



Doctorate of Philosophy

# Schrödinger's Catwalk

Brian Flynn

University of Bristol

December, 2020

# CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LISTINGS

# ACRONYMS

| | |
|---|---|
| AIC | Akaike information criterion. 84, 86 |
| AICC | Akaike information criterion corrected. 84 |
| BF | Bayes factor. 25–27, 33, 35, 48, 51, 55, 57, 61, 63, 64, 80, 81, 87, 89–91, 93–95 |
| BFEER | Bayes factor enhanced Elo ratings. 80, 81, 93, 94, 96 |
| BIC | Bayesian information criterion. 86 |
| CLE | classical likelihood estimation. 13 |
| EDH | experiment design heuristic. 18–23, 26, 35, 45, 55, 56, 83 |
| ES | exploration strategy. 27–31, 33–35, 40, 43, 45, 49, 51, 61, 65, 67, 75, 79, 105, 108 |
| ET | exploration tree. 28, 29, 31, 33–35, 45, 47, 75 |
| FH | Fermi-Hubbard. 58 |
| FP | false negatives. 78 |
| FP | false positives. 78 |
| GA | genetic algorithm. iii, 34, 67, 71, 74, 75, 80, 82, 83, 88, 90, 93, 96–98 |
| GES | genetic exploration strategy. 67, 78, 93, 94, 97 |
| HPD | high particle density. 18 |
| IQLE | interactive quantum likelihood estimation. 13, 14, 93, 94 |
| LTL | log total likelihood. 16 |
| ML | machine learning. 6, 26, 27, 78 |
| MS | model search. 27–29, 31, 35, 43, 45 |
| MVEE | minimum volume enclosing ellipsoid. 18 |

NV nitrogen-vacancy. 9
NVC nitrogen-vacancy centre. 14

OF objective function. iii, 67, 68, 74, 75, 78, 80, 81, 83, 84, 88, 91–93

PGH particle guess heuristic. 19, 20, 45

QHL quantum Hamiltonian learning. 8–14, 16, 18, 20–22, 25–27, 31–35, 40, 43, 48, 55, 56, 75, 93, 94, 105
QL quadratic loss. 17
QLE quantum likelihood estimation. 13, 32
QMLA Quantum Model Learning Agent. ii, iii, vii, 8, 13, 24, 25, 27–31, 34, 35, 40, 43–49, 51, 52, 60, 61, 63–65, 67, 72, 74–77, 83, 87, 88, 90, 91, 93, 94, 96–98, 105

SMC sequential monte carlo. 11–13, 15, 18, 19, 22, 31

TLTL total log total likelihood. 16, 25–27, 35, 84, 91
TN true negatives. 78
TP true positives. 78

# GLOSSARY

| | |
|---|---|
| Jordan Wigner transformation (JWT) | Jordan Wigner transformation . 60, 61, 64 |
| Loschmidt echo (LE) | Quantum chaotic effect described. . 14 |
| chromosome | A single candidate in the space of valid solutions to the posed problem in a genetic algorithm. . 68 |
| gene | Individual element within a chromosome. . 68 |
| hyperparameter | Variable within an algorithm that determines how the algorithm itself proceeds.. 11 |
| instance | a single implementation of the QMLA algorithm. iii, 48, 94, 97, 98, 105 |
| likelihood | Value that represents how likely a hypothesis is.. 10, 13, 15, 18, 31, 33, 34, 36, 88 |
| model | The mathematical description of some quantum system. 24 |
| model space | Abstract space containing all descriptions (within defined constraints such as dimension) of the system as models. 29 |
| probe | Input probe state, $\|\psi\rangle$, which the target system is initialised to, before unitary evolution. plural. 13, 15, 18–22, 54 |
| results directory | Directory to which the data and analysis for a given run of QMLA are stored. . 49 |
| run | collection of QMLA instances. iii, vii, 48, 49, 64, 65, 94, 96, 98, 105 |
| spawn | Process by which new models are generated by combining previously considered models.. 29 |
| success rate | . 48, 49 |

| | |
|---|---|
| term | Individual constituent of a model, e.g. a single operator within a sum of operators, which in total describe a Hamiltonian. . 24 |
| volume | Volume of a parameter distribution's credible region.. 18, 55, 56, 63 |
| win rate | . 48, 49 |

# INTRODUCTION

# 1 INTRODUCTION

# Part I

# CONTEXTUAL REVIEW

# 2

# QUANTUM MECHANICS

# 3

# QUANTUM COMPUTING AND SIMULATION

# 4

# MACHINE LEARNING

machine learning (ML) is the application of statistics, algorithms and computing power to discover meaning and/or devise actions from data.

# Part II

# ALGORITHMS

# OVERIVEW AND CONTRIBUTION

This part details the algorithm which form the basis for the research conducted in this thesis.

Chapter 5 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 sections. The analysis/figures presented in this chapter are unique to this thesis but do not necessarily offer novel insights.

Chapter 6 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 the data it produces. This protocol was initially devised by Dr. Rafaelle Santagati, with Drs. Andreas Gentile, Nathan Wiebe and Chris Granade. I contributed to the refinement of QMLA with Drs. Santagati and Gentile; the version presented in Chapter 6 represents the culmination of several conceptual stages. The protocol has been described in [1], which are described in later Parts.

Chapter 7 describes the implementation of QMLA through an open source software package. I was the principle designer and programmer of the codebase described, which constitutes a large proportion of the output of my research. The results presented in Part III, Part IV are all found using this framework.

# 5

# QUANTUM HAMILTONIAN LEARNING

First suggested in [2] and since developed [3, 4] and implemented [5, 1], QHL is a machine learning algorithm for the optimisation of a given Hamiltonian parameterisation against a quantum system whose model is known apriori. 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 $d \in \{0, 1\}$, according to the expectation value $\left|\langle\psi| e^{-i\hat{H}_0 t} |\psi\rangle\right|^2$. 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 $d$, i.e. $\left|\langle d| e^{-i\hat{H}(\vec{\alpha}_p)t} |\psi\rangle\right|^2$ 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 consistenst with those measured from $Q$.

The procedure is as follows. A *prior* probability distribution $\Pr(\vec{\alpha})$ 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 width 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. An example of imposing such domain-specific prior knowledge is, when choosing the prior for a model representing an $e^-$ spin in a nitrogen-vacancy (NV) centre, to select $GHz$ parameters for the electron spin's rotation terms, and $MHz$ terms for the spin's coupling to nuclei, as proposed in literature. It is important to understand, then, that QHL removes the prior knowledge of precisely the 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* $\left|\langle d| e^{-i\hat{H}(\vec{\alpha}_p)t} |\psi\rangle\right|^2$ 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

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

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). Likewise, each additional experiment incurs the cost of calculation of $N_p$ particles, so the total cost of running QHL for a single 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.

## 5.1 BAYES RULE

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

$$\Pr(\text{hypothesis}|\text{data}) = \frac{\Pr(\text{data}|\text{hypothesis}) \times \Pr(\text{hypothesis})}{\Pr(\text{data})}. \tag{5.1}$$

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

$$\Pr(\vec{\alpha}|\mathcal{D};\mathcal{E}) = \frac{\Pr(\mathcal{D}|\vec{\alpha};\mathcal{E})\ \Pr(\vec{\alpha})}{\Pr(\mathcal{D}|\mathcal{E})}. \tag{5.2}$$

We can then discretise Eq. (5.2) to the level of single particles (individual vectors in the parameter space), sampled from $\Pr(\vec{\alpha})$:

$$\Pr(\vec{\alpha}_p|d;e) = \frac{\Pr(d|\ \vec{\alpha}_p;\ e)\ \Pr(\vec{\alpha}_p)}{\Pr(d|e)} \tag{5.3}$$

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 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.

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 $d$, after evolving $\hat{H}(\vec{\alpha}_p)$ for $t$, i.e.

$$\Pr(d|\vec{\alpha}; e) = \left| \langle d| \, e^{-i\hat{H}(\vec{\alpha}_p)t} \, |\psi\rangle \right|^2, \tag{5.4}$$

which can be simulated clasically or using a quantum simulator (see Section 5.3). It is necessary first to know the datum $d$ (either 0 or 1) which was projected by $Q$ under real 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. (5.3), allowing us to redraw the entire probability distribution – i.e. we compute a *posterior* probability distribution by performing this routine on $N_p$ particles.

## 5.2 SEQUENTIAL MONTE CARLO

In practice, QHL samples from and updates $\Pr(\vec{\alpha})$ via 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 $p$ are updated by Eq. (5.5).

$$w_p^{new} = \frac{\Pr(d|\vec{\alpha}_p; e) \times w_p^{old}}{\sum_p w_p \Pr(\vec{\alpha}_p|d; e)} \tag{5.5}$$

In this way, strong particles (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 thus: we *simulataneously* update sampled particles' weights as well as $\Pr(\vec{\alpha})$. This iterates for the following experiment, using the *same* particles: we do *not* redraw $N_p$ particles for every experiment. Eventually, the weights of most particles fall 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*, i.e. selects new particles, according to the updated $\Pr(\vec{\alpha})$, according to the Liu-West resampling algorithm [6]. 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. 5.2. This procedure is easiest understood through the example presented in Fig. 5.1, where a two-parameter Hamiltonian is learned starting from a uniform distribution.

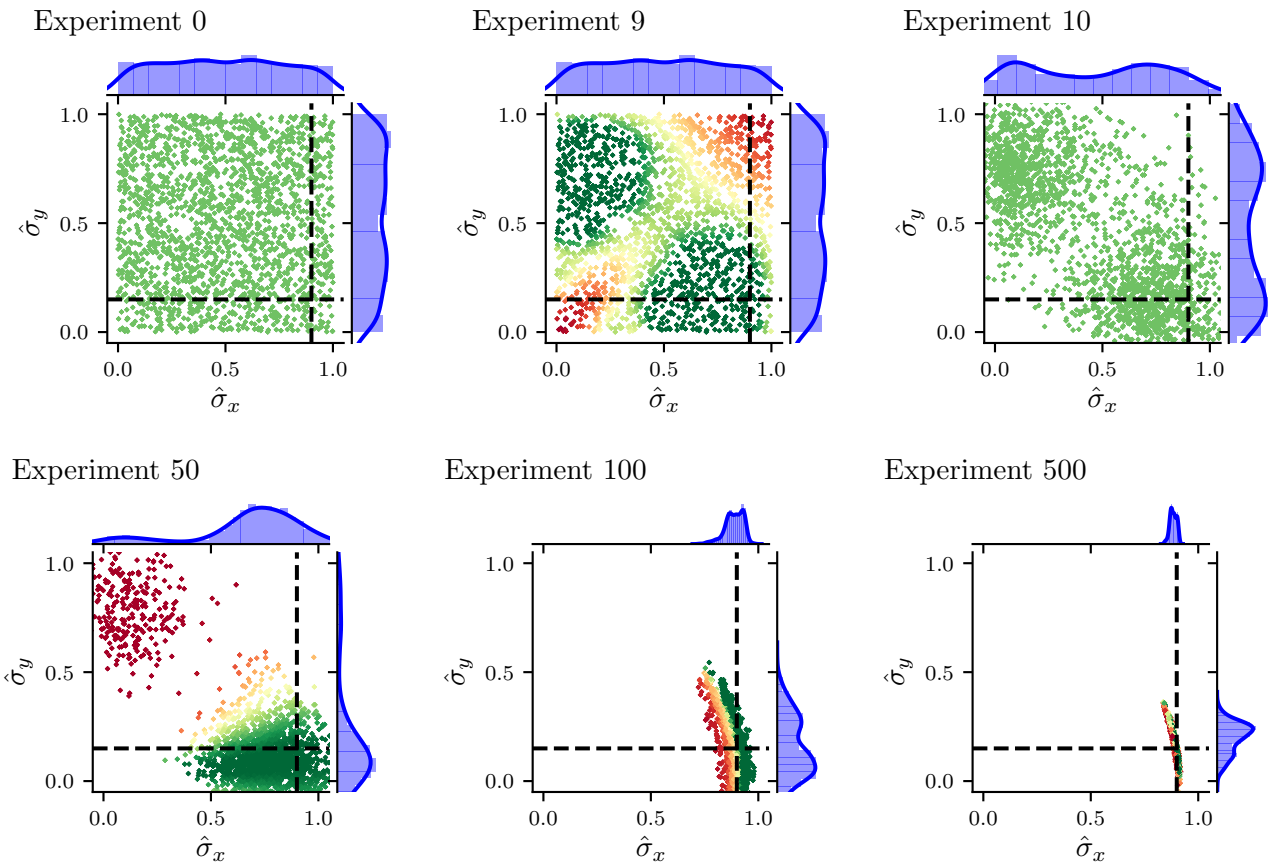


Figure 5.1: Quantum Hamiltonian learning (QHL) via sequential monte carlo (SMC). The studied model has two terms, $\{\hat{\sigma}^x, \hat{\sigma}^y\}$ with true parameters $\alpha_x = 0.9, \alpha_y = 0.15$ (dahsed lines), with $N_e = 500, N_p = 2000$. Crosses represent particles, while the distribution $\Pr(\alpha_p)$ 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),** sequential monte carlo (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. (5.3) and its weight is updated by Eq. (5.5). **(b),** after 9 experiments, the weights of the sampled particles are sufficintly 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),** Afer 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 $\alpha_x$ and $\alpha_y$.

very

## 5.3 LIKELIHOOD

The fundamental step within QHL is the calculation of likelihood in Eq. (5.3). The core of this learning algorithm is that this likelihood can be retrieved from the Born rule, although in principle *any* valid likelihood function can fulfil this equation, provided the calculation of the likelihood captures the probabilty that the present hypothesis produced the present datum.

In general, it is not always possible to derive the analytical likelihood, especially in cases where we wish to vary the probe. When Eq. (5.4) can be computed classically, QHL relies on classical likelihood estimation (CLE), i.e. involving the exponential calculation of Eq. (5.4), whereas quantum likelihood estimation (QLE) uses the same likelihood function computed on a quantum simulator; this is the sole application of quantum simulators in this protocol and indeed the remainder of this thesis. Access to such hardware, operating perfectly, would provide exponential speedup in the calculation of this term, rendering both QHL and the wider QMLA formalism scalable, although in this thesis we do not implement QLE so everything can be viewed as CLE. QLE was implemented in [5].

We adopt notation used by QInfer, which QMLA for the likelihood estimation stage [7]. The expectation value for a the unitary operator is given by

$$\Pr(0) = |\langle\psi| e^{-i\hat{H}_p t} |\psi\rangle|^2 = l(d=0|\hat{H}_p; e), \tag{5.6}$$

i.e. the input basis is assigned the measurement label $d = 0$, and this quantity is the probability of measuring $d = 0$, i.e. measuring the same state as input. However, we assume a binary outcome model, i.e. that the system is measured either in $|\psi\rangle$ (labelled $d = 0$), or it is not ($d = 1$); the likelihood for the latter case is

$$\Pr(1) = l(d=1|\hat{H}_p; e) = \sum_{\{|\psi_\perp\rangle\}} |\langle\psi_\perp| e^{-i\hat{H}_p t} |\psi\rangle|^2 = 1 - \Pr(0). \tag{5.7}$$

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

### 5.3.1 *Interactive Quantum Likelihood Estimation*

An important extension to QLE is interactive quantum likelihood estimation (IQLE), which follows SMC but uses an alternative likelihood function in order to overcome some of its inherent challenges [4]. Two almost identical Hamiltonians will diverge after exponentially small evolution time [8]. This is problematic for QHL because it relies on the likelihood function which is built on the assumption that increasing accuracy of $\hat{H}(\vec{\alpha})$ approximating $\hat{H}_0$ should result in rising likelihood; this result indicates that even extremely accurate approximations become unreliable after very short evolution times. Coupled with the result that small time experiments are uninformative [9], this observation demands exponentially many measurements to approximate the exponentially small likelihoods, rendering the approach inefficient.

The Loschmidt echo (LE) is the measure of the revival resulting from an imperfect time-reversal operation implemented after the standard time evolution. The reversal operation corresponds to some Hamiltonian, $\hat{H}_-$, which is seen as an attempt to un-do the evolution according to the original Hamiltonian, $\hat{H}_+$. As such we say that $\hat{H}_-$ is evolved for $-t$, so its unitary is $e^{-i\hat{H}_-(-t)}$ after $\hat{H}_+$ is evolved for $t$. The LE can be written and characterised as

$$M(t) = \left|\langle\psi|e^{+i\hat{H}_- t}e^{-i\hat{H}_+ t}|\psi\rangle\right|^2 \sim \begin{cases} 1-\mathcal{O}(t^2), & t \leq t_c \\ e^{-\mathcal{O}(t)}, & t_c \leq t \leq t_s \\ 1/\|\hat{H}\|, & t \geq t_s \end{cases} \tag{5.8}$$

where $\hat{H}_-, \hat{H}_+$ are backward and forward time evolutions respsectively, which are assumed almost identical; $\|\hat{H}\|$ is their dimension, 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 [10]. In effect, the LE gaurantees that 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 $\hat{H}_0$ (true) or $\hat{H}(\vec{\alpha})$ (particle or hypothesis), and sampling $\hat{H}_-$ from $\Pr(\vec{\alpha})$, we can adopt Eq. (5.8) as the likelihood function in Eq. (5.4), in the knowledge that this will disinguish between hypotheses based on their similarity to $\hat{H}_0$.

Importantly, IQLE can only be used where we can *reliably* evolve the system under study. In order that the reverse evolution is relibable, 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 [11]. The remaining application for IQLE, and correspondingly QHL, is in the characterisation of untrusted quantum simulators, which can realise such coherent channels [5].

### 5.3.2 *Analytical likelihood*

In some cases, analytical likelihood functions can be derived to describe the dynamics of simple quantum systems [12, 13], for instance encoding the Rabi frequency $\omega$ of an oscillating electron spin in an nitrogen-vacancy centre (NVC),

$$\hat{H}(\omega) = \frac{\omega}{2}\hat{\sigma}_z. \tag{5.9}$$

Then, bearing in mind 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. (5.9) is given by

$$\begin{aligned} U = e^{-i\hat{H}(\omega)t} = e^{-i\frac{\omega t}{2}\hat{\sigma}_z} &= \cos\left(\frac{\omega t\hat{\sigma}_z}{2}\right) - i\sin\left(\frac{\omega t\hat{\sigma}_z}{2}\right) \\ &= \left(\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\left(\frac{\omega t}{2}\right)^{2k}\hat{\sigma}_z^{2k}\right) - i\left(\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}\left(\frac{\omega t}{2}\right)^{2k+1}\hat{\sigma}_z^{2k+1}\right) \\ &= \left(\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k)!}\left(\frac{\omega t}{2}\right)^{2k+1}\right)\hat{\mathbb{1}} - i\left(\sum_{k=0}^{\infty}\frac{(-1)^k}{(2k+1)!}\left(\frac{\omega t}{2}\right)^{2k+1}\right)\hat{\sigma}_z \\ &= \cos\left(\frac{\omega t}{2}\right)\hat{\mathbb{1}} - i\sin\left(\frac{\omega t}{2}\right)\hat{\sigma}_z \end{aligned} \tag{5.10}$$

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

$$\langle\psi_1|\, U\, |\psi_0\rangle = \cos\left(\frac{\omega t}{2}\right)\langle\psi_1|\psi_0\rangle - i\sin\left(\frac{\omega t}{2}\right)\langle\psi_1|\hat{\sigma}_z|\psi_0\rangle\,. \tag{5.11}$$

By initialising and projecting into the same state, say $|\psi_0\rangle = |\psi_1\rangle = |+\rangle$, we have

$$\begin{aligned} \hat{\sigma}_z|+\rangle = |-\rangle \implies \langle\psi_1|\hat{\sigma}_z|\psi_0\rangle &= 0 \\ \langle\psi_1|\psi_0\rangle &= 1 \\ \implies \langle\psi_1|U|\psi_0\rangle &= \cos\left(\frac{\omega t}{2}\right), \end{aligned} \tag{5.12}$$

i.e. if the system measures in $|+\rangle$, we set the datum $d = 1$, otherwise $d = 0$. Then, from Born's rule, and in analogy with Eq. (5.4), we can formulate the likelihood function, where the hypothesis is the single parameter $\omega$, and the sole experimental control is $t$,

$$\Pr(d = 1|\omega; t) = |\langle\psi_1|U|\psi_0\rangle|^2 = \cos^2\left(\frac{\omega t}{2}\right) \tag{5.13}$$

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

## 5.4 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*,

$$l_e = \sum_{p\in\{p\}} \Pr(d|\vec{\alpha}_p; e) \times w_p^{old}. \tag{5.14}$$

For each experiment, we use total likelihood as a measure of how well the distribution performed, 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 noramlisation factor in Eq. (5.5), [14].

$l_e$ are strictly positive, and because the natural logarithm is a monotonically increasing function, we can equivalently work with 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.

Note, we know that

$$\begin{aligned} w_p^0 = \frac{1}{N_p} &\implies \sum_p^{N_p} w_p^0 = 1; \\ \Pr(d|\vec{\alpha}_p; e) \leq 1 &\implies \Pr(d|\vec{\alpha}_p; e) \times w_p^{old} \leq w_p^{old} \\ &\implies \sum_{\{p\}} \Pr(d|\vec{\alpha}_p; e) \times w_p^{old} \leq \sum_{\{p\}} w_p^{old} \leq \sum_p^{N_p} w_p^0; \\ &\implies l_e \leq 1. \end{aligned} \tag{5.15}$$

Eq. (5.14) 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 $l_e \approx 0$.

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),

$$\mathcal{L}_i = \sum_{e \in \mathcal{E}} ln(l_e). \tag{5.16}$$

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, i.e. $\hat{H}_i$ 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 poor agreement $l_e < 1 \Rightarrow ln(l_e) < 0$. Then, in all cases Eq. (5.16) is negative, and across a series of experiments, strong agreement gives low $|\mathcal{L}_i|$, whereas weak agreement gives large $|\mathcal{L}_i|$.

## 5.5 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.

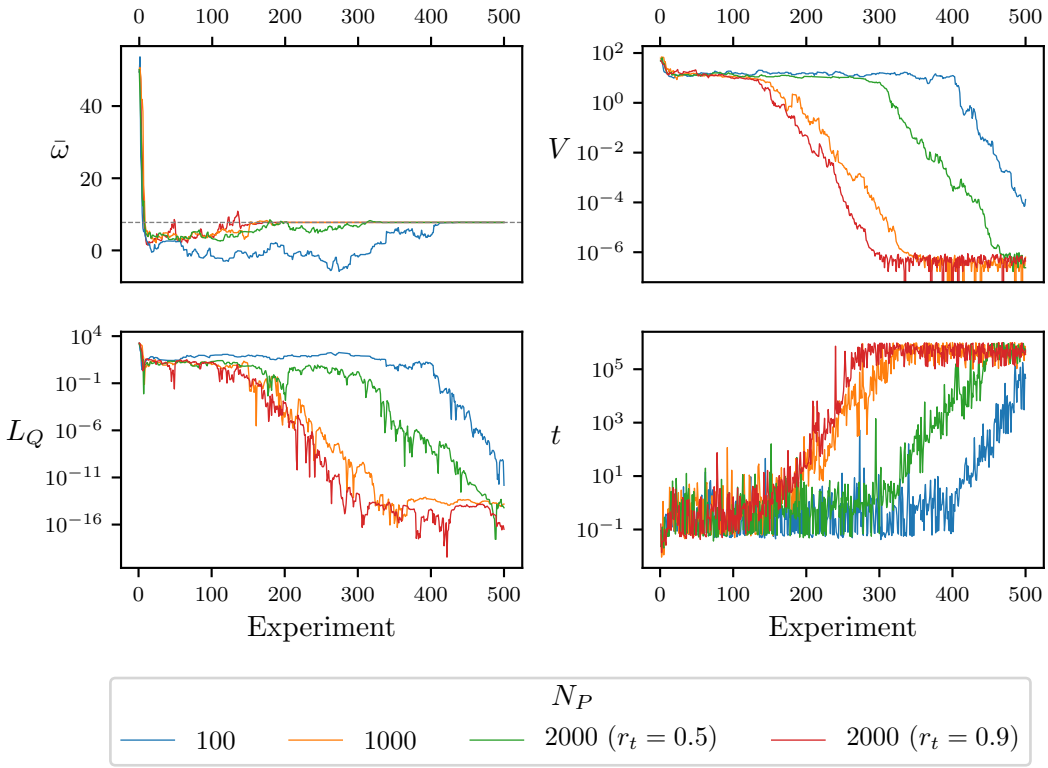


Figure 5.2: Parameter learning for the analyitcal likelihood Eq. (5.13) for varying numbers of particles $N_p$, for $N_e = 500$. For $N_p = 2000$, we show the resampler threshold set to $r = 0.5$ and $r = 0.9$. (a) the parameter estimate, i.e. $\bar{\omega}$, the mean of the posterior distribution after each experiment, approaching $\omega_0 = 7.75$ (dashed line), where the prior is centred on $\omega = 50 \pm 25$. Decrease in (b) volume, $V$, (c) quadratic loss, $L_Q$, and (d) evolution time, $t$, are shown against experiment number.

The most obvious meausre of the progression of parameter estimation is the error between the true parameterisation, $\vec{\alpha}_0$, and the approximation $\vec{\alpha}_p = \text{mean}\left(\Pr(\vec{\alpha})\right)$, which can be captured by a large family of loss functions. Here we use the quadratic loss (QL), which captures this error through the sum of the square difference between each parameter's true and estimated values symetrically, i.e. error above the true parameter is as impactful as error below. We can record the QL at each experiment of our training regime and hence track its performance.

**Definition 5.5.1** (Quadratic Loss)**.** For a true parameterisation $\vec{\alpha}_0$, and a hypothesis $\vec{\alpha}$, the quadratic loss is given by

$$L_Q = \|\vec{\alpha}_0 - \vec{\alpha}\|^2 . \tag{5.17}$$

### 5.5.1 *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 mean $(\Pr(\vec{\alpha}))$ is accurate. For example, for a single parameter $\omega$, our best knowledge of the parameter is mean $(\Pr(\omega))$, 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 [7, 15].

In general, a confidence region, defined by its confidence level $\kappa$, is drawn by grouping particles of high particle density (HPD), $\mathcal{P}$, such that $\sum_{p\in\mathcal{P}} w_p \geq \kappa$. We use the concept of minimum volume enclosing ellipsoid (MVEE) to capture the confidence region [15], calculated as in [16], which are characterised by their covariance matrix, $\Sigma$, which allows us to calculate the volume,

do we posteri credibl

$$V(\Sigma) = \frac{\pi^{|\vec{\alpha}|/2}}{\Gamma(1+\frac{|\vec{\alpha}|}{2})} \det\left(\Sigma^{-\frac{1}{2}}\right), \tag{5.18}$$

where $\Gamma$ 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 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 $\Sigma = \mathrm{cov}(\Pr(\vec{\alpha}))$ after each experiment, and hence track the uncertainty in our parameters across the training experiments [2]. We use volume as a measure of the learning procedure's progress: slowly decreasing or static volume indicates poor learning, possibly highlighting poor experiment design, while fast or exponentially decreasing volume indicates that the parameters are being learned well. When the volume has converged, the learning has saturated and there is little benefit to running further experiments.

## 5.6 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. Usually, the EDH attempts to exploit the information available, adaptively accounting for some aspects of the inference process performed already, although there may be justification for enforcing a non-adaptive schedule, for instance to force QHL to train on a full set of experimental data rather than a restricted set which adaptive methods would advise. We can categorise each EDH as either *online* of *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 [17, 18].

### 5.6.1 *Particle Guess Heuristic*

The default EDH is the particle guess heuristic (PGH) [4], an online method which attempts to design the optimal evolution time based on the posterior. 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, and therefore limits the evolution time for which the posterior can be expected to reasonably mimic $\hat{H}_0$. For example, consider Eq. (5.9) with a single parameter with $\omega_0 = 10$, and current mean $(\Pr(\omega)) = 9, \text{std}\,(\Pr(\omega)) = 2$, we can expect that the approximation $\omega' = \text{mean}\,(\Pr(\omega))$ is valid up to $t_{max} = {}^{1}/_{\text{std}(\Pr(\omega))}$. It is sensible, then, to use $t \sim t_{max}$ for two reasons: (i) smaller times are already well explained by the posterior, so offer little opportunity to learn; (ii) $t_{max}$ is at or near the threshold the posterior can comfortably explain, so it will expose the relative difference in likelihood between the posterior's particles, providing a strong capacity to learn. Informally, as the uncertainty in the posterior shrinks, PGH selects larger times to ensure the training is based on informative eperiments simulataneously with increasing certainty about the paraemters. In the one-dimensional case, this logic can be used to find an optimal time heuristic, where experiment $k$ is assigned $t_k = {}^{1.26}/_{\text{std}(\Pr(\omega))}$ [13].

Rather than directly using the inverse of the standard deviation of $\Pr(\vec{\alpha})$, which relies on the expensive calculation of the covarinace 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

$$t_k = \frac{1}{\|\vec{\alpha}_i - \vec{\alpha}_j\|}, \tag{5.19}$$

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 $\Pr(\vec{\alpha})$.

### 5.6.2 *Alternative experiment design heuristics*

The EDH can be used to suit the requirements of the target system; we test four such examples against a series of models. 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 $t_{max} = 1000$. 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 is not gauranteed 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 between 0 and $t_{max}$, and train on the list twice. 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 $t = (9/8)^k$ [2]. 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 memory.
- PGH: as in Section 5.6.1.

We show how the choie of EDH within this set can influence the training procedure, by testing models of various complexity amd dimension. In particular, we first test a simple 1-qubit model, Eq. (5.20a); followed by more complicated 1-qubit model, Eq. (5.20b); as well as randomly generated 5-qubit Ising, Eq. (5.20c), and 4-qubit Heisenberg models, Eq. (5.20d). Each $\hat{H}_i$ have randomly chosen parameters implicitly assigned to each term.

$$\hat{H}_1 = \hat{\sigma}_1^z \tag{5.20a}$$

$$\hat{H}_2 = \hat{\sigma}_1^x + \hat{\sigma}_1^y + \hat{\sigma}_1^z \tag{5.20b}$$

$$\hat{H}_3 = \hat{\sigma}_1^z\hat{\sigma}_3^z + \hat{\sigma}_1^z\hat{\sigma}_4^z + \hat{\sigma}_1^z\hat{\sigma}_5^z + \hat{\sigma}_2^z\hat{\sigma}_4^z + \hat{\sigma}_2^z\hat{\sigma}_5^z + \hat{\sigma}_3^z + \hat{\sigma}_4^z + \hat{\sigma}_3^z + \hat{\sigma}_5^z \tag{5.20c}$$

$$\hat{H}_4 = \hat{\sigma}_1^z\hat{\sigma}_2^z + \hat{\sigma}_1^z\hat{\sigma}_3^z + \hat{\sigma}_2^x\hat{\sigma}_3^x + \hat{\sigma}_2^z\hat{\sigma}_3^z + \hat{\sigma}_2^x\hat{\sigma}_4^x + \hat{\sigma}_3^z\hat{\sigma}_4^z \tag{5.20d}$$

We show the performance of each of the listed EDHs in Fig. 5.3. We will have cause to use alternative EDHs in paricular circumstances, but we adopt PGH as the default EDH throughout this thesis, unless otherwise stated.

## 5.7 PROBE SELECTION

A final consideration for 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 [5, 13]. 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 $\Psi$, allowing for the same $|\psi\rangle$ to be used for the multiple experiments within the training, e.g. by iterating over $\Psi$. $\Psi$ can be generated with

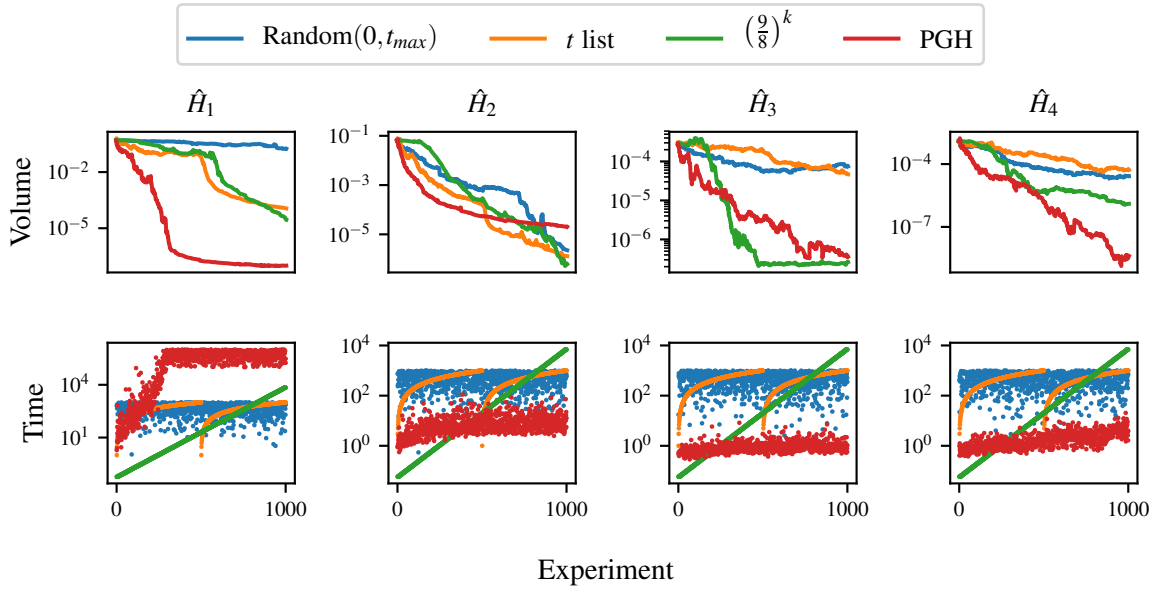


Figure 5.3: The volume of various models when trained through QHL using different EDHs. We show models of various complexity and dimension, each trained using four heuristics, outlined in the main text. Implementation details are listed in Table A.1

respect to the individual learning problem as we will examine later, but it is usually sufficient 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. $|t\rangle$ : $\Psi$ is the tomographic basis set, $N_\psi = 40$;

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

We show the 1-qubit probes within $\Psi$ under each of these strategies on the Bloch sphere in Fig. 5.4.

We test the same set of models as Eq. (5.20) and test each of these probe construction strategies in Fig. 5.5. We can draw a number of useful observations from these simple tests:

- Training on an eigenstate, $|0\rangle$, of the model 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, $|+\rangle$, 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. 5.5c, 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

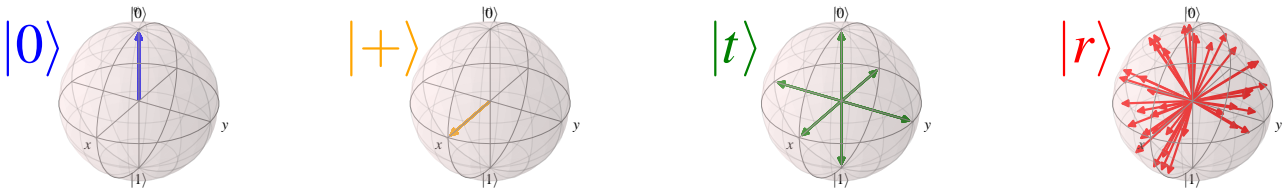


Figure 5.4: 1-qubit probes used for tests in Fig. 5.5.

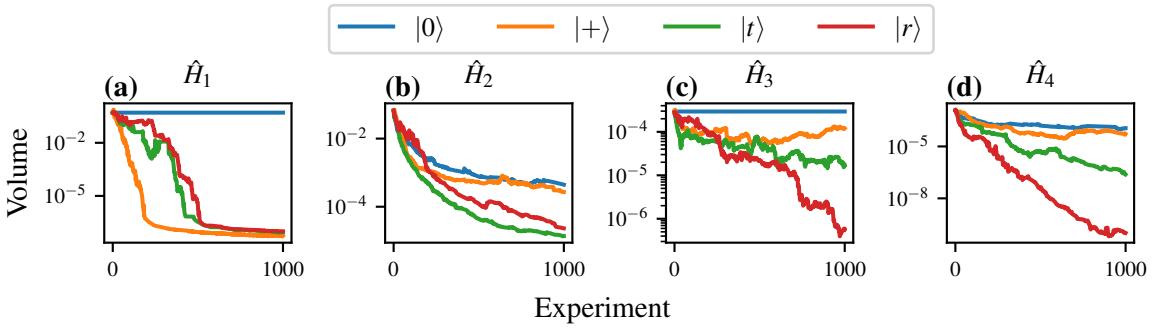


Figure 5.5: The volume of various models when trained through QHL using different initial probe sets. We show models of various complexity and dimension, each trained using random probes, $|r\rangle$, tomographic basis set probes, $|t\rangle$, as well as $|0\rangle$ and $|+\rangle$ probes. In each case the probes are generated for arbitrary numbers of qubits; for $|0\rangle, |1\rangle$, 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

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.

- In general the tomographic and random probe sets perform reliably.

It is an open challenge to identify the optimal probe for training any given model, and the choice of such informative probes could be built into the EDH in general, for example by computing the eigenstates of the candidate, and generating a probe set from superposition between those eigenstates. However, a further consideration is that, for model comparison purposes, it is helpful to have a universal set of probes, upon which all models are learned. This would minimise bias towards particlar models, which might arise from probes which are favourable bases for a subset of models, for example $|+\rangle$ in Fig. 5.5**a**. Careful consideration should be given to $N_\psi$ in the choice of the probe generator, since it is important to ensure probes test the parameterisation across the Hilbert space robustly, However this must be balanced by ensuring SMC has sufficient opportunity to learn within a subspace before moving to the next; we can mitigate this concern by instructing the EDH to repeatedly select a probe from $\Psi$ for a batch of experiments, say five, before moving to the next available probe.

As standard for the remainder of this thesis, unless otherwise stated, we adopt the random probe generator as the defualt mechanism for selecting probes, only iterating between probes after batches of 5 experiments.

# 6

# QUANTUM MODEL LEARNING AGENT

Quantum Model Learning Agent (QMLA) is an algorithm that extends the concept of applying machine learning to the characterisation of Hamiltonians we've seen so far. The extension, and central question of QMLA is: if we do not even have 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 what interactions the system is subject to. Throughout this thesis, we are concerned solely with Hamiltonian models. although, in general, the description of a quantum system of interest need not be Hamiltonian, e.g. it could be Lindbladian for open quantum systems, so we generalise the effort to the search for the most suitable *model*.

For the remainder of this thesis, our objective is to learn the model underlying some target system $Q$. We will first introduce some concepts which will prove useful when discussing QMLA, before describing the protocol in detail in Section 6.3.

## 6.1 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 assosiated with a multipicative 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

$$\hat{H} = \begin{pmatrix} \alpha_0 & \dots & \alpha_n \end{pmatrix} \begin{pmatrix} \hat{t}_1 \\ \vdots \\ \hat{t}_n \end{pmatrix} = \vec{\alpha} \cdot \vec{T} \tag{6.1}$$

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

$$\begin{aligned} \hat{H} &= \begin{pmatrix} \alpha_x & \alpha_y & \alpha_z \end{pmatrix} \cdot \begin{pmatrix} \hat{\sigma}^x \\ \hat{\sigma}^y \\ \hat{\sigma}^z \end{pmatrix} \\ &= \alpha_x \hat{\sigma}^x + \alpha_y \hat{\sigma}^y + \alpha_z \hat{\sigma}^z \\ &= \begin{pmatrix} \alpha_z & \alpha_x - i\alpha_y \\ \alpha_x + i\alpha_y & \alpha_z \end{pmatrix}. \end{aligned} \tag{6.2}$$

Through this formalism, we can say that the sole task of QHL was to optimise $\vec{\alpha}$, given $\vec{T}$. The principle task of QMLA is to identify $\vec{T}$ with the most statistical evidence for describing the target system $Q$. In short, QMLA proposes candidate models $\hat{H}_i$ as hypoteses 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 $\vec{T}$, [19, 20, 21, 22, 23, 24, 25]. While discussing a model $\hat{H}_i$, their *training* then simply means the implementation of QHL, where $\hat{H}_i$ is assumed to represent $Q$, such that $\vec{\alpha}_i$ is optimised as well as it can be, even if $\hat{H}_i$ is entirely inaccurate.

## 6.2 BAYES FACTORS

We can use the tools introduced in Section 5.4 to *compare* 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 5. It is statistically meaningful to compare models via their 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, $\mathcal{E}$ [26].

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),

$$B_{ij} = \frac{\Pr(\mathcal{D}|\hat{H}_i;\mathcal{E})}{\Pr(\mathcal{D}|\hat{H}_j;\mathcal{E})}. \tag{6.3}$$

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 signicative of the stronger model from a pair, at explaining observed data, while favouring models of low cardinality, thereby supressing overfitting models.

We have that, for independent experiments, and recalling Eq. (5.14),

$$\begin{aligned}\Pr(\mathcal{D}|\hat{H}_i;\mathcal{E}) &= \Pr(d_n|\hat{H}_i;e_n)\times\Pr(d_{n-1}|\hat{H}_i;e_{n-1})\times\cdots\times\Pr(d_0|\hat{H}_i;e_0)\\ &= \prod_{e\in\mathcal{E}}\Pr(d_e|\hat{H}_i;e)\\ &= \prod_{e\in\mathcal{E}}(l_e)_i.\end{aligned} \tag{6.4}$$

We also have, from Eq. (5.16)

$$\begin{aligned} \mathcal{L}_i &= \sum_{e\in\mathcal{E}} ln\left((l_e)_i\right) \\ \implies e^{\mathcal{L}_i} &= \exp\left(\sum_{e\in\mathcal{E}} ln\left[(l_e)_i\right]\right) = \prod_{e\in\mathcal{E}} \exp\left(ln\left[(l_e)_i\right]\right) = \prod_{e\in\mathcal{E}} (l_e)_i. \end{aligned} \tag{6.5}$$

So we can write

$$B_{ij} = \frac{\Pr(\mathcal{D}|\hat{H}_i;\mathcal{E})}{\Pr(\mathcal{D}|\hat{H}_j;\mathcal{E})} = \frac{\prod_{e\in\mathcal{E}}(l_e)_i}{\prod_{e\in\mathcal{E}}(l_e)_j} = \frac{e^{\mathcal{L}_i}}{e^{\mathcal{L}_j}} \tag{6.6}$$

$$\implies B_{ij} = e^{\mathcal{L}_i - \mathcal{L}_j} \tag{6.7}$$

This is simply the exponential of the difference between two models' total log-likelihoods 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 $B_{ij} = e^{-90} \ll 1$. Therfore $\left|B_{ij}\right|$ is the strength of the statistical evidence in favour of the interpretation

$$\begin{cases} B_{ij} > 1 & \Rightarrow \hat{H}_i \text{ stronger than } \hat{H}_j \\ B_{ij} < 1 & \Rightarrow \hat{H}_j \text{ stronger than } \hat{H}_i \\ B_{ij} = 1 & \Rightarrow \hat{H}_i \text{ as strong as } \hat{H}_j \end{cases} \tag{6.8}$$

### 6.2.1 *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 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. (6.3), 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. 5.2. Moreover it provides a benefit in reducing the computation 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 converged. We will verify this claim later, after we have introduced some relevant concepts, in Section 10.2.2.1.

figures points.

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 clearly a favourable approach: provided for each model we compute Eq. (5.16) 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 $t > T_2$, 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 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.

## 6.3 QUANTUM MODEL LEARNING AGENT PROTOCOL

Given a target quantum system, $Q$, described by some *true* model $\hat{H}_0$, QMLA distills a model $\hat{H}' \approx \hat{H}_0$. We can think of QMLA as a forest search algorithm[1]: 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 (MS)*, structured according to a bespoke exploration strategy (ES), which we detail in Section 6.4. In short, the MS proceeds by grouping models in *layers*, training each model on each layer independently, layers are then *consolidated* to rank their performance, and new models are then *spawned*.

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

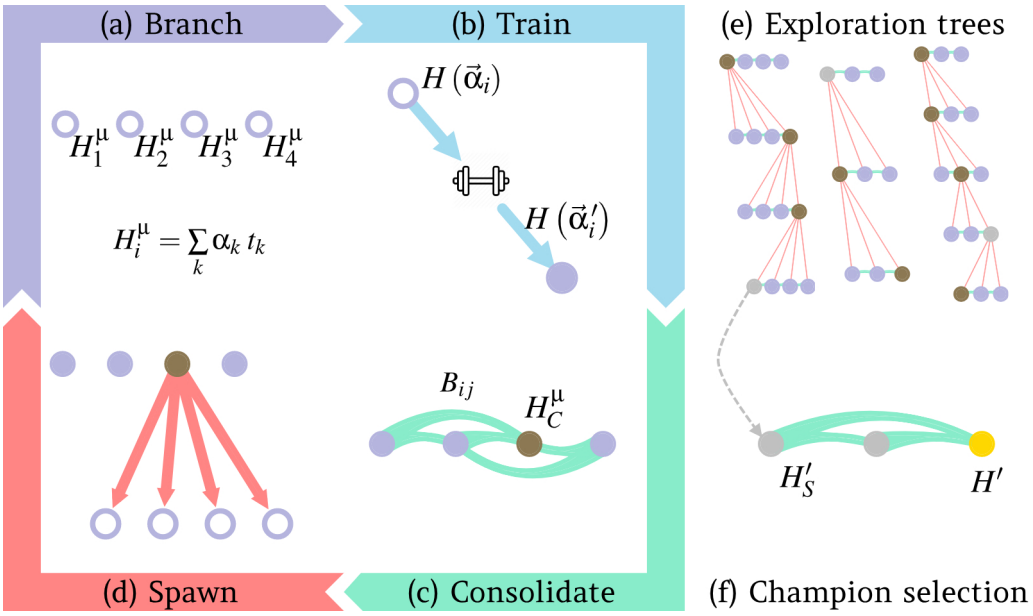


Figure 6.1: Schematic of Quantum Model Learning Agent (QMLA). **(a-d)** Model search (MS) phase within an Exploration strategy (ES). **(a)** Models are placed as (empty, purple) nodes on the *active branch* $\mu$, where each model is a sum of terms $\hat{t}_k$ multiplied by a scalar parameter $\alpha_k$. **(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)** $\mu$ is consolidated, i.e. models are evaluated relative to other models on $\mu$, according to the consolidation mechanism specificed 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 genertion strategy. In this example, models are spawned from a single parent. The newly spawned models are placed on the next layer $\mu + 1$, iterating back to **(a)**. **(e-f)** Higher level of entire QMLA procedure. **(e)** The MS 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 exploration trees. Each ES nominates a champion, $\hat{H}_S'$ (silver), after consolidating its branch champions (bronze). **(f)** $\hat{H}_S'$ from each of the above exploration trees are gathered on a single layer, which is consolidated to give the final champion model, $\hat{H}'$ (gold).

## 6.4 EXPLORATION STRATEGIES

QMLA is implemented by running $N_t$ exploration trees (ETs) concurrently, where each ET corresponds to a unique MS and ultimately nominates a single model as its favoured approximation of $\hat{H}_0$. An ES is the set of rules which guide a single ET throughout its MS. 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 MS phase: instructions for how QMLA should respond at predefined junctions, e.g. whether to cease the MS 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. 6.2. In summary: QMLA sends a request to the ES for a set of models; ES designs models and places them as leaves on one of its branches, and returns the set $\mathbb{H}$; QMLA places $\mathbb{H}$ on a unique layer; 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.

### 6.4.1 *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 successfel 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 MS occurs within some *model space*, the size of which 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[2]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 high quality models and providing a reasonable prospect of uncovering the true model, or a strong approximation thereof.

2 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 7.

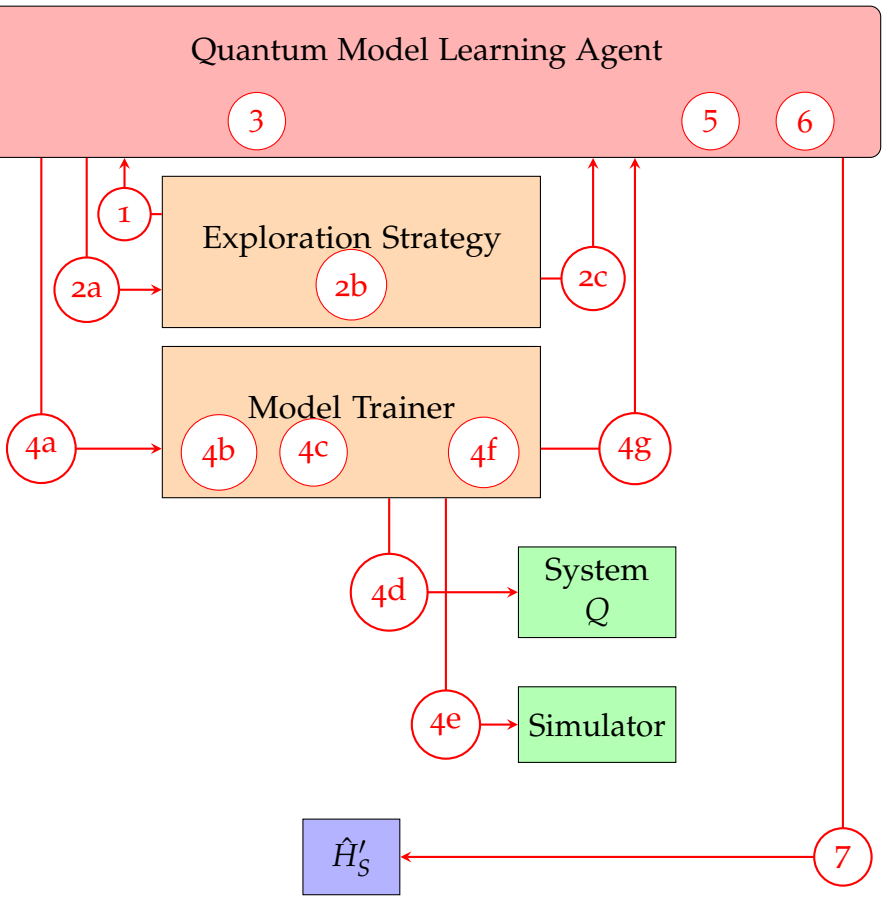


Figure 6.2: Interface between QMLA and a single ES. The main components are the ES, model training subroutine, target quantum system (i.e. black box, $Q$), and (quantum) simulator. The main steps of the algorithm, shown in red with arrows denoting data transferred during that step, are as follows. **1**, 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 layers' models if appropriate. **2b**, ES spawns new models, $\mathbb{H}$; **2c**, ES passes $\mathbb{H}$ to QMLA. **3**, QMLA assigns a new layer $(\mu \leftarrow \mu + 1)$ 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 $P_i$ 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 $\{\vec{\alpha}_1, \ldots, \vec{\alpha}_N\}$ sampled from $P_i$ to retrieve a *likelihood* $l_e$; **4e**, update the prior $P_i$ based on $(d, l_e)$. **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, $\hat{H}'_S$.

### 6.4.2 *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 Algs. Algorithm 1, Algorithm 2, QMLA queries the ES in order to decide what happens next. Here we list the important cases of this behaviour.

i. Parameter-learning settings
   (i) such as the prior distribution to assign each parameter during QHL, and the parameters needed to run SMC.
   (ii) the time scale on which to examine $Q$.
   (iii) the input probes to train upon.

ii. Branch comparison strategy
   (i) How to compare models within a branch (or QMLA layer). Some examples used in this work are (a) a points-ranking where each points are assigned according to Eqn. Eq. (6.8) (b) ranking reflecting each model's log-likelihood after training. All methods in §**??** can be thought of as branch comparison strategies.

iii. MS termination criteria
   (i) e.g. instruction to stop after a fixed number of iterations, or when a certain fitness has been reached.

iv. Champion nomination
   (i) when a single tree is explored, identify a single champion from the branch champions
   (ii) if multiple trees are explored, how to compare champions across trees.

### 6.4.3 *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' likelihood are computed on $S_t$.

### 6.4.4 *Modular functionality*

Finally, there are a number of fundamenetal 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.

i. Model training procedure
   (a) i.e. whether to use QHL or quantum process tomography, etc.
   (b) In this work we always used QHL.

ii. Likelihood function: the method used to estimate the likelihood for use during QLE within QHL, which ultimately depends on the measurement scheme.
   (a) By default, here we use projective measurement back onto the input probe state, $\left|\langle\psi| e^{-i\hat{H}t} |\psi\rangle\right|^2$.
   (b) The role of these functions is to compute $\Pr(0)$ to be used by QInfer, which is not always the same as computing the likelihood, although these are equiavelent when we measure $d = 0$, see .
      i. Generically, without referring to the likelihood, these functions are to compute the expectation value of the unitary operator corresponding to the model of the system.
   (c) It is possible to change this to implement any measurement procedure, for example an experimental procedure where the environment is traced out.

iii. Probes: defining the input probes to be used during training.
   (a) In general it is preferable to use numerous probes in order to avoid biasing particular terms.
   (b) In some cases we are restricted to a small number available input probes, e.g. to match experimental constraints.

iv. Experiment design heuristic: bespoke experiments to maximise the information on which models are individually trained.
   (a) In particular, in this work the experimental controls consist solely of $\{|\psi\rangle, t\}$.
   (b) Currently, probes are generated once according to Section 6.4.4, 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.
   (c) 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 [4]. Alternatively, times may be chosen from a fixed set 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.

v. Model training prior: change the prior distribution, e.g. Fig. Fig. 5.1(a)

### 6.4.5 *Exploration strategy examples*

To solidifiy 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 compared within a branch. Recall that all of these strategies rely on QHL as the model training strategy, so that the run time for training, is $t_{\text{QHL}\sim N_e N_p t_U}$, 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. (5.4). Assume the conditions

- all models considered are represented by 4-qubit models;
  - $t_{U(4)} \sim 10^{-3}$sec.
- each model undergoes a reasonable training regime;
  - $N_e = 1000, N_p = 3000$;
  - $\implies t_{\text{QHL}} = N_e \times N_p \times t_{U(4)} = 3000s \sim 1\text{h}$;
- Bayes factor calculations use
  - $N_e = 500, N_p = 3000$
  - $\implies t_{\text{BF}} \sim 2 \times 500 \times 3000 \times 10^{-3} \sim 1\text{h}$;
- 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, comparing every pair of models
   (i) Training takes $1h$, and $\binom{16}{2} = 120$ comparisons need $8h$
   (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 $12 \times 9 = 108h$;
   (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 ESs, 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 approch in Section 6.4.5 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, Section 6.4.5 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 for all use–cases: specific quantum systems of study demand particular considerations, and the amount of prior information available informs how wide the model search should reach.

In this work we have used two straightforward model generation routines. Firstly, during the study of various physical classes (Ising, Heisenberg, Fermi-Hubbard), a list of lattice configurations were chosen in advance, which were then mapped to Hamiltonian models. This ES is non-adaptive and indeed the model search consists merely of a single branch with no subsequent calls to the model generation routine, as in Section 6.4.5. In the latter section instead we use a GA: this is clearly a far more general strategy, at a significant computational cost, but is suitable for systems where we have less knowledge in advance. In this case, new models are designed based heavily on results from earlier branches of the ET. The genetic algorithm model generation subroutine is listed in Algorithm 3, and can broadly be summed up thus: the best models in a generation $\mu$ produce offspring which constitute models on the next generation. These types of evolutionary algorithms ensure that newly proposed candidates inherit some of the structure which rendered previous candidates (relatively) successful, in the expectation that this will yield ever-stronger candidates.

## 6.5 GENERALITY

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

- *Model* can mean 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* is any subroutine which can train a given model, i.e. optimise a given parameterisation under the assumption that represents the target system.
  - Currently only QHL has been used, but tomography is valid in principle, albeit slower.
  - QHL relies on the calculation of a characteristic likelihood function; this too is not restricted to the generic form of Eq. (5.4) 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 11 where we trace out part of the system in order to represent open systems.

- *Model selection*, or *consoldation* can be as rigourous 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 signicative. However, in **??** we will consider a number of alternative schemes for discerning the strongest models.

### 6.5.1 *Agency*

While the concept of *agency* is contentious [27], we can view our overall protocol as a multi-agent system [28], or even an agent based evolutionary algorithm [29], 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 [30], 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 the designed actions and measures the outcome – the measurement of a datum following the experiment chosen by the EDH;
- a *critic*: the likleihood 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. 6.3. Finally, the model design strategy encoded in the ES *can* allow agency, by permitting the spawn rules autonomy, so we label the entire procedure as the quantum model learning agent.

## 6.6 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 1 shows the overall QMLA algorithm, which is simplified greatly to a loop over the MS of each ES. The MS iteself is listed in Algorithm 2, which contains calls to subroutines for model learning (QHL, Algorithm 4), branch evaluation (which can be based upon BF, Algorithm 5) and centers on the generation of new models, an example of which – based on a genetic algorithm – is given in Algorithm 3.

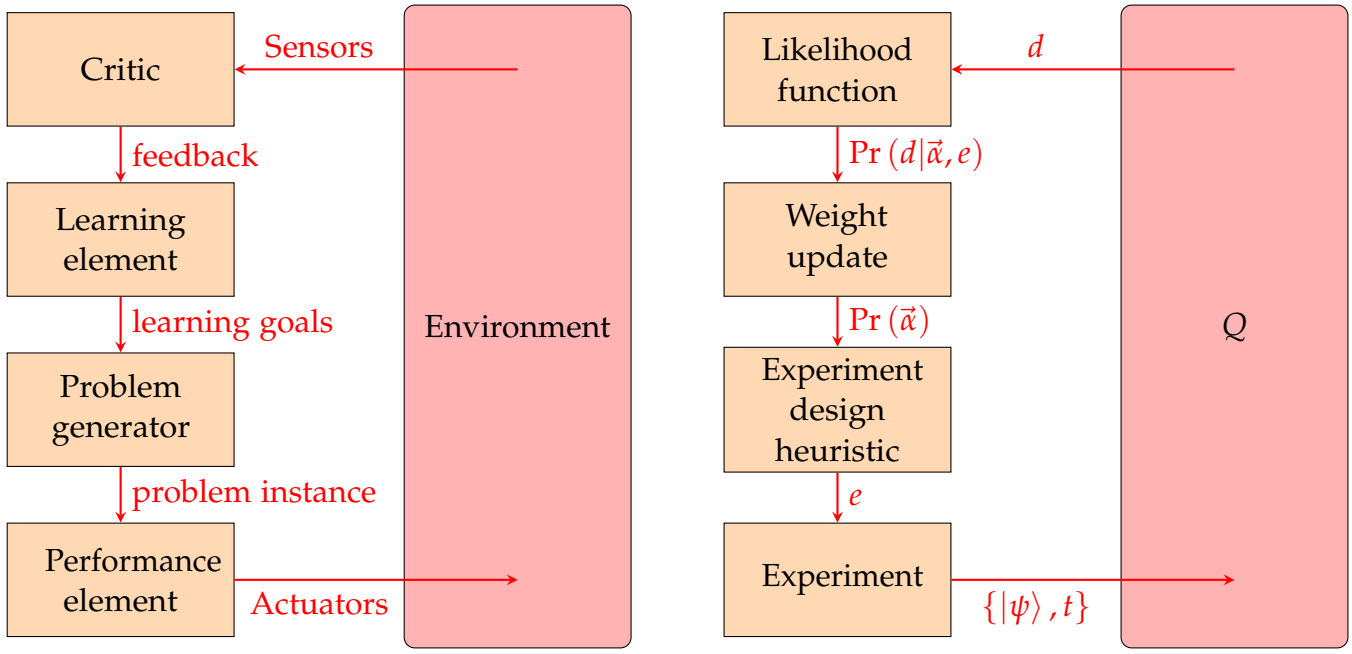


Figure 6.3: Learning agents. **Left**: definition of a learning agent, where an *environment* is affected by *actuators* which realise a *problem instance*, designed by a *problem generator*, through some *performance element*. The result of the agent's action is detected by *sensors*, which the *critic* 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.

**Algorithm 1:** Quantum Model Learning Agent

**Input:** $Q$ // some physically measurable or simulateable 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)
  $\mathbb{H}_c \leftarrow \mathbb{H}_c \cup \{\hat{H}'_S\}$ // add ES champion to collection
**end**
$\hat{H}' \leftarrow$ final_champion($\mathbb{H}_c$)
return $\hat{H}'$

**Algorithm 2:** ES subroutine: model_search

**Input:** $Q$ // some physically measurable or simulateable 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($\nu$) // e.g. Algorithm 3
  **for** $\hat{H}_i \in \mu$ **do**
    $\hat{H}'_i \leftarrow$ S.train($\hat{H}_i$) // e.g. Algorithm 4
  **end**
  $\nu \leftarrow$ S.evaluate($\mu$) // e.g. pairwise via Algorithm 5
  $\hat{H}^{\mu}_c \leftarrow$ S.branch_champion($\nu$) // use $\nu$ 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 3:** ES subroutine: generate_models (example: greedy spawn)

**Input:** $\nu$ // information about models considered to date

return $\mathbb{H}$

**Algorithm 4:** 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 epochs to iterate learning procedure for
**Input:** $N_p$ // number of particles to draw from $\Pr(\vec{\alpha})$
**Input:** $\Lambda(\Pr(\vec{\alpha}))$ // Heuristic algorithm which designs experiments
**Input:** $\mathrm{RS}(\Pr(\vec{\alpha}))$ // Resampling algorithm for redrawing particles
**Output:** $\vec{\alpha}'$ // estimate of Hamiltonian parameters

Sample $N_p$ times from $\Pr(\vec{\alpha}) \leftarrow \mathcal{P}$ // particles

**for** $e \in \{1 \rightarrow N_e\}$ **do**
  $t, |\psi\rangle \leftarrow \Lambda(\Pr(\vec{\alpha}))$ // design an experiment
  **for** $p \in \mathcal{P}$ **do**
    Retrieve particle $p \leftarrow \vec{\alpha}_p$
    Prepare $Q$ in $|\psi\rangle$, evolve and measure after $t \leftarrow d$ // datum
    $|\langle d| e^{-iH(\vec{\alpha}_p)t} |\psi\rangle|^2 \leftarrow Pr(d|\vec{\alpha}_p; t)$ // 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**
    $\mathrm{RS}(\Pr(\vec{\alpha})) \leftarrow \mathcal{P}$ // Redraw particles via resampling algorithm
  **end**
**end**
$\mathrm{mean}(\Pr(\vec{\alpha})) \leftarrow \vec{\alpha}'$
return $\vec{\alpha}'$

**Algorithm 5:** Bayes Factor calculation

**Input:** $Q$ // some physically measurable or simulateable quantum system.
**Input:** $\hat{H}'_j, \hat{H}'_k$ // Hamiltonian models after QHL (i.e. $\vec{\alpha}_j, \vec{\alpha}_k$ already optimised), on which to compare performance.
**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

$\mathcal{E} = \{\mathcal{E}_j \cup \mathcal{E}_k\}$
**for** $\hat{H}'_i \in \{\hat{H}'_j, \hat{H}'_k\}$ **do**
  $\mathcal{L}_i = 0$ // total log-likelihood of $\hat{H}_i$
  **for** $e \in \mathcal{E}$ **do**
    $e \leftarrow t, |\psi\rangle$ // assign time and probe from experiment control set
    Prepare $Q$ in $|\psi\rangle$, evolve and measure after $t \leftarrow d$ // datum
    $\left|\langle d| e^{-i\hat{H}'_i t} |\psi\rangle\right|^2 \leftarrow Pr(d|\hat{H}_i, t)$ // total likelihood for $\hat{H}'_i$ on $e$
    $log\left(Pr(d|\hat{H}_i, t)\right) \leftarrow l_e$ // log total likelihood for $\hat{H}'_i$ on $e$
    $\mathcal{L}_i + l_e \leftarrow \mathcal{L}_i$ // add $l_e$ to total log total likelihood
  **end**
**end**
$\exp\left(\mathcal{L}_j - L_k\right) \leftarrow B_{jk}$ // Bayes factor between models

return $B_{jk}$

# 7 SOFTWARE

All of the details in Chapter 5 and Chapter 6 are implemented in the QMLA software framework, a (mostly) Python codebase which underlies all of the arguments, results and figures in this thesis. The codebase is designed to simplify the process of running QMLA or QHL on novel systems. In particular, the core QMLA algorithm can support a wide range of ESs, allowing for the design of bespoke ESs to account for the specific requirements of any given system. In this chapter we give an overview of the QMLA software, implementation and instructions for its use.

## 7.1 IMPLEMENTATION

In this section we describe the technical details of the implementation of the algorithm described in Chapter 6, as well as a number of relevant subroutines. 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.

### 7.1.1 *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 [31], meaning that we can frame concepts as objects which permit actions to be performed by/to 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 an object, such that the player object has attributes, e.g. number of games played and goals scored in a season, as well as methods for 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* objects inherit 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 specify further details. 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 7.1 to 7.2.

```
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):
        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):
        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 7.1: Parent class, encoding the concept of an athlete.

```
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):
        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):
        self.goals_scored += num_new_goals


mickey = Footballer(
    name = 'Mickey',
    footed = 'left',
    team = 'QECDT-FC',
    birth_day = 1,
    birth_month = 1,
```

```
    birth_year = 1990,
    size = 'Big'
)
mickey.record_goals(num_new_goals = 10)
mickey.summarise()
```

Listing 7.2: Child class, encoding the concept of a footballer, which adopts the abstrat representation of an athlete.

## 7.2 PYTHON FRAMEWORK

A driving motivation for the development of QMLA is generality: we endevaour to make QMLA applicable to any target quantum system. We provide a framework, where users can tailor the inputs and methodology to their needs: we depict the main components of the framework in Fig. 7.1, 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.

### 7.2.1 *Application*

The application of QMLA refers to the choice of target system, $Q$, and how QMLA searches the MS in attempt to discover its model. As outlined in Section 6.4, ESs play the role of 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 $\mathrm{ExplorationStrategy}$ 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 6.4, as well as further details.

#### 7.2.1.1 *Modular functionality*

The most crucial methods[1] of the ES class are modular, described in Section 7.2.1.1, meaning that they can be directly replaced, provided the alternative method fulfils the same role. Our base ES class uses sensbile 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. By default, QHL calls a subroutine to compute Eq. (5.4). This can be replaced by any function which, given a Hamiltonian, evolution time and probe state,

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

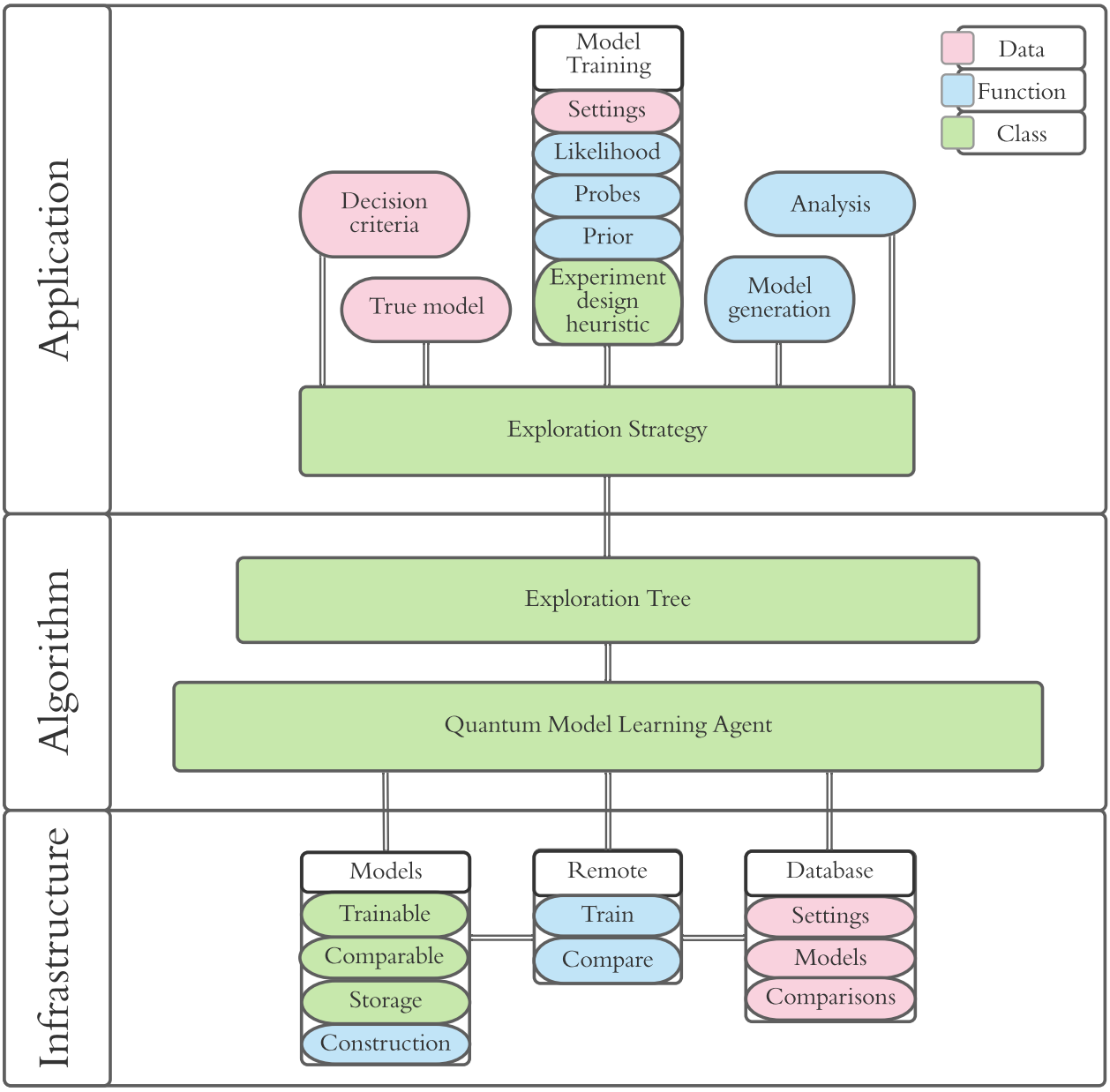


Figure 7.1: Overview of important *objects* in the QMLA framework. Colours encdoe the type of object: red objects are *data*, blue are *functions/methods* and green are *classes*. Objects are grouped broady, with double lines showing communication channels between (groups of) objects. *Infrastructure*: functions for the implementation of model training/comparisons on a remote server. *Algorithm*: implementation of the iterative procedures and decision-making laid out in Chapter 6. *Application*: inter-changeable data/functionality for the unique requirements of a given target system. Users wishing to customise QMLA must choose a valid object for each of those in *applications*, and need not alter any of the underlying framework.

returns the likelihood, according to the experiment you wish to simulate. For example, in Chapter 11, 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. Importantly, QInfer requires the quantity $\Pr(0)$, see Eq. (5.6), so that is the output of these functions.

- Probe generation. The training phase requires a set of probes against which to optimise individual models. Users may wish to specify the design of such probes, for example to match experimental constraints which restrict the realisible 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. We provide a base class implementing PGH, as well as child classes for each of the EDHs listed in Section 5.6.2.
- 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, [6], as well as detailing the true (target) model, $\hat{H}_0$, in the case where $Q$ is a simulated quantum system. We can also specify some ES-specific analyses to examine its internal performance, although this is generally required during development/testing, and less useful thereafter.

### 7.2.2 *Algorithm*

The algorithm layer of Fig. 7.1 implements the core steps of QMLA, as shown in Fig. 6.1, 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.

#### 7.2.2.1 *Parallel implementiation*

The implementation of QMLA seeks to separate the organisation of the MS 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. QMLA distributes jobs to worker processes in a server, i.e. we assume that QMLA is run on a machine with $N_c$ available parallel *processes*[2]. Then, the expensive calculations, namely training and comparing models, are not performed directly within the QMLA class, but instead are farmed out across the server. The role of QMLA then is to collate the outcome of

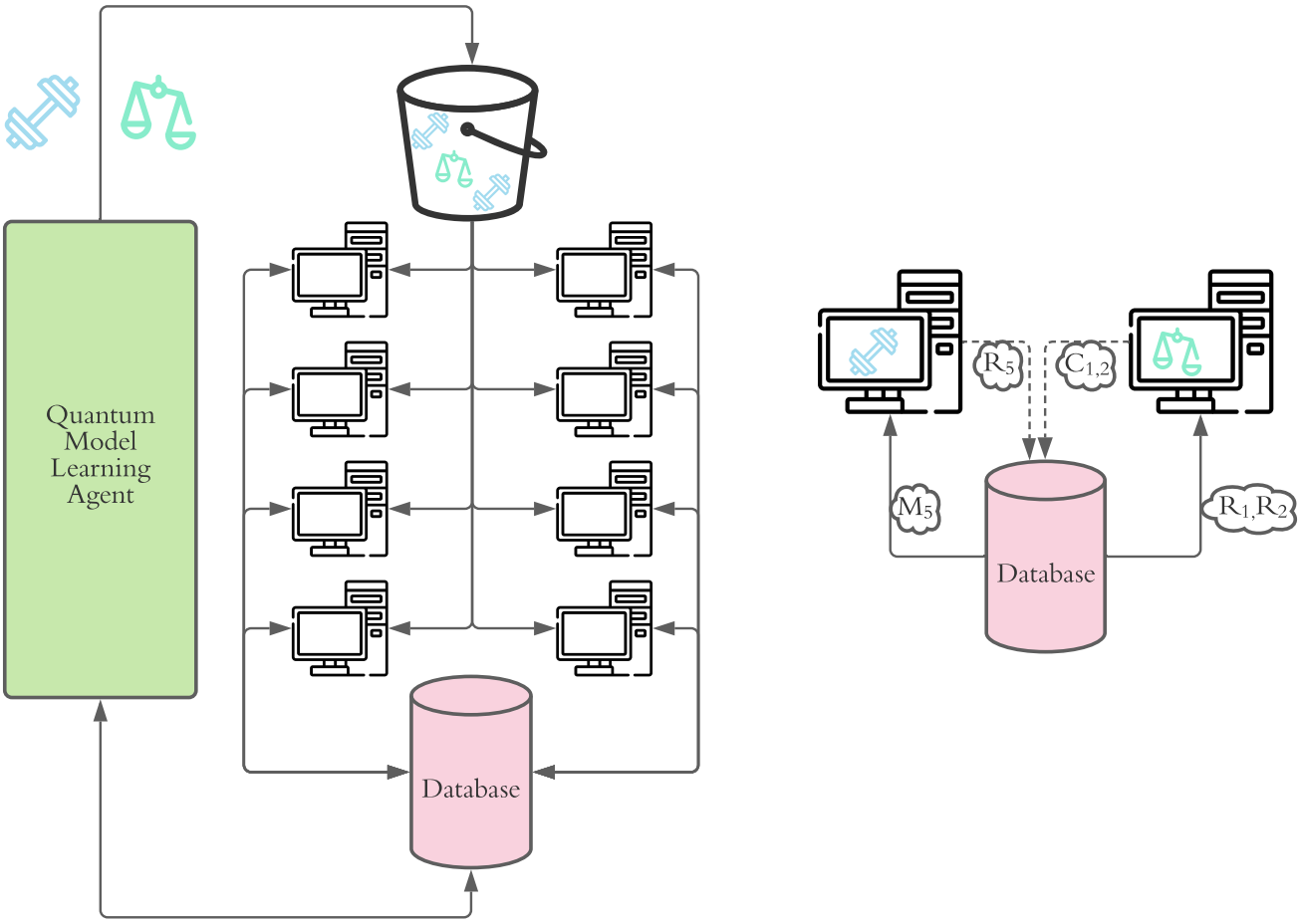


Figure 7.2: Parallel architecture for QMLA. **Left**, QMLA generates tasks – either to train (blue dumbbells) or compare (oragne scales) models – and places them in a task queue. Worker processes (depicted as computers) retrieve those tasks and compute them in parallel, and interact with a database. **Right**, Distributed tasks occuring in parallel. The left-hand process assumes the task of training the model with ID 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 model 5. After training, it sends a packet, $R_5$, summarising the result of model 5's training. The right-hand process compares two models with IDs 1 and 2, by first retrieivng the results packets $R_1, R_2$, then storing the comparison $C_{1,2}$ on the database.

those calculations in conjunction with the set of 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, e.g. to generate a series of plots which demonstrate how the model search progressed, as well as the evidence in favour of $\hat{H}'$, including for example the reproduction of $Q$'s dynamics by $\hat{H}'$. See **??** for further details.

While there are a number of strategies for parallelising code over a cluster, 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 [32]. 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. The QMLA class is hence assigned a single process solely for its considerations, 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 job.

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. There are two types of task for workers:

- to train a candidate model $\hat{H}_i$: the worker first requests some essenital information about the model from the databse, 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.
- 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 master QMLA class can also access said database, and 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 obviously the comparison can not begin until both $R_i, R_j$ are available. This is dealt with easily by using a *blocking* protocol, where 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 layer have been trained before queueing comparisons on that layer, to ensure a comparison can not start without the data needed to compute that comparison.

Models are assigned a unique ID upon creation; models 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 has already been trained, in which case it does not resubmit the model, but instead relies on the previous result. Likewise QMLA can check for the presence of any comparison result $C_{ij}$ before setting it as a new task, ensuring we do not duplicate expensive calculation.

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

We use a redis database and task queue [33, 34]. We depict the structure of this parallel architecture, and the master-worker strategy, in Fig. 7.2.

### 7.2.3 *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 QMLA.

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 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* classs is retained by QMLA for later use.

The tasks which actually implement the calculations (Section 7.2.2.1) are captured by standalone *remote* functions. These functions receive instructions such as `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 sending the compressed result to the database for storage.

To achieve this separation between calculation and analysis, we use a redis database [33], 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

## 7.3 USAGE

Several aspects of QMLA are *probabilistic*. Firstly, the Bayesian updates within model training of QHL relies on Likelihoods which implicitly depend on the measurement datum of a quantum system; in the case where the the measurement collapses $Q$ into the less-likely basis, the likelihoods will reflect that accurate hypotheses were poor, resulting in misguided posterior distributions. It is thus *possible* that the parameter learning will converge on incorrect values. Moreover, the model design subroutine is not gauranteed 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 on the incorrect terms, in the common situation where it does not know which constituent terms are helpful and which aren't.

Overall, then, it is pertinent to run QMLA many times and gather statistics about its performance, rather than making overly-strong claims about $Q$ based on a single *instance* of the algorithm. We say that a QMLA *run* consists of $N_r$ independent *instances*, which can be run in parallel, such that we are primarily concerned with the performance of the run instead of any

individual instance. For example, with $N_r = 100$, we can interpret the *win rate* of every model in the model space as evidence for that model being $\hat{H}_0$. 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 *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 10.2.2.

### 7.3.1 *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[3]. QMLA provides a large amount of anaytics to assess the performance of the protocol. These range from *big picture* perspectives such as the win rate across the entire run, to zooming in on the 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.

The user has control on which plots are generated, in order that the appropriate level of degree is presented, without producing an excessive number of images which can slow the protocol down. Results are categorised across the levels of the framework, for example:

- Run: results across a number of instances.
  - the number of instance wins for champion models.
  - average dynamics reproduced by champion models.
- Instance: performance of a single insance.
  - models generated and the branches on which they reside
- Model: Individual model performance within an instance.
  - parameter estimation through QHL.
- Comparisons Pairwise comparison of models' performance.
  - dynamics of both candidates (with respect to a single basis).
- Exploration strategy: figures specific to the ES
  - model generation metrics

3 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.

Most plots used in this thesis are generated directly by the QMLA framework[4]; the ES class and implementation parameters of each figure is listed in Table A.1.

4 Or are minor modifications of auto-generated plots.

# Part III

# THEORETICAL STUDY

# 8

# PRESCRIBED MODEL SETS

A sensible first case study for the QMLA framework is to prescribe a set of models, where we know that the true model is among them, or at least that we would be satisfied with approximating $\hat{H}_0$ as the best model in the set. 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. (5.4), 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, while miscalibrations will mainfest 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 find identify the miscalibrations. For example, by encoding connections between every pair of device qubits in $\hat{H}_0$, we can compose models with 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.

## 8.1 LATTICES

We first consider $Q$ as some lattice, where QMLA attempts to identify the structure of the lattice. The set of viable models then comprises alternative lattices. Due to simluation constraints, because we train models through exact unitary evolution, we are restricted to $\sim$ 8-qubit Hamiltonians, so we only consider lattices which can be simulated in this limit. The ES in this chapter is then simply to propose a set of models with no further model generation, with comparisons between all pairs of models through 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, $\hat{H}(\vec{\alpha}, \mathcal{C})$. 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. 8.1:

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

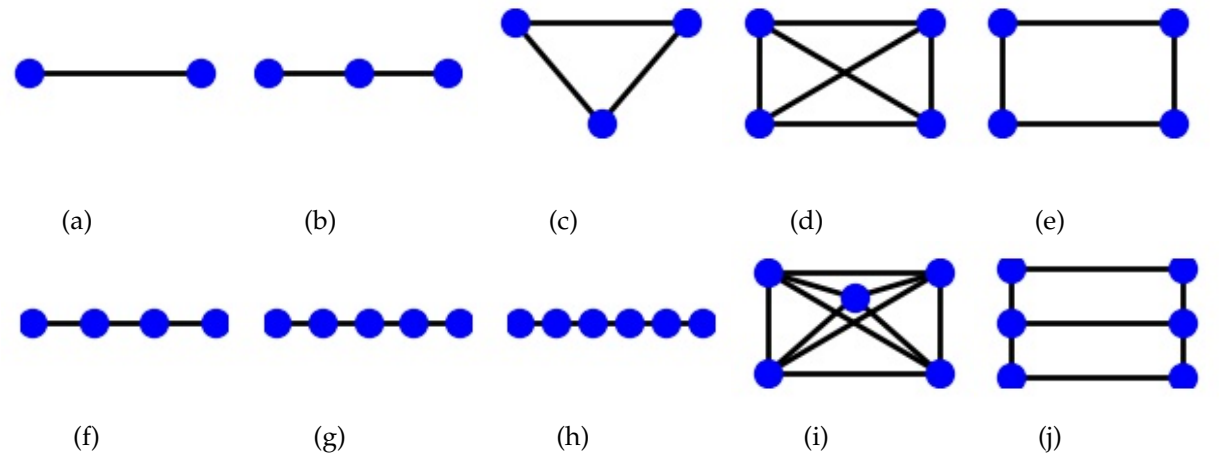


Figure 8.1: Lattices used for prescribed models test for QMLA. Lattices are characterised by the connectivity of their sites; dotted lines show connection between pairs of sites.

- 2-site chain
- 3-site chain
- 3-site fully connected (triangle)
- 4-site fully conneted (square)
- 4-site linearly connected (loop)
- 4-site chain
- 5-site chain
- 6-site chain
- 5-site fully connected (pentagon)
- 6-site partially connected (grid)

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

## 8.2 ISING MODEL

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 [35, 36, 37]. Interactions between spins $\langle k, l \rangle$ have strength $J_{kl}$, and the transverse magnetic field acts on spin $k$ with strength $h_k$. It is usually stated as

$$\hat{H}_I(\mathcal{C}) = \sum_{\langle k,l \rangle \in \mathcal{C}} J_{kl}\, \hat{\sigma}_k^z\, \hat{\sigma}_l^z + \sum_{k=1}^{N} h_k \hat{\sigma}_k^x. \tag{8.1}$$

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

$$\begin{cases} J_{kl} < 0, & \text{ferromagnetic;} \\ J_{kl} > 0, & \text{antiferromagnetic;} \\ J_{kl} = 0, & \text{noninteracting.} \end{cases} \tag{8.2}$$

If all interaction pairs are described by the same case in Eq. (8.2), the entire system can be said belong to that class of magnetism.

### 8.2.1 *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 neglecting the transverse magnetic field, and treating Ising model classically, such that the ground state is found by minimising the energy function

$$E_I = \langle \psi | H_I | \psi \rangle = \sum_{\langle k,l \rangle \in \mathcal{C}} J_{kl} \ \langle \psi | \hat{\sigma}_k^z \ \hat{\sigma}_l^z | \psi \rangle \,, \tag{8.3}$$

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. (8.3) consists only of $\hat{\sigma}^z$ terms, and we have that

$$\hat{\sigma}^z \, |+\rangle = +1 \, |+\rangle \quad ; \quad \hat{\sigma}^z \, |-\rangle = -1 \, |-\rangle \,. \tag{8.4}$$

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

$$\begin{aligned} \langle +_k \, +_l | \, \hat{\sigma}_k^z \ \hat{\sigma}_l^z \, | +_k \, +_l \rangle &= \langle +_k \, +_l | \, (+1)(+1) \, | +_k \, +_l \rangle = +1, \\ \langle +_k \, -_l | \, \hat{\sigma}_k^z \ \hat{\sigma}_l^z \, | +_k \, -_l \rangle &= \langle +_k \, -_l | \, (+1)(-1) \, | +_k \, -_l \rangle = -1, \\ \langle -_k \, +_l | \, \hat{\sigma}_k^z \ \hat{\sigma}_l^z \, | -_k \, +_l \rangle &= \langle -_k \, +_l | \, (-1)(+1) \, | -_k \, +_l \rangle = -1, \\ \langle -_k \, -_l | \, \hat{\sigma}_k^z \ \hat{\sigma}_l^z \, | -_k \, -_l \rangle &= \langle -_k \, +_l | \, (-1)(1) \, | -_k \, -_l \rangle = +1. \end{aligned} \tag{8.5}$$

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. (8.5), such that the energy function

$$E_I(\mathcal{S}) = \langle \psi | \hat{H}_I | \psi \rangle = \sum_{\langle k,l \rangle \in \mathcal{C}} J_{kl} \ \ s_k s_l \tag{8.6}$$

can be minimised by optimising the configuration $\mathcal{S}$, 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. (8.5), meaning we do not have to compute any unitary evolution to evaluate Eq. (8.6), it is still an expensive optimisation,

because effectively it is a search over $\{|\psi\rangle\}$, so the search space has $2^N$ candidates [36, 38]. This allows for a straightforward mapping between ground state search and solving combinatorial optimsiation algorithms, namely MAX-CUT, known to be NP-complete [39], allowing for proposed advantage in mapping computationally challenging problems to quantum hardware [40]. This mapping underlies ongoing research into quantum annealing as a computational platform capable of providing advantage for a specific family of problems [41, 42, 43].

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 *quantum mechanically*: instead of treating Eq. (8.1) as the underlying mechanism for a cost function to be optimised, i.e. Eq. (8.6), we use quantum operators and do not necessarily restrict the probe state $|\psi\rangle$, allowing us to use Eq. (8.1) within the likelihood function Eq. (5.4).

### 8.2.2 *Ising model cases*

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

$$\hat{H}(\mathcal{C}) = J \sum_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}_k^z \hat{\sigma}_l^z + h \sum_{k=1}^{N} \hat{\sigma}_k^x, \tag{8.7}$$

allowing for the compact representation, following Section 6.1,

$$\vec{\alpha}_I = \begin{pmatrix} J & h \end{pmatrix} \tag{8.8a}$$

$$\vec{T}_I = \begin{pmatrix} \sum\limits_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}_k^z \hat{\sigma}_l^z \\ \sum\limits_{k=1}^{N} \hat{\sigma}_k^x \end{pmatrix}. \tag{8.8b}$$

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

$$\mathcal{T}_I = \left\{ \hat{\sigma}_k^z \hat{\sigma}_l^z, \sum_{k=1}^{N} \hat{\sigma}_k^x \right\}_{\langle k,l\rangle \in \mathcal{C}}. \tag{8.9}$$

with unique parameters $J_{kl}$ associated with each interaction term $\hat{\sigma}_k^z \hat{\sigma}_l^z$. We summarise these cases in Table 8.1.

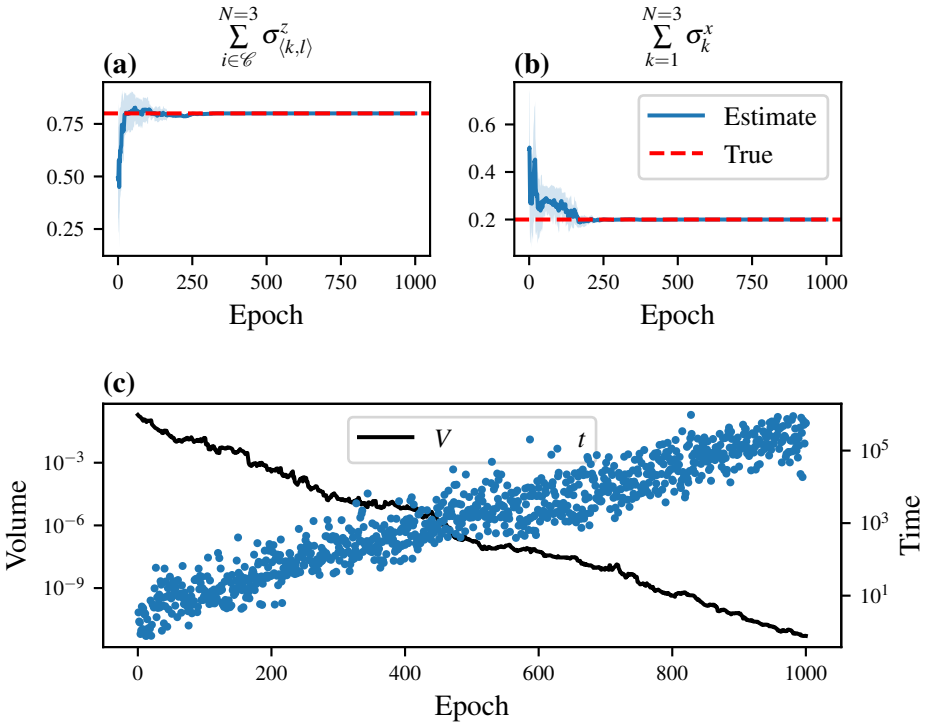


Figure 8.2: QHL for Ising model where terms are grouped by their functionality, as in Eq. (8.1). (a,b) show the parameter estimates progression against epochs (experiments), with the corresponding term written on top of the plot; (c) shows the volume of the parameter distribution at each epoch, as well as the evolution time chosen by the EDH. Implementation details are listed in Table A.1

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

Table 8.1: Types of Ising model. Varying whether parameters $J^z_{\langle k,l\rangle}, h_k$ are shared across sites gives distinct models.

We first construct models under each of these forms to verify QHL is capable of learning in this regime. The former case is the standard form of the Ising model; its training is shown in Fig. 8.2, while the fully paramterised model is shown in Fig. 8.3. Ultimately, these two cases give the same Hamiltonian when $J_{\langle k,l\rangle} = J;\ h_k = h\ \forall k,l$. So, 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 learaned the parameters to high precision, and neither model converged; the volume continues to reduce

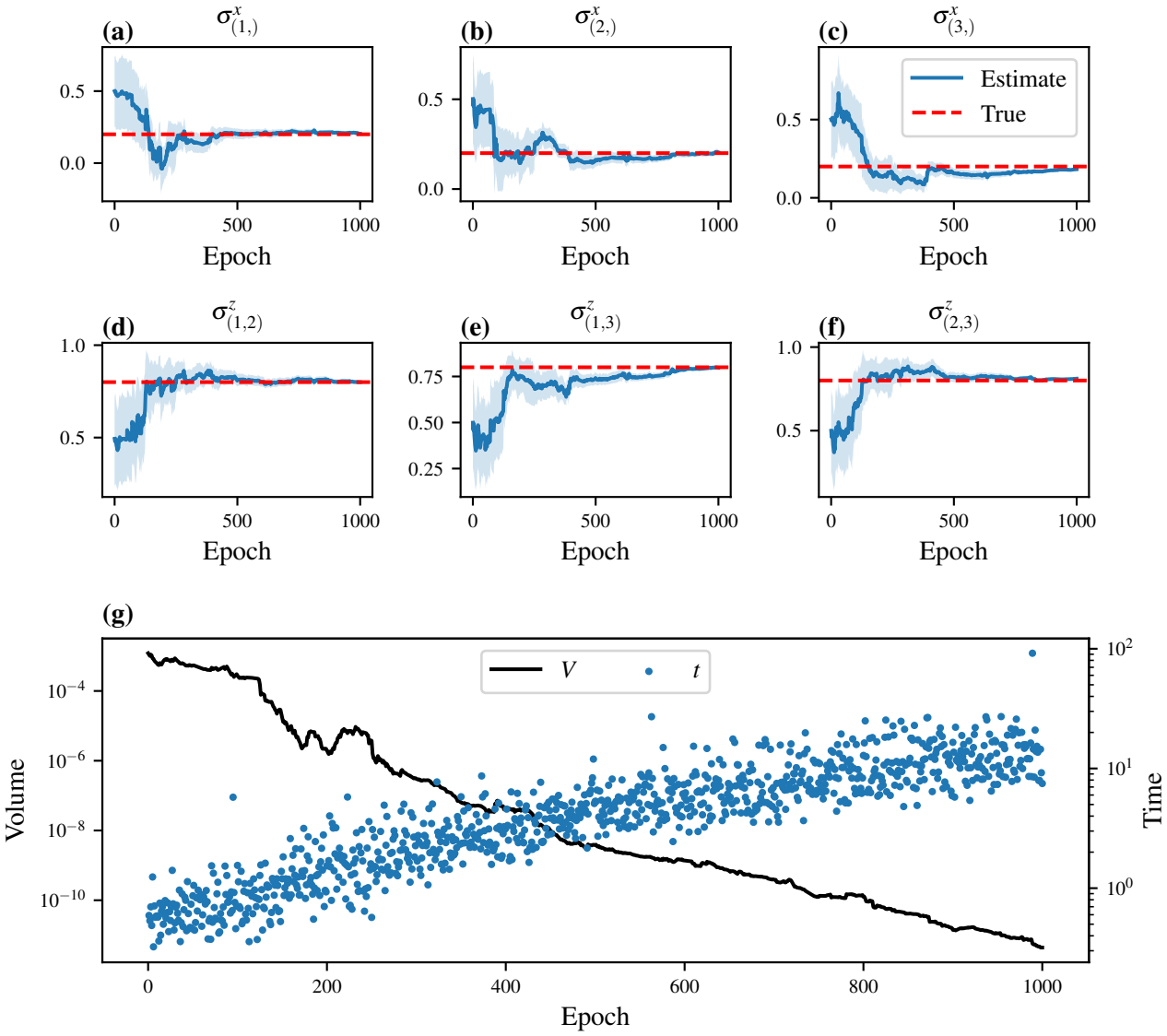


Figure 8.3: QHL for fully parameterised Ising model, where every interaction between pairs of sites are assigned unique parameters, here neglecting the transverse field, as in Eq. (8.9). (a)-(f) show the parameter estimates progression against epochs (experiments), with the corresponding term written on top of the plot; (g) shows the volume of the parameter distribution at each epoch, as well as the evolution time chosen by the EDH. Implementation details are listed in Table A.1

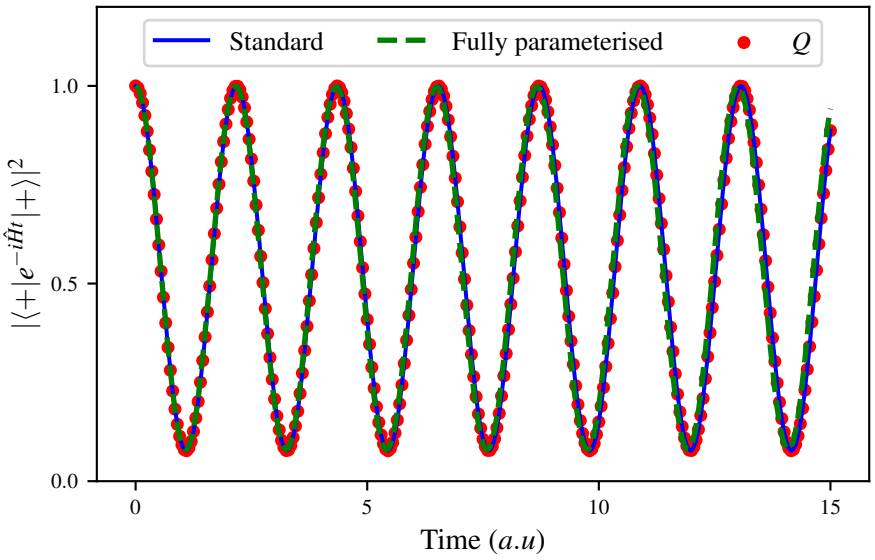


Figure 8.4: Dynamics reproduced by Ising models under standard and fully parameterised formalisms, compared with dynamics for the true system. Implementation details are listed in Table A.1

exponentially in both cases, suggesting it may be impractical to seek saturation in the model training phase for every model, since this may require a very large number of experiments and particles.

The dynamics produced by both models are shown in Fig. 8.4: 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 $10^{19}$. 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 parmaeters, and are difficult to distinguish through human intuition.

## 8.3 HEISENBERG MODEL

Generalising the Ising model, the Heisenberg Hamiltonian is another model for magnetic systems consisting of a set of spins on a lattice [44]. It builds on the Ising model by additionally considering the spins' rotations about the $x-$ and $y-$ axes, generally stated as

$$\hat{H}_H(\mathcal{C}) = \sum_{\langle k,l\rangle\in\mathcal{C}} J^x_{kl}\,\hat{\sigma}^x_k\hat{\sigma}^x_l + \sum_{\langle k,l\rangle\in\mathcal{C}} J^y_{kl}\,\hat{\sigma}^y_k\hat{\sigma}^y_l + \sum_{\langle k,l\rangle\in\mathcal{C}} J^z_{kl}\,\hat{\sigma}^z_k\hat{\sigma}^z_l + \sum_{k=1}^{N} h_k\hat{\sigma}^z_k. \tag{8.10}$$

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 the instances within the family of Heisenberg models in Table 8.2.

| | $J^x_{kl}$ | $J^y_{kl}$ | $J^z_{kl}$ | $h_k$ |
|---|---|---|---|---|
| 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 8.2: Heisenberg model types: varying whether the interaction parameters $J^w_{kl}$ are shared among pairs of spins give distinct descriptions which are all in the family of Heisenberg models.

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. In this chapter we will refer to the Heisenberg-XYZ model, and will consider the fully parameterised Heisenberg model in Chapter 10; the parameters and terms of interest are then captured by Eq. (8.11).

$$\vec{\alpha}_H = \begin{pmatrix} J^x & J^y & J^z & h \end{pmatrix} \tag{8.11a}$$

$$\vec{T}_H = \begin{pmatrix} \sum\limits_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}^x_k \hat{\sigma}^x_l \\ \sum\limits_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}^y_k \hat{\sigma}^y_l \\ \sum\limits_{\langle k,l\rangle \in \mathcal{C}} \hat{\sigma}^z_k \hat{\sigma}^z_l \\ \sum\limits_{k=1}^{N} \hat{\sigma}^z_k \end{pmatrix} \tag{8.11b}$$

## 8.4 HUBBARD MODEL

Another representation of solid state matter systems is given by the Hubbard model [45, 46, 47]. The Hubbard model deals with systems of correlated fermions, allowing spins to *hop* between sites. Note the Hubbard model is synonymous with the Fermi-Hubbard (FH) model, which can be used to distinguish this model of fermions from a similar model of bosons, named the Bose-Hubbard model, which is not studied in this thesis. We use the subscript FH to distinguish

the (Fermi-)Hubbard model from the Heisenberg model $\hat{H}_H$, Eq. (8.10). The Hubbard model is generally stated in second quantisation as

$$\hat{H}_{FH}(\mathcal{C}) = \sum_{s\in\{\uparrow,\downarrow\}} \sum_{\langle k,l\rangle\in\mathcal{C}} t^s_{\langle k,l\rangle} \left(\hat{c}^\dagger_{ks} c_{ls} + \hat{c}^\dagger_{ls} c_{ks}\right) + \sum_k^N U_k \hat{n}_{k\uparrow}\hat{n}_{k\downarrow} + \sum_k^N \mu_k \left(\hat{n}_{k\uparrow} + \hat{n}_{k\downarrow}\right) \tag{8.12}$$

where

- $\hat{c}_{ks}$ and $\hat{c}^\dagger_{ks}$ are respecitvely 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 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 $k$;
- $\mu_k$ is the chemical energy for $k$;
- $N$ is the number of sites in the system.

Again, we can achieve differing physics by controlling whether the parameters are shared, 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 8.3; we will use the standard form in this chapter, i.e.

$$\vec{\alpha}_{FH} = (t^\uparrow \quad t^\downarrow \quad U \quad \mu) \tag{8.13a}$$

$$\vec{T}_{FH} = \begin{pmatrix} \sum_{\langle k,l\rangle\in\mathcal{C}} (\hat{c}^\dagger_{k,\uparrow}\hat{c}_{l,\uparrow} + \hat{c}^\dagger_{l,\uparrow}\hat{c}_{k,\uparrow}) \\ \sum_{\langle k,l\rangle\in\mathcal{C}} (\hat{c}^\dagger_{k,\downarrow}\hat{c}_{l,\downarrow} + \hat{c}^\dagger_{l,\downarrow}\hat{c}_{k,\downarrow}) \\ \sum_{k=1}^N \hat{n}_{k\uparrow}\hat{n}_{k\downarrow} \\ \sum_{k=1}^N \left(\hat{n}_{k\uparrow} + \hat{n}_{k\downarrow}\right) \end{pmatrix} \tag{8.13b}$$

| | $t^\uparrow_{\langle k,l\rangle}$ | $t^\uparrow_{\langle k,l\rangle}$ | $U_k$ | $\mu_k$ |
|---|---|---|---|---|
| Standard | $t^\uparrow$ | $t^\downarrow$ | $U$ | $\mu$ |
| Fully parameterised | $t^\uparrow_{\langle k,l\rangle}$ | $t^\downarrow_{\langle k,l\rangle}$ | $U_k$ | $\mu_k$ |

Table 8.3: Types of Hubbard model. Varying whether parameters $t^s_{\langle k,l\rangle}, U_k, \mu_k$ are shared across sites gives distinct models.

### 8.4.1 *Jordan Wigner transformation*

In order that the Hubbard model is simulateable with qubits[2], 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) [48, 49]. We implement the JWT within QMLA through OpenFermion's fermilib package [50].

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*,

$$|\psi_f\rangle = |n_{m_1}, n_{m_2}, \dots, n_{m_n}\rangle\,, \tag{8.14}$$

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 from (to) $m_i$:

$$\hat{c}^\dagger_{m_i}|\psi_f\rangle = |n_{m_1}, \dots, n_{m_i}+1, \dots, n_{m_n}\rangle\,, \tag{8.15a}$$

$$\hat{c}_{m_i}|\psi_f\rangle = |n_{m_1}, \dots, n_{m_i}-1, \dots, n_{m_n}\rangle\,. \tag{8.15b}$$

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

$$|\psi_{FH}\rangle = |n_{1\uparrow}, n_{1\downarrow}, \dots, n_{N\uparrow}, n_{N\downarrow}\rangle\,. \tag{8.16}$$

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 oeprators $\hat{\sigma}^+, \hat{\sigma}^-$ such that

$$\hat{\sigma}^+ = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \implies \hat{\sigma}^+|0\rangle = |1\rangle \tag{8.17a}$$

$$\hat{\sigma}^- = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \implies \hat{\sigma}^-|1\rangle = |0\rangle \tag{8.17b}$$

Then, to map the number basis of Eq. (8.16) 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 8.4. The JWT can be summarised by mapping – for the mode $m$ – the

2 Or simulations of qubits, as in this thesis.

| Mode | Site | Spin | Qubit |
|---|---|---|---|
| 1 | 1 | ↑ | 1 |
| 2 | 1 | ↓ | 2 |
| 3 | 2 | ↑ | 3 |
| 4 | 2 | ↓ | 4 |
| | ⋮ | | |
| $2N-1$ | $N$ | ↓ | $2N-1$ |
| $2N$ | $N$ | ↑ | $2N$ |

Table 8.4: Jordan Wigner mode/qubit indices.

creation (annihilation) operator $\hat{c}_m^\dagger$ ($\hat{c}_m$), to an operator which adds a spin to the corresponding state through the operator $\hat{\sigma}_m^+$ ($\hat{\sigma}_m^-$).

$$\hat{c}_m \to (\hat{\sigma}^z)^{\otimes k-1} \otimes \hat{\sigma}^- \otimes (\hat{\sigma}^z)^{\otimes 2N-1} \tag{8.18a}$$

$$\hat{c}_m^\dagger \to (\hat{\sigma}^z)^{\otimes k-1} \otimes \hat{\sigma}^+ \otimes (\hat{\sigma}^z)^{\otimes 2N-1} \tag{8.18b}$$

Note the JWT acts on all modes/qubits other than the target with $\hat{\sigma}^z$, since

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

$$\hat{c}_{2\uparrow}^\dagger |0000\rangle = \hat{c}_3^\dagger |0000\rangle = \hat{\sigma}_1^z \hat{\sigma}_2^z \hat{\sigma}_3^+ \hat{\sigma}_4^z |0000\rangle = |0010\rangle \tag{8.19}$$

### 8.4.2 *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. (8.16) is effectively projected into the subspace spanned by half-filled basis states. For example, with $N = 2$

$$\{|1100\rangle, |1010\rangle, |1001\rangle, |0101\rangle, |0110\rangle, |0011\rangle\} \tag{8.20}$$

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

## 8.5 MODEL LEARNING FOR LATTICES

Finally, then, we can use the lattice systems introduced in Sections 8.1 to 8.4 as first case studies for QMLA. Each $\mathcal{C} \in \mathbb{C}$ can specify a unique model under the standard model formalism for

each of Ising (Eq. (8.8)), Heisenberg (Eq. (8.11)) and Hubbard (Eq. (8.13)) models. We can then devise a simple ES which only tests the models corresponding to $\mathbb{C}$, with no further model generation, i.e. Algorithm 6, and compares every pair of models through BF, deeming the champion as that which wins the largest number of comparisons, as in Algorithm 7.

**Algorithm 6:** Lattice exploration strategy: model generation

**Input:** $\mathbb{C}$ // Set of lattice configurations
**Output:** $\{\hat{H}_i\}$ // Set of models to tests

$\mathbb{H} = \{\ \}$
**for** $\mathcal{C} \in \mathbb{C}$ **do**
  $\hat{H}_i \leftarrow$ map_lattice_to_model($\mathcal{C}$)
  $\mathbb{H} \leftarrow \mathbb{H} \cup \{\hat{H}_i\}$
**end**
return $\mathbb{H}$

**Algorithm 7:** 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 5
    **if** $B_{ij} > 1$ **then**
      $s_i \leftarrow s_i + 1$ // $\hat{H}_i$'s score increases if it is the stronger model
    **end**
  **end**
**end**
$\hat{H}' \leftarrow \arg\max_{s_i} (\hat{H}_i)$
return $\hat{H}'$

For example, we adopt the fully connected four site lattice ($d$ in Fig. 8.1) as the true lattice specifying $\hat{H}_0$, under the Ising formalism (Eq. (8.7)). We run QMLA by training the ten models corresponding to the ten lattices, Fig. 8.5**a-b**; comparing the models predictive power, Fig. 8.5**c-d**,

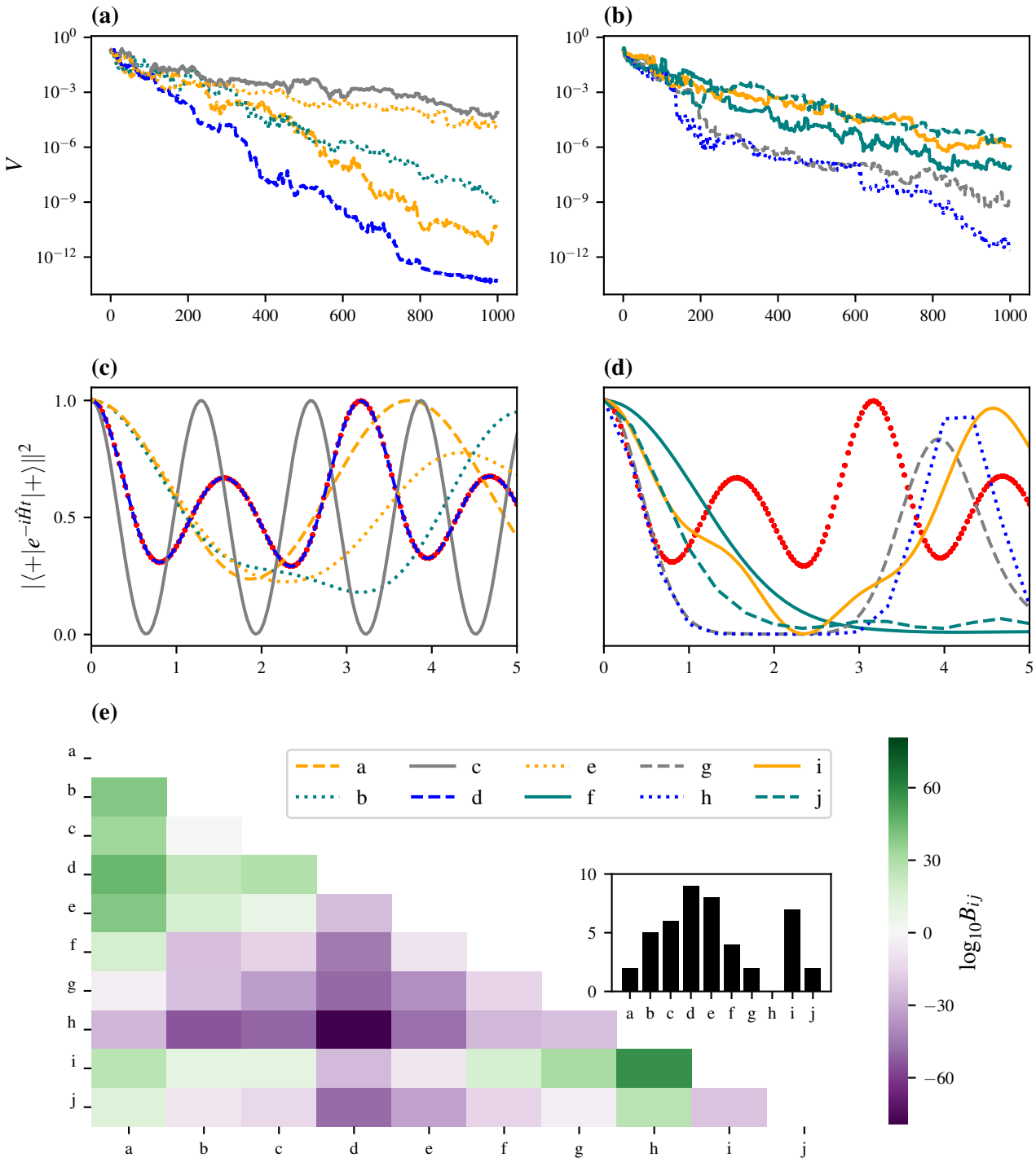


Figure 8.5: QMLA for set of lattices under Ising formalism. The lattice indices correspond to those in Fig. 8.1, and the true system is given by lattice $d$. (a,b) show the decrease in volume for each model's training phase. (spread over two plots for readibility) (c,d), trained models are used to reproduce dynamics, compared with the dynamics of the true system. (e) Heatmap of $\log_{10} B_{ij}$ between every pair of models. The BF is read as $i$ versus $j$, where $i$ is the model on the $y$-axis and $j$ 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. The inset shows the number of BF comparisons won by each model, i.e. the models' scores. Implementation details are listed in Table A.1

through BF (Fig. 8.5**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, and is hence determined as $\hat{H}'$.

## 8.6 COMPLETE QUANTUM MODEL LEARNING AGENT RUNS FOR LATTICE SETS

In order to test QMLA robustly, we can use each of the lattices shown in Fig. 8.1 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[3]. Moreover, we can extend this test to the Heisenberg and Hubbard formalisms; note that due to the overhead given by the JWT (Section 8.4.1), i.e. the requirement of two qubits per site, we restrict study of the Hubbard model to lattices $a - e$ for practicality. 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. 8.6, finding in all cases that QMLA identifies $\hat{H}_0$ with success rates at least 70%.

We take this test case as evidence that the BF is a fair mechanism by which to distinguish between models. 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. Then, 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: 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.

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

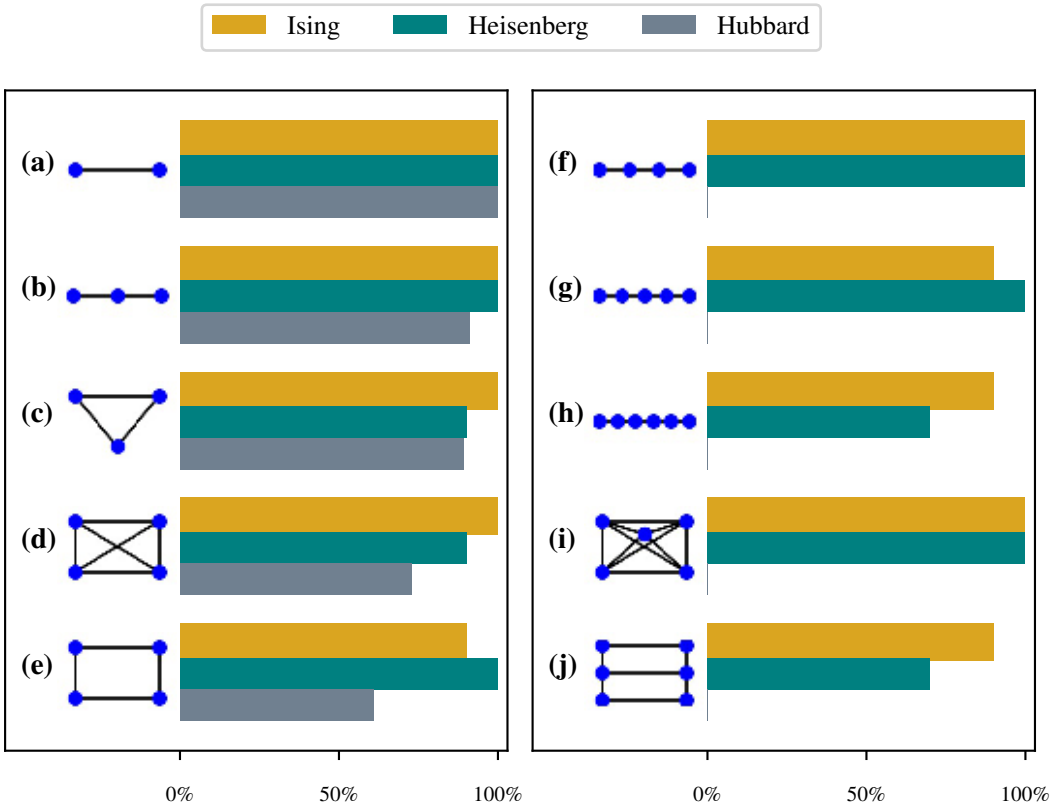


Figure 8.6: Rates of success for QMLA under various conditions. Each lattice is set as the true model $\hat{H}_0$ for ten independent instances. In each instance, the ES considers the available lattices (**a-j** for Ising and Heisenberg cases and **a-e** for the Hubbard case), and selects a champion 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

# 9

# BLACK BOX QUANTUM SYSTEMS

# 10 GENETIC ALGORITHMS

The QMLA framework lends itself easily to the family of optimsation techniques called *evolutionary algorithms*, where individuals, sampled from a population of candidates, are considered, in generations, as solutions to the given problem, and iterative generations aim to efficiently search the available population, by mimicing biological evolutionary mechanisms [51]. In particular, we develop a ES which incorporates an GA in the generation 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 [52]. Here we will first introduce the concept of an GA, before describing the adaptations which allow us to build a genetic exploration strategy (GES).

## 10.1 GENETIC ALGORITHM DEFINITION

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 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, mimicing the biological mechanism whereby the genetic makeup of offspring is an even mixture of both parents. 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. Buidling 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.

Pseudocode for a generic GA is given in Algorithm 8, but we can also informally define the procedure as follows. Given access to the population, $\mathcal{P}$,

1. Sample $N_m$ candidates from the population at random
   (a) call this group of candidates the first generation, $\mu$.
2. Evaluate each candidate $\gamma_j \in \mu$.
   (a) each $\gamma_j$ is assigned a fitness, $g_j$
   (b) the fitness is computed through an objective function acting on the candidate, $g(\gamma_j)$.
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, $p_1, p_2$, from $\mu$
      i. Each candidate's probability of being chosen is given by their $s_j$
   (c) Cross over $p_1, p_2$ to produce children candidates, $c_1, c_2$.
      i. mutate $c_1, c_2$ according to some random probabilistic process
      ii. keep $c_i$ only if it is not already in $\mu$, 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 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.

### 10.1.1 *Example: knapsack problem*

One commonly referenced combinatorial optimisation problem is the *knacksack problem*: given a set of objects, where each object has a defined mass and also a defined 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 object sets as candidate vectors $\vec{\gamma}_j$, whose genes are binary, and simply indicate whether or not the associated object is included in the set. For example, with $n = 6$,

$$\gamma_j = 100001 \Longrightarrow \vec{\gamma}_j = (1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1), \tag{10.1}$$

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[1] if the weight of the corresponding set of objects is less than the capacity of the knapsack, i.e. $\vec{w}_j \cdot \vec{\gamma}_j \leq w_{max}$.

1 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.

**Algorithm 8:** Genetic algorithm

**Input:** $\mathcal{P}$ // Population of candidate models
**Input:** $g()$ // objective funtion
**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:** $N_g$ // number of generations
**Input:** $N_m$ // number of candidates per generation

**Output:** $\gamma'$ // 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' $s_j$
    $c_1, c_2 \leftarrow$ crossover$(p_1, p_2)$ // generate offspring candidates based on parents
    **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 $\gamma'$

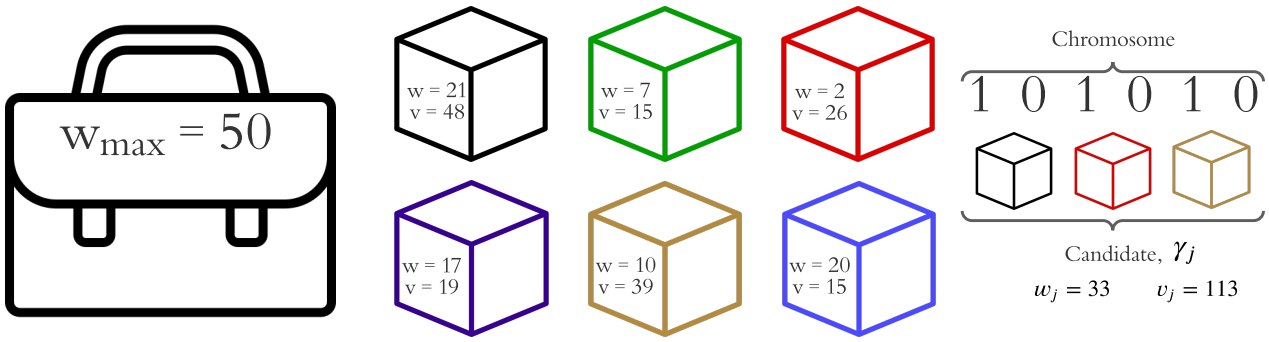


Figure 10.1: Depiction of the knapsack problem. **Left**, A knapsack which can hold any number of objects but is constrained by the total weight it can support, $w_{max} = 50$. **Centre**, A set of objects are available, each with associated weight, $w$, and value $v$. The objective is to find the subset of objects which maximise the total value, while not exceeding the capacity of the knapsack. **Right**, An example chromosome, i.e. candidate $\gamma_j$, where the bits of the chromosome indicate whether the corresponding object is included, allowing for calculation of the total weight and value of the candidate, $w_j, v_j$.

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

$$\vec{v} = (48 \quad 15 \quad 26 \quad 19 \quad 39 \quad 15) \implies v_j = \vec{\gamma}_j \cdot \vec{v} = 48 + 15 = 63; \tag{10.2a}$$

$$\vec{w} = (21 \quad 7 \quad 2 \quad 17 \quad 10 \quad 20) \implies w_j = \vec{\gamma}_j \cdot \vec{w} = 21 + 20 = 41. \tag{10.2b}$$

We can hence assess the fitness of $\gamma_j$ 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.

| Name | Candidate | Value | Weight | Valid |
|---|---|---|---|---|
| $\gamma_1$ | 110011 | 117 | 58 | No |
| $\gamma_2$ | 101010 | 113 | 33 | Yes |
| $\gamma_3$ | 011110 | 99 | 36 | Yes |
| $\gamma_4$ | 011011 | 95 | 39 | Yes |
| $\gamma_5$ | 111000 | 89 | 30 | Yes |
| $\gamma_6$ | 010111 | 88 | 54 | No |
| $\gamma_7$ | 100010 | 87 | 31 | Yes |
| $\gamma_8$ | 110001 | 78 | 48 | Yes |
| $\gamma_9$ | 011101 | 75 | 46 | Yes |
| $\gamma_{10}$ | 110000 | 63 | 28 | Yes |
| $\gamma_{11}$ | 000011 | 54 | 30 | Yes |
| $\gamma_{12}$ | 000101 | 34 | 37 | Yes |

Table 10.1: Candidate solutions to the knapsack problem.

The strongest (valid) candidates from Table 10.1 are $101010$, $011110$. By spawning from these candidates through a one-point crossover at the midpoint, we get $\gamma_{c_1} = 101110, \gamma_{c_2} = 011010$, 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 to consider, 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 which say $\sim 1000$ 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[2].

### 10.1.2 *Selection mechanism*

A key subroutine of every GA is the mechanism through which it nominates candidates from generation $\mu$ as parents to offsping candidates in $\mu + 1$ [53]. All mechanisms have in common that they act on a set of candidates from the previous generation, where each candidate, $\gamma_j$, has

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

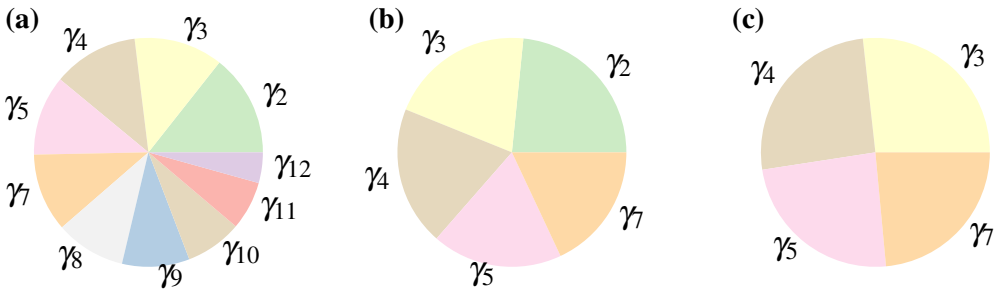


Figure 10.2: Roulette wheels showing selection probability $s_i$ for corresponding candidates $\gamma_i$. Colours here only distinguish candidates, they do not encode any information. **b**, The set of potential parents is truncated to include only the strongest five candidates. **a**, All valid candidates are assigned selection probability based on their value in Table 10.1. **c**, After one parent ($\gamma_2$) has been chosen, it is removed from the roulette wheel and the remaining candidates' probabilities are renormalised for the selection of the second parent.

been evaluated and has fitness value, $g_j$. Among the viable schemes for selecting individual parents from the set of candidates, $\mu$ 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 $\mu$, 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 later within QMLA, the common fitness proportional selection, known as *roulette selection* [53]. This is a straightforward strategy where we directly map candidates' fitness, $g_i$ to a selection probability, $s_i$, simply by normalising $\{g_i\}$, allowing us to visualise a roulette wheel of uneven wedges, eachh of which correspond to a candidate. Then we need only conceptually spin the roulette wheel to select the first parent, $\gamma_{p_1}$. We then remove $\gamma_{p_1}$ from the set of potential parents, renormalise the remaining $\{s_i\}$, and spin the wheel again to choose the second parent, $\gamma_{p_2}$.

Practically, we repeat the process outlined until the next generation is filled, usually we have $|\mu| = N_m$, and desire that every generation should contain the same $N_m$ candidates, so we repeat the roulette selection $N_m/2$ times per generation, since every pair of parents yield 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 ten models, 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

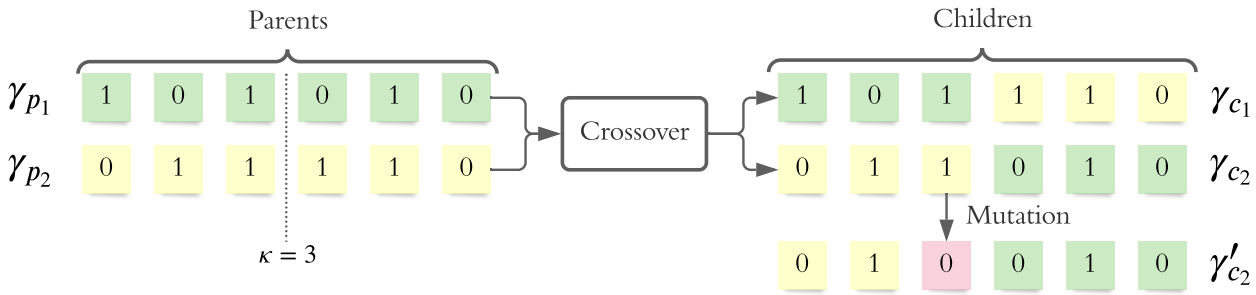


Figure 10.3: Crossover and mutation of chromosomes. Two parents, $\gamma_{p_1}, \gamma_{p_2}$, are nominated from the process in Fig. 10.2. They are then crossed-over via a one-point crossover with crossing point $\kappa = 3$, resulting in children candidates $\gamma_{c_1}, \gamma_{c_2}$. One child chromosome is mutated to yield a new candidate, $\gamma'_{c_2}$. The candidates added to the next generation are then $\{\gamma_{c_1}, \gamma'_{c_2}\}$.

$\mu$ and retain only the highest-fitness $\frac{N_m}{2}$ models as selectable parents. The roulette selection is shown in Fig. 10.2.

### 10.1.3 *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, $\kappa$: for candidates of $n$ genes, the first $\kappa$ genes of $\gamma_{p_1}$ are conjoined with the latter $n - \kappa$ genes of $\gamma_{p_2}$. Often $\kappa$ is restricted to the midpoint of the chromosomes, although in general we need not impose this: 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. 10.3, recalling the chromosome structure from Section 10.1.1.

By allowing $\kappa$ 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 potentital 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 10.1, for example:

| Parent 1 | Parent 2 | $\kappa$ | $s_{ij}$ |
|---|---|---|---|
| $\gamma_2$ | $\gamma_3$ | 2 | $11,187\ (= 113 \times 99)$ |
| $\gamma_2$ | $\gamma_3$ | 3 | $11,187$ |
| $\gamma_2$ | $\gamma_3$ | 4 | $11,187$ |
| $\gamma_2$ | $\gamma_4$ | 2 | $10,735\ (= 113 \times 95)$ |
| $\gamma_2$ | $\gamma_4$ | 3 | $10,735$ |
| $\gamma_2$ | $\gamma_4$ | 4 | $10,735$ |
| | | $\vdots$ | |
| $\gamma_5$ | $\gamma_7$ | 2 | $7,743\ (= 89 \times 87)$ |
| $\gamma_5$ | $\gamma_7$ | 3 | $7,743$ |
| $\gamma_5$ | $\gamma_7$ | 4 | $7,743$ |

Table 10.2: 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 $\kappa$, and all $\kappa$ are equally likely.

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 [54], but for brevity we only describe those used in this thesis. For each proposed child candidate, $\gamma_c$, we probabilistically mutate each gene with some mutation rate $r_m$: if a mutation occurs, the child is replaced by $\gamma_c'$. That is, $\gamma_c'$ is added to the next generation, and $\gamma_c$ is discarded. $r_m$ is a *hyperparameter* of the GA: the performance of the algorithm can be optimised by finding the best $r_m$ for a given problem.

### 10.1.4 *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 used above, candidates were evaluated by the value of their contennts, but also by whether they would fit in the knapsack. Idenitfiyng the appropriate method by which to evaluate candidates is arguably the most important aspect of designing a GA: while the choice of hyperperameters ($N_g, N_m, r_m$) 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 (OF).

Unlike previous aspects of generic GAs, in the context of QMLA, here we must deviate from default mechanisms. Recalling the overarching goal of QMLA, to characterise some black box quantum system, $Q$, we do not have access to a natural OF. We wish to optimise the modelling of $\hat{H}'$, but assume we do not know the target $\hat{H}_0$, so we can not simply invoke some loss function, 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 an ES based on a GA. We propose and discuss a number of options in Section 10.3.

Common to all proposed OFs, however, is that candidates should first be trained before evaluation, so that their assessment is based on their actual power in explaining the target system, rather than some initial paramterisation which may not capture their potential. This is a tenet of QMLA: for each candidate $\hat{H}_j(\vec{\alpha}_j)$, we use a subroutine to optimise $\vec{\alpha}_j$, again for this study we rely on QHL.

## 10.2 ADAPTATION TO QMLA FRAMEWORK

Ultimately, the conceived role of a GA within QMLA is to generate the sets of models to place on successive branches of the ETs in Fig. 6.1. The apparatus for this is to implement an exploration strategy (ES) whose model generation subroutine calls an external GA. Recall from Section 6.4.1, 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 models[3], 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 idenitifcation of the precise true model to be successful, as insight can be gained from approximate models which capture the physics of the target system. We introduce metrics for success in Section 10.2.2. 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. 6.1, is unchanged. In a genetic exploration strategy (GES):

- models are still grouped in branches, here called generations;
- models are still trained, again through QHL;
- branches are evaluated according the the OF to be described in Section 10.3;
- new models are spawned through the genetic algorithm by selecting pairs of parents for crossover, with the resultant offspring models probabilistically mutated.

We detail the corresponding $\mathrm{generate\_models}$ subroutine in Algorithm 9. We can restate the informal description of GAs, now in the context of QMLA, as

1. Sample $N_m$ models from $\mathcal{P}$ at random
   (a) this is the first generation, $\mu$.
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 Recall that models impose structure on sets of terms: $\hat{H}_j = \vec{\alpha}_j \cdot \vec{T}_j = \sum_{k \in \mathcal{T}_j} \alpha_k \hat{t}_k$.

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 $\mu$
      i. Each model's probability of being chosen is given by 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
      ii. keep $\hat{H}_{c_i}$ only if it is not already in $\mu$, 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, $\hat{H}'$.

### 10.2.1 *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 10.1.1. 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 10.3.

### 10.2.2 *$F_1$-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 mertic to act as proxy for model quality: the $F_1$-score. 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 will define the concept formally next. Note that here we are able to compute $f$ for candidate models because the target $\hat{H}_0$ is simulated, i.e. we know the true terms $\mathcal{T}_0$; this would not be available for a real system with unknown $\hat{H}_0$, but is useful for the analysis of the algorithm itself.

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 [3, 55]. Therefore we can consider QMLA as a

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

Table 10.3: Mapping between QMLA's models and chromosomes used by a genetic algorithm. Example shown for a three-qubit system with six possible terms, $\hat{\sigma}^w_{i,j} = \hat{\sigma}^w_i \hat{\sigma}^w_j$. Model terms are mapped to binary genes: if the gene registers 0 then the corresponding term is not present in the model, and if it registers 1 the term is included. 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 $\gamma_{p_1}$ is mixed with the second half of $\gamma_{p_2}$ to produce two new children 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 $0 \to 1$ to mutate $\gamma_{c_2}$ to $\gamma'_{c_2}$. The next generation of the genetic algorithm will then include $\gamma_{c_1}, \gamma'_{c_2}$ (assuming $\gamma_{c_1}$ 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.

classification algorithm, 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 $\hat{H}_0$, which has $\mathcal{T}_0$. 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 (FP): number of terms in $\mathcal{T}_0$ which are not in $\mathcal{T}_i$.

These concepts 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 10.3a;
- *sensitivity*: how sensitive is $\hat{H}_i$ to $\hat{H}_0$, i.e. if a term is in $\mathcal{T}_0$, how likely $\mathcal{T}_i$ is to include it, Eqn. 10.3b.

$$\text{precision} = \frac{TP}{TP + FP} \tag{10.3a}$$

$$\text{sensitivity} = \frac{TP}{TP + FN} \tag{10.3b}$$

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. $F_\beta$-score is a measure which balances these, with $F_1$-score in particular giving them equal importance.

$$F_1 = \frac{2 \times (precision) \times (sensitivity)}{(precision + sensitivity)} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}. \tag{10.4}$$

We give an example of these quantities in Fig. 10.4, where $TP = 3, TN = 4, FP = 1, FN = 2$, giving $precision = 3/4$ and $sensitivity = 3/5$, with a final $f = 0.67$, i.e. the average of the indicators of model quality which we care about.

We adopt $F_1$-score as an indication of model quality because we are concerned both with precision and sensitivity of output models. 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 knoweldge 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 simualated, e.g. in the case of device calibration. Our search for an effective OF can then be guided by seeking the method which correlates most strongly with $F_1$-score in test-cases.

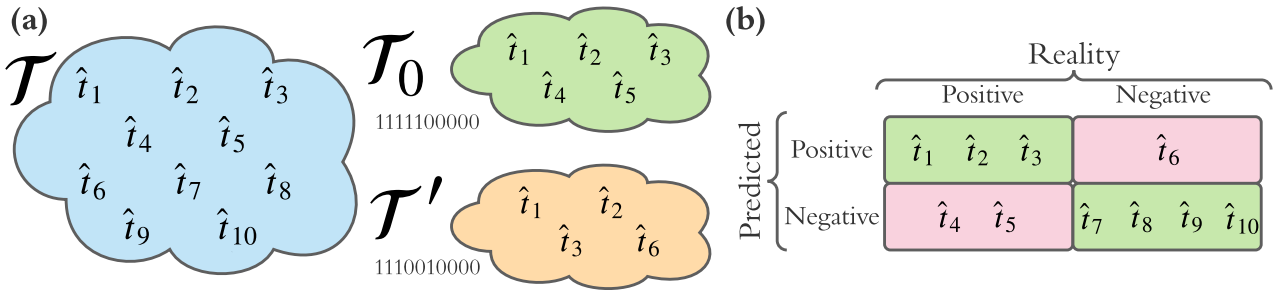


Figure 10.4: Concepts used for classification. **a**, the set of available terms $\mathcal{T}$ containing individual terms $\hat{t}_1$ to $\hat{t}_{10}$. The true model $\hat{H}_0$ is constructed from the set $\mathcal{T}_0$. Suppose a candidate $\hat{H}'$ has the set $\mathcal{T}'$. **b**, the confusion 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).

**Algorithm 9:** ES subroutine: generate_models via genetic algorithm

**Input:** $\nu$ // information about models considered to date

**Input:** $g(\hat{H}_i)$ // objective function

**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
$\mathbb{H}_t \leftarrow \text{truncate}(r, \frac{N_m}{2})$ // truncate models by rank: only keep $\frac{N_m}{2}$
$s \leftarrow \text{normalise}(\{g_i\})\ \forall \hat{H}_i \in \mathbb{H}_t$ // normalise remaining models' fitness
$\mathbb{H} = \{\}$ // 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 $\mathbb{H}$

#### 10.2.2.1 *Distinguishing $F_1$-score through Bayes factors*

We have so far relied on BF as the means by which to distinguish models' ability to explain data from the target system. We conjecture that models of higher $F_1$-score are usually stronger at this task than those of lower $F_1$-score, which will allow us to incorporate these statistical tools into the design of OFs. We can test this simply by training models of equally spaced $F_1$-score, and computing BF between all pairs.

In **??**, we show the relationships between $F_1$-score and or 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. In Fig. 10.5b, we run a complete model training subroutine, but compute the BF based on fewer experiments and particles (retaining a fraction 0.2). This verifies an earlier claim from Section 6.2.1: 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 BFs calculations. On the other extreme, we see in Fig. 10.5**c**, where models are trained with, and BFs based upon, even greater resources, 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, as would be required for rating models according to Bayes factor enhanced Elo ratings (BFEER), in Fig. 10.5**d**. In essence, we can see that the insight is largely the same from the most and least expensive training/comparison strategies, and by exploiting the available evidence through Elo ratings, rather than brute-force computing as much evidence as possible, 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 $N_m = 28$, these rise to 190 and 378 respectively. This saving depends on the user's choice of graph connectivity, see **??**, though is typically a factor between 2-3; in general, though, assuming we can reduce the resources used within the BF, this phase is considerably less cumbersome than the model training itself, so it is feasible to compute all available BFs to inform the BFEER.

### 10.2.3 *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 yperparameters to find a suitable configuration: in Fig. 10.6 we show how the choice of parameters affect the success rate of preciesly identifying the target chromosome, which is chosen at random for each instance, and we run 20 instance of each configuration. The studied hyperparameters[4] are

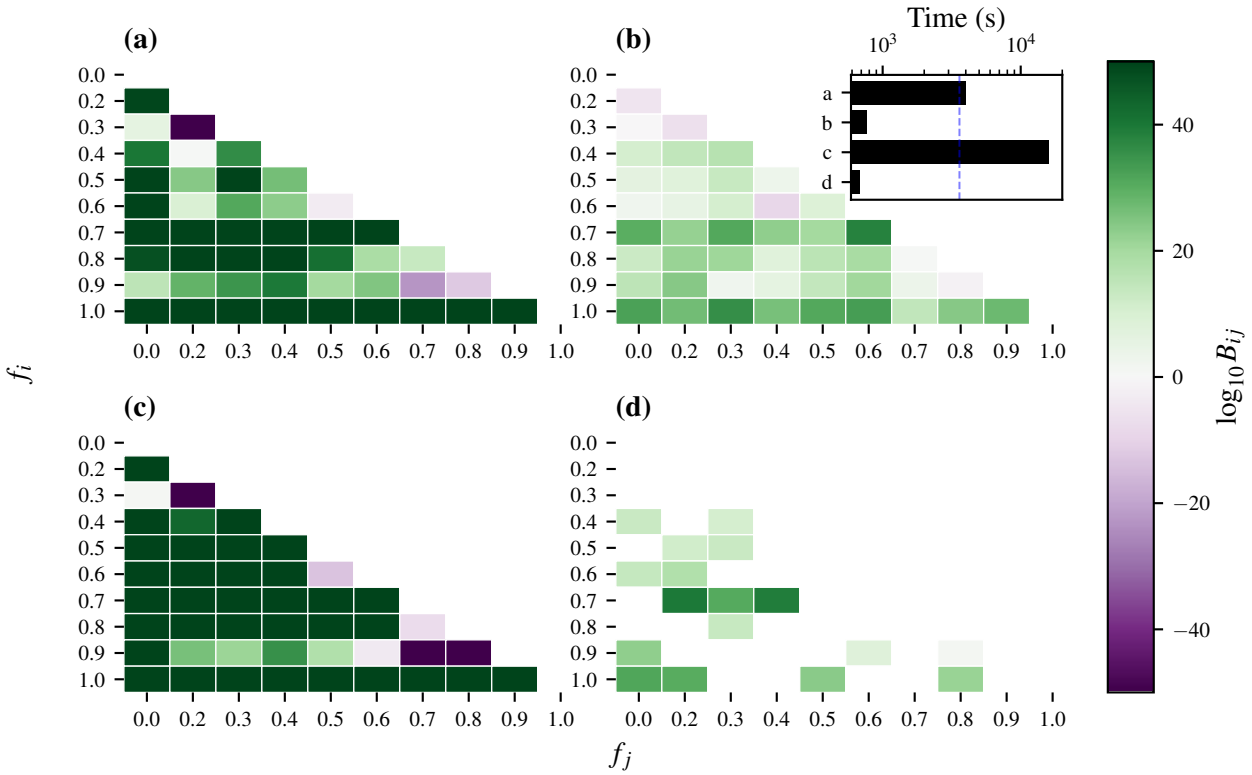


Figure 10.5: Pairwise Bayes factor, $B_{ij}$ by $F_1$-score of candidates $\hat{H}_i$ ($f_i$ on the $y$-axis) and $\hat{H}_j$ ($f_j$ on the $x$-axis). $\log_{10} B_{ij} > 0$ ($< 0$), green (purple), indicates statistical evidence that $\hat{H}_i$ ($\hat{H}_j$) is the better model with respect to the 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 uis used in the calculation of BFs. **b**, $N_e = 500, N_p = 2500$ using only a fraction (0.2) of experiments/particles for BF calculation. **c**, $N_e = 1000, N_p = 5000$, using all available data in the calculation of BF. **d**, $N_e = 500, N_p = 2500$, comparing only a subset of pairs of models through BF, and using only a fraction (0.2) of experiments/particles for those calculations. This pairwise comparison strategy is used for the BFEER OF. **Inset**, timings for each approach in seconds, with $t = 1$hr marked vertically in blue.

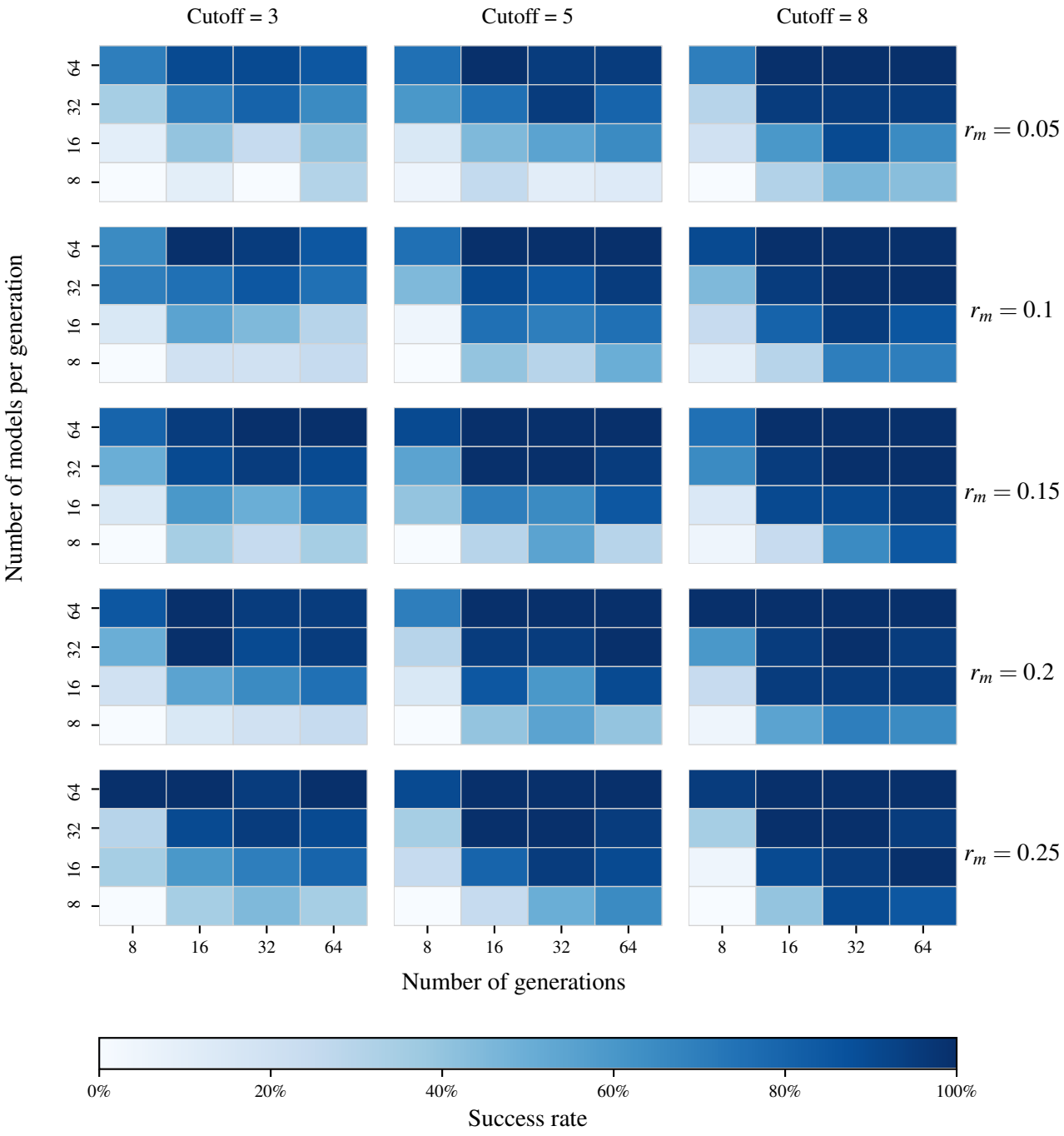


Figure 10.6: Genetic algorithm parameter sweep. Each subplot shows the success rates for varying numbers of generations, $N_G \in \{8, 16, 32, 64\}$, and numbers of models per generation, $N_m \in \{8, 16, 32, 64\}$. A subplot is generated for ranges of the mutation rate, $r_m$ and the number of generations for which the elite model is unchanged after which the GA is cut off.

1. number of generations;
2. number of models per generation;
3. mutation rate, $r_m$;
4. number of generation 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 badly with these, since training and comparing models are such expensive subroutines. Our goal is therefore to identify the set of hyperparameters which best searches the model space with minmial $N_g, N_m$. We see that, unsurprisingly, the GA performs poorly when run with few resources, but broadly the performances are similiar provided it is run with sufficient resources. We can bound the parameters $r_m \geq 0.1, cutoff \geq 5, N_m \geq 16, N_g \geq 16$ 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.

## 10.3 OBJECTIVE FUNCTIONS

We have alluded to the central probelm in building a GA into QMLA: how to evaluate trained candidate models in the absence of a natural OF. Here we will propose and analyse a number of potential OFs, some of which will underlie later studies in this thesis. Readers may prefer to skip to Section 10.3.8, where we conclude this study by choosing a single OF for consideration in this chapter.

We will show how each OF computes $g_i$ for candidate models $\hat{H}_i$, and summarise the outcomes in Table 10.4. For each $\hat{H}_i$, we may refer to

- $\mathcal{L}_i$, total log total likelihood (TLTL), introduced in Section 5.4
- $k_i$: the model's cardinality, i.e. number of terms in its parameterisation;
- $\mathcal{E}_i$: the bespoke set of experiments composed by the EDH solely for trainng $\hat{H}_i$;
- $n = |\mathcal{E}_i|$: the number of samples used in trainng $\hat{H}_i$.

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

In Table 10.4, we consider six models as examples of the outcome using each OF; the models, randomly generated of varying quality with respect to the target $\hat{H}_0$, are

$$\begin{aligned}
\hat{H}_0 &= \sigma^z_{(1,2)}\sigma^z_{(1,3)}\sigma^z_{(2,3)}\sigma^z_{(2,5)}\sigma^z_{(3,5)};\\
\hat{H}_a &= \sigma^z_{(1,5)}\sigma^z_{(3,4)}\sigma^z_{(4,5)};\\
\hat{H}_b &= \sigma^z_{(1,4)}\sigma^z_{(1,5)}\sigma^z_{(2,5)}\sigma^z_{(3,4)};\\
\hat{H}_c &= \sigma^z_{(1,2)}\sigma^z_{(1,5)}\sigma^z_{(2,4)}\sigma^z_{(2,5)}\sigma^z_{(4,5)};\\
\hat{H}_d &= \sigma^z_{(1,3)}\sigma^z_{(1,4)}\sigma^z_{(1,5)}\sigma^z_{(2,4)}\sigma^z_{(2,5)}\sigma^z_{(3,4)}\sigma^z_{(3,5)};\\
\hat{H}_e &= \sigma^z_{(1,2)}\sigma^z_{(1,3)}\sigma^z_{(1,5)}\sigma^z_{(2,3)}\sigma^z_{(2,5)}\sigma^z_{(4,5)};\\
\hat{H}_f &= \sigma^z_{(1,2)}\sigma^z_{(1,3)}\sigma^z_{(2,3)}\sigma^z_{(2,4)}\sigma^z_{(2,5)}\sigma^z_{(3,4)}\sigma^z_{(3,5)}.
\end{aligned} \tag{10.5}$$

### 10.3.1 *Inverse Log-likelihood*

$\mathcal{L}_i$ can be thought of as a measure of the success of a given model at explaining data from any set of experiments, $\mathcal{E}$. This can be immediately interpreted as an OF, provided each candidate model computes a meaningful 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

$$g^L_i = \frac{-1}{\mathcal{L}_i}. \tag{10.6}$$

In our tests, Eqn. 10.6 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 gauranteed in advance, or at least risks introducing strong bias.

### 10.3.2 *Akaike Information Criterion*

A common metric in model selection is Akaike information criterion (AIC) [56]. Incorporating TLTL, AIC objectively quantifies how well a given model accounts for data from the target system, and punishes models which use extraneous parameters, by incurring a penalty on $k_i$. AIC is given by

$$AIC_i = 2k_i - 2\mathcal{L}_i. \tag{10.7}$$

In practice we use a slightly modified form of Eqn. 10.7 which corrects for the number of samples $n = |\mathcal{E}_i|$, called the Akaike information criterion corrected (AICC),

$$AICC_i = AIC_i + 2k_i\frac{k_i+1}{n-k_i-1}. \tag{10.8}$$

| 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 |
| | $\overline{l_e}$ | $0.86 \pm 0.29$ | $0.84 \pm 0.29$ | $0.77 \pm 0.27$ | $0.78 \pm 0.29$ | $0.79 \pm 0.26$ | $0.79 \pm 0.26$ |
| | $\mathcal{L}_i$ | -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 |
| | $\mathrm{w}_i^B$ | 5.49e-66 | 1.26e-70 | 1.97e-62 | 1.11e-72 | 8.43e-61 | 8.95e-62 |
| | $\mathrm{g}_i^B$ | 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 | $\mathrm{g}_i^p$ | 0 | 2 | 3 | 2 | 3 | 5 |
| | % | 0 | 13 | 20 | 13 | 20 | 33 |
| Ranking points | Ranking | 6 | 5 | 2 | 4 | 3 | 1 |
| | $\mathrm{g}_i^R$ | 0 | 0 | 0.3 | 0.1 | 0.2 | 0.4 |
| | % | 0 | 0 | 30 | 10 | 20 | 40 |
| Elo rating | Rating | 909 | 944 | 1042 | 1007 | 1011 | 1084 |
| | $\mathrm{g}_i^E$ | 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 |
| | $\mathrm{g}_i^r$ | 0.753 | 0.729 | 0.785 | 0.743 | 0.836 | 0.862 |
| | % | 23 | 0 | 24 | 0 | 26 | 27 |

Table 10.4: Examples of how each objective function, $g$ as described in Section 10.3.1 to Section 10.3.7, assign selection probability (denoted %) to the same set of candidate models, $\{\hat{H}_i\}$, when attempting to learn data from $\hat{H}_0$, listed in Eq. (10.5). For each model we first summarise its average likelihood $l_e$ (Eqn. 6.5), total log-likelihood $\mathcal{L}_i$ (Eqn. 5.16), as well as $F_1$-score and number of terms $k$. 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.

Model selection from a set of candidates occurs simply by selecting the model with $AICC_{\min}$. A suggestion to retrieve selection probability, by using Eqn. 10.8 as a measure of *relative likelihood*, is to compute *Akaike weights* (as defined in in Chapter 2 of [56]),

$$w_i^A = exp\left(\frac{AICC_{\min} - AICC_i}{2}\right), \tag{10.9}$$

where $AICC_{\min}$ is the lowest $AICC$ observed among the models under consideration e.g. all models in a given generation.

Clearly, Akaike weights impose quite strong penalties on models which do not explain the data well, but also punish models with extra parameters, i.e. 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 10.4 that this results in most models being assigned negligible weight, which is not useful for parent selection. Instead we compute a straightforward quantity as the AIC–inspired fitness, Eqn. 10.10,

$$g_i^A = \left(\frac{1}{AICc_i}\right)^2, \tag{10.10}$$

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

### 10.3.3 *Bayesian Information Criterion*

Related to the idea of AIC, Eqn. 10.7, is that of Bayesian information criterion (BIC),

$$BIC_i = k_i\, ln(n_i) - 2\mathcal{L}_i, \tag{10.11}$$

where $k_i, n_i$ and $\mathcal{L}_i$ are as defined on Page 84. Analagously to Akaike weights, *Bayes weights* as proposed in §7.7 of [57], are given by

$$w_i^B = exp\left(-\frac{BIC_i}{2}\right). \tag{10.12}$$

BIC is harsher than AIC in its punishment of the number of parameters in each model, therefore requiring strong statistical justification for the addition of any parameters. Again, this may be overly cumbersome for our use case: with such a relatively small number of parameters, the punishment is disproportionate; moreover since we are trying to uncover physical interactions, we do not necessarily want to suppress models merely for their cardinality, since this might result in favouring simple models which do not capture the physics. As with Akaike weights, then, we opt instead for a simpler objective function,

$$g_i^B = \left(\frac{1}{BIC_i}\right)^2. \tag{10.13}$$

### 10.3.4 *Bayes factor points*

A cornerstone of model selection within QMLA is the calculation of Bayes factor (BF) (see Section 6.2). We can compute the pairwise BF between two candidate models, $B_{ij}$, according to Eqn. 6.7. $B_{ij}$ 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. 6.8) 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 simply 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 $\left|\mathcal{E}_j\right|$ experiments, each requiring $N_P$ particles[5], where each particle corresponds to a unitary evolution and therefore the caluclation of a matrix exponential. The combinatorial scaling of the model space is then quite a heavy disadvantage. However, in the case where all pairwise BF are performed, we can assign a point to $\hat{H}_i$ for every comparison which favours it.

$$g_i^p = \sum_{j \in \mu} b_{ij}, \quad b_{ij} = \begin{cases} 1, & B_{ij} > 1 \\ 0, & \text{otherwise} \end{cases} \tag{10.14}$$

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.

### 10.3.5 *Ranking*

Related to Section 10.3.4, we can rank models in a generation based on their number of BF points. BF points are assigned as in Eqn. 10.14, 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 $\mu$, meaning we remove the worse-performing models and retain only $N'_m$ models, before calculating $R$. This is because computing $R$ using all $N_m$ models results in less distinct selection probabilities.

$$g_i^R = \frac{N'_m - R_i + 1}{\sum_{n=1}^{N'_m} n}, \tag{10.15}$$

where $R_i$ is the rank of $\hat{H}_i$ and $N'_m$ is the number of models retained after truncation.

5 Caveat the reduction in overhead outlined in Section 6.2.1.

### 10.3.6 *Residuals*

Recall at each experiment, $N_P$ particles are compared against a single experimental datum, $d$. For consistency with QInfer [7] – on which QMLA's code base extends – we call the expectation value for the system $\Pr_Q(0)$, and that of each particle $\Pr_p(0)$, recall Section 5.3. Typically, $\Pr_Q(0) = \left|\langle\psi| e^{-i\hat{H}_0 t} |\psi\rangle\right|^2$, but this can be changed to match given experimental schemes, e.g. the Hahn-echo sequence applied in [1]. By definition, the datum $d$ is the binary outcome of the measurement on the system 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 0 (usually the same as the input probe state $|\psi\rangle$)? However, in practice we often have access also to the likelihood , i.e. rather than a binary value, a number representing the probability that $Q$ will project on to 0 for a given experiment $e$, $\Pr_Q(0|e)$. 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, $r_p^e$, as well as the mean residual across all particles in a single experiment $r^e$:

$$\begin{aligned} r_p^e &= \left|Pr_Q(0|e) - Pr_p(0|e)\right| \\ r^e &= \underset{p}{\text{mean}}\{r_p^e\} \end{aligned} \tag{10.16}$$

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

$$g_i^r = \left|1 - \underset{e\in\mathcal{E}}{\text{mean}}\{r^e\}\right|^2. \tag{10.17}$$

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 average to ensure $0 \leq r \leq 1$. $g_i^r$ encapsulates how well a candidate model can reproduce dynamics from the target system, as a proxy for whether 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 likelihood that $\hat{H}_i = \hat{H}_0$. However, 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 simulate $\hat{H}_0$, then this OF gaurantees 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 does not account for the cardinality, $k_i$, of the candidate models, which could result in favouring severely overfitting models in order to gain marginal improvement in residuals, which all machine learning protocols aim to avoid in general.

### 10.3.7 *Bayes factor enhanced Elo-ratings*

A popular tool for rating individual competitors in sports and games is the *Elo rating* scheme, e.g. used to rate chess players and soccer teams [58, 59], also finding application in the study of animal hierarchies [60]. Elo ratings allow for evaluating 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 [61]. We recognise a parallel with these types of competitions by noting that in our case, we similarly have a pool of individuals (models), which we can place in direct competition, and quantify the comparitive outcome through BF.

Elo ratings are transitive: given some interconnectivity 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 is tested robustly. We can take advantage of this transitivity to reduce the combinatorial overhead usually associated with computing bespoke BF between all models (i.e. using their own training data $\mathcal{E}_i$ instead of a generic $\mathcal{E}_v$). In practice, we map models within a generation to nodes on a graph, which is then sparsely connected. In composing the list of edges for this graph, we primarily prioritise each node having a similar number of edges, and secondarily the distance between any two nodes. For example, with 14 nodes we overlay edges such that each node is connected with 5,6 or 7 other nodes, and all nodes at least share a competitor in common.

The Elo rating scheme is 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 $\Delta R_i$ proportionally changes the credence we assign to $\hat{H}_i$. It achieves this by first computing the *expected* result of a given comparison with respect to each model, based on the current ratings,

$$E_i = \frac{1}{1 + 10^{\frac{R_j - R_i}{400}}}; \tag{10.18a}$$

$$E_i + E_j = 1, \tag{10.18b}$$

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

$$\begin{cases} B_{ij} > 1 & \Rightarrow \; S_i = 1; \; S_j = 0 \\ B_{ij} < 1 & \Rightarrow \; S_i = 0; \; S_j = 1 \end{cases} \tag{10.19}$$

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

$$\Delta R_i = \eta \times (S_i - E_i)\,. \tag{10.20}$$

| | Model | $R_i$ | $E_i$ | $S_i$ | $B_{ij}$ | $\log_{10}(B_{ij})$ | $\Delta R_i$ | $R'_i$ |
|---|---|---|---|---|---|---|---|---|
| $\hat{H}_a > \hat{H}_b$ | $\hat{H}_a$ | 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 10.5: Example of Elo rating updates. We have two models, where $\hat{H}_a$ is initially believed to be a stronger candidate than $\hat{H}_b$, i.e. has a higher starting Elo rating. We show the effect of strong evidence[6] in favour of each model following BF comparison, $B_{ij} \sim 10^{100}$. In the case where $\hat{H}_a$ defeats $\hat{H}_b$, because this was so strongly expected given their initial ratings, the reward for $\hat{H}_a$ (and cost to $\hat{H}_b$) is relatively small, compared with the case where – contrary to prediction – $\hat{H}_b$ defeats $\hat{H}_a$.

An important detail is the choice of $\eta$, 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 $\eta$ is the number of goals by which one team won – we weight the change by the strength of our belief in the outcome: $\eta \propto \left|B_{ij}\right|$. That is, similarly to the interpretation of Eqn. 6.8, we use the evidence in favour of the winning model to transfer points from the loser to the winner, albeit we temper this 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

$$R'_i = R_i + \log_{10}(B_{ij}) \left( S_i - \frac{1}{1 + 10^{\frac{R_j - R_i}{400}}} \right). \tag{10.21}$$

This procedure is easiest to understand by following the example in Table 10.5.

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 we impose details specific to the QMLA GA: 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 defintion. Therefore by maintaining a constant $R_i^0$, i.e. a model born at generation 12 gets the same $R_i^0$ as $\hat{H}_e^1$ – which was born several generations prior and has been winning BF comparisons ever since – we bias the GA to continue to favour the elite models. Instead, we would prefer that newly born models can overtake the Elo rating of elite models. We achieve this through an imprecise mechanism:

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

newly born models are given the Elo rating of the second-most-elite model, $\hat{H}_e^2$. This performs three key functions:

i. new models are immediately *within range* of the elite models; if they perform well enough, they have a realistic and fair chance of overtaking them;

ii. the strongest model retains some of its advantage gained over previous generations – in order that the ratings are meaningful, there must be some advantage accrued over series of victories;

iii. $\hat{H}_e^2$ is allowed continue to compete, but has no advantage: in order to retain its status as elite, it must perform well *again* in this generation, so it can not simply rely on results from previous generations – against inferior opposition – to remain active in the gene pool.

Given the arbitrary scaling of the Elo rating scheme, and in order to derive a meaningful selection probability, we ought to ground the raw Elo rating somehow at each generation $\mu$. We do this by subtracting the lowest rating among the entertained models, $R^{\mu}_{\text{min}}$. This serves to ensure the range of remaining $R_i$ is defined only by the difference between models as assessed within $\mu$: a very strong model might have much higher $R_i$ than its contemporaries, but that difference was earned exclusively by comparison within $\mu$, so it deserves higher fitness. We perform this step before truncation, so that the reamining models post-truncation all have non-zero fitness. Finally, then, we name this OF the *Bayes-factor enhanced Elo rating*, whereby the fitness of each model is attained directly from its rating after undergoing Elo updates in the current generation minus the minimum rating of any model in the same generation $\mu$,

$$g_i^E = R_i^{\mu} - R^{\mu}_{\text{min}}. \tag{10.22}$$

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.

### 10.3.8 *Choice of objective function*

Having proposed a series of possible objective functions, we are now in a position to analyse which of those are most appropriate for QMLA. Recall from Section 10.2.2 the figure of merit we use for models, $F_1$-score, which we will use to distinguish between the outputs of each OF.

First we can remark on the examples listed in Table 10.4. 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 for poor models, e.g. $\hat{H}_a, \hat{H}_c$. This underlines the risk in buidling $\mathcal{E}_v$, which can be biased towards weak models, for example resulting in high selection probability for $\hat{H}_a$ which has $f = 0$, while

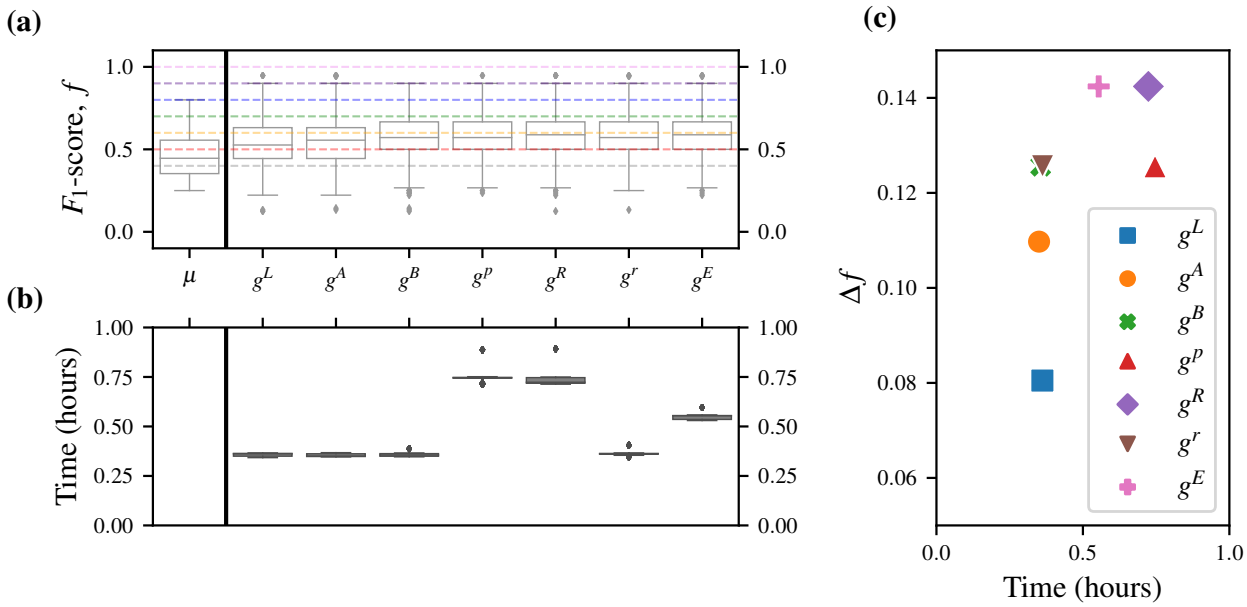


Figure 10.7: Comparison between proposed OFs. Each OF trains the same initial generation of $N_m = 28$ models with resources $N_E = 500, N_P = 2000$, and then design a new set of $N_m$ models through the same roulette strategy, such that the only difference between OF's output is how they assign selection probability. We run each OF 25 times for the same target system, a 4–qubit Heisenberg–XYZ model. **(a)** shows the box–plot of new models' $F_1$-score, $f$, where the median and inter–quartile ranges are indicated by the boxes, as well as those of the initial generation $\mu$ centered on $f_\mu = 0.45$. We mark $f = \{0.4, 0.5, ..., 1.0\}$ for ease of interpretation. **(b)** shows box–plots of the time taken to compute the single generation in each case. In **(c)** we report the difference between the median $f$ among the newly proposed models from $f_\mu$, $\Delta f$, plotted against the time to achieve the result.

$\hat{H}_d$, 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 using each OF, judging them by the quality of models they produce. 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. We plot the distribution of $F_1$-score that each OF produces in Fig. 10.7, 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.

Overall, then, we can see that a strong balance of outcome with resource considerations are achieved by the Bayes factor enhanced Elo rating strategy, Section 10.3.7 so we use that 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. In particular, we will later use the OF based on residuals, Section 10.3.6, to study a much larger model space under assumptions of perfect parameter learning, Chapter 12.

## 10.4 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 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 IQLE[7]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 $\mu$ are first truncated (the worst-performing $N_m/2$ are discarded), and then assigned selection probability based on their fitness.

- Models are selected to become parents sequentially using roulette selection. Highly favoured models can parent many children 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 $\mu$ are included on the $\mu + 1$.
- If, after 5 generations, the highest-fitness (elite) model is unchanged, we terminate the search and declare that model as the champion, $\hat{H}'$.
- Otherwise, after $N_g$ generations, the highest-fitness model on the final generation is declared $\hat{H}'$.

We will use a four-qubit model space under the Heisenberg formalism, Eq. (8.10), such that any pair of sites $\langle k, l \rangle$ can be coupled by any of $\hat{\sigma}^x_{\langle k,l \rangle}, \hat{\sigma}^y_{\langle k,l \rangle} m \hat{\sigma}^z_{\langle k,l \rangle}$, 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[8], we set $N_m = 28$ and $N_g = 32$, 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 $0.5 \pm 0.125$. We choose $\hat{H}_0$ at random to contain half the available terms[9],

$$\hat{H}_0 = \sigma^{yz}_{(1,2)} \sigma^z_{(1,3)} \sigma^y_{(1,4)} \sigma^{xy}_{(2,3)} \sigma^x_{(2,4)} \sigma^{xz}_{(3,4)}. \tag{10.23}$$

### 10.4.1 *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 $\mu$, a given model $\hat{H}_i$ undergoes some $N^i_{BF} < N_m$ BF comparisons. In Fig. 10.8 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 $\sim 20\%$ of contemporary models on $\mu$. 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 (its rating).

We extend the single model analysis of Fig. 10.8 to all $N_m$ models in the first generation in Fig. 10.9. 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

7 IQLE assumes complete access to the target system see Section 5.3.1. This restricts the present analysis to simulateable, rather than physical, use cases, e.g. device calibration.

8 This is to ensure, with 15 available worker nodes, and accounting for some slowly-learning models, that all $N_m$ models in a generation are trained within $2t_{qhl}$, where $t_{qhl}$ is the time to train a single model.

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

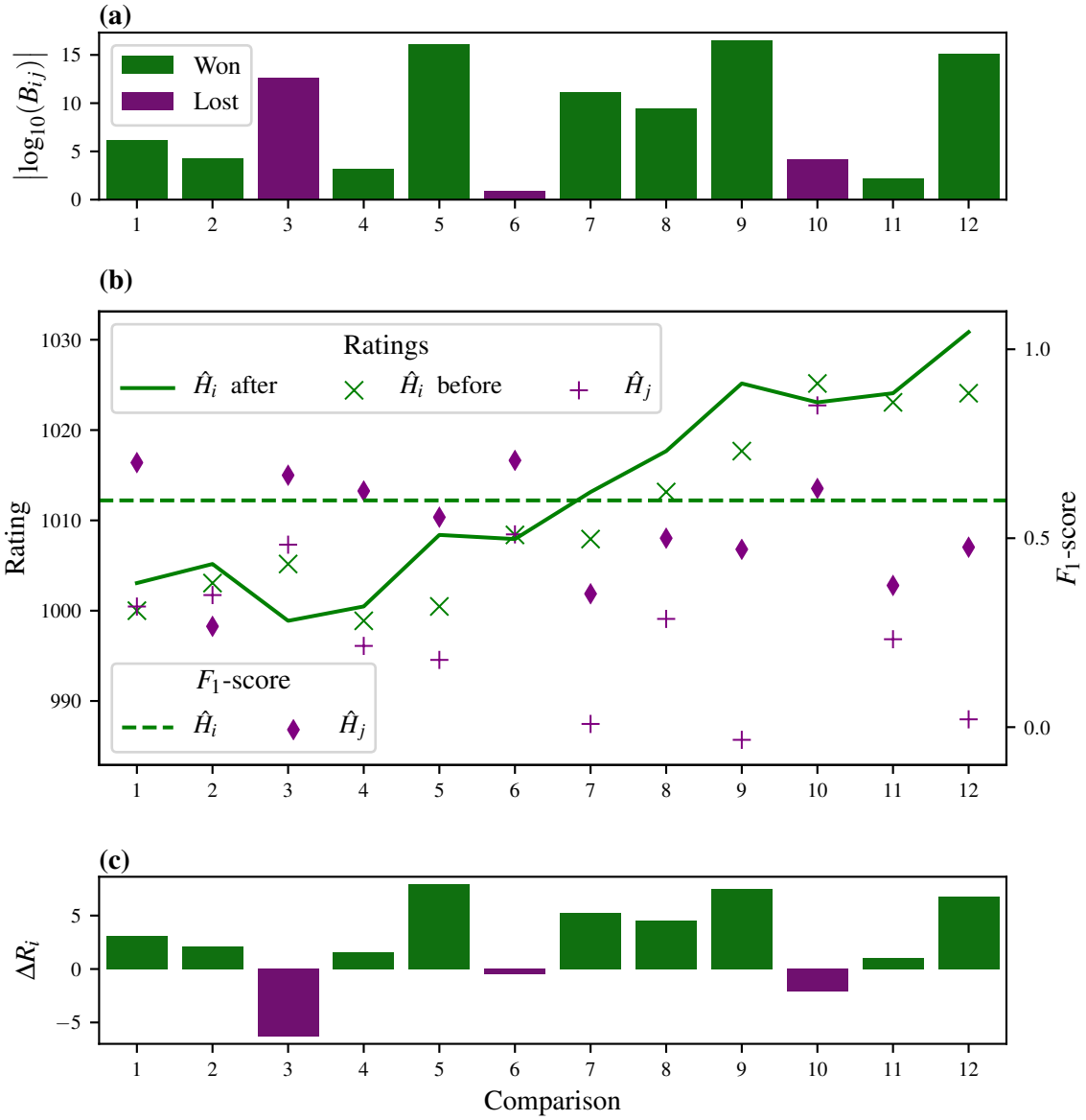


Figure 10.8: Bayes factor enhanced Elo ratings (BFEER) progression for a single candidate, $\hat{H}_i$, within a single generation. **a**, The BFs between $\hat{H}_i$ and some opponents, $\{\hat{H}_j\}$, from the perspective where $\hat{H}_i$ *wins* given $B_{ij} > 1 \Rightarrow \log_{10} B_{ij} > 0$, 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, $\Delta R_i$.

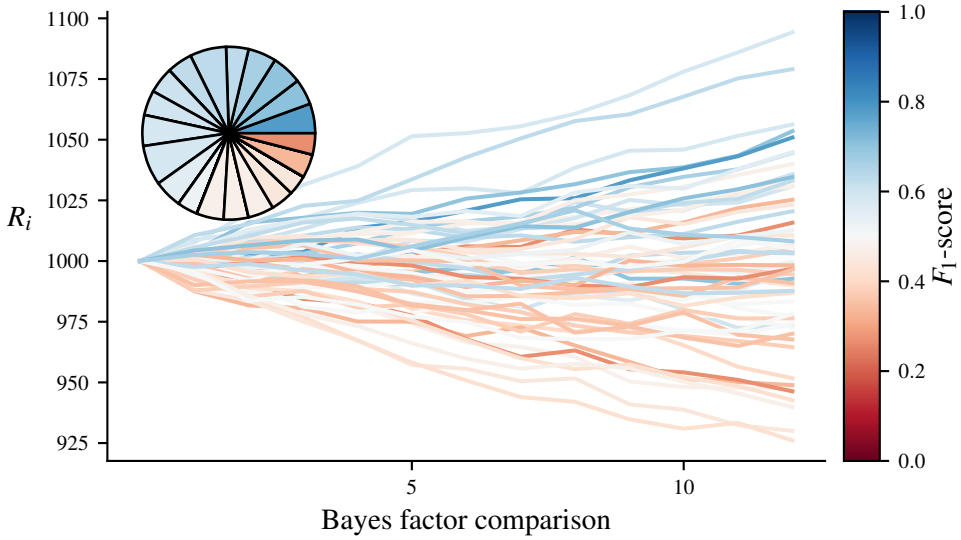


Figure 10.9: Ratings of all models in a single GA generation. Each line represents a unique model and is coloured by the $F_1$-score of that model. **Inset**, the selection probabilities resulting from the final ratings of this generation. Only a fraction of models are assigned selection probability, while the remaining poorer-performing models are truncated.

truncated to retain only the strongest candidates, which are assigned probability of being chosen to become a parent during roulette selection, as in Section 10.1.3. The very strongest two models are granted a position in the subsequent generation: these are the *elite* models.

Similarly we can consider the quality of models across at each generation, and their ratings. In Fig. 10.10 we see the trend suggested by Fig. 10.9 continue, i.e. that models of higher quality overall tend to achieve higher BFEER. The gene pool as a whole tends towards a homogeneous set of models, all with $f0.9$. Consequently, even in cases 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 meaningul insight on the system's physics.

Finally, to understand the performance of the QMLA algorithm overall, we run 50 independent instances in a run, Fig. 10.11. We see that, while the overall model space can be characterised by the distribution of models' $F_1$-score, where we sample where $f = 0.5 \pm 0.13$, that QMLA quickly moves to the subspace of high-quality models, and ultimately nominates champion models, $\{\hat{H}'\}$ with $f > 0.9$ in , and precisely identifies $\hat{H}' = \hat{H}_0$ in % of instances. Considering the big picture, where the remit of QMLA is to identify the interactions the target system is subject to, we show how often each term/gene is included in $\hat{H}'$ in Fig. 10.11**d**. Crucially, we see that terms which really are within the true Hamiltonian, $\hat{t} \in \mathcal{T}_0$, are found significantly more frequently than those without, $\hat{t} \notin \mathcal{T}_0$. This level of analysis can be used to post-validate the outcome of

fill in f
fill in f

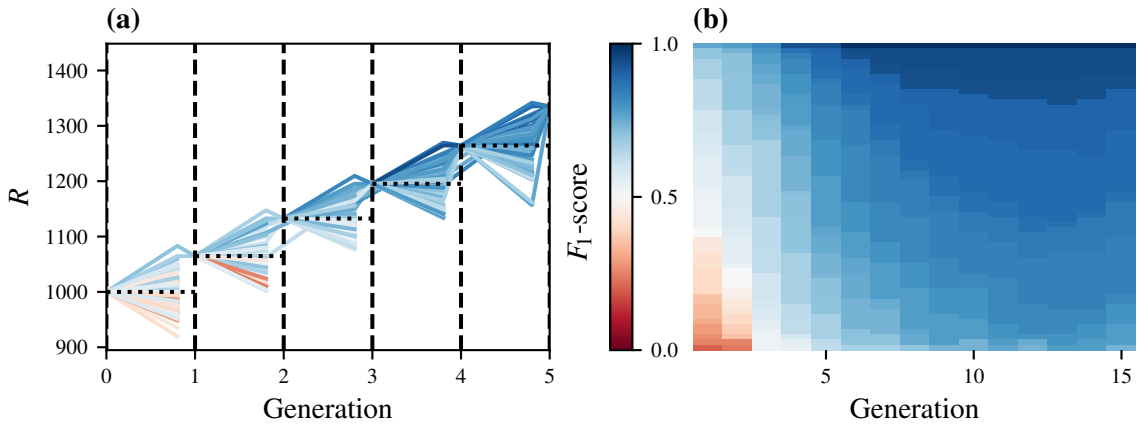


Figure 10.10: A single instance of the QMLA GA. **a**, Ratings of all models for the first five generations. Each line in each generation represents a model by its $F_1$-score. Horizontal dotted lines show the starting rating at that generation. **b**, Gene pool progression for $N_m = 60, N_g = 15$. Each time at each generation represents a model by its $F_1$-score.

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.

### 10.4.2 *Device characterisation*

This adaptive GES may 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, and QMLA can effectively reconstruct which operations were and were not faithfully computed.

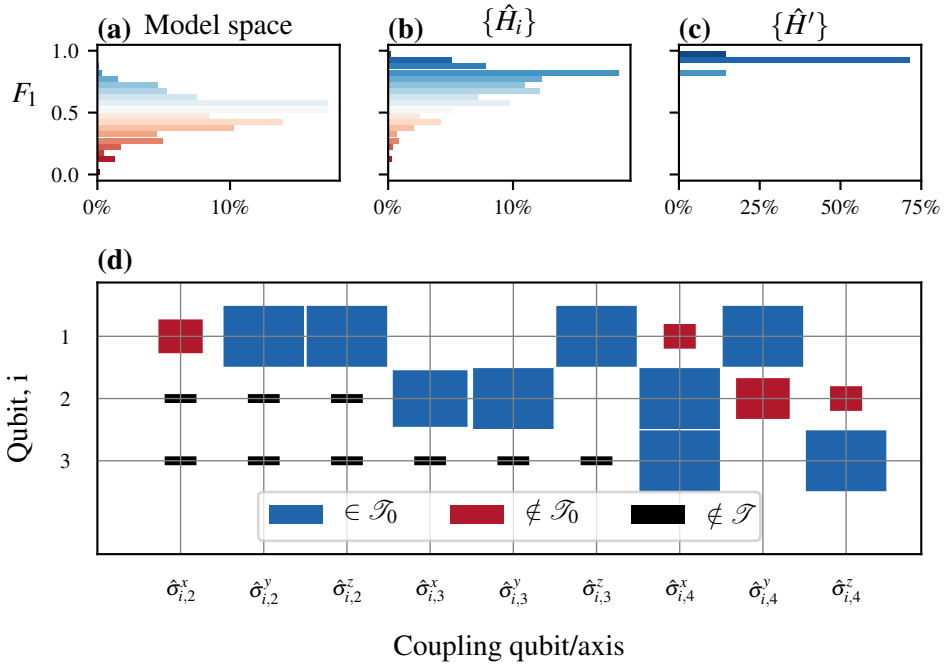


Figure 10.11: A run of the QMLA GA, consisting of 50 independent instances. **(a)**, The model space contains $2^{18} \approx 250,000$ candidate models; normally distributed around $f = 0.5 \pm 0.14$. **(b)**, The models explored during the model search of all instances combined, $\{\hat{H}_i\}$, show that QMLA tends towards stronger models overall. **(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 32% of cases. **(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 in the true model (blue) or not (red). Terms are couplings between two qubits, e.g $\hat{\sigma}^x_{(1,3)}$ couples the first and third qubits along the $x$-axis. We test four qubits with full connectivity, resulting in 18 unique terms (terms with black rectangles are not considered by the GA).

# Part IV

# EXPERIMENTAL STUDIES

# 11

# NITROGEN VACANCY CENTRE

## 11.1 SHORT TIME DYNAMICS

# 12

# EXTENSION TO MANY QUBITS

## 12.1 GENETIC ALGORITHM

# Part V

# CONCLUSION

# 13

# OUTLOOK FOR MODEL LEARNING METHODOLOGIES

There is a large and growing interest in automatically identifying the models of quantum systems [62, 63, 55, 64, 65]. Automated methodologies for identifying quantum mechanical models are becoming feasible with the development of quantum devices capable of simulating nature at the quantum level.

# APPENDIX

# A

## FIGURE REPRODUCTION

Most of the figures presented in the main text are generated directly by the QMLA framework. Here we list the implementation details of each figure so they may be reproduced by ensuring the configuration in Table A.1 are set in the launch script. The default behaviour of QMLA is to generate a results folder uniquely identified by the date and time the run was launched, e.g. results can be found at the *results directory* qmla/Launch/Jan_01/12_34. Given the large number of plots available, ranging from high-level run perspective down to the training of individual models, we introduce a plot_level $\in \{1, ..., 6\}$ for each run of QMLA: higher plot_level informs QMLA to generate more plots.

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.
- single_instance_plots: 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.

In order to produce the results presented in this thesis, the configurations listed in Table A.1 were input to the launch script. The launch scripts in the QMLA codebase consist of many configuration settings for running QMLA; only the lines in snippet in Listing A.1 need to be set according to altered to retrieve the corresponding figures. Note that the runtime of QMLA grows quite quickly with $N_E, N_P$ (except for the AnalyticalLikelihood ES), especially for the entire QMLA algorithm; running QHL is feasible on a personal computer in $< 30$ minutes for $N_e = 1000; N_p = 3000$.

```
#!/bin/bash
```

```
###############
# QMLA run configuration
###############
num_instances=1
run_qhl=1 # perform QHL on known (true) model
run_qhl_mulit_model=0 # perform QHL for defined list of models.
exp=200 # number of experiments
prt=1000 # number of particles

###############
# QMLA settings
###############
plot_level=6
debug_mode=0

###############
# Choose an exploration strategy
###############

exploration_strategy='AnalyticalLikelihood'
```

Listing A.1: "QMLA Launch scipt"

| Figure | Exploration Strategy | Algorithm | $N_E$ | $N_P$ | Data |
|---|---|---|---|---|---|
| Fig. 5.2 | AnalyticalLikelihood | QHL | 500 | 2000 | Nov_16/14_28 |
| Fig. 8.2 | DemoIsing | QHL | 500 | 5000 | Nov_18/13_56 |
| Fig. 8.3 | DemoIsing | QHL | 1000 | 5000 | Nov_18/13_56 |
| Fig. 8.4 | DemoIsing | QHL | 1000 | 5000 | Nov_18/13_56 |
| Fig. 8.5 | IsingLatticeSet | QMLA | 1000 | 4000 | Nov_19/12_04 |
| | IsingLatticeSet | QMLA | 1000 | 4000 | Sep_30/22_40 |
| Fig. 8.6 | HeisenbergLatticeSet | QMLA | 1000 | 4000 | Oct_22/20_45 |
| | FermiHubbardLatticeSet | QMLA | 1000 | 4000 | Oct_02/00_09 |
| Fig. 10.5 | DemoBayesFactorsByFscore | QMLA | 500 | 2500 | Dec_09/12_29 |
| | DemoFractionalResourcesBayesFactorsByFscore | QMLA | 500 | 2500 | Dec_09/12_29 |
| | DemoBayesFactorsByFscore | QMLA | 1000 | 5000 | Dec_09/12_29 |
| | DemoBayesFactorsByFscoreEloGraphs | QMLA | 500 | 2500 | Dec_09/12_29 |

Table A.1: Implementation details for figures used in the main text.

# B

# EXAMPLE EXPLORATION STRATEGY RUN

A complete example of how to run the ;sqmla framework, including how to implement a custom ES, and generate/interpret analysis, is given.

# 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, 2020.

[2] Christopher E Granade, C Ferrie, N Wiebe, and D G Cory. Robust online Hamiltonian learning. *New Journal of Physics*, 14(10):103013, October 2012.

[3] Nathan Wiebe, Christopher Granade, Christopher Ferrie, and David Cory. Quantum hamiltonian learning using imperfect quantum resources. *Physical Review A*, 89(4):042314, 2014.

[4] 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.

[5] 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.

[6] 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.

[7] Christopher Granade, Christopher Ferrie, Steven Casagrande, Ian Hincks, Michal Kononenko, Thomas Alexander, and Yuval Sanders. QInfer: Library for statistical inference in quantum information, 2016.

[8] Rodolfo A Jalabert and Horacio M Pastawski. Environment-independent decoherence rate in classically chaotic systems. *Physical review letters*, 86(12):2490, 2001.

[9] Nathan Wiebe, Christopher Granade, and David G Cory. Quantum bootstrapping via compressed quantum hamiltonian learning. *New Journal of Physics*, 17(2):022005, 2015.

[10] Arseni Goussev, Rodolfo A Jalabert, Horacio M Pastawski, and Diego Wisniacki. Loschmidt echo. *arXiv preprint arXiv:1206.6348*, 2012.

[11] 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.

[12] 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.

[13] 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.

[14] Christopher E Granade. Characterization, verification and control for large quantum systems. page 92, 2015.

[15] Christopher Ferrie. High posterior density ellipsoids of quantum states. *New Journal of Physics*, 16(2):023006, 2014.

[16] 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.

[17] 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.

[18] Lukas J Fiderer, Jonas Schuff, and Daniel Braun. Neural-network heuristics for adaptive bayesian quantum estimation. *arXiv preprint arXiv:2003.02183*, 2020.

[19] 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.

[20] Stefan Krastanov, Sisi Zhou, Steven T Flammia, and Liang Jiang. Stochastic estimation of dynamical variables. *Quantum Science and Technology*, 4(3):035003, 2019.

[21] 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.

[22] 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 moir\'{e}-enhanced swap spectroscopy with deep evolutionary algorithm. *arXiv preprint arXiv:1912.04368*, 2019.

[23] Eliska Greplova, Christian Kraglund Andersen, and Klaus Mølmer. Quantum parameter estimation with a neural network. *arXiv preprint arXiv:1711.05238*, 2017.

[24] Andrey Y Lokhov, Marc Vuffray, Sidhant Misra, and Michael Chertkov. Optimal structure and parameter learning of ising models. *Science advances*, 4(3):e1700791, 2018.

[25] 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.

[26] Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995.

[27] 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.

[28] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[29] Ruhul A Sarker and Tapabrata Ray. Agent based evolutionary approach: An introduction. In *Agent-Based Evolutionary Search*, pages 1–11. Springer, 2010.

[30] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[31] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[32] Roger W Hockney and Chris R Jesshope. *Parallel Computers 2: architecture, programming and algorithms*. CRC Press, 2019.

[33] Redis, Nov 2020. [Online; accessed 7. Nov. 2020].

[34] RQ: Simple job queues for Python, Nov 2020. [Online; accessed 7. Nov. 2020].

[35] Ernst Ising. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, 1925.

[36] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117, 1944.

[37] Stephen G Brush. History of the lenz-ising model. *Reviews of modern physics*, 39(4):883, 1967.

[38] Francisco Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[39] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[40] Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014.

[41] 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.

[42] 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.

[43] 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.

[44] Walter Greiner, Ludwig Neise, and Horst Stöcker. *Thermodynamics and statistical mechanics*. Springer Science & Business Media, 2012.

[45] 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.

[46] Richard T Scalettar. An introduction to the hubbard hamiltonian. *Quantum Materials: Experiments and Theory*, 6, 2016.

[47] Editorial. The hubbard model at half a century. *Nature Physics*, 2013.

[48] Pascual Jordan and Eugene Paul Wigner. über das paulische äquivalenzverbot. In *The Collected Works of Eugene Paul Wigner*, pages 109–129. Springer, 1993.

[49] Mark Steudtner and Stephanie Wehner. Fermion-to-qubit mappings with varying resource requirements for quantum simulation. *New Journal of Physics*, 20(6):063010, 2018.

[50] 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.

[51] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[52] 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.

[53] Sean Luke. 1 essentials of metaheuristicsl. 1.

[54] Lothar M Schmitt. Theory of genetic algorithms. *Theoretical Computer Science*, 259(1-2):1–61, 2001.

[55] Eyal Bairey, Itai Arad, and Netanel H Lindner. Learning a local hamiltonian from local measurements. *Physical review letters*, 122(2):020504, 2019.

[56] Burnham KP Anderson DR. Model selection and multimodel inference: a practical information-theoretic approach, 2002.

[57] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[58] Arpad E Elo. *The rating of chess players, past and present*. Arco Pub., 1978.

[59] FIFA.com. Who we are - news - 2026 fifa world cup™: Fifa council designates bids for final voting by the fifa congress, Jun 2018.

[60] 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.

[61] 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.

[62] Jonas B Rigo and Andrew K Mitchell. Machine learning effective models for quantum systems. *Physical Review B*, 101(24):241105, 2020.

[63] 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.

[64] Chris J Pickard and RJ Needs. Ab initio random structure searching. *Journal of Physics: Condensed Matter*, 23(5):053201, 2011.

[65] 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.