^{-i\delta t}}{2}\left\{(1 + e^{-i\delta(t'-t)})\,|0\rangle + (1 - e^{-i\delta(t'-t)})\,|1\rangle\right\}. \tag{9.3}$$

Eq. (9.3) has the precise form (up to the phase factor $e^{-i\delta t}$) of a spin having undergone a *Ramsey sequence*, i.e. a $\pi/2$ pulse, followed by free evolution for t, and a final $\pi/2$ pulse. That is, for $t' \neq t$, the Hahn echo sequence behaves like a Ramsey sequence with a delay $(t' - t)$ [198]. $|\psi_H(t, t' \neq t)\rangle$ therefore does *not* decouple the spin from its dominant dephasing contributions, but rather it reverses the dephasing due to *slowly varying* components of the environment. As such, this provides a platform for examining only the *fast* effects on the spin, i.e. the dominant Markovian decoherence processes, which are expected to be dominated by coupling with the nearest C atom of strength $\mathcal{O}(\text{MHz})$.

In Chapter 10, we consider the system more broadly, and endeavour to characterise its coupling with several nuclei, located farther from the spin than the nearest C, i.e. the slower-varying contributions to the spin's dynamics, which an be examined via the former scheme with $t' = t$. For the remainder of this chapter, however, we will exploit the latter scheme, Eq. (9.3), in order to model the decoherence of the spin via its relationship with a single C atom.

To relate the experimental procedure to the parameter learning technique described in Chapter 4 which fulfils the training stage of QMLA, consider the overall Hahn echo sequence. We depict the stages of the experiment more generally in Fig. 9.2, starting from the initialised computational state, $|\psi_0\rangle = |0\rangle$, through to its final state which is read out through PL, both of which as described in Section 9.1. In particular, the final state, $|\psi\rangle_5$, is effectively read out by projection onto $|0\rangle$; we can interpret the normalised PL after evolution time t as the likelihood that the NVC is found in $|0\rangle$ after evolution of its *true*[3] Hamiltonian, $\hat{H}_0$ for t. That is, we assign this projection as the quantity $\Pr(0|\hat{H}_0, t)$ (the likelihood), and it can be used within likelihood estimation in order to refine a candidate model $\hat{H}_j$, effectively[4] by changing the structure of $\hat{H}_j$ until $\Pr(0|\hat{H}_0, t) \approx \Pr(0|\hat{H}_j, t)\ \forall t$.

By varying the evolution time, t, used within the Hahn echo sequence, we can map the likelihood against time, which we can view as capturing the *dynamics* of the NVC spin, Fig. 9.3. We vary the evolution time up to $t \approx 4\mu s$ in intervals of $\Delta t = 50$ns, so we have 425 data points. Note the data for the studied NVC is taken once and analysed offline, i.e. QMLA does not have

2 The timescales on which these interactions decohere the spin are orders of magnitude higher then available through alternative decoupling schemes, e.g. $T_2 = 242\,\mu$s in [194].

3 Note: we refer to the target $\hat{H}_0$ as the system's *true* Hamiltonian. This is a matter of convention: here $\hat{H}_0$ is not the Hamiltonian of the complete NVC system/environment, but captures only the precession and fast decoherence processes, i.e. $\hat{H}_0$ is simply the name assigned to the Hamiltonian which models the interactions we aim to uncover.

4 Of course this is a gross simplification of quantum Hamiltonian learning (QHL) which is described fully in Chapter 4

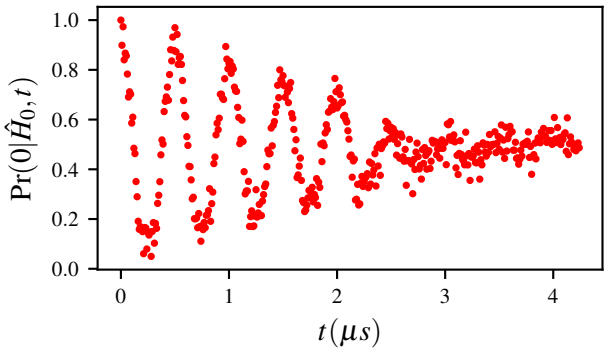


Figure 9.3: Raw data for the nitrogen-vacancy centre's dynamics. The y-axis shows the normalised photoluminescence of the NVC, equivalently the likelihood $\Pr(0|\hat{H}_0, t)$.

complete authority to design experiments to run on the NVC, although it can aim to choose the most informative t available in the predefined set; we will discuss the consequences of this restriction in Section 9.4.

9.2 TARGET SYSTEM

We take the axis of the NVC, i.e. the axis connecting the ^{14}N with the lattice vacancy, as the z-axis. While the NVC is subject to myriad interactions which result in decoherence, we choose to focus on its dominant interactions with proximal environmental nuclei. These interactions are characterised by hyperfine terms [199]. The complete Hamiltonian for such systems, where the set of nuclear sites is $\{\chi\}$, is expected to be given by

$$\hat{H}_{\text{full}} = \Delta_{\text{gs}}\hat{S}_z^2 + \mu_B g\mathbf{B}\cdot\mathbf{S} + \mathbf{S}\cdot\sum_\chi\left(\mathbf{A}_\chi\cdot\hat{I}_\chi\right) + P\hat{I}_z^2 + \mu_n g\mathbf{B}\cdot\sum_\chi\hat{I}_\chi. \tag{9.4}$$

Our overarching intention is to design an approximate model $\hat{H}'$, i.e. a subset of the terms in $\hat{H}_{\text{full}}$ which can explain the observed dynamics and decoherence of the NVC. It is therefore prudent only to retain terms which *may* contribute to the spin's decoherence and precession. First, we will describe each term in Eq. (9.4), as well as approximations which enable us to drastically reduce the space of terms to consider for inclusion in $\hat{H}'$.

ISOLATED-SPIN TERMS

Describe the spin independent of the environmental nuclei.

– $\Delta_{\text{gs}}\hat{S}_z^2$: the *zero-field* splitting, or ground state splitting between the computational basis states. $\Delta_{\text{gs}}\sim\mathcal{O}(\text{GHz})$ is a constant offset which does not contribute to the decoherence, so it is excluded from our study.

– $\mu_B g \mathbf{B} \cdot \mathbf{S}$: the spin's precession about the magnetic field, $\mathbf{B} = \begin{pmatrix} B_x & B_y & B_z \end{pmatrix}$, via the total spin operator[5] $\mathbf{S} = \begin{pmatrix} \hat{S}_x & \hat{S}_y & \hat{S}_z \end{pmatrix}$, where μ_B is the Bohr magneton and g is the electron g-factor (≈ 2, simplified from the g-factor tensor). Misalignment of the magnetic field is a decoherence effect, so one core aim of QMLA in this setting is to identify whether such terms dominate.

HYPERFINE TERMS

– $\hat{S} \cdot \sum_\chi \left(\mathbf{A}_\chi \cdot \hat{I}_\chi \right)$: The NVC total spin operator $\mathbf{S}$ couples the spin with each site, χ. At each site there is a nucleus which has total spin operator $\mathbf{I}_\chi = \begin{pmatrix} \hat{I}_x & \hat{I}_y & \hat{I}_z \end{pmatrix}_\chi$. $\mathbf{A}$ is the hyperfine tensor, containing the hyperfine parameters of interest. The coupling between the NVC and these nuceli is one of the primary decoherence mechanisms, so is essential to any model aiming to capture those dynamics.

BATH-ONLY TERMS

Describe the other nuceli independent of the spin

– $P \hat{I}_z^2$: the quadrupole splitting, which provides another constant shift, and is therefore not of interest when modelling the spin's decoherence, and can be neglected.

– $\mu_n g \mathbf{B} \cdot \sum_\chi \hat{I}_\chi$: nuclear precession terms. μ_n is the nuclear magneton and g is the nuclear g-factor (again from the g-factor tensor). These terms represent the nuclei independent of the spin – these terms lead to decoherence at much higher times than we have access to, since the Hahn echo sequence reverses the contribution on the spin from the bath, Section 9.1.1. For the short-time model targeted here, then, these terms can be excluded.

Given that we are modelling the spin's decoherence, we are interested only in the spin and its interactions with the environment, so we can immediately drop the bath-only terms, by assuming the bath is static apart from its interactions with the NVC. This is a usual assumption in the treatment of open system dynamics, to allow for focus on the dominant interactions in the processes of interest [43]. Additionally, since the zero field splitting contributes a constant shift in energy, we can safely omit it by moving to the rotating frame. We are then left only with the second and third terms of Eq. (9.4), from which to define the space of terms in which QMLA will search:

$$\mu_B g \mathbf{B} \cdot \mathbf{S}; \tag{9.5a}$$

$$\mathbf{S} \cdot \sum_\chi \left(\mathbf{A}_\chi \cdot \hat{\mathbf{I}}_\chi \right). \tag{9.5b}$$

5 We invoke an inexact representation of high dimensional tensors here for ease of interpretation. For instance, the total nuclear spin operator exists in arbitrary dimension (depending on the number of sites modelled), but we present it simply as $\mathbf{I} = \begin{pmatrix} \hat{I}_x & \hat{I}_y & \hat{I}_z \end{pmatrix}$ at each site to convey that we can separate the terms in the construction of models.

9.2.1 *Mapping to model terms*

Next we will focus on mapping the remaining terms to operators to compose the set of terms $\mathcal{T}$ to use in our ES. In our modelling, the NVC spin is represented by the first logical qubit, with a further $|\{\chi\}|$ qubits, each representing a unique nuclear site, as discussed later in this section. As standard, we take the axis[6] of the NVC as parallel to the qubit's z-axis.

The first terms included, Eq. (9.5a), come from the spin's precession about the magnetic field. It is usually assumed that the external, applied magnetic field is well-aligned with the spin qubit's z-axis: if the field is misaligned, it leads to decoherence effects. Determining the alignment is treated as a core role of QMLA, i.e. we will endeavour to establish whether the x-, y-axis components of the magnetic field are important for describing the spin's decoherence. Then, we have

$$\mu_B g \mathbf{B} \cdot \mathbf{S} = \mu_B g (B_x \quad B_y \quad B_z) \cdot (\hat{S}_x \quad \hat{S}_y \quad \hat{S}_z) \rightarrow \alpha_x \hat{S}_x + \alpha_y \hat{S}_y + \alpha_z \hat{S}_z, \tag{9.6}$$

with $\alpha_i = \mu_B g B_i$. The spin's rotation terms to be included in QMLA's deliberations are therefore

$$\mathcal{T}_s = \{\hat{S}_x, \quad \hat{S}_y, \quad \hat{S}_z\}. \tag{9.7}$$

Next, we consider the hyperfine coupling term. In general we sum over the nuclear sites $\{\chi\}$, since the NVC spin will interact with every nucleus within a certain range. We show in [1] that a realistic system requires modelling a finite-size bath of $|\{\chi\}|\sim$15 nuclei to capture the dynamics of interest, which is infeasible for complete characterisation via classical simulation, where we are limited to $\sim$11 qubit calculations[7]. Instead, by focusing only on the *short-time* dynamics of the NVC, we can isolate the effects of dominant interactions, most notably with a single nearby C. Indeed, by assigning a first qubit as representing the NVC spin, we can map the entire environment onto a generic second *environmental qubit*, representing the amalgamation of said interactions, though we can think of the two-qubit system as the NVC coupled with a single ^{14}N [199].

$$\mathbf{S} \cdot \sum_{\chi} (\mathbf{A}_\chi \cdot \mathbf{I}_\chi) \rightarrow \mathbf{S} \cdot \mathbf{A} \cdot \mathbf{I} \tag{9.8}$$

This reduces the dimension of our approximation: the number of qubits required, n_q reduces from $n_q = 1 + |\{\chi\}|$ to $n_q = 2$, since now we only retain qubits for the NVC and the ^{14}N (which also represents the entire bath). The hyperfine tensor $\mathbf{A}$ consists of the hyperfine parameters, i.e. the strength of corresponding interactions.

$$\mathbf{A} = \begin{pmatrix} A_\perp & 0 & 0 \\ 0 & A_\perp & 0 \\ 0 & 0 & A_\parallel \end{pmatrix}, \tag{9.9}$$

6 The quantisation axis, i.e. the axis along the ^{14}N and lattice vacancy.

where $A_\perp$ is the non-axial hyperfine coupling term and $A_\parallel$ is the axial coupling term, since the axis of the NVC is used to define the z-axis for our qubits.

The total spin operators are then those of the NVC operating on the first logical qubit, e.g. $\hat{S}_x^{(1)}$, and those of the environmental qubit on the second, e.g. $\hat{I}_x^{(2)}$. They can be summarised as

$$\begin{aligned} \mathbf{S} &= (\hat{S}_x^{(1)} \quad \hat{S}_y^{(1)} \quad \hat{S}_z^{(1)}) \\ \mathbf{I} &= (\hat{I}_x^{(2)} \quad \hat{I}_y^{(2)} \quad \hat{I}_z^{(2)}) \end{aligned} \tag{9.10}$$

So we can write,

$$\begin{aligned} \mathbf{S} \cdot \mathbf{A} \cdot \mathbf{I} =& A_\perp \hat{S}_x \hat{I}_x + A_\perp \hat{S}_y \hat{I}_y + A_\parallel \hat{S}_y \hat{I}_y \\ &+ A_{xy} \left(\hat{S}_x \hat{I}_y + \hat{S}_y \hat{I}_x \right) \\ &+ A_{xz} \left(\hat{S}_x \hat{I}_z + \hat{S}_z \hat{I}_x \right) \\ &+ A_{yz} \left(\hat{S}_y \hat{I}_z + \hat{S}_z \hat{I}_y \right) \end{aligned} \tag{9.11}$$

Similarly to α_i in Eq. (9.6), we replace the expected (and theoretically computable) scalar parameters, e.g. $A_\perp$, with generic parameters α, to be learned. Off-diagonal terms, referred to hereafter as *transverse* terms ($\hat{S}_i \hat{I}_j$ where $i \neq j$), are usually neglected [193]. Here we will employ QMLA to determine whether the transverse contributions are worthy of inclusion in the decoherence model, although we consider only $\{\hat{S}_x \hat{I}_y, \hat{S}_x \hat{I}_z, \hat{S}_y \hat{I}_z\}$ for brevity. The hyperfine terms to be entertained by QMLA are then

$$\mathcal{T}_{HF} = \begin{Bmatrix} \hat{S}_x \hat{I}_x, & \hat{S}_y \hat{I}_y, & \hat{S}_z \hat{I}_z, \\ \hat{S}_x \hat{I}_y, & \hat{S}_x \hat{I}_z, & \hat{S}_y \hat{I}_z \end{Bmatrix}. \tag{9.12}$$

Finally, combining Eq. (9.7) and Eq. (9.12), we have the full set of terms to incorporate into the ES for the QMLA model search:

$$\mathcal{T}_{\text{NV}} = \begin{Bmatrix} \hat{S}_x, & \hat{S}_y, & \hat{S}_z, \\ \hat{S}_x \hat{I}_x, & \hat{S}_y \hat{I}_y, & \hat{S}_z \hat{I}_z, \\ \hat{S}_x \hat{I}_y, & \hat{S}_x \hat{I}_z, & \hat{S}_y \hat{I}_z \end{Bmatrix}. \tag{9.13}$$

We introduce a shorthand notation to ease model representation for the remainder of this chapter. Recall that we have defined a two-qubit Hilbert space for model construction. Terms which affect only the spin act only on the first qubit, $\hat{S}_i = \hat{S}_i^{(1)} = \hat{\sigma}_i \otimes \hat{\mathbb{1}}$, where $\hat{\sigma}_i$ is the Pauli operator giving rotation about the i-axis, and $\hat{\mathbb{1}}$ is the one-qubit identity matrix. Retaining the hyperfine notation, for the expectedly-dominant diagonal terms, we denote $\hat{A}_i = \hat{S}_i^{(1)} \hat{I}_i^{(2)} =$

7 This limitation arises from the requirement to compute the total evolution of the global state, involving calculation of $e^{-i\hat{H}t}$, i.e. the characterisation of an n_q-qubit model depends on classical exponentiation of the $2^{n_q} \times 2^{n_q}$ Hamiltonian for each particle and experiment in classical likelihood estimation (CLE), which is prohibitively expense.

$\hat{\sigma}_i \otimes \hat{\sigma}_i$. We refer to the transverse terms as $\hat{T}_{kl} = \hat{S}_k^{(1)} \hat{I}_l^2 = \hat{\sigma}_k \otimes \hat{\sigma}_l$. We can hence rewrite Eq. (9.13) as

$$\mathcal{T}_{\text{NV}} = \begin{Bmatrix} \hat{S}_x, & \hat{S}_y, & \hat{S}_z, \\ \hat{A}_x, & \hat{A}_y, & \hat{A}_z, \\ \hat{T}_{xy}, & \hat{T}_{xz}, & \hat{T}_{yz} \end{Bmatrix}. \tag{9.14}$$

We also use a succinct representation for brevity, e.g. $\hat{S}_{xy}\hat{A}_z = \hat{S}_x + \hat{S}_y + \hat{A}_z$, where parameters $\alpha_x, \alpha_y, \alpha_z$ are implicitly assumed.

9.2.2 *Prior knowledge*

QMLA will construct models using the pool of terms defined in Eq. (9.14). Recall from Chapter 5 that each model considered must be trained independently, where the purpose of model training is to optimise the parameter vector $\vec{\alpha}$ which characterises the model. For example, the model $\hat{H}_i = \hat{S}_{x,y}\hat{A}_z = \alpha_1\hat{S}_x + \alpha_2\hat{S}_y + \alpha_3\hat{A}_z$, is trained to retrieve the optimal $\vec{\alpha}_i' = (\alpha_1' \quad \alpha_2' \quad \alpha_3')$. Models are trained through QHL, described in Chapter 4, which iteratively updates a probability distribution for the associated parameters, $\Pr(\vec{\alpha})$. As such, a *prior* distribution must be drawn, from which QHL begins its training. While QHL can redraw the probability distribution iteratively, and even find parameters entirely outside of the initial range, it is necessary at least to identify the order of magnitude where the true parameter should be found. The algorithm therefore demands that the user specifies the *range* of each parameter in which to search, which can be based on domain knowledge and theoretical predictions. For example, recall from Section 9.2 that the zero field splitting, Δ_{gs} in Eq. (9.4) (and excluded in our modelling), is expected to be $\mathcal{O}$ (GHz): in order to provide a reasonable chance at learning the true parameter, here we would propose a prior distribution of 5 ± 2GHz. We must similarly identify the broad range in which we reasonably expect to find parameters associated with each term in Eq. (9.14).

The spin-only terms, $\hat{S}_i$, are consequences of the magnetic field, expected in the range $\mathcal{O}(2 - 3\text{MHz})$. Likewise, the hyperfine terms, $\hat{A}_i$ are expected in the range of $\mathcal{O}$ (MHz) [200], while in the *secular approximation* only the z-component is expected to contribute substantially [201]. The non-axial hyperfine terms, i.e. the transverse terms $\hat{T}_{kl}$ are not usually included in effective models, but can be found of order $\mathcal{O}(10\,\text{kHz})$ [202]. We utilise this prior understanding of the system to inform the parameter range used for training candidate models: for each of the terms in Eq. (9.14), we will adopt a normal prior distribution of 4 ± 1.5MHz. This range is sufficiently specific to ensure the training subroutine operates in a physically meaningful – and likely appropriate – space, while also broad enough to allow for significant differences between expectation and reality. Moreover this distribution supports hypotheses where each parameter is zero: if these prove favourable, negligible contributions can be identified and excluded from the model.

9.3 EXPLORATION STRATEGY

We may now turn to the specific implementation details by which QMLA is applied to the study of this NVC system. Recalling the terminology of QMLA from Chapter 5, we design an exploration strategy (ES) specifically for the system under study. The ES will account for the details listed in this chapter so far, in summary:

- we aim to assign a model, $\hat{H}'$, to the NVC to describe its decoherence processes
  - we especially focus on its hyperfine interactions;
- we use a 2-qubit approximation
  - the first qubit represents the spin itself;
  - the second qubit represents the environment in which the NVC resides;
- we query the NVC by performing Hahn echo experiments (Fig. 9.2);
- the outcome of those experiments are thought of as the system's likelihoods (Fig. 9.3);
- candidate models are composed of the terms defined in Eq. (9.14)
  - likelihoods are used for the training of individual candidate models through QHL;
  - we assign approximate ranges to the scalar parameters corresponding to each term based on theoretical arguments;
    - ∗ those parameters are to be learned precisely by QHL.

As outlined in Section 5.4, the central role of any ES is to specify the model generation procedure, which QMLA relies upon for deciding the next set of candidate models to test. In this case, we exploit some intuition and prior knowledge of how such systems work, to design a bespoke model generation subroutine: we can think of this as a midway point between the completely specified ESs used for identifying the underlying lattices from a prescribed set in Chapter 7, and the entirely general genetic algorithm which does not restrict model generation, of Chapter 8. We use the standard structure of exploration trees (ETs) introduced in Section 5.4, where models are placed on consecutive branches, μ, and branches are consolidated by pairwise comparisons between all models on μ, where comparisons are computed through Bayes factors (BFs). The outcome of consolidation on μ is the determination of a single *branch champion*, $\hat{H}_{C(\mu)}$.

We use a *greedy search rule*: terms are added one-by-one to gradually increase the complexity of candidate models until terms are exhausted [103]. We break the ET into three distinct tiers, each corresponding to an intuitive degree of complexity: the first tier involves the spin-only terms, $\mathcal{T}_1 = \{\hat{S}_i\}$; the second considers the hyperfine terms, $\mathcal{T}_2 = \{\hat{A}_i\}$; the final tier the transverse terms, $\mathcal{T}_3 = \{\hat{T}_i\}$. Within each tier, terms are added greedily to the previous branch's champion, $\hat{H}_{C(\mu)}$. So, the first branch is given by $\mu = \{\hat{S}_x, \hat{S}_y, \hat{S}_z\}$; say $\hat{H}_{C(\mu)} = \hat{S}_y$, then $\mu + 1$ determines that $\hat{S}_x, \hat{S}_z$ are not yet considered, so it constructs the models $\{\hat{S}_{x,y}, \hat{S}_{y,z}\}$, e.g. Fig. 9.4. After exhausting all tiers, we consolidate the set of branch champions, $\mathbb{H}_C = \{\hat{H}_{C(\mu)}\}$, to determine the best model considered globally, $\hat{H}'$.

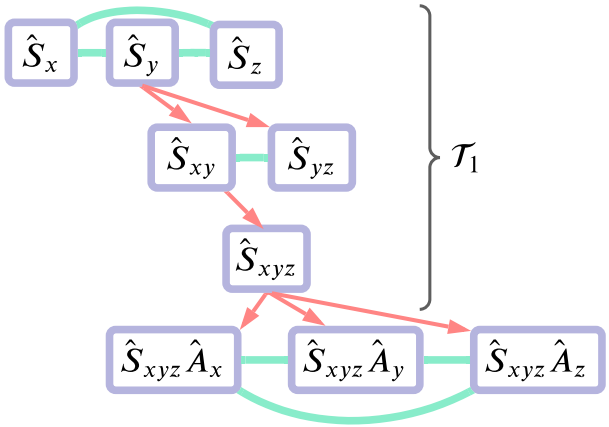


Figure 9.4: Greedy model search. Models (purple) are placed on branches, trained and consolidated (green) as in Fig. 5.1, with the branch champion spawning (red) candidates to place on the subsequent branch. Branches are grouped in tiers, corresponding to levels of approximation: the first tier of the model generation strategy is shown, where $\mathcal{T}_1 = \{\hat{S}_x, \hat{S}_y, \hat{S}_z\}$ is explored. The final champion from the first tier seeds the second tier.

Clearly, this growth rule is partially deterministic, insofar as some models are guaranteed to be considered, while others are not reachable. Indeed, the space of available models are heavily constrained, in particular models in later tiers will always involve all of the tier 1 terms, e.g. $\hat{S}_x\hat{A}_y$ can not occur organically. In general, restrictions of this kind undermine the ES and are considered a weakness. To account for this, we add a final test of reducibility on the champion model, triggered if any of the parameters of $\vec{\alpha}'$ are potentially negligible, i.e. the posterior distribution of any parameter assigns credibility to the hypothesis that the parameter is 0. This champion reducibility test simply removes the negligible-parameter terms from $\hat{H}'$, yielding a reduced global champion, $\hat{H}'_r$. We then compute the BF between $\hat{H}', \hat{H}'_r$: if the BF indicates strong evidence in favour of the reduced version, we replace the champion model, $\hat{H}' \leftarrow \hat{H}'_r$. In effect, we thus verify the statistical significance of each term included in $\hat{H}'$.

The total model in Eq. (9.4) supports $N_t = 1 + 3 + 6|\chi| + 3 + 3|\chi| = 7 + 9|\chi|$ terms, which we reduced to a space of $N_t = 3 + 6|\chi| = 9$ through several approximations. Even so, the remaining 2^9 permitted models were reduced further by building the logic of this ES from our intuition around existing knowledge of typical NVC systems. As such, the described ES will only ever consider < 20 models per instance. The described ES seems overly prescriptive, but should be viewed as a first attempt at a generalisable approach: essentially we can view the tiers of the greedy search as characterising the system at various approximations, e.g. the first tier examines one-qubit terms, while subsequent tiers inspect 2-qubit terms. We can envision future work

where the greedy search is gradually extended to less rigid approximations, enabling study of more complex quantum systems. This leads to some important remarks:

1. Realistic, near-term applications of QMLA can not be thought of as a solution to black-box characterisation problems: it must be used in conjunction with domain expertise for the system under study.
2. While this test-case yields promising results, the outcome of QMLA here may not be especially insightful, since the available model terms were so deliberately constrained – we demonstrate a use-case in Chapter 10 where a broader scope is enabled in simulation.

A common charge against QMLA supposes to first write down the most complex model, train it fully, and then infer which terms are negligible, in a similar process to the champion reduction test outlined here. While this may be feasible in the case described here, with $N_t = 9$ and a closed term set, it is unscalable: adding just a second nuclear site increases the model search to a space of $N_t = 15$. Models of higher cardinality ($|\vec{\alpha}|$) demand higher N_E, N_P to train well, so immediately training the most involved model would require infeasible resources[8], and risks significantly overfitting the data. It seems more appropriate to work "from the ground up", testing terms and only keeping those justified, rather than training all terms and attempting to decouple their effects post-hoc.

9.3.1 *Test in simulation*

Before considering the real experimental data (Fig. 9.3), we first test the ES in simulation under ideal conditions. That is, we assume the ability to prepare arbitrary probe states, and use a random probe set (see Section 4.7), and use the full expectation value as the likelihood, $|\langle\psi|e^{-i\hat{H}_j t}|\psi\rangle|^2$. Of course, this is infeasible since we presume access to the full state at the time of measurement, but this can be seen as a best-case scenario for this application, because the realistic case loses information by tracing out the environmental qubit at measurement. We vary the target $\hat{H}_0$, among a series of ten models, which are all valid models achievable by the ES.

9.4 EXPERIMENT DESIGN CONSTRAINTS

Moving to analyse the experimental setup, there are a number of constraints which we must account for in training models. Firstly, the $\pi/2$-pulse applied to the prepared qubit ($|\psi_0\rangle \rightarrow |\psi_1\rangle$ in Fig. 9.2) means that the state before evolution is always $|+\rangle$ in the computational basis; this is a severe limitation on model training, as we saw in Section 4.6. Moreover, this places a bias on the interactions QMLA is likely to identify: we show in Fig. 9.2 how QHL performs in training

8 Note: in the case studies presented in this thesis, it was found that the same resources were sufficient for the simplest and most complex models, due to the relatively small number of terms therein. We expect for larger models, e.g. $|\vec{\alpha}| > 10$, that the resources allocated ought to be proportional to the cardinality, which is an in-built option in the QMLA software.

the same model using (i) the probe set available experimentally; (ii) a more general (random) probe set. This bias adds a caveat to the outcome of this study: the suppression of terms means we are more likely to find some genuine interactions than others, so the champion model is capturing the decoherence with respect only to one basis.

The experiment was run with increasing t for the duration of the first decay of the NVC, i.e. until it had dephased, so the data available for examination terminate at $t_{\max} \sim 4\mu s$, see Fig. 9.3. As discussed in Section 4.6, usually it is helpful to allow an experiment design heuristic (EDH) to choose the experimental controls, including the evolution time, t, against which the model is trained at each experiment; the default particle guess heuristic (PGH) attempts to select t at the upper boundary of times where the model is expected to be predictive, to maximise the information gained by the experiment (see Section 4.6.1). Here, however, we can not allow the EDH to select arbitrary t, since we do not have data beyond $t_{\max}$.

We require a custom EDH to account for the constraints outlined, with the following considerations:

1. We may only assume access to the probe $|+\rangle$ on the spin qubit
   (a) we further assume the environmental spin is polarised by the same microwave pulse, such that the global probe available is $|\psi\rangle = |+\rangle\,|+'\rangle$, with $|+'\rangle = \frac{|0\rangle + e^{i\phi}|1\rangle}{\sqrt{2}}$ and ϕ is random [203].
2. We can not allow the choice of any t:
   (a) Any $t > t_{\max}$, arising from a thin parameter distribution, must be mapped to some $0 < t \leq t_{\max}$.
   (b) All nominated t must be mapped to the nearest available t in the dataset so that the likelihoods are as close as possible to simulating the true system.
3. Much of the physics of interest occurs at relatively high times, i.e. because the rotation (MHz) terms dominate, the decay of the peaks can be seen as evidence of the bath, notably through hyperfine terms in the model.
   (a) We therefore wish to enforce that all models are trained on those data ($t \geq 2\mu s$), even if their parameter distribution is insufficiently narrow to yield those times naturally.

Accounting for these, we construct an EDH which mixes the robust, adaptive nature of PGH, useful for refining an initially broad $\Pr(\vec{\alpha})$, with a primitive, linear time-selection, useful to ensure the trained parameters at least attempt to account for the physics we are actually interested in. That is, with each model trained for N_E experiments, we train according to the standard PGH for the first $N_e/2$, but force the training to mediate over the available data for the latter $N_e/2$.

9.5 RESULTS

We apply the ES described in Section 9.3 to the raw data of Fig. 9.3: the results are summarised in Fig. 9.5. We first focus on the overall outcomes: the most blunt figure of merit of interest is simply whether QMLA overfits or underfits the true parameterisation. In preliminary analysis we run 500 instances with varying $\hat{H}_0$, varying the cardinality of $\hat{H}_0$, so we can broadly gauge the tendency towards over- and under-fitting: we see that in $\sim$50% of instances the correct cardinality is found, rising to $\sim$86% by allowing ± 1 term, Fig. 9.5**(b)**. In general, the champion models from each instance are highly predictive: the median *coefficient of determination* between the systems' and corresponding champion models' data is $R^2 = 0.84$.

Then, considering the performance of the algorithm on whole, we perform runs of 100 instances on the experimental data as well as simulated data, where the simulation assumes[9] $\hat{H}_0 = \hat{S}_{xyz}\hat{A}_z$. The set of models selected most frequently are shown in Fig. 9.5**(c)**, and each model is trained with $N_e = 1000$, $N_p = 3000$, with the volumes of those models (in the experimental case) shown in Fig. 9.5**(d)**. In particular, the most prominent models, $\{\hat{S}_{x,y,z}\hat{A}_z, \hat{S}_{x,y,z}\hat{A}_{y,z}, \hat{S}_{x,y,z}\hat{A}_{x,z}, \hat{S}_{y,z}\hat{A}_z\}$ are found collectively in 74% (87%) of instances on the experimental (simulated) data; the win rate and R^2 of all models (which won at least one instance) are reported in Table 9.1. It is noteworthy that even in the simulated case, the same models mislead QMLA: this suggests that the resultant physics from these models is substantially similar to that of the true model[10]. These models are defensible with respect to the descriptions of Section 9.2, since in each case they detect the interaction between the spin qubit and the environmental qubit, i.e. the hyperfine terms $\hat{A}_i$, especially $\hat{A}_z$ which occurs in 97% (99%) of champion models, reported in Table 9.1. We discuss some physical insights from these results in Section 9.5.1.

The most frequently identified model, $\hat{H}' = \hat{S}_{x,y,z}\hat{A}_z$, is found in 45% (61%) of instances on experimental (simulated) data: we show its attempt to reproduce the dynamics of Fig. 9.3 in Fig. 9.5**(e)**, showing excellent agreement with the raw data, with $R^2 = 0.82$. This serves as an essential sanity check: we can intuitively see that QMLA has distilled a model which captures at least *some* of the most important physical interactions the target NVC system is subject to; otherwise we would not see such clear overlap between the predicted and true dynamics.

Finally we display the model search as a directed acyclic graph (DAG) in Fig. 9.5**(f)**, where models are represented on nodes on the graph's layers (equivalent to ET branches), and their parents are resident on the branch immediately above their own. Comparisons between models, $\left(\hat{H}_i, \hat{H}_j\right)$, are shown as edges between nodes on the graph, coloured by the strength of evidence of the outcome, i.e. the Bayes factor, B_{ij}. Each layer, μ, nominates their branch champion, $\hat{H}_{C(\mu)}$; the set of branch champions are consolidated to determine the global champion, $\hat{H}'_C$.

9 Here we work backwards by setting the target model as that which QMLA deemed most appropriate for the available data. We posit that this choice is arbitrary and doesn't fundamentally change the discussion of this chapter, merely aiding in analysing the performance of the algorithm with respect to a concrete example.

10 Alternatively, that the same systematic error misdirects the search in both cases.

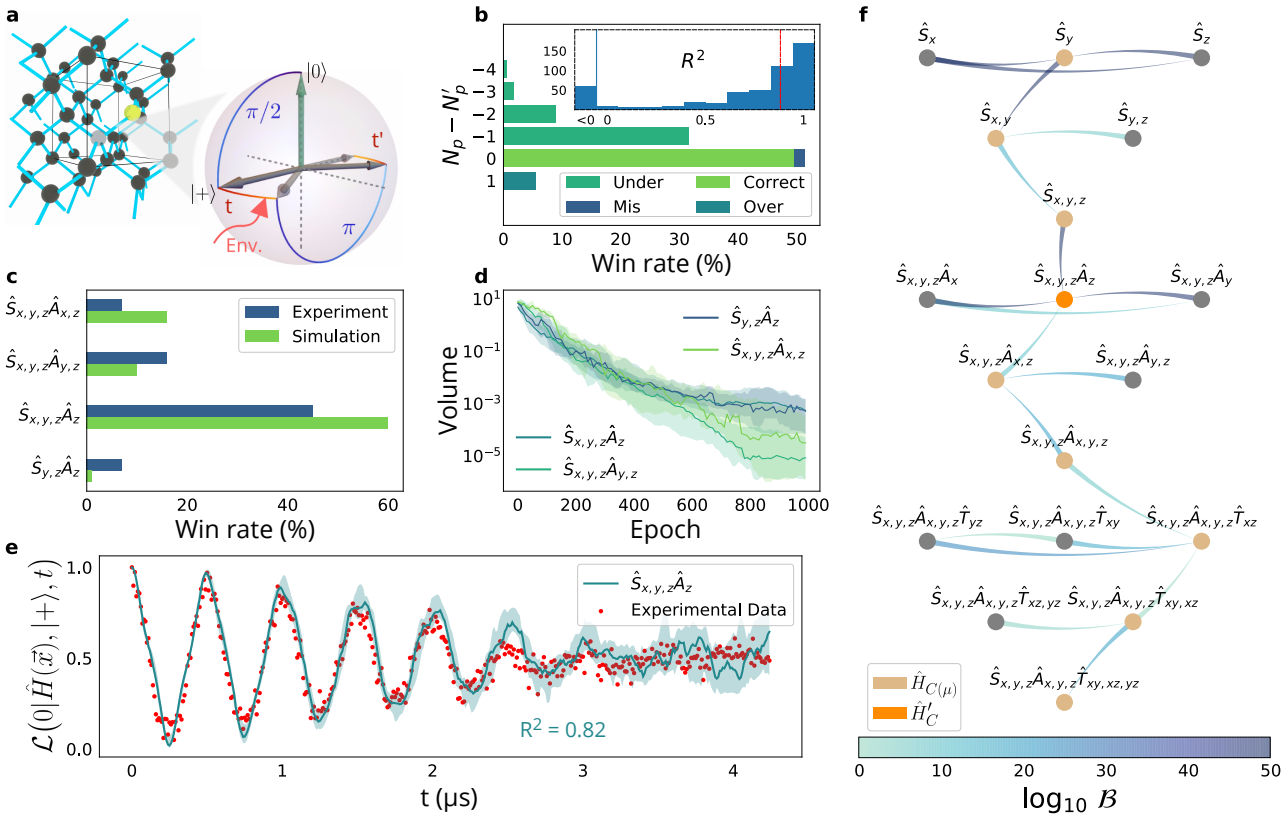


Figure 9.5: QMLA results on simulated and experimental data, describing a NVC (nitrogen-vacancy centre) system. Figure reproduced from [1]. **a**, The carbon lattice providing the outer environment for the NVC, along with the time evolution of the electron spin state (represented on a Bloch sphere) during the pulses for the Hahn echo sequences. These steps are expanded in Fig. 9.2, although here the final $\pi/2$ pulse is omitted. **b**, Simulation of 500 independent QMLA instances, where $\hat{H}_0$ is chosen randomly. The win rate is reported against the difference $(N_p - N_p')$ between the number of parameters in $\hat{H}'$ and $\hat{H}_0$, respectively. The *under-parameterised (over-parameterised)* class refers to models with less (more) parameters than $\hat{H}_0$. *Correct* indicates that exactly $\hat{H}_0$ was found. The *mis-parameterised* class groups models with the same parameterisation cardinality as $\hat{H}_0$, but different Hamiltonian terms. **Inset**, Histogram of occurrences of R^2 values for each retrieved $\hat{H}'$ against a sampling of datapoints from $\hat{H}_0$. The blue vertical line groups together all those instance champions with $R^2 < 0$, and the median $R^2 = 0.84$ is shown as a red dotted line. **c**, Win rates of the top four models for 100 QMLA instances, against both simulated and experimental data. On experimental data, $\hat{H}_0$ is unknown, while simulations use $\hat{H}_0 = \hat{S}_{x,y,z}\hat{A}_z$. **d**, Total volume spanned by the parameters' probability distribution across progressive epochs, for the models in **(c)**. The shaded area show the 67% confidence region of volumes, taken from instances where those models were deemed $\hat{H}'$. **e**, Simulated likelihoods reproduced by the model with the highest win rate ($\hat{S}_{x,y,z}\hat{A}_z$, turquoise), compared with corresponding NV-centre system experimental data (red dots, extracted from the observed photoluminescence of the first Hahn echo decay). Error bars are smaller than the dots. The shaded area indicates the 67% confidence region of likelihoods predicted from the instances where $\hat{H}' = \hat{S}_{x,y,z}\hat{A}_z$ **f**, A single QMLA instance depicted as a directed acyclic graph. The thin end of each edge points to the favoured model; the colour of the edges depict the strength of evidence, $\log_{10}\mathcal{B}$, where $\mathcal{B}$ is the BF between those two models. Champions of each layer, $\hat{H}_{C(\mu)}$, are in light brown, whereas the global champion $\hat{H}'$ is in orange and all other candidate models are grey.

Model	Experiment		Simulation	
	Wins	R^2	Wins	R^2
$\hat{S}_{y,z}\hat{A}_z$	9	0.8	1	0.26
$\hat{S}_y\hat{A}_{x,z}$	2	0.63		
$\hat{S}_{x,y,z}\hat{A}_z$	45	0.86	61	0.97
$\hat{S}_{x,y,z}\hat{A}_y$			1	-0.54
$\hat{S}_{x,y,z}\hat{A}_{x,y}$	3	0.81		
$\hat{S}_{x,y,z}\hat{A}_{y,z}$	14	0.83	10	0.96
$\hat{S}_{x,y,z}\hat{A}_{x,z}$	6	0.64	15	0.99
$\hat{S}_{x,y,z}\hat{A}_{x,y,z}$	2	0.72	5	0.97
$\hat{S}_{x,y,z}\hat{A}_{x,z}\hat{T}_{xz}$			1	0.68
$\hat{S}_{x,y,z}\hat{A}_{x,y,z}\hat{T}_{xz}$			5	0.77
$\hat{S}_y\hat{A}_{x,y,z}\hat{T}_{xy,xz,yz}$	2	0.31		
$\hat{S}_{x,y,z}\hat{A}_{x,y,z}\hat{T}_{xy,xz}$	4	0.67	1	0.32

Table 9.1: QMLA win rates and R^2 for models based on experimental data and simulations. We state the number of QMLA instances won by each model and the average R^2 for those instances as an indication of the predictive power of winning models.

9.5.1 *Analysis*

Here we offer some further perspectives, considering the runs summarised in Fig. 9.5. Fig. 9.6 first details all models considered in the 200 instances comprising the experimental and simulated QMLA runs, as well as the win rate of each model. This ES is designed to study a small subspace of the overall available space: only 40 unique models are constructed. We highlight a number of *credible* models which we deem especially valid approximations of the target system, i.e. which contain the most viable approximations.

Fig. 9.7 shows the reproduction of dynamics of the top[11] four models from both simulated and experimental runs. We see that each model faithfully captures the essential dynamics arising from the respective target systems; this alone is insufficient to conclude that the true model has been identified, but serves as a valuable *sanity-check*, convincing us that the output of QMLA is at least a sensible approximation of $\hat{H}_0$, if not the absolute true model.

11 *Top* models in the context of QMLA are those models within the run with the highest win rates, i.e. which won more instances than other candidates from the model space.

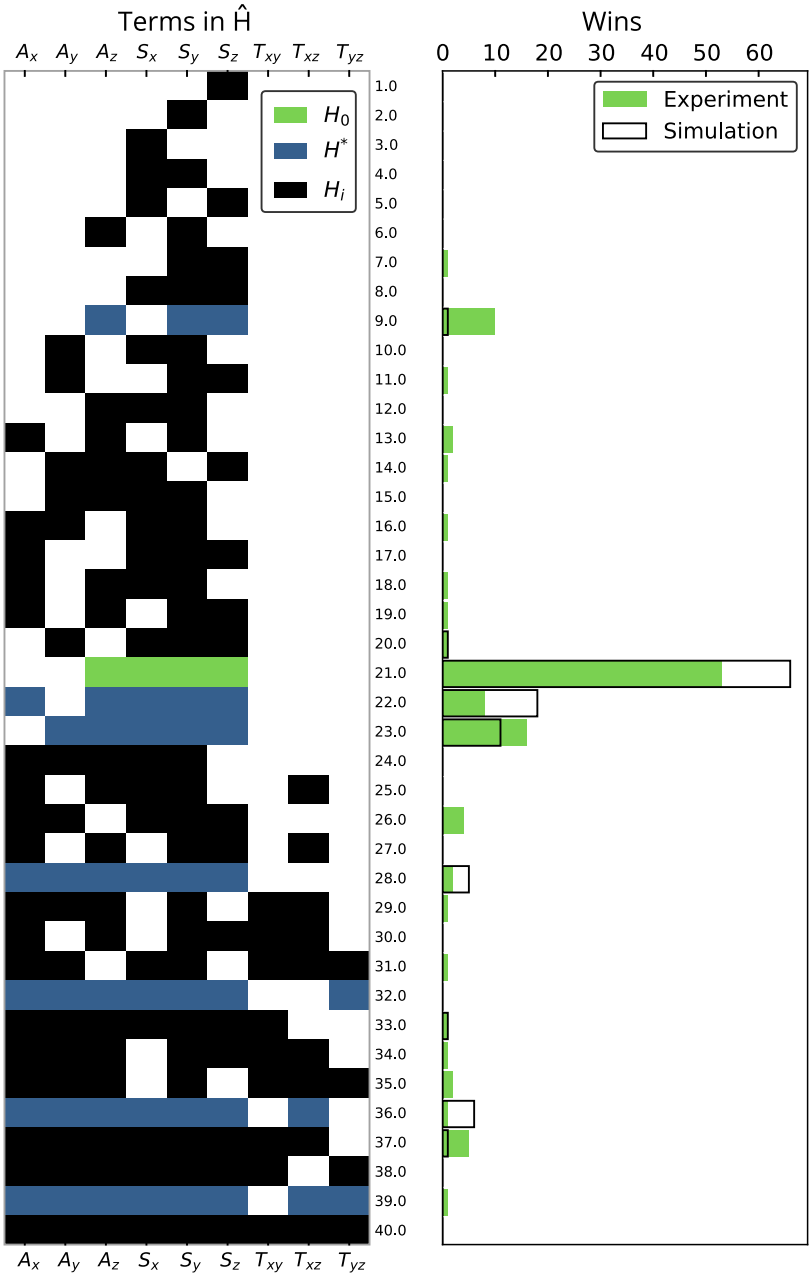
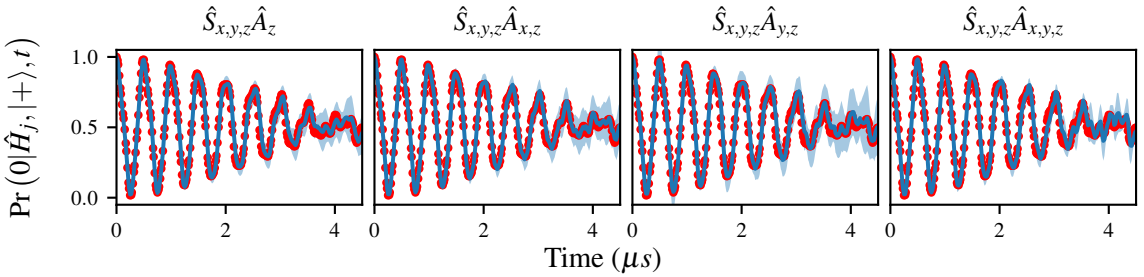
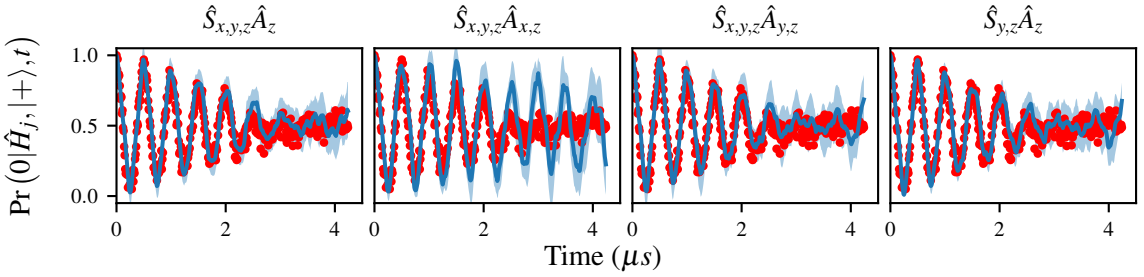


Figure 9.6: **Left,** map of the various Hamiltonian terms included in each of the possible 40 candidate models explored by QMLA during any of the 100 instances on either simulated or experimental data. IDs of candidate models are on the vertical axis, and labels for the terms on the horizontal axis. The true model $\hat{H}_0$ for the simulated case is highlighted in green and a subset of credible models in blue, i.e. models which may reasonably be expected to describe the targeted nitrogen-vacancy centre from theoretical arguments. **Right,** number of wins for each of the candidate models out of 100 independent QMLA runs. Cases adopting simulated data are shown by empty bars, with those using the experimental dataset shown by green bars. Implementation details are listed in Table A.1.



(a) Simulated data



(b) Experimental data

Figure 9.7: Dynamics reproduced by various QMLA champion models for **(a)** simulated and **(b)** experimental data. Likelihoods, $\Pr(0)$ are shown on y–axis with time on x–axis. Red dots give the true dynamics of $\hat{H}_0$, while the blue lines show the median reconstruction by $\hat{H}'$ from all the instances where that model was deemed $\hat{H}'$, with light blue showing the 67% confidence region. $\hat{H}'$ is listed on top of each plot; the number of instances won by each model can be read from Table 9.1. Implementation details are listed in Table A.1.

The key insight promised by QMLA is to identify the interactions present in the studied system, which in this case was the decoherence processes of an NVC. In Fig. 9.8 we show the number of times each of the terms permitted, i.e. $\hat{t} \in \mathcal{T}_{\text{NV}}$ from Eq. (9.14), are included in the champion model, as well as the distribution of parameter estimates for those terms. From the simulated case, we see that those terms which are in $\hat{H}_0$, i.e. $\hat{t} \in \mathcal{T}_0$, are found in almost all instances. Furthermore, while most instances find a champion model which includes some erroneous term(s), each $\hat{t} \notin \mathcal{T}_0$ is found with less than a quarter of the frequency of those $\hat{t} \in \mathcal{T}_0$. Hence terms outside of $\mathcal{T}_0$ may be reasonably ruled out in post-processing the QMLA results, by manually considering the relative frequency with which each term is found. The inaccurate terms found most often are seen to have (almost) negligible parameters: in conjunction with domain expertise, users can determine whether the inclusion of these terms are meaningful or simply artefacts of slight overfitting. In the experimental run, on the other hand, we see a similar gulf in frequency between some terms. Namely, $\{\hat{S}_x, \hat{S}_y, \hat{S}_z, \hat{A}_z\}$ are found in over 50 instances *more* than all other terms: we therefore conclude that those terms contribute most strongly to the NVC's decoherence process. The resultant model, $\hat{S}_{xyz}\hat{A}_z$ is in agreement with theoretical

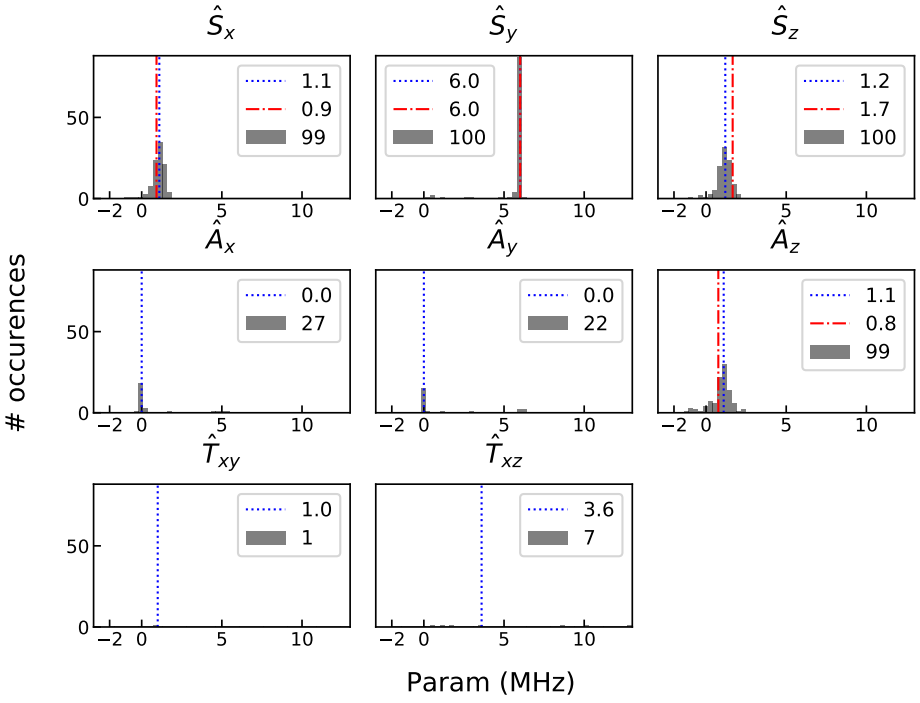
expectations, and shows that we can describe the intricate processes involved in decoherence through a relatively simple Hamiltonian, proving that QMLA can perform an important role in aiding the understanding of quantum systems.

9.5.2 *Finite size effects*

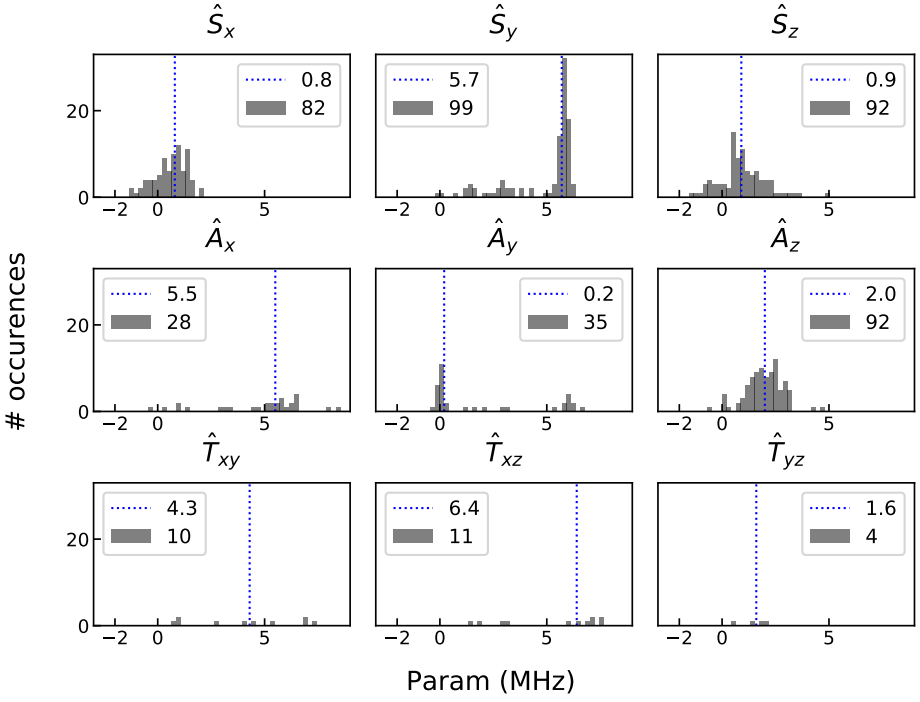
The case examined in this chapter is fundamentally limited by the nature of the approximation: by assigning a closed Hamiltonian to the NVC, we may never retrieve a complete description of the system. Closed, conservative dynamical systems – with a finite spectrum of permissible states – will return to their initial state after some finite time, according to the Poincaré recurrence theorem [43]. For quantum systems, then, the Poincaré recurrence times $\{t_R\}$ correspond to the times at which the *global* system's state, $|\psi_g\rangle$, returns to its starting state, $\left|\langle\psi_g(t_R)|\psi_g(0)\rangle\right|^2 \approx 1$ [197]. Such a system is therefore expected to exhibit *revivals*; in the context of quantum many-body systems, this is where the system *rephases* some time after it had dephased. These revivals are not necessarily physical, but rather an artefact of the approximation: realistic systems are seen to decohere completely after some decoherence time without revivals, i.e. *irreversible decoherence*. The Poincaré recurrence time scales polynomially with the number of degrees of freedom, n, within the closed system: $t_R \sim n!$. That is, even though the system of many interacting components *will* rephase, the time required to do so may be greater than the age of the universe [204]. Conversely, in our simplistic two-qubit model, the spectrum of eigenenergies is artificially restricted, and so revivals will appear unrealistically quickly.

Fig. 9.9 shows the predicted revivals of $\hat{H}' = \hat{S}_{x,y,z}\hat{A}_z$, the model found by QMLA to describe the NVC in the above analysis. The predicted revivals are unlikely to correspond to further experimental data; we can see intuitively that the system had decohered completely and irreversibly after 4.2 µs. When modelling such systems in practice, these revivals are often suppressed by a time-dependent damping $e^{-\Gamma t}$, with Γ determined phenomononoligcally, in order to yield the irreversible decoherence observed in real experimental systems, including the one used throughout this chapter. Here our aim was to model the interactions of the NVC spin with its nearest environmental qubit, which would be undermined by including such a drastic time-dependent effect: we omit this factor in this work, however it will prove important to account for this behaviour in future applications of QMLA to open systems.

Invoking a finite-size bath to approximate the genuine open-system nature of the NVC permits us to examine the interaction of the spin with few neighbours, sufficient to characterise the dominant decoherence processes of a system, as we have done in this chapter. However, the approximation can not capture all of the spin's interactions, so we are motivated to consider models involving more nuclei, and ultimately models of open quantum system. In Chapter 10 we consider a similar NVC setup in simulation, to explore QMLA's capability while relaxing the effect of the finite size bath, and we note in Chapter 11 that in order for QMLA to meaningfully characterise genuine quantum systems, it must explore models of open systems.



(a) Simluated QMLA instances. Red dotted lines show the true parameters.



(b) Experimental QMLA instances.

Figure 9.8: Parameter values for terms found by QMLA. For each term found within the champion model of at least one instance of QMLA, a histogram is shown for the values of the term's associated parameter. The terms are listed along the top of each subplot with results for (**a**) simulated and (**b**) experimental data. Blue dotted lines indicate the median for that parameter, while red dotted lines give the true parameter in the simulated case. Grey blocks show the number of champion models which found the parameter to have that value, and the number listed in the legend reports the total number of champion models which contained that term. Implementation details are listed in Table A.1.

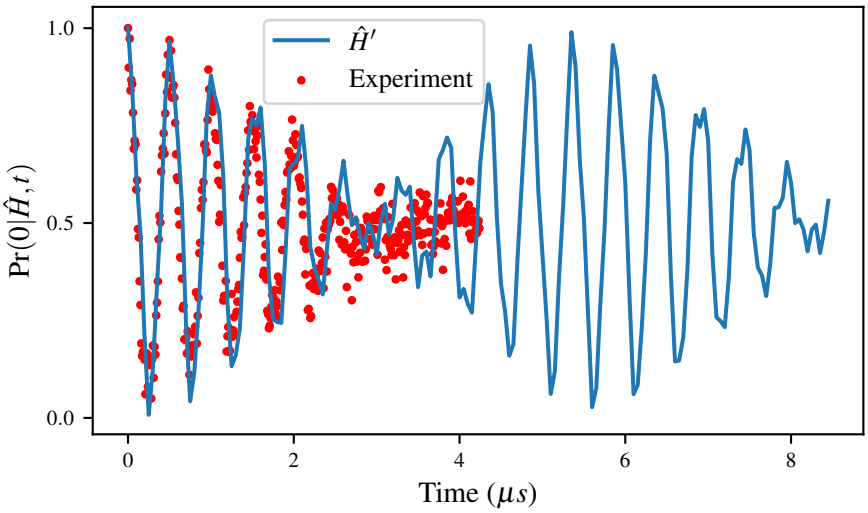


Figure 9.9: Predicted revivals of QMLA champion model, $\hat{H}'$ (blue), against the experimental data (red).

9.5.3 *Outlook*

We have thus characterised the interactions which dominate the decoherence process for a given NVC. In doing so, we identified not only the terms present, but also the strength (i.e. parameters) of those terms. Automated characterisation of quantum systems will be essential in the development of quantum technologies, whether for calibration of controlled devices, or alignment of experimental systems for optimal results. While this demonstration must be understood in the context of its limitations, e.g. the restricted basis studied, and the constrained model space searched, it represents a crucial proof of principle that QMLA is applicable to the task of automated quantum system characterisation. In the next chapter, we extend QMLA in simulation, to overcome some of the limitations mentioned here.

10 LARGER REALISTIC SYSTEMS

Chapter 9 concerned a two-qubit approximation of the short-time dynamics of an nitrogen-vacancy centre (NVC). It is valid criticism that the corresponding model space searched was reduced substantially through prior knowledge, and it therefore remains to test Quantum Model Learning Agent (QMLA) in a large model space, on physically meaningful data. In this chapter, we extend QMLA to consider approximations of NVC systems using more qubits, representing several nuclear sites, which aim to capture the interactions between the target NVC and the environment more thoroughly. Here we will simulate the target system, allowing us to make definite statements on the performance of QMLA, unlike the experimental data where we can not be sure of the dynamics' generator.

10.1 TARGET SYSTEM

A realistic model may be expected from considering the environment as a finite-size bath, consisting of n_s nuclear spins in addition to the NVC spin, i.e. the total number of qubits of such a model is $n_q = 1 + n_s$. The effects of nuclear spins are expected to manifest at higher times than those studied in Chapter 9, e.g. the decoherence of the NVC is only effected by the nuclear spins' independent precession at higher times. To study these *slowly varying* contributions to the spin's decoherence, we must modify the experimental procedure: the Hahn echo sequence isolates these types of dynamics when the second free evolution runs for the same duration as the first, $t' = t$ [198], detailed in Section 9.1.1.

Since we are simulating the target system, we may choose the approximation we wish to invoke. Starting from the Hamiltonian expected to describe NVCs in [199], we again exclude the zero-field and quadrupole splitting (as in Eq. (9.4)), and assume the complete Hamiltonian to describe the long-time dynamics of an NVC,

$$\hat{H}_{\text{long}} = \mu_B g \mathbf{B} \cdot \mathbf{S} + \gamma \cdot \mathbf{B} \cdot \mathbf{I} + \mathbf{A} \cdot \mathbf{S} \cdot \mathbf{I}, \tag{10.1}$$

where[1]

- $\mu_B = 9.274 \times 10^{-24} \text{J T}^{-1}$ is the Bohr magneton;
- $g \approx 2$ is the electron g-factor;
- $\mathbf{B}$ is the magnetic field with magnitude $B = 11\,\text{mT}$;
- $\mathbf{S}$ is the total electron spin operator;
- $\mathbf{I}$ is the total nuclear spin operator.
- γ is gyromagnetic ratio, with dominant axial contributions from ^{13}C of $\gamma_z = 10.8\,\text{MHz T}^{-1}$ and non-axial contribution from proximal nuclei of $\gamma_x = \gamma_y = 42.6\,\text{MHz T}^{-1}$ [205];

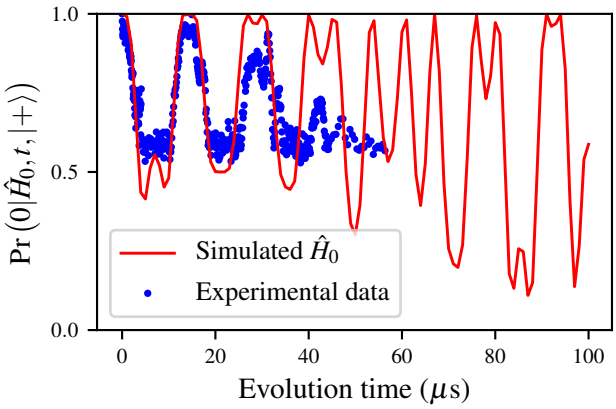


Figure 10.1: Long-time dynamics for nitrogen-vacancy centre, red, showing revivals, generated by $\hat{H}_0$ from Eq. (10.4), via Hahn echo measurement with $t' = t$. For comparison, experimentally generated dynamics are shown in blue.

- **A** is the hyperfine tensor, coupling the electron with the nuceli, here we take it to consist of $A = A_x = A_y = A_z = 0.2\,\text{MHz}$ [206].

We perform the same mapping to qubits as in Section 9.2: we let the first qubit represent the electron, and reduce the spin tensors to sums over nuclear sites, $\{\chi\}$. The complete model is then given by

$$\begin{aligned}\hat{H}_{\text{long}} =& \mu_B g B \sum_{w\in\{x,y,z\}} \hat{S}_w \\ &+ A \sum_{w\in\{x,y,z\}} \sum_{\chi\in\{\chi\}} \hat{S}_w \cdot \hat{A}^{\chi}_w \\ &+ B \sum_{w\in\{x,y,z\}} \gamma_w \sum_{\chi\in\{\chi\}} \hat{I}^{\chi}_w.\end{aligned} \tag{10.2}$$

For the purpose of testing QMLA, we can choose a subset of terms from Eq. (10.2) to constitute the true model: to set $\hat{H}_0$, we use the *secular* approximation, i.e. we assume the magnetic field is perfectly aligned along the z-axis [195, 201]. In the secular approximation, the NVC spin qubit rotates only about the z-axis, and coupling between the NVC and nuclear qubits are only via $\hat{S}_z \cdot \hat{A}^{\chi}_z$. Here we will include the effect of the nuclear spins' rotations, which are much weaker

1 Note: the parameter values used here do not necessarily correspond to a physical system as they are taken from a range of sources. The precise values should not matter to the discussion, but are intended to generate dynamics using realistic values, which are similar to a genuine system, by inspection of Fig. 10.1.

Term	$\hat{t}$	Meaning	Parameter (Hz)	$\in \hat{H}_0$
$\hat{S}_x$	$\hat{\sigma}_x^1$	electron spin rotation about x-axis	2×10^9	No
$\hat{S}_y$	$\hat{\sigma}_y^1$	electron spin rotation about y-axis	2×10^9	No
$\hat{S}_z$	$\hat{\sigma}_z^1$	electron spin rotation about z-axis	2×10^9	Yes
$\hat{S}_x \cdot \hat{A}_x^j$	$\hat{\sigma}_x^1\hat{\sigma}_x^j$	hyperfine coupling with j^{th} nuclear qubit, x-axis	0.2×10^6	No
$\hat{S}_y \cdot \hat{A}_y^j$	$\hat{\sigma}_y^1\hat{\sigma}_y^j$	hyperfine coupling with j^{th} nuclear qubit, y-axis	0.2×10^6	No
$\hat{S}_z \cdot \hat{A}_z^j$	$\hat{\sigma}_z^1\hat{\sigma}_z^j$	hyperfine coupling with j^{th} nuclear qubit, z-axis	0.2×10^6	Yes
$\hat{I}_x^j$	$\hat{\sigma}_x^j$	j^{th} nuclear spin rotation about x-axis	66×10^3	Yes
$\hat{I}_y^j$	$\hat{\sigma}_y^j$	j^{th} nuclear spin rotation about y-axis	66×10^3	Yes
$\hat{I}_z^j$	$\hat{\sigma}_z^j$	j^{th} nuclear spin rotation about z-axis	15×10^3	Yes

Table 10.1: Terms permitted in the QMLA GA (genetic algorithm) when modelling the extended nitrogen-vacancy centre systems. Each term is permitted in the model search from expectations about the system under study. The succinct representation is listed as $\hat{t}$, along with an interpretation of the term's physical contribution to the system. The parameter values used in simulations can be found from from the listings of Eq. (10.1). The presence of the term in the true model is indicated in the final column: $\hat{t} \notin \hat{H}_0$ do not contribute to the dynamics of Fig. 10.1 but are available to the genetic algorithm (GA) when constructing models.

and only influence the NVC's decoherence at long times. In total then, the set of nuclear spins, $\{\chi\}$, are mapped to n_s qubits, and we define the true model as

$$\hat{H}_0 = \hat{S}_z + \sum_{j=2}^{n_q} \hat{S}_z \cdot \hat{A}_z^j + \sum_{w \in \{x,y,z\}} \sum_{j=2}^{n_q} \hat{I}_w^j, \tag{10.3}$$

with the parameters of Eq. (10.2) now absorbed, listed in Table 10.1, giving the dynamics in Fig. 10.1. For simplicity, we restate this in terms only of the Pauli matrices, where the first qubit refers to the NVC and the remaining qubits give the interactions and nuclear terms.

$$\hat{H}_0 = \hat{\sigma}_z^1 + \sum_{j=2}^{n_q} \hat{\sigma}_z^1\hat{\sigma}_z^j + \sum_{w \in \{x,y,z\}} \sum_{j=2}^{n_q} \hat{\sigma}_w^j, \tag{10.4}$$

so in total, the set of terms for Q, $\mathcal{T}_0$, has 1 term for the NVC qubit, n_s terms for hyperfine couplings and $3n_s$ terms for the nuclei: $|\mathcal{T}_0| = 1 + 4n_s$.

We set the goal of QMLA as finding the approximation of Eq. (10.4), by allowing it to consider a wider set of terms. The permissible terms are then all terms from Eq. (10.2), i.e. the spin rotation terms about all axes, as well as all nuclei rotation terms, and the coupling terms:

$$\mathcal{T} = \left\{ \begin{array}{c} \hat{S}_w = \hat{\sigma}^1_w, \\ \hat{I}^j_w = \hat{\sigma}^j_w, \\ \hat{S}_w \cdot \hat{A}_w = \hat{\sigma}^1_w \hat{\sigma}^j_w \end{array} \right\} \tag{10.5}$$

for $w = \{x, y, z\}$ and $j \in \{2, ..., n'_q\}$. Note that n'_s is the number of nuclear spins considered by QMLA, but not necessarily the same number of nuclear spins, n_s, present in $\hat{H}_0$: in general $n'_s + 1 = n'_q \neq n_q$. In total, $|\mathcal{T}| = 3 + 3n'_s + 3n'_s = 3 + 6n'_s$.

Our aim is to test QMLA, so the choice of n_s and n'_s are arbitrary: we will allow QMLA to explore a larger model space than is required to capture the true model, in order to give QMLA the means to overfit, as a robust test. For the target system we use $n_s = 3$ proximal spins, so that $|\mathcal{T}_0| = 13$; we allow candidates up to $n'_s = 5$, so $|\mathcal{T}| = 33$. In the most general sense, irrespective of the underlying physics we are simulating, here QMLA is aiming to identify the 13 terms truly present in Q, while searching the space of 33 permissible terms. Without imposing any restrictions on which combinations of terms are allowed, each term is simply either in $\hat{H}'$ or not, so can be thought of as binary variables: the total model space is therefore of size $2^{|\mathcal{T}|} = 2^{33} \approx 10^{10}$.

10.2 GENETIC ALGORITHM

Genetic algorithms (GAs) provide a robust and thoroughly tested paradigm for searching large candidate spaces; this is a natural framework through which we can explore such an unrestricted model space as described above. We have already extensively discussed the formalism of GAs in Section 3.3, and specifically in the context of QMLA in Chapter 8. Here we will use the same exploration strategy (ES) as described in Section 8.1, i.e. where model generation is driven by a GA, and models are cast to chromosomes. In particular, candidate model's fitness will be computed from the residuals between their and the system's dynamics, described fully in Section 8.2.6. This objective function (OF) relies on the definition of a validation dataset, $\mathcal{E}_v$, which we compose of tomographic probes (see Section 4.7) and times generated uniformly up to $t_{max} = 100\mu s$, Fig. 10.2.

10.2.1 *Parameter learning*

Our primary goal in this chapter is to validate QMLA's performance in a very large model space, with over 10^{10} valid candidates. Our focus, then, is on model *generation*, and not concerned with parameter learning: we do *not* train models individually, but rather we assume access to a *perfect* parameter learning subroutine. That is, for each candidate, $\hat{H}_i$, considered, we simply

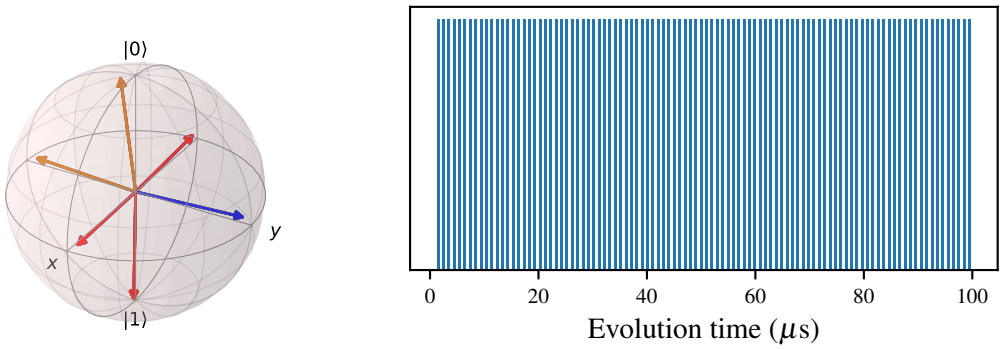


Figure 10.2: Evaluation dataset, $\mathcal{E}_v$, for nitrogen-vacancy centre genetic algorithm. **Left**, Set of 1-qubit probe states the NVC qubit is prepared in for evaluation, i.e. Ψ_v is close to the tomographic basis. Shown are the one-qubit probes on the Bloch sphere, which are combined to form n-qubit probes used when evaluating candidate models. **Right**, Time comb evaluated against, i.e. uniformly distributed times up to $t_{max} = 100\mu$s are used for experiments in $\mathcal{E}_v$.

assume knowledge of its parameters, $\vec{\alpha}_i$. This assumption is a major caveat to the results of this chapter: no such perfect training scheme is known, so it remains to examine the detrimental effects of imprecisely finding $\vec{\alpha}'_i \approx \vec{\alpha}_i$. Moreover, while it is possible to extract information on the nuclear qubits from measuring only the NVC qubit, as in the Hahn echo measurements, it is uncertain whether any technique can simultaneously detect parameters of significantly varying orders of magnitude. For instance, some terms in Table 10.1 are $\mathcal{O}$(GHz), while others are $\mathcal{O}$(kHz); it is likely to prove difficult to discern the kHz parameters well, given that their contribution is equivalent to errors of order $\mathcal{O}(10^{-6})$ in the dominant GHz terms. Finally, the terms in candidate models which are *not* in $\hat{H}_0$ must be assigned learned parameters as though they had undergone quantum Hamiltonian learning (QHL); we do not know how QHL would treat such terms in reality, and here we simply assume such terms would learn the parameter listed in Table 10.1.

Given these considerations, therefore, we must caution that the results presented here, while demonstrating that QMLA *can* operate in large model spaces, are not immediately applicable to experimental systems, since there are outstanding challenges in the assessment of individual candidates, which must be overcome before the technique outlined can realistically succeed.

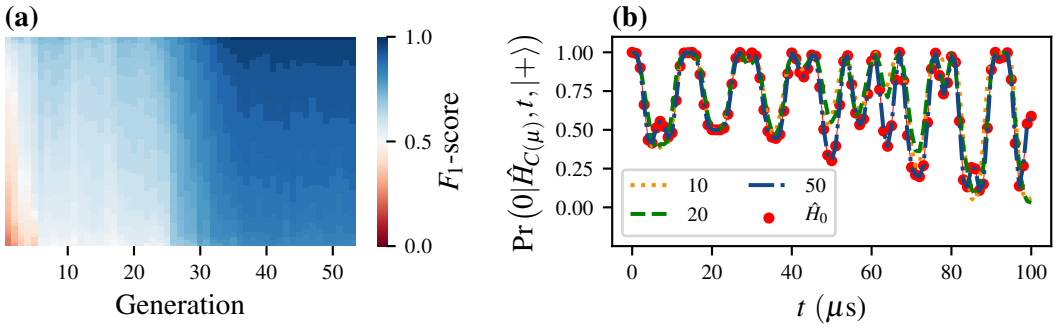


Figure 10.3: Instance of the genetic algorithm (GA) for simulated nitrogen-vacancy centre system with four qubits. **a**, Gene pool progression for the GA. Each tile represents a candidate model by its F_1-score. Each generation considers $N_m = 72$ models; the GA runs for $N_g = 53$ generations. **b**, Branch champions' dynamics. Each generation, μ, nominates a branch champion, $\hat{H}_{C(\mu)}$. Here, progressive generations' champions dynamics are shown against those of the target system, $\hat{H}_0$ (red).

10.2.2 *Results*

At the instance level, we can see that the gene pool tends towards models of higher quality, captured[2] by their F_1-score, Fig. 10.3**(a)**. The improvement in modelling is reflected in the branch champions' predictive power at reproducing data generated by the system, Fig. 10.3**(b)**.

Considering the overall run, we see that QMLA is searching in a vast model space where randomly sampled models have poor F_1-score on average, Fig. 10.4**(a)**. QMLA efficiently explores the space by quickly moving into a subspace of high F_1-score, nominating $\hat{H}' = \hat{H}_0$ precisely in 85% of instances, Fig. 10.4**(b,c)**. The number of times each of the terms considered, Eq. (10.5), are present in $\hat{H}'$ offers the most important insight from QMLA, namely the evidence in favour of each term's presence, which can be used to infer the most likely underlying physics. Here, $\hat{t} \in \mathcal{T}_0$ are found in $\geq 94\%$ of instances, while $\hat{t} \notin \mathcal{T}_0$ are found in $\leq 11\%$, shown in Fig. 10.5 and listed in Table 10.2. Such a discrepancy, as well as the win rates for the models, allows for the clear declaration of the model $\hat{H}_0$ as the favoured representation for the quantum system.

By simulating a realistic system, we have hereby shown QMLA's ability to operate on physically meaningful data in large model spaces. The model search guided by a GA instructs QMLA to consider only $\mathcal{O}(10^3)$ models out of the total space of $\mathcal{O}(10^{10})$, clearly showing a drastic speedup in characterising Q when compared with brute force search. This demonstration is moderated by the presumptions which enabled us to perform simulations quickly and assume perfect outcomes from model training. However, taken together with the results of examining experimental data from Chapter 9, and the earlier confirmations of QMLA's operating principles

2 The use of F_1-score as a figure of merit for candidate models in the QMLA search is described in Section 8.1.2.

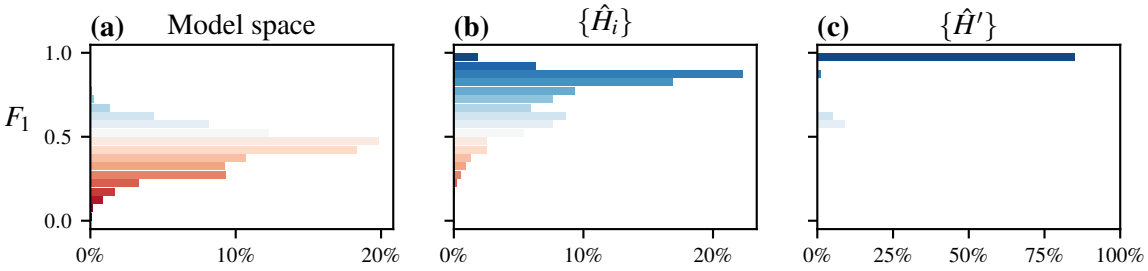


Figure 10.4: Nitrogen-vacancy centre genetic algorithm run. **a**, F_1-score of 10^6 samples from the model space of $2^{33} \approx 10^{10}$ candidate models, normally distributed around $f = 0.44 \pm 0.12$. **b**, The models explored during the model search of all instances combined, $\{\hat{H}_i\}$, show that QMLA tends towards stronger models overall, with $f = 0.79 \pm 0.16$ from $\mathcal{O}(140,000)$ chromosomes across the 100 instances, i.e. each instance tests $\mathcal{O}(1400)$ distinct models. **c**, Champion models from each instance, showing QMLA finds strong models in general, and in particular finds the true model ($\hat{H}_0$, with $f = 1$) in 85% of cases.

in Part III, QMLA shows promise for characterising genuine small-to-medium quantum systems in the near future.

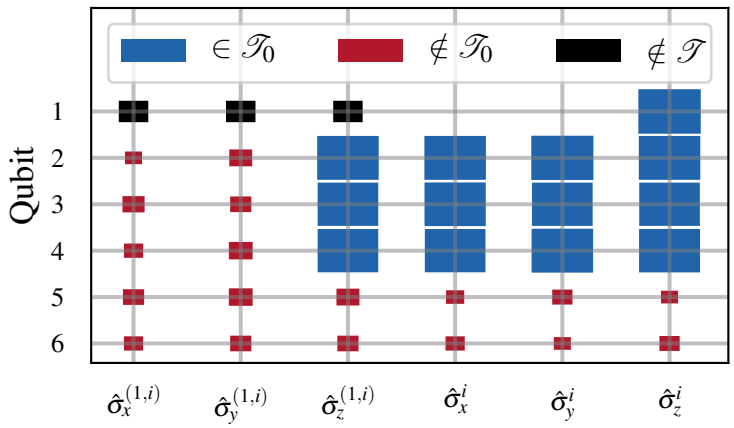


Figure 10.5: Hinton diagram of terms found for 4-qubit nitrogen-vacancy centre model. Terms are either in the target model ($\in \mathcal{T}_0$, blue) or not ($\notin \mathcal{T}_0$, red), or else not considered ($\notin \mathcal{T}$, black). Terms acting solely on the first qubit are the NVC spin's rotation terms, $\hat{\sigma}_w^1$, while each nuclear site also has rotation terms $\hat{\sigma}_w^j$. Hyperfine terms, $\hat{\sigma}_w^{(1,j)}$, couple the NVC qubit with the j^{th} nuclear spin. The precise rate at which each term is detected can be read from Table 10.2.

Qubit	$\hat{\sigma}_x^{(1,i)}$	$\hat{\sigma}_y^{(1,i)}$	$\hat{\sigma}_z^{(1,i)}$	$\hat{\sigma}_x^i$	$\hat{\sigma}_y^i$	$\hat{\sigma}_z^i$
1	-	-	-	0	0	100
2	5	11	97	97	99	97
3	10	9	94	96	94	94
4	7	12	94	94	97	95
5	9	12	11	6	8	5
6	7	9	9	7	5	8

Table 10.2: Percentage of instances for which each term is found by QMLA GA studying NVC system.

Part V

CONCLUSION

11

OUTLOOK FOR MODEL LEARNING METHODOLOGIES

Optimal control techniques are a crucial component in improving quantum technologies, such that imperfect near-term devices may be leveraged to achieve some meaningful quantum advantages. The developments presented in this thesis contribute to the growing interest in automatic characterisation and verification of quantum systems and devices. Namely, the introduction of the Quantum Model Learning Agent (QMLA) represents an important advancement, whereby quantum systems can be completely characterised starting with little prior knowledge. The majority of this thesis was dedicated to the rigorous testing of QMLA, gradually moving from ideal scenarios in simulation to genuine experimental quantum systems.

We described the implementation of QMLA as an open source software platform in Part II, detailing numerous tunable aspects of the protocol, and their impact on training candidate models in Chapter 4. QMLA facilitates customisation of its core elements and subroutines, such that it is applicable to a wide range of target quantum systems, as described in Chapters 5 to 6. This malleability enables users to easily adapt the framework to their own needs, and formed the basis for the cases studied in the remainder of the thesis: we tested QMLA by devising a series of exploration strategies, each corresponding to a different target quantum system.

In Part III we considered ideal theoretical quantum systems in simulation. Initial tests in Chapter 7 showed that QMLA could distinguish between different physical scenarios and internal configurations. In Chapter 8, we explored much larger model spaces by incorporating a genetic algorithm into QMLA's model design; the genetic algorithm showed promise for characterising complex quantum systems by successfully identifying the target model. The performance of the genetic algorithm, however, came at the expense of relying on a restrictive subroutine – used for training individual candidate models – drastically reducing its applicability to realistic systems. However, the restriction is permitted in the scope of characterising *controlled* quantum systems, for example new, untrusted quantum simulators.

We concluded the thesis by considering realistic quantum systems in Part IV. Experimental data from an electron spin in a nitrogen-vacancy centre was treated in Chapter 9; this too relied upon tailoring QMLA's procedure with respect to the system under study. A theoretically justified Hamiltonian is proposed by QMLA to describe the decoherence of the electron spin, yielding a highly predictive model in agreement with the system's measured dynamics, albeit exploring a small model space. To overcome concerns that the model search was artificially constrained in the context of realistic systems, Chapter 10 exercised QMLA in a vast model space, spanning terms which represent plausible interactions for the same type of system. Here, again, QMLA achieved high success rates, but with caveats on the subroutines assumed for model training, and resorting to simulated data.

In summary, this thesis has provided extensive tests of the QMLA algorithm, but each may be undermined by its individual constraints. In outlook, near-term developments of model learning

methodologies in the context of quantum systems must address these shortcomings, for instance by unifying the strategies described in this thesis. Further, we anticipate immediate application in the study of open quantum systems, by replacing the Hamiltonian formalism examined here with a Lindbladian representation, permitted within the QMLA apparatus. Through the advancements presented herein, we hope to have provided a solid foundation upon which these constraints may be relaxed, ultimately with a view to providing an automated platform for the complete characterisation of quantum systems. We envision QMLA as a straightforward but powerful utility for quantum engineers in the design of near term quantum devices, expecting continued development of the framework alongside the burgeoning open-source quantum software ecosystem.

APPENDIX

A

FIGURE REPRODUCTION

The default behaviour of the Quantum Model Learning Agent (QMLA) software framework is to generate a *results directory* for every run of the algorithm, uniquely identified by the date and time the run was launched, e.g. qmla/launch/Jan_01/12_34. Most of the figures presented in the main text analyse data from runs of the QMLA framework; the results directory corresponding to each figure are listed (as *Data*) in Table A.1, and all data presented herein can be downloaded from [207]. The presented figures are usually available as default analyses through the QMLA framework, and therefore can be reproduced by implementing the corresponding configuration. In Table A.1 we list the core implementation details – the number of experiments N_E, and particles, N_P and the exploration strategy (ES) – used to achieve those results. In some cases, further configuration details are necessary to produce equivalent runs; complete details and information are available at [207]. Instructions for installing, configuring and running the QMLA software are given in Appendix C.

A.1 STRUCTURE OF RESULTS DIRECTORY

Within the results directory, the outcome of the run's instances are stored, with analysis plots broadly grouped as

- evaluation: plots of probes and times used as the evaluation dataset.
- instances: outcomes of an individual QMLA instance, grouped by the instance ID. Includes results of training of individual models (in model_training), as well as sub-directories for anlaysis at the branch level (in branches) and comparisons.
- combined_datasets: pandas dataframes containing most of the data used during analysis of the run. Note that data on the individual model/instance level may be discarded so some minor analyses can not be performed offline.
- exploration_strategy_plots plots specifically required by the ES at the run level.
- champion_models: analysis of the models deemed champions by at least one instance in the run, e.g. average parameter estimation for a model which wins multiple instances.
- performance: evaluation of the QMLA run, e.g. the win rate of each model and the number of times each term is found in champion models.
- meta analysis of the algorithm' implementation, e.g. timing of jobs on each process in a cluster; generally users need not be concerned with these.

Figure	Exploration Strategy	N_E	N_P	Data
Fig. 4.4	DemoHeuristicPGH	1000	3000	Nov_27/19_39
	DemoHeuristicNineEighths	1000	3000	Nov_27/19_40
	DemoHeuristicTimeList	1000	3000	Nov_27/19_42
	DemoHeuristicRandom	1000	3000	Nov_27/19_47
Fig. 4.5	DemoProbesPlus	1000	3000	Nov_27/14_43
	DemoProbesZero	1000	3000	Nov_27/14_45
	DemoProbesTomographic	1000	3000	Nov_27/14_46
	DemoProbes	1000	3000	Nov_27/14_47
Fig. 4.2	AnalyticalLikelihood	500	2000	Nov_16/14_28
Fig. 7.2	DemoIsing	500	5000	Nov_18/13_56
Fig. 7.3	DemoIsing	1000	5000	Nov_18/13_56
Fig. 7.4	DemoIsing	1000	5000	Nov_18/13_56
Fig. 7.5	IsingLatticeSet	1000	4000	Nov_19/12_04
	IsingLatticeSet	1000	4000	Nov_19/12_04
	IsingLatticeSet	1000	4000	Nov_19/12_04
Fig. 7.6	IsingLatticeSet	1000	4000	Sep_30/22_40
	HeisenbergLatticeSet	1000	4000	Oct_22/20_45
	FermiHubbardLatticeSet	1000	4000	Oct_02/00_09
Fig. 7.7	IsingReducedLatticeSet	125	500	Feb_16/09_12
	HeisenbergReducedLatticeSet	125	500	Feb_16/09_14
	HubbardReducedLatticeSet	125	500	Feb_16/09_16
	IsingReducedLatticeSet	250	1000	Feb_15/21_49
	HeisenbergReducedLatticeSet	250	1000	Feb_15/21_47
	HubbardReducedLatticeSet	250	1000	Feb_15/21_45
	IsingReducedLatticeSet	500	2000	Feb_16/09_20
	HeisenbergReducedLatticeSet	500	2000	Feb_16/09_19
	HubbardReducedLatticeSet	500	2000	Feb_16/09_18
	IsingReducedLatticeSet	1000	4000	Feb_16/18_33
	HeisenbergReducedLatticeSet	1000	4000	Feb_16/18_34
	HubbardReducedLatticeSet	1000	4000	Feb_16/18_35

Table A.1: Implementation details for figures used in the main text. For each *Figure*, we list the *Exploration Strategy* used by the QMLA framework to achieve the corresponding results, which are stored in the results directory labelled by *Data*. We list the number of experiments, N_E, and particles, N_P, used in the training of all models within that run. Equivalent runs of QMLA can be launched as outlined in Appendix C. Continued in Table A.2.

Figure	Exploration Strategy	N_E	N_P	Data
Fig. 8.3	N/A	N/A	N/A	Dec_07/22_04
Fig. 8.2	DemoBayesFactorsByFscore	500	2500	Dec_09/12_29
	DemoFractionalResourcesBayesFactorsByFscore	500	2500	Dec_09/12_31
	DemoBayesFactorsByFscore	1000	5000	Dec_09/12_33
	DemoBayesFactorsByFscoreEloGraphs	500	2500	Dec_09/12_32
Fig. 8.5	HeisenbergGeneticXYZ	500	2500	Dec_10/14_40
Fig. 8.6	HeisenbergGeneticXYZ	500	2500	Dec_10/14_40
	HeisenbergGeneticXYZ	500	2500	Dec_10/14_40
Fig. 8.7	HeisenbergGeneticXYZ	500	2500	Mar_07/12_40
	HeisenbergGeneticXYZ	500	2500	Dec_10/16_12
Fig. 8.8	HeisenbergGeneticXYZ	500	2500	Dec_18/20_12
Fig. 9.6	NVCentreExperimentalData	1000	3000	2019/Oct_02/18_01
	SimulatedExperimentNVCentre	1000	3000	2019/Oct_02/18_16
Fig. 9.7	NVCentreExperimentalData	1000	3000	2019/Oct_02/18_01
Fig. 9.6	SimulatedExperimentNVCentre	1000	3000	2019/Oct_02/18_16
Fig. 9.8	SimulatedExperimentNVCentre	1000	3000	2019/Oct_02/18_16
	NVCentreExperimentalData	1000	3000	2019/Oct_02/18_01
Fig. 10.2	NVCentreGenticAlgorithmPrelearnedParameters	2	5	Sep_09/12_00
	NVCentreGenticAlgorithmPrelearnedParameters	2	5	Sep_09/12_00
Fig. 10.3	NVCentreGenticAlgorithmPrelearnedParameters	2	5	Sep_09/12_00
	NVCentreGenticAlgorithmPrelearnedParameters	2	5	Sep_09/12_00
Fig. 10.4	NVCentreGenticAlgorithmPrelearnedParameters	2	5	Sep_08/23_58
Fig. 10.5	NVCentreGenticAlgorithmPrelearnedParameters	2	5	Sep_08/23_58

Table A.2: [Continued from Table A.1] Implementation details for figures used in the main text. For each *Figure*, we list the *Exploration Strategy* used by the QMLA framework to achieve the corresponding results, which are stored in the results directory labelled by *Data*. We list the number of experiments, N_E, and particles, N_P, used in the training of all models within that run. Equivalent runs of QMLA can be launched as outlined in Appendix C.

B

FUNDAMENTALS

There are a number of concepts which are fundamental to any discussion of quantum mechanics (QM), but are likely to be known to most readers, and are therefore cumbersome to include in the main body of the thesis. We include them here for completeness[1].

B.1 LINEAR ALGEBRA

Here we review the language of linear algebra and summarise the basic mathematical techniques used throughout this thesis. We will briefly recall some definitions for reference.

Definition of	Representation
Vector (or *ket*)	$\|\psi\rangle$
Dual Vector (or *bra*)	$\langle\psi\|$
Tensor Product	$\|\psi\rangle \otimes \|\phi\rangle$
Complex conjugate	$\|\psi^*\rangle$
Transpose	$\|\psi\rangle^T$
Adjoint	$\|\psi\rangle^\dagger = (\|\psi\rangle^*)^T$

Table B.1: Linear algebra definitions.

The dual vector of a vector, or *ket* $|\psi\rangle$, is given by its *bra*, $\langle\psi| = |\psi\rangle^\dagger$.

The *adjoint* of a matrix replaces each matrix element with its own complex conjugate, and then switches its columns with rows.

$$M^\dagger = \begin{pmatrix} M_{0,0} & M_{0,1} \\ M_{1,0} & M_{1,1} \end{pmatrix}^\dagger = \begin{pmatrix} M^*_{0,0} & M^*_{0,1} \\ M^*_{1,0} & M^*_{1,1} \end{pmatrix}^T = \begin{pmatrix} M^*_{0,0} & M^*_{1,0} \\ M^*_{0,1} & M^*_{1,1} \end{pmatrix}. \tag{B.1}$$

1 Much of this description is reproduced from my undergraduate thesis [208].

The *inner product* of two vectors, $|\psi\rangle = \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_n \end{pmatrix}$ and $|\phi\rangle = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix}$ is given by

$$\langle\phi|\psi\rangle = (|\phi\rangle^\dagger)\,|\psi\rangle = (\phi_1^* \ \phi_2^* \ \dots \ \phi_n^*) \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_n \end{pmatrix} = \phi_1^*\psi_1 + \phi_2^*\psi_2 + \cdots + \phi_n^*\psi_n. \tag{B.2}$$

$|\psi\rangle_i, |\phi\rangle_i$ are complex numbers, and therefore the above is simply a sum of products of complex numbers. The inner product is often called the *scalar product*, which is in general complex.

B.2 POSTULATES OF QUANTUM MECHANICS

There are numerous statements of the postulates of quantum mechanics. Each version of the statements aims to achieve the same foundation, so we endeavour to explain them in the simplest terms.

1 Every moving particle in a conservative force field has an associated state vector, known as its wavefunction, $|\psi\rangle \in \mathcal{H}$, where $\mathcal{H}$ is the *Hilbert space* to which the particle belongs. Normalised linear combinations of state vectors are valid state vectors, known as *superpositions*. From this wavefunction, it is possible to determine all physical information about the system.

2 Particles have physically *observable* properties, Q, corresponding to a linear operator $\hat{Q}$. Measurement of such an observable – i.e. $\hat{Q}$ acting on $|\psi\rangle$ – will only ever result in an *eigenvalue* of that operator, q, with the system immediately after measurement in the corresponding eigenstate, $|q\rangle$:

$$\hat{Q}\,|\psi\rangle = q\,|\psi\rangle\,. \tag{B.3}$$

The set of eigenvalues, $\{q\}$, is the spectrum of $\hat{Q}$: in the case where the spectrum is discrete, the possible outcomes of measurement are *quantised*.

3 Any such operator $\hat{Q}$ is Hermitian,

$$\hat{Q}^\dagger = \hat{Q}. \tag{B.4}$$

4 The set of eigenfunctions for an observable operator $\hat{Q}$ forms a complete set of linearly independent functions, i.e. a *basis* of the Hilbert space.

5 For a system with wavefunction $|\psi\rangle$, the expectation value of an observable Q with respect to an operator $\hat{Q}$, denoted $\langle q \rangle$, is given by

$$\langle q \rangle = \langle \psi | \hat{Q} | \psi \rangle = \sum_n q_n \langle q_n | \hat{Q} | q_n \rangle . \tag{B.5}$$

6 The time evolution of $|\psi\rangle$ is given by the time dependent *Schrödinger Equation*

$$i\hbar \frac{\partial |\psi\rangle}{\partial t} = \hat{H} |\psi\rangle , \tag{B.6}$$

where $\hat{H}$ is the system's Hamiltonian.

Using these building blocks, we can begin to construct a language to describe quantum systems.

B.3 STATES

An orthonormal basis consists of vectors of unit length which do not overlap, e.g. $|x_1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |x_2\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow \langle x_1 | x_2 \rangle = 0$. In general, if $\{|x\rangle\}$ are the eigenstates of a system, then the system can be written as some state vector, $|\psi\rangle$, in general a superposition over the basis-vectors:

$$|\psi\rangle = \sum_x a_x |x\rangle \tag{B.7a}$$

$$\text{subject to} \quad \sum_x |a_x|^2 = 1, \quad a_x \in \mathbb{C}. \tag{B.7b}$$

The *state space* of a physical system (classical or quantum) is then the set of all possible states the system can exist in, i.e. the set of all possible values for $|\psi\rangle$ such that Eq. (B.7b) are satisfied.

For example, photons can be polarised horizontally ($\leftrightarrow$) or vertically ($\updownarrow$); take those two conditions as observable states to define the eigenstates of a two-level system, so we can designate the photon as a qubit. Then we can map the two states to a 2-dimensional, x-y plane: a general vector on such a plane can be represented by a vector with coordinates $\begin{pmatrix} x \\ y \end{pmatrix}$. These polarisations can then be thought of as standard basis vectors in linear algebra. Denote $\leftrightarrow$ as the eigenstate $|0\rangle$ and $\updownarrow$ as $|1\rangle$

$$|\leftrightarrow\rangle = |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \text{A unit vector along x-axis} \tag{B.8a}$$

$$|\updownarrow\rangle = |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \text{A unit vector along y-axis} \tag{B.8b}$$

Now, in relation to the concept of superposition, we can consider, for example, a photon in an even superposition of the vertical and horizontal polarisations, evenly splitting the two basis vectors. As such, we would require that, upon measurement, it is equally likely that the photon will *collapse* into the polarised state along x as it is to collapse along y. That is, we want $\Pr(\updownarrow) = \Pr(\leftrightarrow)$ so assign equal modulus amplitudes to the two possibilities:

$$|\psi\rangle = a\,|\leftrightarrow\rangle + b\,|\updownarrow\rangle\,, \quad \text{with} \quad \Pr(\updownarrow) = \Pr(\leftrightarrow) \Rightarrow |a|^2 = |b|^2 \tag{B.9}$$

We consider here a particular case, due to the significance of the resultant basis, where $\leftrightarrow$-polarisation and $\updownarrow$-polarisation have real amplitudes $a, b \in \mathbb{R}$.

$$\begin{aligned} &\Rightarrow a = \pm b \quad \text{but also} \quad |a|^2 + |b|^2 = 1 \\ \Rightarrow \quad &a = \frac{1}{\sqrt{2}} \quad ; \quad b = \pm\frac{1}{\sqrt{2}} \\ &\Rightarrow |\psi\rangle = \frac{1}{\sqrt{2}}\,|\leftrightarrow\rangle \pm \frac{1}{\sqrt{2}}\,|\updownarrow\rangle \\ &\Rightarrow |\psi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle \pm \frac{1}{\sqrt{2}}\,|1\rangle \end{aligned} \tag{B.10}$$

These particular superpositions are of significance:

$$|+\rangle = \frac{1}{\sqrt{2}}\,(|0\rangle + |1\rangle) \tag{B.11a}$$

$$|-\rangle = \frac{1}{\sqrt{2}}\,(|0\rangle - |1\rangle) \tag{B.11b}$$

This is called the Hadamard basis: it is an equally valid vector space as the standard basis which is spanned by $\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, as it is simply a rotation of the standard basis.

B.3.1 *Mulitpartite systems*

In reality, we often deal with systems of multiple particles, represented by multiple qubits. Mathematically, we consider the state vector of a system containing n qubits as being the tensor

product of the n qubits' individual state vectors[2]. For instance, suppose a 2-qubit system, $|\psi\rangle$ consisting of two independent qubits $|\psi_A\rangle$ and $|\psi_B\rangle$:

$$|\psi\rangle = |\psi_A\rangle\,|\psi_B\rangle = |\psi_A\psi_B\rangle = |\psi_A\rangle \otimes |\psi_B\rangle\,. \tag{B.12}$$

Consider first a simple system of 2 qubits. Measuring in the standard basis, these qubits will have to collapse in to one of the basis states $|0,0\rangle, |0,1\rangle, |1,0\rangle, |1,1\rangle$. Thus, for such a 2-qubit system, we have the general superposition

$$|\psi\rangle = \alpha_{0,0}|0,0\rangle + \alpha_{0,1}|0,1\rangle + \alpha_{1,0}|1,0\rangle + \alpha_{1,1}|1,1\rangle.$$

where $\alpha_{i,j}$ is the amplitude for measuring the system as the state $|i,j\rangle$. This is perfectly analogous to a classical 2-bit system necessarily occupying one of the four possibilities $\{(0,0),(0,1),(1,0),(1,1)\}$.

Hence, for example, if we wanted to concoct a two-qubit system composed of one qubit in the state $|+\rangle$ and one in $|-\rangle$

$$\begin{aligned}
|\psi\rangle &= |+\rangle \otimes |-\rangle \\
|\psi\rangle &= \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) \\
&= \frac{1}{2}\left[|00\rangle - |01\rangle + |10\rangle - |11\rangle\right] \\
&= \frac{1}{2}\left[\begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} - \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} - \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix}\right] \\
&= \frac{1}{2}\left[\begin{pmatrix}1\\0\\0\\0\end{pmatrix} - \begin{pmatrix}0\\1\\0\\0\end{pmatrix} + \begin{pmatrix}0\\0\\1\\0\end{pmatrix} - \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right]. \\
\Rightarrow |\psi\rangle &= \frac{1}{2}\begin{pmatrix}1\\-1\\-1\\1\end{pmatrix}
\end{aligned} \tag{B.13}$$

That is, the two qubit system – and indeed any two qubit system – is given by a linear combination of the four basis vectors

$$\{|00\rangle\,, |01\rangle\,, |10\rangle\,, |11\rangle\} = \left\{\begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\1\end{pmatrix}\right\}. \tag{B.14}$$

2 We will later discuss entangled states, which can not be described thus.

We can notice that a single qubit system can be described by a linear combination of two basis vectors, and that a two qubit system requires four basis vectors to describe it. In general we can say that an n-qubit system is represented by a linear combination of 2^n basis vectors.

B.3.2 *Registers*

A *register* is generally the name given to an array of controllable quantum systems; here we invoke it to mean a system of multiple qubits, specifically a subset of the total number of available qubits. For example, a register of ten qubits can be denoted $|x[10]\rangle$, and we can think of the system as a register of six qubits together with a register of three and another register of one qubit.

$$|x[10]\rangle = |x_1[6]\rangle \otimes |x_2[3]\rangle \otimes |x_3[1]\rangle$$

B.4 ENTANGLEMENT

Another unique property of quantum systems is that of *entanglement*: when two or more particles interact in such a way that their individual quantum states can not be described independent of the other particles. A quantum state then exists for the system as a whole instead. Mathematically, we consider such entangled states as those whose state can not be expressed as a tensor product of the states of the individual qubits it's composed of: they are dependent upon the other.

To understand what we mean by this dependence, consider a counter-example. Consider the Bell state,

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right), \tag{B.15}$$

if we measure this state, we expect that it will be observed in either eigenstate $|00\rangle$ or $|11\rangle$, with equal probability due to their amplitudes' equal magnitudes. The bases for this state are simply the standard bases, $|0\rangle$ and $|1\rangle$. Thus, according to our previous definition of systems of multiple qubits, we would say this state can be given as a combination of two states, like Eq. (B.12),

$$\begin{aligned} |\Phi^+\rangle &= |\psi_1\rangle \otimes |\psi_2\rangle \\ &= (a_1|0\rangle + b_1|1\rangle) \otimes (a_2|0\rangle + b_2|1\rangle) \\ &= a_1a_2|00\rangle + a_1b_2|01\rangle + b_1a_2|10\rangle + b_1b_2|11\rangle \end{aligned} \tag{B.16}$$

However we require $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, which would imply $a_1b_2 = 0$ and $b_1a_2 = 0$. These imply that either $a_1 = 0$ or $b_2 = 0$, and also that $b_1 = 0$ or a_2=0, which are obviously invalid since we require that $a_1a_2 = b_1b_2 = \frac{1}{\sqrt{2}}$. Thus, we cannot express $|\Phi^+\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$; this inability to describe the first and second qubits independently from each other is termed *entanglement*.

B.5 UNITARY TRANSFORMATIONS

A fundamental concept in quantum mechanics is that of performing transformations on states. *Quantum transformations*, or *quantum operators*, map a quantum state into a new state within the same Hilbert space. There are certain restrictions on a physically possible quantum transformation: in order that U is a valid transformation acting on some superposition $|\psi\rangle = a_1 |\psi_1\rangle + a_2 |\psi_2\rangle + \dots a_k|\psi_k\rangle$, U must be linear

$$U(a_1 |\psi_1\rangle + a_2 |\psi_2\rangle + \dots a_k|\psi_k\rangle) = a_1(U |\psi_1\rangle) + a_2(U |\psi_2\rangle) + \cdots + a_k(U|\psi_k\rangle). \tag{B.17}$$

To fulfil these properties, we require that U *preserve the inner product*:

$$\langle\psi_0|U^\dagger U |\psi\rangle = \langle\psi_0 |\psi\rangle .$$

That is, we require that any such transformation be *unitary*:

$$UU^\dagger = I \Rightarrow U^\dagger = U^{-1} \tag{B.18}$$

Unitarity is a sufficient condition to describe any valid quantum operation: any quantum transformation can be described by a unitary transformation, and any unitary transformation corresponds to a physically implementable quantum transformation.

Then, if U_1 is a unitary transformation that acts on the space $\mathcal{H}_1$ and U_2 acts on $\mathcal{H}_2$, the product of the two unitary transformations is also unitary. The tensor product $U_1 \otimes U_2$ acts on the space $\mathcal{H}_1 \otimes \mathcal{H}_2$. So, then, supposing a system of two separable qubits, $|\psi_1\rangle$ and $|\psi_2\rangle$ where we wish to act on $|\psi_1\rangle$ with operator U_1 and on $|\psi_2\rangle$ with U_2, we perform it as

$$(U_1 \otimes U_2)(|\psi_1\rangle \otimes |\psi_2\rangle) = (U_1 |\psi_1\rangle) \otimes (U_2 |\psi_2\rangle) . \tag{B.19}$$

B.6 DIRAC NOTATION

In keeping with standard practice, we employ *Dirac notation* throughout this thesis. Vectors are denoted by *kets* of the form $|a\rangle$. For example, the standard basis is represented by,

$$\begin{aligned} |x\rangle = |0\rangle &= \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \\ |y\rangle = |1\rangle &= \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \end{aligned} \tag{B.20}$$

We saw in Table B.1 that for every such ket, $|\psi\rangle$, there exists a *dual vector*: its complex conjugate transpose, called the *bra* of such a vector, denoted $\langle\psi|$. That is,

$$\begin{aligned} \langle\psi|^\dagger &= |\psi\rangle , \\ |\psi\rangle^\dagger &= \langle\psi| . \end{aligned} \tag{B.21}$$

$$|\psi\rangle = \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_n \end{pmatrix} \Rightarrow \langle\psi| = (\psi_1^* \quad \psi_2^* \quad \dots \quad \psi_n^*). \tag{B.22}$$

Then if we have two vectors $|\psi\rangle$ and $|\phi\rangle$, their *inner product* is given as $\langle\psi|\phi\rangle = \langle\phi|\psi\rangle$.

$$\begin{aligned} |\psi\rangle = \begin{pmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_n \end{pmatrix} \quad ; \quad |\phi\rangle = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_n \end{pmatrix} \\ \Rightarrow \langle\phi| = (\phi_1^* \quad \phi_2^* \quad \phi_3^* \quad \dots \quad \phi_n^*) \\ \Rightarrow \langle\phi|\,|\psi\rangle = (\phi_1^* \quad \phi_2^* \quad \phi_3^* \quad \dots \quad \phi_n^*) \begin{pmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_n \end{pmatrix} \\ \Rightarrow \langle\phi|\,|\psi\rangle = \phi_1^*\psi_1 + \phi_2^*\psi_2 + \phi_3^*\psi_3 + \cdots + \phi_n^*\psi_n \end{aligned} \tag{B.23}$$

Example B.6.1.

$$\begin{aligned} |\psi\rangle = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad ; \quad |\phi\rangle = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix} \\ \Rightarrow \langle\phi|\,|\psi\rangle = (4 \quad 5 \quad 6) \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \\ = (4)(1) + (5)(2) + (6)(3) = 32 \end{aligned} \tag{B.24}$$

Similarly, their *outer product* is given as $|\phi\rangle\langle\psi|$. Multiplying a column vector by a row vector thus gives a matrix. Matrices generated by a outer products then define operators:

Example B.6.2.

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} \begin{pmatrix} 3 & 4 \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 6 & 8 \end{pmatrix} \tag{B.25}$$

Then we can say, for $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

$$|0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}; \tag{B.26a}$$

$$|0\rangle\langle 1| = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}; \tag{B.26b}$$

$$|1\rangle\langle 0| = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}; \tag{B.26c}$$

$$|1\rangle\langle 1| = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}. \tag{B.26d}$$

And so any 2-dimensional linear transformation in the standard basis $|0\rangle, |1\rangle$ can be given as a sum

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = a|0\rangle\langle 0| + b|0\rangle\langle 1| + c|1\rangle\langle 0| + d|1\rangle\langle 1|. \tag{B.27}$$

This is a common method of representing operators as outer products of vectors. A transformation that *exchanges* a particle between two states, say $|0\rangle \leftrightarrow |1\rangle$ is given by the operation

$$\hat{Q} : \begin{cases} |0\rangle \to |1\rangle \\ |1\rangle \to |0\rangle \end{cases}$$

Which is equivalent to the outer product representation

$$\hat{Q} = |0\rangle\langle 1| + |1\rangle\langle 0|.$$

For clarity, here we will prove this operation

Example B.6.3.

$$\begin{aligned} \hat{Q} &= |0\rangle\langle 1| + |1\rangle\langle 0| \\ &= \begin{pmatrix} 1 \\ 0 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{aligned}$$

$$= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$$
$$= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

So then, acting on $|0\rangle$ *and* $|1\rangle$ *gives*

$$\hat{Q}|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

$$\hat{Q}|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

To demonstrate how Dirac notation simplifies this:

$$\begin{aligned} \hat{Q}|0\rangle &= (|0\rangle\langle 1| + |1\rangle\langle 0|)|0\rangle \\ &= |0\rangle\langle 1|0\rangle + |1\rangle\langle 0|0\rangle \\ &= |0\rangle\langle 1|0\rangle + |1\rangle\langle 0|0\rangle \end{aligned}$$

Then, since $|0\rangle$ and $|1\rangle$ are orthogonal basis, their inner product is 0 and the inner product of a vector with itself is 1, i.e. $\langle 1|1\rangle = \langle 0|0\rangle = 1, \langle 0|1\rangle = \langle 1|0\rangle = 0$. So,

$$\begin{aligned} \hat{Q}|0\rangle &= |0\rangle(0) + |1\rangle(1) \\ &\Rightarrow \hat{Q}|0\rangle = |1\rangle \end{aligned} \tag{B.28}$$

And similarly for $\hat{Q}|1\rangle$. This simple example then shows why Dirac notation can significantly simplify calculations across quantum mechanics, compared to standard matrix and vector notation. To see this more clearly, we will examine a simple 2-qubit state under such operations. The method generalises to operating on two or more qubits generically: we can define any operator which acts on two qubits as a sum of outer products of the basis vectors $|00\rangle, |01\rangle, |10\rangle$ and $|11\rangle$. We can similarly define any operator which acts on an n qubit state as a linear combination of the 2^n basis states generated by the n qubits.

Example B.6.4. *To define a transformation that will exchange basis vectors* $|00\rangle$ *and* $|11\rangle$*, while leaving* $|01\rangle$ *and* $|10\rangle$ *unchanged (i.e. exchanging* $|01\rangle \leftrightarrow |01\rangle, |10\rangle \leftrightarrow |10\rangle$*) we define an operator*

$$\hat{Q} = |00\rangle\langle 11| + |11\rangle\langle 00| + |10\rangle\langle 10| + |01\rangle\langle 01| \tag{B.29}$$

Then, using matrix calculations this would require separately calculating the four outer products in the above sum and adding them to find a 4×4 *matrix to represent* $\hat{Q}$*, which then acts on a state* $|\psi\rangle$*. Instead, consider first that* $|\psi\rangle = |00\rangle$*, i.e. one of the basis vectors our transformation is to change:*

$$\hat{Q}|00\rangle = (|00\rangle\langle 11| + |11\rangle\langle 00| + |10\rangle\langle 10| + |01\rangle\langle 01|)\ |00\rangle \tag{B.30}$$

And as before, only the inner products of a vector with itself remains:

$$\begin{aligned} &= |00\rangle \langle 11 |00\rangle + |11\rangle \langle 00 |00\rangle + |10\rangle\langle 10 |00\rangle + |01\rangle \langle 01 |00\rangle \\ &= |00\rangle (0) + |11\rangle (1) + |10\rangle (0) + |01\rangle (0) \\ \Rightarrow &\hat{Q} |00\rangle = |11\rangle \end{aligned} \tag{B.31}$$

i.e the transformation has performed $\hat{Q} : |00\rangle \rightarrow |11\rangle$ as expected. Then, if we apply the same transformation to a state which does not depend on one of the target states, eg,

$$\begin{aligned} |\psi\rangle =& a |10\rangle + b |01\rangle \\ \hat{Q} |\psi\rangle =& \Big(|00\rangle \langle 11| + |11\rangle \langle 00| + |10\rangle \langle 10| + |01\rangle \langle 01| \Big) \Big(a |10\rangle + b |01\rangle \Big) \\ =& a \Big(|00\rangle \langle 11| |10\rangle + |11\rangle \langle 00| |10\rangle + |10\rangle \langle 10| |10\rangle + |01\rangle \langle 01| |10\rangle \Big) \\ &+ b \Big(|00\rangle \langle 11| |01\rangle + |11\rangle \langle 00| |01\rangle + |10\rangle \langle 10| |01\rangle + |01\rangle \langle 01| |01\rangle \Big) \end{aligned} \tag{B.32}$$

And since the inner product is a scalar, we can factor terms such as $\langle 11|10\rangle$ to the beginning of expressions, eg $|00\rangle \langle 11| |10\rangle = \langle 11 |10\rangle |00\rangle$, and we also know

$$\begin{aligned} \langle 11|10\rangle = \langle 00|10\rangle = \langle 01|10\rangle = \langle 11|01\rangle = \langle 00|01\rangle = \langle 10|01\rangle = 0 \\ \langle 10|10\rangle = \langle 01|01\rangle = 1 \end{aligned} \tag{B.33}$$

We can express the above as

$$\begin{aligned} \hat{Q} |\psi\rangle = & a \Big((0) |00\rangle + (0) |11\rangle + (1) |10\rangle + (0) |01\rangle \Big) \\ & + b \Big((0) |00\rangle + (0) |11\rangle + (0) |10\rangle + (1)|01\rangle \Big) \\ & = a |10\rangle | + b|01\rangle \\ & = |\psi\rangle . \end{aligned} \tag{B.34}$$

Then it is clear that, when $|\psi\rangle$ is a superposition of states unaffected by transformation $\hat{Q}$, then $\hat{Q} |\psi\rangle = |\psi\rangle$.

This method generalises to systems with greater numbers of particles (qubits). If we briefly consider a 3 qubit system - and initialise all qubits in the standard basis state $|0\rangle$ - then the system is represented by $|000\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. This quantity is an 8-row vector. To calculate the outer product $\langle 000|000\rangle$, we would be multiplying an 8-column bra $\langle 000|$ by an 8-row ket $|000\rangle$. Clearly then we will be working with 8×8 matrices, which will become quite difficult to maintain effectively and efficiently quite fast. As we move to systems of larger size, standard matrix multiplication becomes impractical for hand-written analysis, although of course remains tractable computationally up to $n = \mathcal{O}(10)$ qubits. It is obvious that Dirac's bra/ket notation is a helpful, mathematically precise tool for QM.

C

EXAMPLE EXPLORATION STRATEGY CUSTOMISATION AND RUN

Here we provide a complete example of how to run the Quantum Model Learning Agent (QMLA) framework, including how to implement a custom exploration strategy (ES), and generate/interpret analysis. Note: these examples are included in the Quantum Model Learning Agent (QMLA) documentation in a format that may be easier to follow – where possible, we recommend readers follow the Tutorial section of [4].

First, *fork* the QMLA codebase from [3] to a Github user account (referred to as username in Listing C.6). Now, we must download the code base and ensure it runs properly; these instructions are implemented via the command line[1].

The steps of preparing the codebase are

1. install redis;
2. create a virtual Python environment for installing QMLA dependencies without damaging other parts of the user's environment;
3. download the QMLA codebase from the forked Github repository;
4. install packages upon which QMLA depends.

```
# Install redis (database broker)
sudo apt update
sudo apt install redis-server

# make directory for QMLA
cd
mkdir qmla_test
cd qmla_test

# make Python virtual environment for QMLA
# note: change Python3.6 to desired version
sudo apt-get install python3.6-venv
python3.6 -m venv qmla-env
source qmla-env/bin/activate
```

1 Note: these instructions are tested for Linux and presumed to work on Mac, but untested on Windows. It is likely some of the underlying software (redis servers) can not be installed on Windows, so running on *Windows Subsystem for Linux* is advised.

```
# Download QMLA
git clone --depth 1 https://github.com/username/QMLA.git #
    REPLACE username

# Install dependencies
cd QMLA
pip install -r requirements.txt
```

Listing C.1: QMLA codebase setup.

Note there may be a problem with some packages in the requirements.txt arising from the attempt to install them all through a single call to pip install. Ensure these are all installed before proceeding.

When all of the requirements are installed, test that the framework runs. QMLA uses redis databases to store intermittent data: we must manually initialise the database. Run the following (note: here we list redis-4.0.8, but this must be corrected to reflect the version installed on the user's machine in the above setup section):

```
~/redis-4.0.8/src/redis-server
```

Listing C.2: Launch redis database.

which should give something like Fig. C.1.

Figure C.1: Terminal running redis-server.

In a text editor, open qmla_test/QMLA/launch/local_launch.sh; here we will ensure that we are running the quantum Hamiltonian learning (QHL) algorithm, with 5 experiments and 20 particles, on the exploration strategy (ES) named TestInstall. Ensure the first few lines of local_launch.sh read:

```
#!/bin/bash

##### ------------------------------------------------ #####
# QMLA run configuration
##### ------------------------------------------------ #####
num_instances=2 # number of instances in run
run_qhl=0 # perform QHL on known (true) model
run_qhl_multi_model=0 # perform QHL for defined list of models
\glspl{experiment}=2 # number of \glspl{experiment}
particles=10 # number of particles
plot_level=5


##### ------------------------------------------------ #####
# Choose an exploration strategy
# This will determine how QMLA proceeds.
##### ------------------------------------------------ #####
exploration_strategy="TestInstall"
```

Listing C.3: local_launch script.

Ensure the terminal running redis is kept active, and open a separate terminal window. We must activate the Python virtual environment configured for QMLA, which we set up in Listing C.6. Then, we navigate to the QMLA directory, and launch:

```
# activate the QMLA Python virtual environment
source qmla_test/qmla-env/bin/activate

# move to the QMLA directory
cd qmla_test/QMLA
# Run QMLA
cd launch
./local_launch.sh
```

Listing C.4: Launch QMLA.

There may be numerous warnings, but they should not affect whether QMLA has succeeded; QMLA will raise any significant error. Assuming the run has completed successfully, QMLA stores the run's results in a subdirectory named by the date and time it was started. For example, if the run was initialised on January 1^{st} at 01:23, navigate to the corresponding directory by

```
cd results/Jan_01/01_23
```

Listing C.5: QMLA results directory.

For now it is sufficient to notice that the code has run successfully: it should have generated (in results/Jan_01/01_23) files like storage_001.p and results_001.p.

C.1 CUSTOM EXPLORATION STRATEGY

Next, we design a basic ES, for the purpose of demonstrating how to run the algorithm. ESs are placed in the directory qmla/exploration_strategies. To make a new one, navigate to the exploration strategies directory, make a new subdirectory, and copy the template file.

```
cd ~/qmla_test/QMLA/exploration_strategies/
mkdir custom_es

# Copy template file into example
cp template.py custom_es/example.py
cd custom_es
```

Listing C.6: QMLA codebase setup.

Ensure QMLA will know where to find the ES by importing everything from the custom ES directory into to the main exploration_strategy module. Then, in the custom_es directory, make a file called __init__.py which imports the new ES from the example.py file. To add any further ESs inside the directory custom_es, include them in the custom __init__.py, and they will automatically be available to QMLA.

```
# inside qmla/exploration_strategies/custom_es
#   __init__.py
from qmla.exploration_strategies.custom_es.example import *

# inside qmla/exploration_strategies, add to the existing
# __init__.py
from qmla.exploration_strategies.custom_es import *
```

Listing C.7: Providing custom exploration strategy to QMLA.

Now, change the structure (and name) of the ES inside custom_es/example.py. Say we wish to target the true model

$$\begin{aligned} \vec{\alpha} &= (\alpha_{1,2} \quad \alpha_{2,3} \quad \alpha_{3,4}) \\ \vec{T} &= \begin{pmatrix} \hat{\sigma}_z^1 \otimes \hat{\sigma}_z^2 \\ \hat{\sigma}_z^2 \otimes \hat{\sigma}_z^3 \\ \hat{\sigma}_z^3 \otimes \hat{\sigma}_z^4 \end{pmatrix} \\ \implies \hat{H}_0 &= \hat{\sigma}_z^{(1,2)} \hat{\sigma}_z^{(2,3)} \hat{\sigma}_z^{(3,4)} \end{aligned} \tag{C.1}$$

QMLA interprets models as strings, where terms are separated by +, and parameters are implicit. So the target model in Eq. (C.1) will be given by

pauliSet_1J2_zJz_d4+pauliSet_2J3_zJz_d4+pauliSet_3J4_zJz_d4.

Adapting the template ES slightly, we can define a model generation strategy with a small number of hard coded candidate models introduced at the first branch of the exploration tree. We will also set the parameters of the terms which are present in $\hat{H}_0$, as well as the range in which to search parameters. Keeping the imports at the top of the example.py, rewrite the ES as:

```
class ExampleBasic(
    exploration_strategy.ExplorationStrategy
):

    def __init__(
        self,
        exploration_rules,
        true_model=None,
        **kwargs
    ):
        self.true_model = 'pauliSet_1J2_zJz_d4+
            pauliSet_2J3_zJz_d4+pauliSet_3J4_zJz_d4'
        super().__init__(
            exploration_rules=exploration_rules,
            true_model=self.true_model,
            **kwargs
        )

        self.initial_models = None
        self.true_model_terms_params = {
            'pauliSet_1J2_zJz_d4' : 2.5,
```

```
            'pauliSet_2J3_zJz_d4' : 7.5,
            'pauliSet_3J4_zJz_d4' : 3.5,
        }
        self.tree_completed_initially = True
        self.min_param = 0
        self.max_param = 10

    def generate_models(self, **kwargs):

        self.log_print(["Generating models; spawn step {}".format
            (self.spawn_step)])
        if self.spawn_step == 0:
            # chains up to 4 sites
            new_models = [
                'pauliSet_1J2_zJz_d4',
                'pauliSet_1J2_zJz_d4+pauliSet_2J3_zJz_d4',
                'pauliSet_1J2_zJz_d4+pauliSet_2J3_zJz_d4+
                    pauliSet_3J4_zJz_d4',
            ]
            self.spawn_stage.append('Complete')

        return new_models
```

Listing C.8: ExampleBasic exploration strategy.

To run[2] the example ES for a meaningful test, return to the local_launch of Listing C.3, but change some of the settings:

```
particles=2000
\glspl{experiment}=500
run_qhl=1
exploration_strategy=ExampleBasic
```

Listing C.9: local_launch configuration for QHL.

Run locally again as in Listing C.4; then move to the results directory as in Listing C.5.

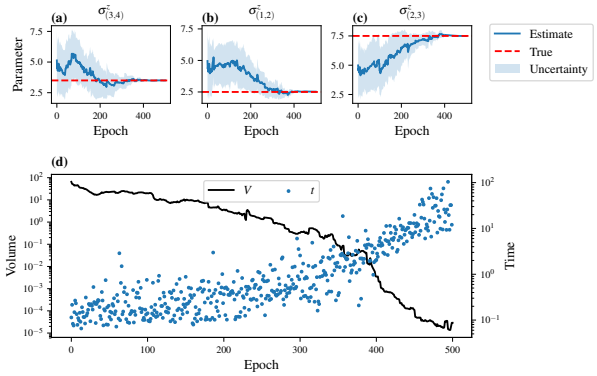
2 Note this will take up to 15 minutes to run. This can be reduced by lowering the values of particles, experiments, which is sufficient for testing but note that the outcomes will be less effective than those presented in the figures of this section.

C.2 ANALYSIS

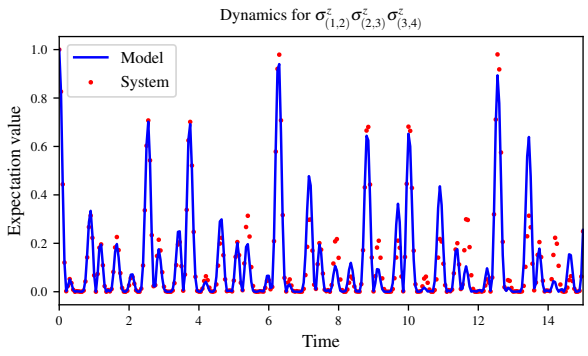
QMLA stores results and generates plots over the entire range of the algorithm[3], i.e. the run, instance and models. The depth of analysis performed automatically is set by the user control plot_level in local_launch.sh; for plot_level=1, only the most crucial figures are generated, while plot_level=6 generates plots for every individual model considered. For model searches across large model spaces and/or considering many candidates, excessive plotting can cause considerable slow-down, so users should be careful to generate plots only to the degree they will be useful. Next we show some examples of the available plots.

C.2.1 *Model analysis*

We have just run quantum Hamiltonian learning (QHL) for the model in Eq. (C.1) for a single instance, using a reasonable number of particles and experiments, so we expect to have trained the model well. Instance-level results are stored (e.g. for the instance with qmla_id=1) in Jan_01/01_23/instances/qmla_1. Individual models' insights can be found in model_training, e.g. the model's learning_summary Fig. C.2a, and dynamics in Fig. C.2b.



(a)



(b)

Figure C.2: Model analysis plots, stored in (for example) Jan_01/01_23/instances/qmla_1/model_training. **a**, learning_summary_1. Displays the outcome of QHL for the given model: Subfigures (a-c) show the estimates of the parameters; (d) shows the total parameterisation volume against experiments trained upon, along with the evolution times used for those experiments. **b**, dynamics_1 The model's attempt at reproducing dynamics from $\hat{H}_0$.

3 Recall that a single implementation of QMLA is called an instance, while a series of instances – which share the same target model – is called the run.

C.2.2 *Instance analysis*

Now we can run the full QMLA algorithm, i.e. train several models and determine the most suitable. QMLA will call the generate_models method of the ExampleBasic ES, set in Listing C.8, which tells QMLA to construct three models on the first branch, then terminate the search. Here we need to train and compare all models so it takes considerably longer to run: for the purpose of testing, we reduce the resources so the entire algorithm runs in about 15 minutes. Some applications will require significantly more resources to learn effectively. In realistic cases, these processes are run in parallel, as we will cover in Appendix C.3.

Reconfigure a subset of the settings in the local_launch.sh script (Listing C.3) and run it again:

```
\glspl{experiment}=250
particles=1000
run_qhl=0
exploration_strategy=ExampleBasic
```

Listing C.10: local_launch configuration for QMLA.

In the corresponding results directory, navigate to instances/qmla_1, where instance level analysis are available.

```
cd results/Jan_01/01_23/instances/qmla_1
```

Listing C.11: Navigating to instance results.

Figures of interest here show the composition of the models (Fig. C.3a), as well as the Bayes factors between candidates (Fig. C.3b). Individual model comparisons – i.e. Bayes factor (BF) – are shown in Fig. C.3c, with the dynamics of all candidates shown in Fig. C.4c. The probes used during the training of all candidates are also plotted (Fig. C.3e).

C.2.3 *Run analysis*

Considering a number of instances together is a *run*. In general, this is the level of analysis of most interest: an individual instance is liable to errors due to the probabilistic nature of the model training and generation subroutines. On average, however, we expect those elements to perform well, so across a significant number of instances, we expect the average outcomes to be meaningful.

Each results directory has an analyse.sh script to generate plots at the run level.

```
cd results/Jan_01/01_23
./analyse.sh
```

Listing C.12: Analysing QMLA run.

Run level analysis are held in the main results directory and several sub-directories created by the analyse script. Here, we recommend running a number of instances with very few resources so that the test finishes quickly[4]. The results will therefore be meaningless, but allow fo elucidation of the resultant plots. First, reconfigure some settings of Listing C.3 and launch again.

```
num_instances=10
\glspl{experiment}=20
particles=100
run_qhl=0
exploration_strategy=ExampleBasic
```

Listing C.13: local_launch configuration for QMLA run.

Some of the generated analysis are shown in Figs. C.4 to C.5. The number of instances for which each model was deemed champion, i.e. their *win rates* are given in Fig. C.4a. The *top models*, i.e. those with highest win rates, analysed further: the average parameter estimation progression for $\hat{H}_0$ – including only the instances where $\hat{H}_0$ was deemed champion – are shown in Fig. C.4b. Irrespecitve of the champion models, the rate with which each term is found in the champion model ($\hat{t} \in \hat{H}'$) indicates the likelihood that the term is really present; these rates – along with the parameter values learned – are shown in Fig. C.4c. The champion model from each instance can attempt to reproduce system dynamics: we group together these reproductions for each model in Fig. C.5.

C.3 PARALLEL IMPLEMENTATION

We provide utility to run QMLA on parallel processes. Individual models' training can run in parallel, as well as the calculation of BF between models. The provided script is designed for portable batch system (PBS) job scheduler running on a compute cluster. It will require a few adjustments to match the system being used. Overall, though, it has mostly a similar structure as the local_launch.sh script used above.

QMLA must be downloaded on the compute cluster as in Listing C.6; this can be a new fork of the repository, though it is sensible to test installation locally as described in this chapter so far, then *push* that version, including the new ES, to Github, and cloning the latest version. It is again advisable to create a Python virtual environment in order to isolate QMLA and its

4 This run will take about ten minutes

dependencies[5]. Open the parallel launch script, QMLA/launch/parallel_launch.sh, and prepare the first few lines as

```
#!/bin/bash

##### ------------------------------------------------ #####
# QMLA run configuration
##### ------------------------------------------------ #####
num_instances=10 # number of \glspl{instance} in run
run_qhl=0 # perform QHL on known (true) model
run_qhl_multi_model=0 # perform QHL for defined list of models
\glspl{experiment}=250
particles=1000
plot_level=5


##### ------------------------------------------------ #####
# Choose an exploration strategy
# This will determine how QMLA proceeds.
##### ------------------------------------------------ #####
exploration_strategy="ExampleBasic"
```

Listing C.14: parallel_launch script.

When submitting jobs to schedulers like PBS, we must specify the time required, so that it can determine a fair distribution of resources among users. We must therefore *estimate* the time it will take for an instance to complete: clearly this is strongly dependent on the numbers of experiments (N_E) and particles (N_P), and the number of models which must be trained. QMLA attempts to determine a reasonable time to request based on the max_num_models_by_shape attribute of the ES, by calling QMLA/scripts/time_required_calculation.py. In practice, this can be difficult to set perfectly, so the timing_insurance_factor attribute of the ES can be used to correct for heavily over- or under-estimated time requests. Instances are run in parallel, and each instance trains/compares models in parallel. The number of processes to request, N_c for each instance is set as num_processes_to_parallelise_over in the ES. Then, if there are N_r instances in the run, we will be requesting the job scheduler to admit N_r distinct jobs, each requiring N_c processes, for the time specified.

The parallel_launch script works together with launch/run_single_qmla_instance.sh, though note a number of steps in the latter are configured to the cluster and may need to be adapted. In particular, the first command is used to load the redis utility, and later lines are used to initialise

5 Indeed it is sensible to do this for any Python development project.

a redis server. These commands will probably not work with most machines, so must be configured to achieve those steps.

```
module load tools/redis-4.0.8

...

SERVER_HOST=$(head -1 "$PBS_NODEFILE")
let REDIS_PORT="6300 + $QMLA_ID"

cd $LIBRARY_DIR
redis-server RedisDatabaseConfig.conf --protected-mode no --port
    $REDIS_PORT &
redis-cli -p $REDIS_PORT flushall
```

Listing C.15: run_single_qmla_instance script.

When the modifications are finished, QMLA can be launched in parallel similarly to the local version:

```
source qmla_test/qmla-env/bin/activate

cd qmla_test/QMLA/launch
./parallel_launch.sh
```

Listing C.16: run_single_qmla_instance script.

Jobs are likely to queue for some time, depending on the demands on the job scheduler. When all jobs have finished, results are stored as in the local case, in QMLA/launch/results/-Jan_01/01_23, where analyse.sh can be used to generate a series of automatic analyses.

C.4 CUSTOMISING EXPLORATION STRATEGIES

User interaction with the QMLA codebase should be achieveable primarily through the exploration strategy (ES) framework. Throughout the algorithm(s) available, QMLA calls upon the ES before determining how to proceed. The usual mechanism through which the actions of QMLA are directed, is to set attributes of the ES class: the complete set of influential attributes are available at [4].

QMLA directly uses several methods of the ES class, all of which can be overwritten in the course of customising an ES. Most such methods need not be replaced, however, with the exception of generate_models, which is the most important aspect of any ES: it determines which

models are built and tested by QMLA. This method allows the user to impose any logic desired in constructing models; it is called after the completion of every branch of the exploration tree on the ES.

C.4.1 *Greedy search*

A first non-trivial ES is to build models greedily from a set of *primitive* terms, $\mathcal{T} = \{\hat{t}\}$. New models are constructed by combining the previous branch champion with each of the remaining, unused terms. The process is repeated until no terms remain.

We can compose an ES using these rules, say for

$$\mathcal{T} = \left\{\hat{\sigma}_x^1,\ \hat{\sigma}_y^1,\ \hat{\sigma}_x^1 \otimes \hat{\sigma}_x^2,\ \hat{\sigma}_y^1 \otimes \hat{\sigma}_y^2\right\}$$

as follows. Note the termination criteria must work in conjunction with the model generation routine. Users can overwrite the method check_tree_completed for custom logic, although a straightforward mechanism is to use the spawn_stage attribute of the ES class: when the final element of this list is Complete, QMLA will terminate the search by default. Also note that the default termination test checks whether the number of branches (spawn_step) exceeds the limit max_spawn_depth, which must be set artifically high to avoid ceasing the search too early, if relying solely on spawn_stage. Here we demonstrate how to impose custom logic to terminate the seach also.

```
class ExampleGreedySearch(
    exploration_strategy.ExplorationStrategy
):
    r"""
    From a fixed set of terms, construct models iteratively,
    greedily adding all unused terms to separate models at each
       call to the generate_models.

    """

    def __init__(
        self,
        exploration_rules,
        **kwargs
    ):

        super().__init__(
            exploration_rules=exploration_rules,
            **kwargs
```

```
        )
        self.true_model = 'pauliSet_1_x_d3+pauliSet_1J2_yJy_d3+
            pauliSet_1J2J3_zJzJz_d3'
        self.initial_models = None
        self.available_terms = [
            'pauliSet_1_x_d3', 'pauliSet_1_y_d3',
            'pauliSet_1J2_xJx_d3', 'pauliSet_1J2_yJy_d3'
        ]
        self.branch_champions = []
        self.prune_completed_initially = True
        self.check_champion_reducibility = False

    def generate_models(
        self,
        model_list,
        **kwargs
    ):
        self.log_print([
            "Generating models in tiered greedy search at spawn
                step {}.".format(
                  self.spawn_step,
            )
        ])
        try:
            previous_branch_champ = model_list[0]
            self.branch_champions.append(previous_branch_champ)
        except:
            previous_branch_champ = ""

        if self.spawn_step == 0 :
            new_models = self.available_terms
        else:
            new_models = greedy_add(
                current_model = previous_branch_champ,
                terms = self.available_terms
            )

        if len(new_models) == 0:
            # Greedy search has exhausted the available models;
```

```
                # send back the list of branch champions and
                    terminate search.
                new_models = self.branch_champions
                self.spawn_stage.append('Complete')

            return new_models

    def greedy_add(
        current_model,
        terms,
    ):
        r"""
        Combines given model with all terms from a set.

        Determines which terms are not yet present in the model,
        and adds them each separately to the current model.

        :param str current_model: base model
        :param list terms: list of strings of terms which are to be
            added greedily.
        """

        try:
            present_terms = current_model.split('+')
        except:
            present_terms = []
        nonpresent_terms = list(set(terms) - set(present_terms))

        term_sets = [
            present_terms+[t] for t in nonpresent_terms
        ]

        new_models = ["+".join(term_set) for term_set in term_sets]

        return new_models
```

Listing C.17: ExampleGreedySearch exploration stategy.

This run can be implemented locally or in parallel as described above[6], and analysed as in Listing C.12, generating figures in accordance with the plot_level set by the user in the launch

script. Outputs can again be found in the instances subdirectory, including a map of the models generated, as well as the branches they reside on, and the BFs between candidates, Fig. C.7.

C.4.2 *Tiered greedy search*

We provide one final example of a non-trivial ES: tiered greedy search. Similar to the idea of Appendix C.4.1, except terms are introduced hierarchically: sets of terms $\mathcal{T}_1, \mathcal{T}_2, \ldots \mathcal{T}_n$ are each examined greedily, where the overall strongest model of one tier forms the seed model for the subsequent tier. This is depicted in the main text in Fig. 9.4. A corresponding ES is given as follows.

```
class ExampleGreedySearchTiered(
    exploration_strategy.ExplorationStrategy
):
    r"""
    Greedy search in tiers.

    Terms are batched together in tiers;
    tiers are searched greedily;
    a single tier champion is elevated to the subsequent tier.

    """

    def __init__(
        self,
        exploration_rules,
        **kwargs
    ):
        super().__init__(
            exploration_rules=exploration_rules,
            **kwargs
        )
        self.true_model = 'pauliSet_1_x_d3+pauliSet_1J2_yJy_d3+
           pauliSet_1J2J3_zJzJz_d3'
        self.initial_models = None
        self.term_tiers = {
            1 : ['pauliSet_1_x_d3', 'pauliSet_1_y_d3', '
               pauliSet_1_z_d3' ],
```

6 We advise reducing plot_level to 3 to avoid excessive/slow figure generation.

```
            2 : ['pauliSet_1J2_xJx_d3', 'pauliSet_1J2_yJy_d3', '
                pauliSet_1J2_zJz_d3'],
            3 : ['pauliSet_1J2J3_xJxJx_d3', '
                pauliSet_1J2J3_yJyJy_d3', 'pauliSet_1J2J3_zJzJz_d3
                '],
        }
        self.tier = 1
        self.max_tier = max(self.term_tiers)
        self.tier_branch_champs = {k : [] for k in self.
            term_tiers}
        self.tier_champs = {}
        self.prune_completed_initially = True
        self.check_champion_reducibility = True

    def generate_models(
        self,
        model_list,
        **kwargs
    ):
        self.log_print([
            "Generating models in tiered greedy search at spawn
                step {}.".format(
                self.spawn_step,
            )
        ])

        if self.spawn_stage[-1] is None:
            try:
                previous_branch_champ = model_list[0]
                self.tier_branch_champs[self.tier].append(
                    previous_branch_champ)
            except:
                previous_branch_champ = None

        elif "getting_tier_champ" in self.spawn_stage[-1]:
            previous_branch_champ = model_list[0]
            self.log_print([
                "Tier champ for {} is {}".format(self.tier,
                    model_list[0])
            ])
```

```
            self.tier_champs[self.tier] = model_list[0]
            self.tier += 1
            self.log_print(["Tier now = ", self.tier])
            self.spawn_stage.append(None) # normal processing

            if self.tier > self.max_tier:
                self.log_print(["Completed tree for ES"])
                self.spawn_stage.append('Complete')
                return list(self.tier_champs.values())
        else:
            self.log_print([
                "Spawn stage:", self.spawn_stage
            ])

        new_models = greedy_add(
            current_model = previous_branch_champ,
            terms = self.term_tiers[self.tier]
        )
        self.log_print([
            "tiered search new_models=", new_models
        ])

        if len(new_models) == 0:
            # no models left to find - get champions of branches
              from this tier
            new_models = self.tier_branch_champs[self.tier]
            self.log_print([
                "tier champions: {}".format(new_models)
            ])
            self.spawn_stage.append("getting_tier_champ_{}".
              format(self.tier))
        return new_models

    def check_tree_completed(
        self,
        spawn_step,
        **kwargs
    ):
        r"""
```

```
        QMLA asks the exploration tree whether it has finished
            growing;
        the exploration tree queries the exploration strategy
            through this method
        """
        if self.tree_completed_initially:
            return True
        elif self.spawn_stage[-1] == "Complete":
            return True
        else:
            return False


def greedy_add(
    current_model,
    terms,
):
    r"""
    Combines given model with all terms from a set.

    Determines which terms are not yet present in the model,
    and adds them each separately to the current model.

    :param str current_model: base model
    :param list terms: list of strings of terms which are to be
        added greedily.
    """


    try:
        present_terms = current_model.split('+')
    except:
        present_terms = []
    nonpresent_terms = list(set(terms) - set(present_terms))

    term_sets = [
        present_terms+[t] for t in nonpresent_terms
    ]

    new_models = ["+".join(term_set) for term_set in term_sets]
```

```
        return new_models
```

Listing C.18: ExampleGreedySearchTiered exploration stategy.

with corresponding results in Fig. C.8.

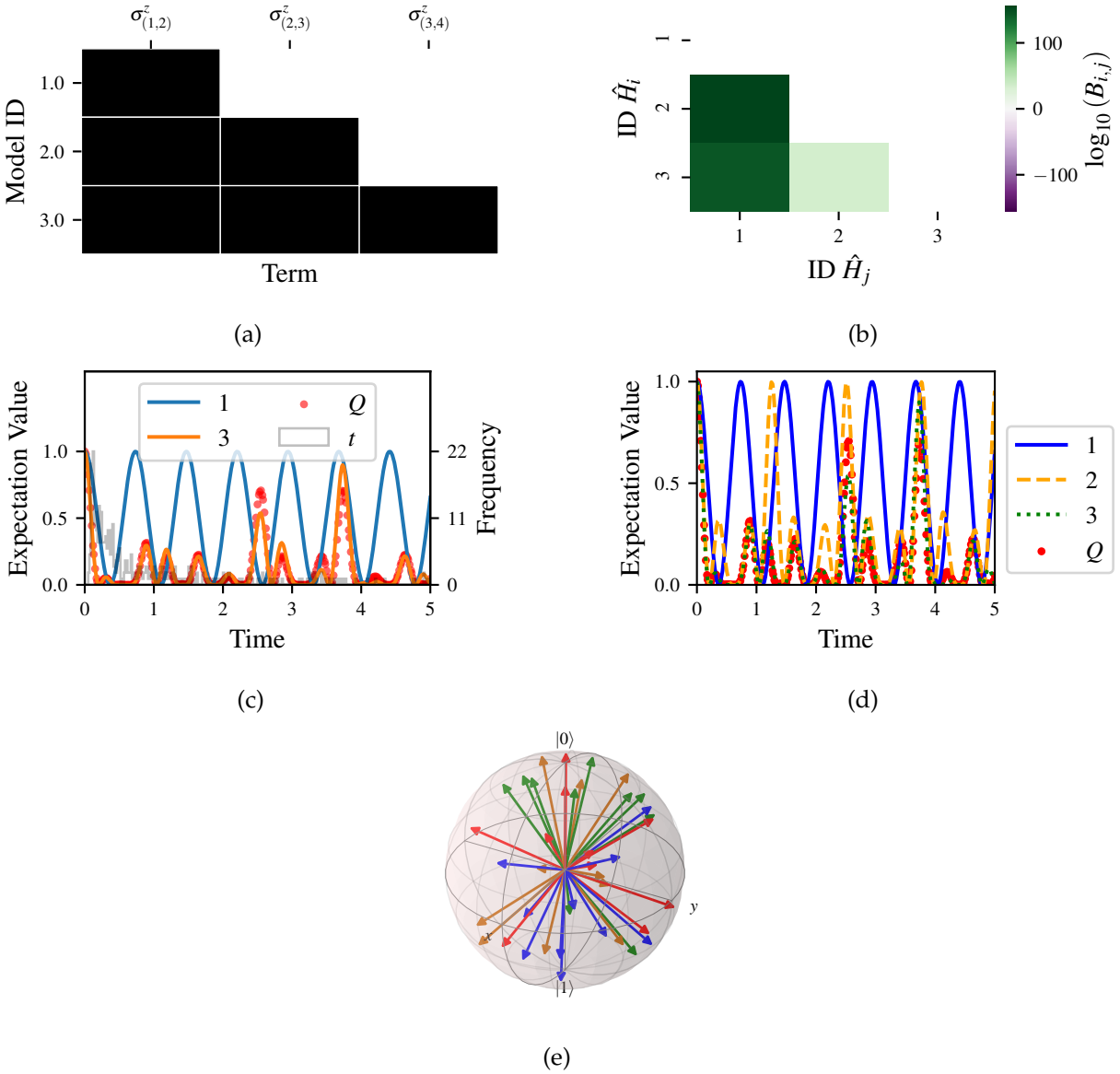


Figure C.3: QMLA plots; found within instance directory e.g. Jan_01/01_23/instances/qmla_1, and its subdirectories. **a** composition_of_models: constituent terms of all considered models, indexed by their model IDs. Here model 3 is $\hat{H}_0$. **b** bayes_factors: Bayes factor (BF) comparisons between all models. Bayes factors (BFs) are read as $B_{i,j}$ where i is the model with lower ID, e.g. $B_{1,2}$ rather than $B_{2,1}$. Thus $B_{ij} > 0$ (< 0) indicates $\hat{H}_i$ ($\hat{H}_j$), i.e. the model on the y-axis (x-axis) is the stronger model. **c** comparisons/BF_1_3: direct comparison between models with IDs 1 and 3, showing their reproduction of the system dynamics (red dots, Q), as well as the times (experiments) against which the BF was calculated. **d** branches/dynamics_branch_1: dynamics of all models considered on the branch compared with system dynamics (red dots, Q). **e** probes_bloch_sphere: probes used for training models in this instance (only showing 1-qubit versions).

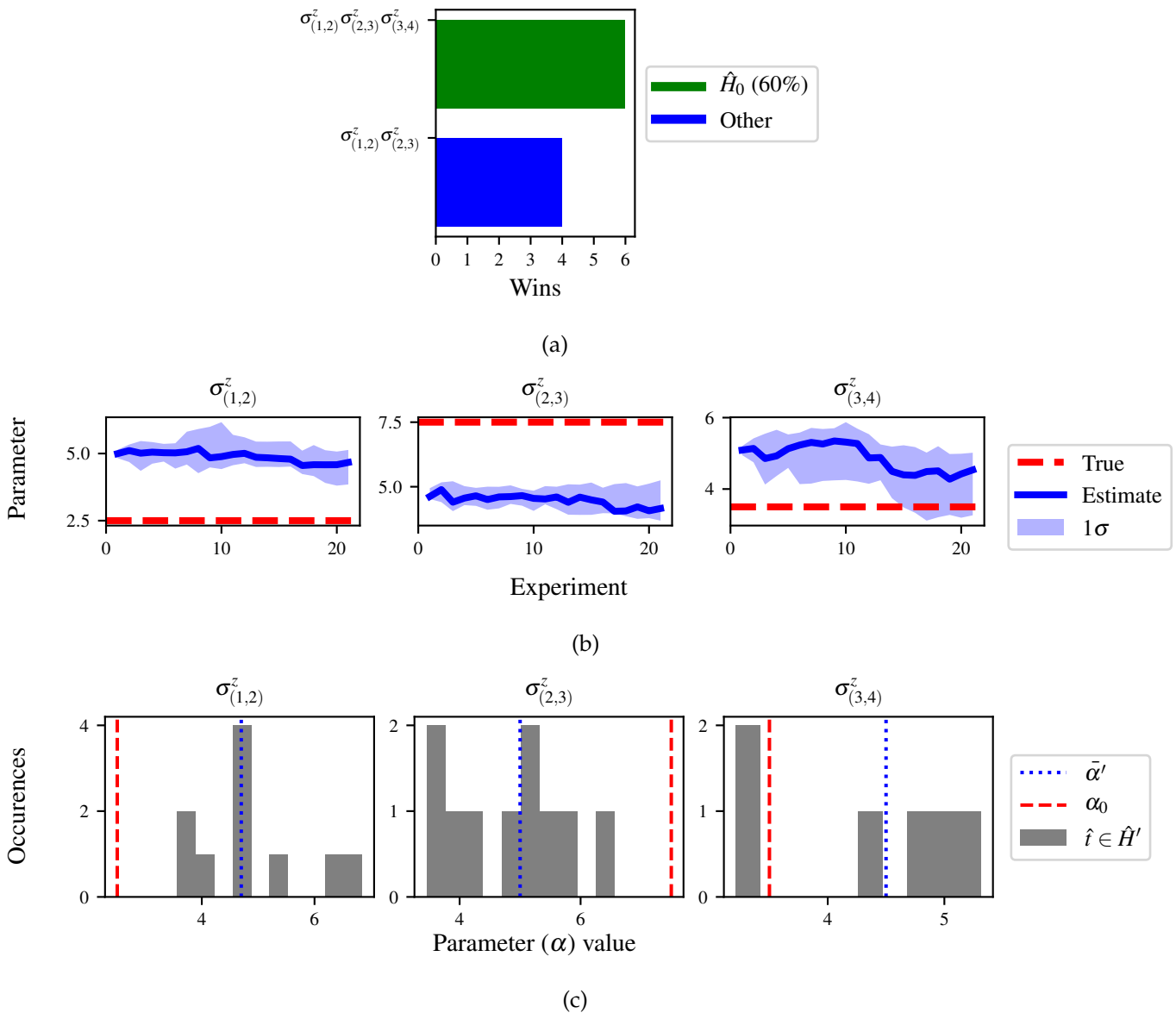


Figure C.4: QMLA run plots; found within run directory e.g. Jan_01/01_23/. **a** performace/model_wins: number of instance wins achieved by each model. **b** champion_models/params_params_pauliSet_1J2_zJz_d4+pauliSet_2J3_zJz_d4+pauliSet_3J4_zJz_d4: parameter estimation progression for the true model , only for the instances where it was deemed champion. **c** champion_models/terms_and_params: histogram of parameter values found for each term which appears in any champion model, with the true parameter (α_0) in red and the median learned parameter ($\bar{\alpha}'$) in blue.

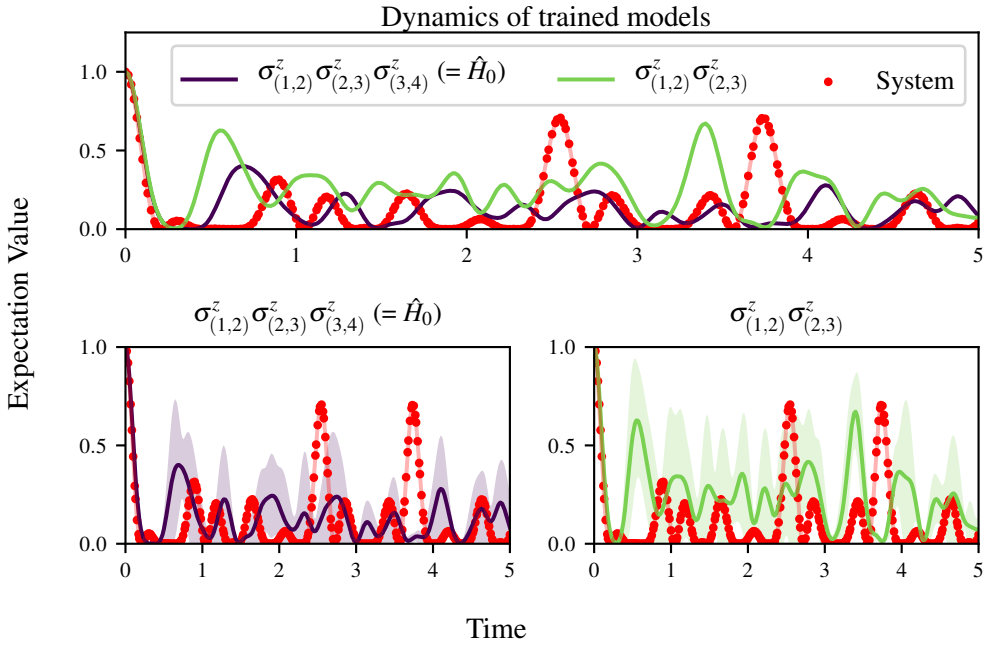


Figure C.5: Run plot performace/dynamics: median dynamics of the champion models. The models which won most instances are shown together in the top panel, and individually in the lower panels. The median dynamics from the models' learnings in its winning instances are shown, with the shaded region indicating the 66% confidence region.

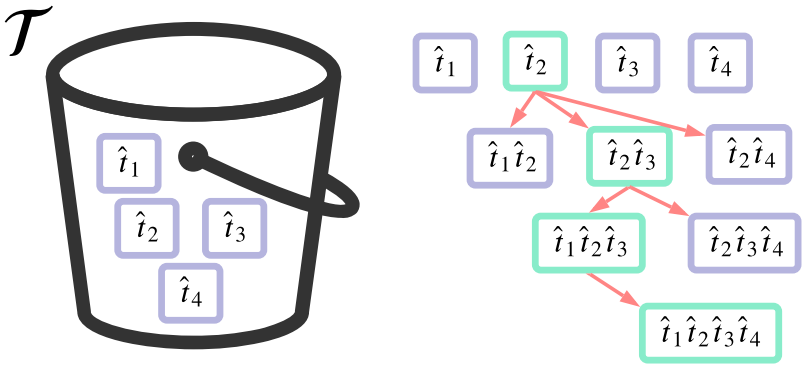
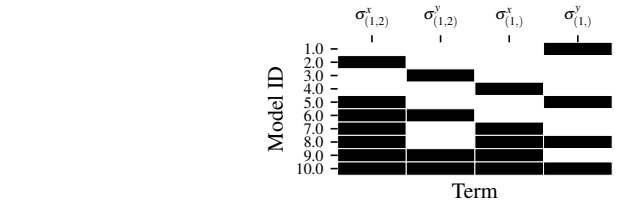
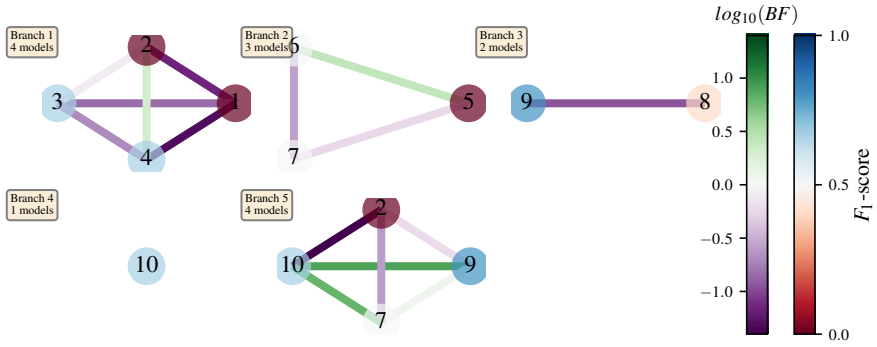


Figure C.6: Greedy search mechanism. **Left**, a set of primitive terms, $\mathcal{T}$, are defined in advance. **Right**, models are constructed from $\mathcal{T}$. On the first branch, the primitve terms alone constitute models. Thereafter, the strongest model (marked in green) from the previous branch is combined with all the unused terms.

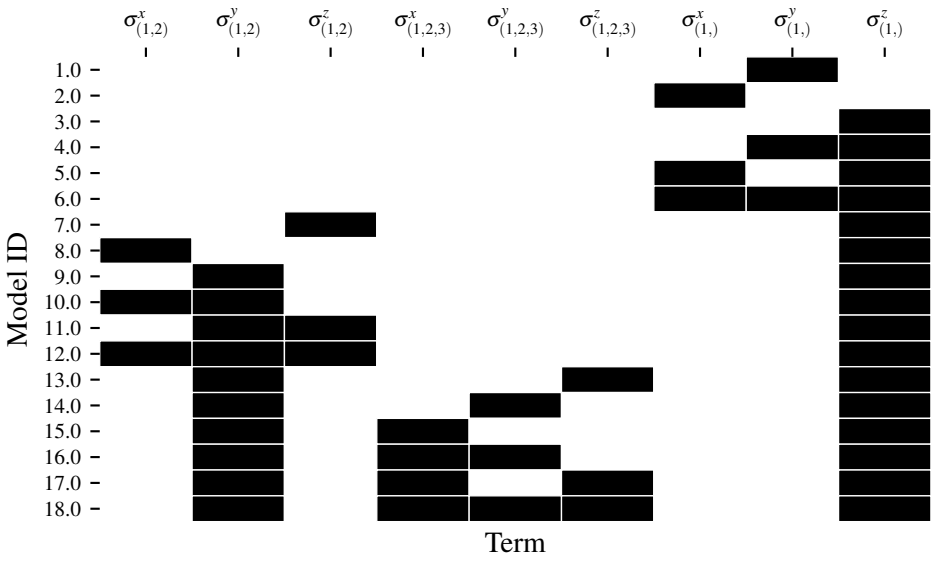


(a)

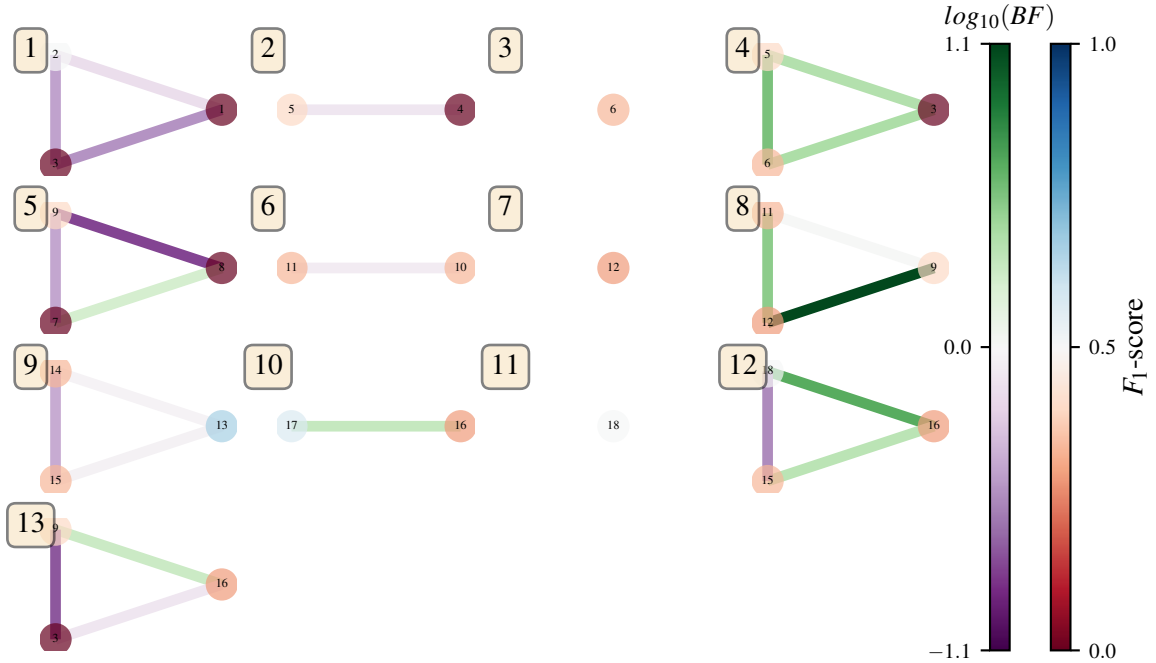


(b)

Figure C.7: Greedy exploration strategy. **a**, composition_of_models. **b**, graphs_of_branches_ExampleGreedySearch: shows which models reside on each branches of the exploration tree. Models are coloured by their F_1-score, and edges represent the BF between models. The first four branches are equivalent to those in Fig. C.6 , while the final branch considers the set of branch champions, in order to determine the overall champion.



(a)



(b)

Figure C.8: Tiered greedy exploration strategy. **a**, composition_of_models. **b**, graphs_of_branches_ExampleGreedySearchTiered: shows which models reside on each branches of the exploration tree. Models are coloured by their F_1-score, and edges represent the BF between models. In each tier, three branches greedily add terms, and a fourth branch considers the champions of the first three branches in order to nominate a tier champion. The final branch consists only of the tier champions, to nominate the global champion, $\hat{H}'$.

BIBLIOGRAPHY

[1] Antonio A Gentile, Brian Flynn, Sebastian Knauer, Nathan Wiebe, Stefano Paesani, Christopher E Granade, John G Rarity, Raffaele Santagati, and Anthony Laing. Learning models of quantum systems from experiments. *Nature Physics*, pages 1–7, 2021.

[2] Brian Flynn, Antonio A. Gentile, Raffaele Santagati, Nathan Wiebe, and Anthony Laing. Quantum model learning agent: quantum systems' characterisation through machine learning. In preparation, 2021.

[3] Brian Flynn. Codebase: Quantum model learning agent. https://github.com/flynnbr11/QMLA, 2021.

[4] Quantum model learning agent documentation. https://quantum-model-learning-agent.readthedocs.io/en/latest/, Jan 2021. [Online; accessed 12. Jan. 2021].

[5] Max Jammer et al. *The conceptual development of quantum mechanics*. McGraw-Hill New York, 1966.

[6] Albert Einstein. On a heuristic point of view concerning the production and transformation of light. *Annalen der Physik*, pages 1–18, 1905.

[7] Max Born. Quantenmechanik der stoßvorgänge. *Zeitschrift für Physik*, 38(11-12):803–827, 1926.

[8] Erwin Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Physical review*, 28(6):1049, 1926.

[9] Werner Heisenberg. Über quantentheoretische umdeutung kinematischer und mechanischer beziehungen. In *Original Scientific Papers Wissenschaftliche Originalarbeiten*, pages 382–396. Springer, 1985.

[10] John Von Neumann. *Mathematical foundations of quantum mechanics: New edition*. Princeton university press, 2018.

[11] Orazio Svelto and David C Hanna. *Principles of lasers*, volume 1. Springer, 2010.

[12] Bart Van Zeghbroeck. Principles of semiconductor devices. *Colarado University*, 34, 2004.

[13] Richard P Feynman. Simulating physics with computers. *Int. J. Theor. Phys*, 21(6/7), 1982.

[14] Seth Lloyd. Universal quantum simulators. *Science*, pages 1073–1078, 1996.

[15] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[16] Han-Sen Zhong, Hui Wang, Yu-Hao Deng, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Jian Qin, Dian Wu, Xing Ding, Yi Hu, et al. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020.

[17] Lov K Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical review letters*, 79(2):325, 1997.

[18] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *Chemical reviews*, 119(19):10856–10915, 2019.

[19] Yudong Cao, Jhonathan Romero, and Alán Aspuru-Guzik. Potential of quantum computing for drug discovery. *IBM Journal of Research and Development*, 62(6):6–1, 2018.

[20] Andrew M Childs, Dmitri Maslov, Yunseong Nam, Neil J Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, 2018.

[21] Jonas B Rigo and Andrew K Mitchell. Machine learning effective models for quantum systems. *Physical Review B*, 101(24):241105, 2020.

[22] Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems*, 33, 2020.

[23] Eyal Bairey, Itai Arad, and Netanel H Lindner. Learning a local hamiltonian from local measurements. *Physical review letters*, 122(2):020504, 2019.

[24] Chris J Pickard and RJ Needs. Ab initio random structure searching. *Journal of Physics: Condensed Matter*, 23(5):053201, 2011.

[25] Eli Chertkov and Bryan K Clark. Computational inverse method for constructing spaces of quantum models from wave functions. *Physical Review X*, 8(3):031029, 2018.

[26] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[27] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[28] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[29] Supercomputing top500. https://www.top500.org/statistics/perfdevel/, Jan 2021. [Online; accessed 13. Mar. 2021].

[30] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, 28(2):39–55, 2008.

[31] Hongxiang Chen, Michael Vasmer, Nikolas P Breuckmann, and Edward Grant. Machine learning logical gates for quantum error correction. *arXiv preprint arXiv:1912.10063*, 2019.

[32] Agnes Valenti, Evert van Nieuwenburg, Sebastian Huber, and Eliska Greplova. Hamiltonian learning for quantum error correction. *Physical Review Research*, 1(3):033092, 2019.

[33] Alexander Hentschel and Barry C Sanders. Machine learning for precise quantum measurement. *Physical review letters*, 104(6):063603, 2010.

[34] DT Lennon, H Moon, LC Camenzind, Liuqi Yu, DM Zumbühl, GAD Briggs, MA Osborne, EA Laird, and N Ares. Efficiently measuring a quantum device using machine learning. *npj Quantum Information*, 5(1):1–8, 2019.

[35] David J Griffiths and Darrell F Schroeter. *Introduction to quantum mechanics*. Cambridge University Press, 2018.

[36] Leonard Susskind and Art Friedman. *Quantum mechanics: the theoretical minimum*. Basic Books, 2014.

[37] Eleanor G Rieffel and Wolfgang H Polak. *Quantum computing: A gentle introduction*. MIT Press, 2011.

[38] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[39] Paul Adrien Maurice Dirac. *The principles of quantum mechanics*. Number 27. Oxford university press, 1981.

[40] T Mart. How do i introduce schrodinger equation during the quantum mechanics course? *arXiv preprint arXiv:2010.15589*, 2020.

[41] Edward Nelson. Derivation of the schrödinger equation from newtonian mechanics. *Physical review*, 150(4):1079, 1966.

[42] Leonard Susskind and George Hrabovsky. *Classical mechanics: the theoretical minimum*. Penguin Books, 2014.

[43] Heinz-Peter Breuer, Francesco Petruccione, et al. *The theory of open quantum systems*. Oxford University Press on Demand, 2002.

[44] Daniel Manzano. A short introduction to the lindblad master equation. *AIP Advances*, 10(2):025106, 2020.

[45] John Preskill. Lecture notes for physics 229: Quantum information and computation. *California Institute of Technology*, 16, 1998.

[46] Jonathan P Dowling and Gerard J Milburn. Quantum technology: the second quantum revolution. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 361(1809):1655–1674, 2003.

[47] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum-enhanced measurements: beating the standard quantum limit. *Science*, 306(5700):1330–1336, 2004.

[48] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Advances in quantum metrology. *Nature photonics*, 5(4):222, 2011.

[49] Charles H Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *arXiv preprint arXiv:2003.06557*, 2020.

[50] Artur K Ekert. Quantum cryptography based on bell's theorem. *Physical review letters*, 67(6):661, 1991.

[51] Nicolas Gisin, Grégoire Ribordy, Wolfgang Tittel, and Hugo Zbinden. Quantum cryptography. *Reviews of modern physics*, 74(1):145, 2002.

[52] Yu I Manin. Vychislimoe i nevychislimoe (computable and noncomputable), moscow: Sov, 1980.

[53] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of statistical physics*, 22(5):563–591, 1980.

[54] Paul Benioff. Quantum mechanical hamiltonian models of turing machines. *Journal of Statistical Physics*, 29(3):515–546, 1982.

[55] J Ignacio Cirac and Peter Zoller. Goals and opportunities in quantum simulation. *Nature Physics*, 8(4):264–266, 2012.

[56] Benjamin P Lanyon, James D Whitfield, Geoff G Gillett, Michael E Goggin, Marcelo P Almeida, Ivan Kassal, Jacob D Biamonte, Masoud Mohseni, Ben J Powell, Marco Barbieri, et al. Towards quantum chemistry on a quantum computer. *Nature chemistry*, 2(2):106–111, 2010.

[57] Sam McArdle, Suguru Endo, Alan Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 92(1):015003, 2020.

[58] Alán Aspuru-Guzik, Anthony D Dutoi, Peter J Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704–1707, 2005.

[59] Chris Sparrow, Enrique Martín-López, Nicola Maraviglia, Alex Neville, Christopher Harrold, Jacques Carolan, Yogesh N Joglekar, Toshikazu Hashimoto, Nobuyuki Matsuda, Jeremy L O'Brien, et al. Simulating the vibrational quantum dynamics of molecules using photonics. *Nature*, 557(7707):660–667, 2018.

[60] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[61] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[62] Charles H Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM journal on Computing*, 26(5):1510–1523, 1997.

[63] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997.

[64] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[65] John Watrous. Quantum computational complexity. *arXiv preprint arXiv:0804.3401*, 2008.

[66] Peter W Shor. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 56–65. IEEE, 1996.

[67] Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error rate. *SIAM Journal on Computing*, 2008.

[68] David P DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik: Progress of Physics*, 48(9-11):771–783, 2000.

[69] Travis S Humble, Himanshu Thapliyal, Edgard Munoz-Coreas, Fahd A Mohiyaddin, and Ryan S Bennink. Quantum computing circuits and devices. *IEEE Design & Test*, 36(3):69–94, 2019.

[70] Pieter Kok, William J Munro, Kae Nemoto, Timothy C Ralph, Jonathan P Dowling, and Gerard J Milburn. Linear optical quantum computing with photonic qubits. *Reviews of Modern Physics*, 79(1):135, 2007.

[71] Jeremy C Adcock, Jueming Bao, Yulin Chi, Xiaojiong Chen, Davide Bacco, Qihuang Gong, Leif K Oxenløwe, Jianwei Wang, and Yunhong Ding. Advances in silicon quantum photonics. *IEEE Journal of Selected Topics in Quantum Electronics*, 27(2):1–24, 2020.

[72] John Michael Kovac, EM Leitch, C Pryke, JE Carlstrom, NW Halverson, and WL Holzapfel. Detection of polarization in the cosmic microwave background using dasi. *Nature*, 420(6917):772–787, 2002.

[73] Christopher Gerry, Peter Knight, and Peter L Knight. *Introductory quantum optics*. Cambridge university press, 2005.

[74] Robert Raussendorf, Daniel E Browne, and Hans J Briegel. Measurement-based quantum computation on cluster states. *Physical review A*, 68(2):022312, 2003.

[75] Caterina Vigliar, Stefano Paesani, Yunhong Ding, Jeremy C Adcock, Jianwei Wang, Sam Morley-Short, Davide Bacco, Leif K Oxenløwe, Mark G Thompson, John G Rarity, et al. Error protected qubits in a silicon photonic chip. *arXiv preprint arXiv:2009.08339*, 2020.

[76] Stefano Paesani, Massimo Borghi, Stefano Signorini, Alexandre Maïnos, Lorenzo Pavesi, and Anthony Laing. Near-ideal spontaneous photon sources in silicon quantum photonics. *Nature communications*, 11(1):1–6, 2020.

[77] Fumihiro Kaneda and Paul G Kwiat. High-efficiency single-photon generation via large-scale active time multiplexing. *Science advances*, 5(10):eaaw8586, 2019.

[78] Michel H Devoret, Andreas Wallraff, and John M Martinis. Superconducting qubits: A short review. *arXiv preprint cond-mat/0411174*, 2004.

[79] Morten Kjaergaard, Mollie E Schwartz, Jochen Braumüller, Philip Krantz, Joel I-J Wang, Simon Gustavsson, and William D Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11:369–395, 2020.

[80] M Kjaergaard, ME Schwartz, A Greene, GO Samach, A Bengtsson, M O'Keeffe, CM McNally, J Braumüller, DK Kim, P Krantz, et al. A quantum instruction set implemented on a superconducting quantum processor. *arXiv preprint arXiv:2001.08838*, 2020.

[81] John M Martinis and A Megrant. Ucsb final report for the csq program: Review of decoherence and materials physics for superconducting qubits. *arXiv preprint arXiv:1410.5793*, 2014.

[82] Göran Wendin. Quantum information processing with superconducting circuits: a review. *Reports on Progress in Physics*, 80(10):20, 2017.

[83] Ioan M Pop, Kurtis Geerlings, Gianluigi Catelani, Robert J Schoelkopf, Leonid I Glazman, and Michel H Devoret. Coherent suppression of electromagnetic dissipation due to superconducting quasiparticles. *Nature*, 508(7496):369–372, 2014.

[84] Jay M Gambetta, Jerry M Chow, and Matthias Steffen. Building logical qubits in a superconducting quantum computing system. *npj Quantum Information*, 3(1):1–7, 2017.

[85] David Kielpinski, Chris Monroe, and David J Wineland. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417(6890):709–711, 2002.

[86] Christopher Monroe and Jungsang Kim. Scaling the ion trap quantum processor. *Science*, 339(6124):1164–1169, 2013.

[87] Norbert M Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

[88] John P Gaebler, Ting Rei Tan, Y Lin, Y Wan, R Bowler, Adam C Keith, S Glancy, K Coakley, E Knill, D Leibfried, et al. High-fidelity universal gate set for be 9+ ion qubits. *Physical review letters*, 117(6):060505, 2016.

[89] Ye Wang, Mark Um, Junhua Zhang, Shuoming An, Ming Lyu, Jing-Ning Zhang, L-M Duan, Dahyun Yum, and Kihwan Kim. Single-qubit quantum memory exceeding ten-minute coherence time. *Nature Photonics*, 11(10):646–650, 2017.

[90] AH Myerson, DJ Szwer, SC Webster, DTC Allcock, MJ Curtis, G Imreh, JA Sherman, DN Stacey, AM Steane, and DM Lucas. High-fidelity readout of trapped-ion qubits. *Physical Review Letters*, 100(20):200502, 2008.

[91] VM Schäfer, CJ Ballance, K Thirumalai, LJ Stephenson, TG Ballance, AM Steane, and DM Lucas. Fast quantum logic gates with trapped-ion qubits. *Nature*, 555(7694):75–78, 2018.

[92] Colin D Bruzewicz, John Chiaverini, Robert McConnell, and Jeremy M Sage. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2):021314, 2019.

[93] Ryan LaRose. Overview and comparison of gate level quantum software platforms. *Quantum*, 3:130, 2019.

[94] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[95] Aram W Harrow and Ashley Montanaro. Quantum computational supremacy. *Nature*, 549(7671):203–209, 2017.

[96] David Poulin, Angie Qarry, Rolando Somma, and Frank Verstraete. Quantum simulation of time-dependent hamiltonians and the convenient illusion of hilbert space. *Physical review letters*, 106(17):170501, 2011.

[97] Dominic W Berry, Andrew M Childs, Richard Cleve, Robin Kothari, and Rolando D Somma. Simulating hamiltonian dynamics with a truncated taylor series. *Physical review letters*, 114(9):090502, 2015.

[98] Dominic W Berry, Andrew M Childs, and Robin Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 792–809. IEEE, 2015.

[99] Laura Clinton, Johannes Bausch, and Toby Cubitt. Hamiltonian simulation algorithms for near-term quantum hardware. *arXiv preprint arXiv:2003.06886*, 2020.

[100] The Difference Between AI and Machine Learning, Jan 2021. [Online; accessed 7. Jan. 2021].

[101] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[102] Alan M Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009.

[103] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[104] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[105] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[106] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[107] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[108] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

[109] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[110] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[111] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[112] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[113] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

[114] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[115] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[116] Jeremy Adcock, Euan Allen, Matthew Day, Stefan Frick, Janna Hinchliff, Mack Johnson, Sam Morley-Short, Sam Pallister, Alasdair Price, and Stasja Stanisic. Advances in quantum machine learning. *arXiv preprint arXiv:1512.02900*, 2015.

[117] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 217–228, 2019.

[118] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, 2014.

[119] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Quantum neuron: an elementary building block for machine learning on quantum computers. *arXiv preprint arXiv:1711.11240*, 2017.

[120] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014.

[121] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.

[122] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Neural-network quantum state tomography. *Nature Physics*, 14(5):447–450, 2018.

[123] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014.

[124] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[125] Kenneth De Jong. Evolutionary computation: a unified approach. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 327–342, 2020.

[126] Sean Luke. Essentials of metaheuristics. 1.

[127] Lothar M Schmitt. Theory of genetic algorithms. *Theoretical Computer Science*, 259(1-2):1–61, 2001.

[128] Christopher E Granade, C Ferrie, N Wiebe, and D G Cory. Robust online Hamiltonian learning. *New Journal of Physics*, 14(10):103013, October 2012.

[129] Nathan Wiebe, Christopher Granade, Christopher Ferrie, and David Cory. Quantum hamiltonian learning using imperfect quantum resources. *Physical Review A*, 89(4):042314, 2014.

[130] N Wiebe, C Granade, C Ferrie, and D G Cory. Hamiltonian Learning and Certification Using Quantum Resources. *Physical Review Letters*, 112(19):190501–5, May 2014.

[131] Jianwei Wang, Stefano Paesani, Raffaele Santagati, Sebastian Knauer, Antonio A Gentile, Nathan Wiebe, Maurangelo Petruzzella, Jeremy L O'Brien, John G Rarity, Anthony Laing, et al. Experimental quantum hamiltonian learning. *Nature Physics*, 13(6):551–555, 2017.

[132] Ian D Kivlichan, Nathan Wiebe, Ryan Babbush, and Alán Aspuru-Guzik. Bounding the costs of quantum simulation of many-body physics in real space. *Journal of Physics A: Mathematical and Theoretical*, 50(30):305301, 2017.

[133] Andrew M Childs and Nathan Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *arXiv preprint arXiv:1202.5822*, 2012.

[134] Jane Liu and Mike West. Combined parameter and state estimation in simulation-based filtering. In *Sequential Monte Carlo methods in practice*, pages 197–223. Springer, 2001.

[135] Alessandro Rudi, Leonard Wossnig, Carlo Ciliberto, Andrea Rocchetto, Massimiliano Pontil, and Simone Severini. Approximating hamiltonian dynamics with the nyström method. *Quantum*, 4:234, 2020.

[136] Christopher Granade, Christopher Ferrie, Steven Casagrande, Ian Hincks, Michal Kononenko, Thomas Alexander, and Yuval Sanders. QInfer: Library for statistical inference in quantum information, 2016.

[137] Arseni Goussev, Rodolfo A Jalabert, Horacio M Pastawski, and Diego Wisniacki. Loschmidt echo. *arXiv preprint arXiv:1206.6348*, 2012.

[138] Nathan Wiebe, Christopher Granade, and David G Cory. Quantum bootstrapping via compressed quantum hamiltonian learning. *New Journal of Physics*, 17(2):022005, 2015.

[139] Bas Hensen, Hannes Bernien, Anaïs E Dréau, Andreas Reiserer, Norbert Kalb, Machiel S Blok, Just Ruitenberg, Raymond FL Vermeulen, Raymond N Schouten, Carlos Abellán, et al. Loophole-free bell inequality violation using electron spins separated by 1.3 kilometres. *Nature*, 526(7575):682–686, 2015.

[140] Alexandr Sergeevich, Anushya Chandran, Joshua Combes, Stephen D Bartlett, and Howard M Wiseman. Characterization of a qubit hamiltonian using adaptive measurements in a fixed basis. *Physical Review A*, 84(5):052315, 2011.

[141] Christopher Ferrie, Christopher E Granade, and David G Cory. How to best sample a periodic probability distribution, or on the accuracy of hamiltonian finding strategies. *Quantum Information Processing*, 12(1):611–623, 2013.

[142] Christopher E Granade. Characterization, verification and control for large quantum systems. page 92, 2015.

[143] Christopher Ferrie. High posterior density ellipsoids of quantum states. *New Journal of Physics*, 16(2):023006, 2014.

[144] Michael J Todd and E Alper Yıldırım. On khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids. *Discrete Applied Mathematics*, 155(13):1731–1744, 2007.

[145] Ian Hincks, Thomas Alexander, Michal Kononenko, Benjamin Soloway, and David G Cory. Hamiltonian learning with online bayesian experiment design in practice. *arXiv preprint arXiv:1806.02427*, 2018.

[146] Lukas J Fiderer, Jonas Schuff, and Daniel Braun. Neural-network heuristics for adaptive bayesian quantum estimation. *arXiv preprint arXiv:2003.02183*, 2020.

[147] Sheng-Tao Wang, Dong-Ling Deng, and Lu-Ming Duan. Hamiltonian tomography for quantum many-body systems with arbitrary couplings. *New Journal of Physics*, 17(9):093017, 2015.

[148] Stefan Krastanov, Sisi Zhou, Steven T Flammia, and Liang Jiang. Stochastic estimation of dynamical variables. *Quantum Science and Technology*, 4(3):035003, 2019.

[149] Emmanuel Flurin, Leigh S Martin, Shay Hacohen-Gourgy, and Irfan Siddiqi. Using a recurrent neural network to reconstruct quantum dynamics of a superconducting qubit from physical observations. *Physical Review X*, 10(1):011006, 2020.

[150] Murphy Yuezhen Niu, Vadim Smelyanskyi, Paul Klimov, Sergio Boixo, Rami Barends, Julian Kelly, Yu Chen, Kunal Arya, Brian Burkett, Dave Bacon, et al. Learning non-markovian quantum noise from moire-enhanced swap spectroscopy with deep evolutionary algorithm. *arXiv preprint arXiv:1912.04368*, 2019.

[151] Eliska Greplova, Christian Kraglund Andersen, and Klaus Mølmer. Quantum parameter estimation with a neural network. *arXiv preprint arXiv:1711.05238*, 2017.

[152] Andrey Y Lokhov, Marc Vuffray, Sidhant Misra, and Michael Chertkov. Optimal structure and parameter learning of ising models. *Science advances*, 4(3):e1700791, 2018.

[153] Giovanni Acampora, Vittorio Cataudella, Pratibha R Hegde, Procolo Lucignano, Gianluca Passarelli, and Autilia Vitiello. An evolutionary strategy for finding effective quantum 2-body hamiltonians of p-body interacting systems. *Quantum Machine Intelligence*, 1(3):113–122, 2019.

[154] Daniel Burgarth and Ashok Ajoy. Evolution-free hamiltonian parameter estimation through zeeman markers. *Physical Review Letters*, 119(3):030402, 2017.

[155] Agnes Valenti, Guliuxin Jin, Julian Léonard, Sebastian D Huber, and Eliska Greplova. Scalable hamiltonian learning for large-scale out-of-equilibrium quantum dynamics. *arXiv preprint arXiv:2103.01240*, 2021.

[156] Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995.

[157] Ruhul A Sarker and Tapabrata Ray. Agent based evolutionary approach: An introduction. In *Agent-Based Evolutionary Search*, pages 1–11. Springer, 2010.

[158] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[159] Mark D Hill and Michael R Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[160] Roger W Hockney and Chris R Jesshope. *Parallel Computers 2: architecture, programming and algorithms*. CRC Press, 2019.

[161] Redis. https://redis.io, Nov 2020. [Online; accessed 7. Nov. 2020].

[162] RQ: Simple job queues for Python. https://python-rq.org, Nov 2020. [Online; accessed 7. Nov. 2020].

[163] Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, 1925.

[164] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117, 1944.

[165] Stephen G Brush. History of the lenz-ising model. *Reviews of modern physics*, 39(4):883, 1967.

[166] Francisco Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[167] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[168] Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014.

[169] Giuseppe E Santoro and Erio Tosatti. Optimization using quantum mechanics: quantum annealing through adiabatic evolution. *Journal of Physics A: Mathematical and General*, 39(36):R393, 2006.

[170] Victor Bapst, Laura Foini, Florent Krzakala, Guilhem Semerjian, and Francesco Zamponi. The quantum adiabatic algorithm applied to random optimization problems: The quantum spin glass perspective. *Physics Reports*, 523(3):127–205, 2013.

[171] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[172] Walter Greiner, Ludwig Neise, and Horst Stöcker. *Thermodynamics and statistical mechanics*. Springer Science & Business Media, 2012.

[173] John Hubbard. Electron correlations in narrow energy bands. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 276(1365):238–257, 1963.

[174] Richard T Scalettar. An introduction to the hubbard hamiltonian. *Quantum Materials: Experiments and Theory*, 6, 2016.

[175] Editorial. The hubbard model at half a century. *Nature Physics*, 2013.

[176] Pascual Jordan and Eugene Paul Wigner. über das paulische äquivalenzverbot. In *The Collected Works of Eugene Paul Wigner*, pages 109–129. Springer, 1993.

[177] Mark Steudtner and Stephanie Wehner. Fermion-to-qubit mappings with varying resource requirements for quantum simulation. *New Journal of Physics*, 20(6):063010, 2018.

[178] Jarrod McClean, Nicholas Rubin, Kevin Sung, Ian David Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, Eric Schuyler Fried, Craig Gidney, Brendan Gimby, et al. Openfermion: the electronic structure package for quantum computers. *Quantum Science and Technology*, 2020.

[179] James O Berger and Luis R Pericchi. The intrinsic bayes factor for model selection and prediction. *Journal of the American Statistical Association*, 91(433):109–122, 1996.

[180] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[181] Burnham KP Anderson DR. Model selection and multimodel inference: a practical information-theoretic approach, 2002.

[182] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[183] Arpad E Elo. *The rating of chess players, past and present*. Arco Pub., 1978.

[184] FIFA.com. Who we are - news - 2026 fifa world cup™: Fifa council designates bids for final voting by the fifa congress, Jun 2018.

[185] Christof Neumann, Julie Duboscq, Constance Dubuc, Andri Ginting, Ade Maulana Irwan, Muhammad Agil, Anja Widdig, and Antje Engelhardt. Assessing dominance hierarchies: validation and advantages of progressive evaluation with elo-rating. *Animal Behaviour*, 82(4):911–921, 2011.

[186] Lars Magnus Hvattum and Halvard Arntzen. Using elo ratings for match result prediction in association football. *International Journal of forecasting*, 26(3):460–470, 2010.

[187] Marcus W Doherty, Neil B Manson, Paul Delaney, Fedor Jelezko, Jörg Wrachtrup, and Lloyd CL Hollenberg. The nitrogen-vacancy colour centre in diamond. *Physics Reports*, 528(1):1–45, 2013.

[188] Gordon Davies and MF Hamer. Optical studies of the 1.945 ev vibronic band in diamond. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 348(1653):285–298, 1976.

[189] J Meijer, B Burchard, M Domhan, C Wittmann, Torsten Gaebel, I Popa, F Jelezko, and J Wrachtrup. Generation of single color centers by focused nitrogen implantation. *Applied Physics Letters*, 87(26):261909, 2005.

[190] AM Edmonds, UFS D'Haenens-Johansson, RJ Cruddace, ME Newton, K-MC Fu, C Santori, RG Beausoleil, DJ Twitchen, and ML Markham. Production of oriented nitrogen-vacancy color centers in synthetic diamond. *Physical Review B*, 86(3):035201, 2012.

[191] A Lenef and SC Rand. Electronic structure of the n-v center in diamond: Theory. *Physical Review B*, 53(20):13441, 1996.

[192] Sebastian Knauer. *Photonic Structure Coupling and Strain Sensing with Single Photon Emitters*. PhD thesis, University of Bristol, 2016.

[193] MS Blok, Cristian Bonato, ML Markham, DJ Twitchen, VV Dobrovitski, and R Hanson. Manipulating a qubit through the backaction of sequential partial measurements and real-time feedback. *Nature Physics*, 10(3):189–193, 2014.

[194] L Childress, MV Gurudev Dutt, JM Taylor, AS Zibrov, F Jelezko, J Wrachtrup, PR Hemmer, and MD Lukin. Coherent dynamics of coupled electron and nuclear spin qubits in diamond. *Science*, 314(5797):281–285, 2006.

[195] L_G Rowan, EL Hahn, and WB Mims. Electron-spin-echo envelope modulation. *Physical Review*, 137(1A):A61, 1965.

[196] Forrest T Charnock and TA Kennedy. Combined optical and microwave approach for performing quantum spin operations on the nitrogen-vacancy center in diamond. *Physical Review B*, 64(4):041201, 2001.

[197] Antonio Andreas Gentile. *Operating practical quantum devices in the pre-threshold regime*. PhD thesis, University of Bristol, 2020.

[198] Lilian Isabel Childress. *Coherent manipulation of single quantum systems in the solid state*. Harvard University, 2007.

[199] Benjamin Smeltzer, Lilian Childress, and Adam Gali. 13c hyperfine interactions in the nitrogen-vacancy centre in diamond. *New Journal of Physics*, 13(2):025021, 2011.

[200] Adam Gali, Maria Fyta, and Efthimios Kaxiras. Ab initio supercell calculations on nitrogen-vacancy center in diamond: Electronic structure and hyperfine tensors. *Physical Review B*, 77(15):155206, 2008.

[201] MV Gurudev Dutt, L Childress, L Jiang, E Togan, J Maze, F Jelezko, AS Zibrov, PR Hemmer, and MD Lukin. Quantum register based on individual electronic and nuclear spin qubits in diamond. *Science*, 316(5829):1312–1316, 2007.

[202] P-Y Hou, L He, F Wang, X-Z Huang, W-G Zhang, X-L Ouyang, X Wang, W-Q Lian, X-Y Chang, and L-M Duan. Experimental hamiltonian learning of an 11-qubit solid-state quantum spin register. *Chinese Physics Letters*, 36(10):100303, 2019.

[203] David A Broadway, Jean-Philippe Tetienne, Alastair Stacey, James DA Wood, David A Simpson, Liam T Hall, and Lloyd CL Hollenberg. Quantum probe hyperpolarisation of molecular nuclear spins. *Nature communications*, 9(1):1–8, 2018.

[204] Maximilian Schlosshauer. Decoherence, the measurement problem, and interpretations of quantum mechanics. *Reviews of Modern physics*, 76(4):1267, 2005.

[205] Eugen Kubala, Kim A Muñoz-Álvarez, Geoffrey Topping, Christian Hundshammer, Benedikt Feuerecker, Pedro A Gómez, Giorgio Pariani, Franz Schilling, Steffen J Glaser, Rolf F Schulte, et al. Hyperpolarized 13c metabolic magnetic resonance spectroscopy and imaging. *Journal of visualized experiments: JoVE*, (118), 2016.

[206] S Felton, AM Edmonds, ME Newton, PM Martineau, D Fisher, DJ Twitchen, and JM Baker. Hyperfine interaction in the ground state of the negatively charged nitrogen vacancy center in diamond. *Physical Review B*, 79(7):075203, 2009.

[207] Brian Flynn. Schrodinger's catwalk source code. https://github.com/flynnbr11/schrodingers_catwalk., 2021.

[208] Brian Flynn. Mathematical introduction to quantum computation, 2015. Undergraduate thesis.