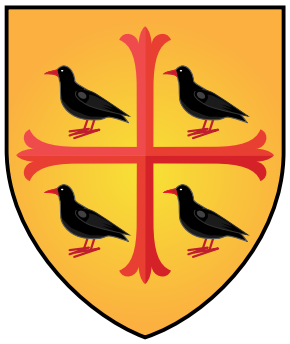
Autoregressive neural quantum states for *ab initio* quantum chemistry



Aleksei Malyshev

St Edmund Hall

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2024

Abstract

Neural quantum states are a powerful and versatile family of ansatzes for variational Monte Carlo simulations of quantum many-body systems. However, their utility for *ab initio* quantum chemistry is yet to be demonstrated. The main complications are (i) taking into account multiple quantum number symmetries, inherent to molecules; (ii) the peaked structure of the molecular wave functions, which impedes sampling; and (iii) large numbers of terms in second quantised Hamiltonians, which hinder scaling to larger molecule sizes.

In this thesis, we address these issues with *autoregressive* neural quantum states (ANQS). ANQS enjoy the expressibility of deep neural networks, and enable fast and unbiased sampling.

First, we develop a framework to make the autoregressive sampling compliant with arbitrary numbers of quantum number symmetries. We run electronic structure calculations for a range of molecules with multiple symmetries of this kind and reach the accuracy reported in previous works with more than an order of magnitude speedup.

Second, we argue that the peaked structure might be key to drastically more efficient calculations. We introduce a novel algorithm for autoregressive sampling without replacement and a procedure to calculate a computationally cheaper energy surrogate. We complement them with a custom modification of the stochastic reconfiguration technique and a highly optimised GPU implementation. As a result, our calculations require substantially less resources and exhibit an additional order of magnitude speedup. On a single GPU we study molecules comprising up to 118 qubits and outperform the "golden standard" CCSD(T) benchmark in Hilbert spaces of $\sim 10^{15}$ Slater determinants, which is orders of magnitude larger than what was previously achieved. We believe that our work underscores the prospect of ANQS for challenging quantum chemistry calculations and serves as a favourable ground for the future method development.

To my loved ones, and all free people of this world

Contents

The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble. It there fore becomes desirable that approximate practical methods of applying quantum mechanics should be developed, which can lead to an explanation of the main features of complex atomic systems without too much computation.

— Paul Dirac, "Quantum mechanics of many-electron systems." *Proceedings of the Royal Society A,* 1929 [1].

Introduction

Despite being almost a hundred years old, the Dirac's quote is as relevant as ever.[1] The Schrödinger equation remains to provide the most accurate description of the non-relativistic microscopic phenomena. The dimension of the Hilbert space required to represent N interacting quantum systems still grows exponentially with N, making exact solving the Schrödinger equation limited to the cases where N is at most few dozen. And, consequently, our progress in understanding quantum phenomena still largely rests on various approximations made at the right time and place.

One of the most successful approaches to making such approximations is known as *variational* and is built on the following premises. For the sake of simplicity, let us assume that we aim to find the ground state of a system described by the Hamiltonian $\hat{H}$.[2] It is equivalent to finding a state $|\psi_{\mathrm{GS}}\rangle$, which minimises the energy of the system:

$$|\psi_{\mathrm{GS}}\rangle \coloneqq \arg\min_{|\psi\rangle} \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle}. \tag{1}$$

The main idea of the variational approach is to step away from an exponentially large Hilbert space and instead restrict ourselves to a certain class of quantum states $|\psi_\theta\rangle$ parameterised with polynomially many numbers θ. Such class of states is referred to as *an ansatz*. Having chosen an ansatz, we try to find the state

[1] And PhD students across the world keep citing it in their theses. For better or worse, this thesis is no exception.

[2] A non-exhaustive list of other problems admitting variational formulation includes simulating the real-time evolution of quantum states [2–4], targeting the excited states of a system [5] and quantum state tomography [6, 7]

which has the minimum energy among all possible ansatz states, and thus serves as the best available approximation to the true ground state. In other words, we are solving the following problem:

$$\text{Find } \theta_{\text{best}} \coloneqq \arg\min_{\theta} E(\theta) = \arg\min_{\theta} \frac{\langle\psi_\theta|\hat{H}|\psi_\theta\rangle}{\langle\psi_\theta|\psi_\theta\rangle}. \tag{2}$$

In order to succeed with the variational approach, one needs to address two key issues. First, the variational approach inevitably trades off accuracy for efficiency, and thus it is crucial to pick an expressive enough ansatz, capable of capturing the relevant properties of the considered system. Second, one needs an optimisation algorithm to find the best set of ansatz parameters. In practice, any such algorithm must be able to efficiently evaluate the cost function in Eq. (2) given a state $|\psi_\theta\rangle$, or, more generally, to efficiently evaluate observables of interest.

A plethora of ansatz families was developed along with various ways to optimise them, which, in spite of the conceptual simplicity of the method, remarkably advanced our understanding of quantum many-body systems. For example, the list of condensed matter systems successfully tackled with the variational approach includes, but is not limited to, quantum Hall systems described in terms of Laughlin wave functions [8], interacting spin systems in 1D studied with tensor networks [9], and high-temperature superconductors approached with Slater-Jastrow wave functions [10].

In this thesis we study a specific family of ansatzes known as *autoregressive neural quantum states (ANQS)*. ANQS are a special type of *neural quantum states (NQS)*, which use a neural network to parameterise the wave function of a system at consideration and were proposed by Carleo in Troyer in Ref. [2]. As opposed to the most applications of neural networks, *no* "training dataset" is required in this case. Instead, NQS are variationally optimised in the framework of *variational Monte Carlo (VMC)*. It means that the cost function in Eq. (2) and its gradient, as well as observables of interest, are evaluated *stochastically* after sampling from the Born distribution encoded by the network *itself*. The estimated gradient is further used to optimise the ansatz parameters.

Specifically, we study the application of ANQS to the problem of *ab initio* quantum chemistry. Its goal is to deduce the properties of a molecule from the first principles by directly solving the Schrödinger equation for a system of interacting electrons and nuclei constituting the molecule. In principle, knowing just the ground state of a molecular Hamiltonian and its energy already provides insight into the magnetic, optical, electrical, and mechanical behavior of a molecule, as well as into its thermodynamic characteristics, and reactivity. Thus, the *ab initio* quantum chemistry remains a long standing and cherished endeavour of computational quantum physics.

The main focus of this thesis is improving the accuracy and the computational performance of *ab initio* quantum chemistry calculations performed with ANQS. The manuscript consists of three parts.

Part I provides the necessary background for the rest of the thesis. In Chapter 1 we cover the main ingredients of variational Monte Carlo, explain what neural networks are and expose the reader to the field of neural quantum states. In Chapter 2 we introduce the problem of *ab initio* quantum chemistry in second quantisation and discuss conventional quantum chemistry methods to solve it. We also survey existing applications of NQS to quantum chemistry. Chapter 3 is dedicated to autoregressive NQS. We overview their main feature — a procedure for fast and unbiased sampling — and explain why it particularly suits quantum chemistry applications. As well, we cover two further advances of ANQS: autoregressive statistics sampling and autoregressive sampling without replacement.

Part II explores how taking into account symmetries of a quantum system affects ANQS optimisation. In Chapter 4 we consider *quantum number* symmetries and devise an algorithm to incorporate them into ANQS optimisation. We numerically showcase the benefits of this algorithm in Chapter 5. We identify quantum number symmetries inherent to quantum chemistry and show that symmetry-aware ANQS calculations are superior to the existing NQS quantum chemistry applications. Specifically, we reach the level of accuracy reported in previous works with more

than an order of magnitude speedup and achieve chemical accuracy for all studied molecules, which was a milestone unreported at the time of the study.

Part III is concerned with improving the computational performance of ANQS optimisation, known to be extremely challenging in terms of the required compute. In Chapter 6 we describe a composite approach addressing bottlenecks of sampling and energy evaluation inherent to quantum chemistry calculations. This approach also aims to enhance the general optimisation convergence via advanced gradient postprocessing technique. Chapter 7 deals with the practical aspect of the proposed approach, namely with its efficient implementation on a graphical processing unit (GPU). Finally, in Chapter 8 we present the numerical results highlighting the advantages of our method: it requires substantially less resources and exhibits more than order of magnitude speedup compared to the previous works. Using only a single GPU, we surpass conventional quantum chemistry methods such as CISD, CCSD and CCSD(T) for system sizes that have been previously inaccessible to NQS (118 qubits, 10^{15} Slater determinants, $3 \cdot 10^6$ Hamiltonian terms).

We believe that expanding the ANQS symmetry toolbox as well as boosting the optimisation performance brings ANQS closer to being a competitive choice for challenging quantum chemistry calculations. We also hope that this thesis offers the reader a few enjoyable insights, as the author undoubtedly experienced many while working on it.

Part I

Background

1

Variational Monte Carlo and neural quantum states

Contents

We start the background part of the thesis with a brief overview of variational Monte Carlo and deep learning, the two parent fields of neural quantum states. First, we explain how observables of interest and their gradients can be evaluated via sampling from the ansatz Born distribution, which is a key premise of variational Monte Carlo. Following this, we cover the basics of Markov chain Monte Carlo sampling and the Metropolis-Hastings algorithm, which enable Born distribution sampling for ansatzes of the most generic nature. We wrap up the discussion of VMC with a description of stochastic reconfiguration, a powerful gradient postprocessing

technique that significantly improves the convergence of NQS optimisation.

In the latter part of this chapter we provide a self-contained introduction to neural networks. As a primary example, we consider feedforward neural networks, known to be universal function approximators. We detail the optimisation of these networks via the backpropagation algorithm and briefly touch upon other neural network architectures.

Finally, we present NQS as a family of potent variational ansatzes and highlight the most important advances which took place since the field inception.

1.1 Variational Monte Carlo

In this thesis we focus on systems of N interacting qubits whose states are described with a superposition of 2^N computational basis vectors:

$$|\psi\rangle = \sum_{\mathbf{x}=0}^{2^N-1} \psi(\mathbf{x}) \, |\mathbf{x}\rangle \, ; \; \mathbf{x} \in \{0,1\}^{\otimes N} \text{ and } \psi(\mathbf{x}) \in \mathbb{C}. \tag{1.1}$$

The variational Monte Carlo (VMC) approach relies on the fact that quantum expectation values can be cast in the form of mean values with respect to the Born distribution $p(\mathbf{x}) \coloneqq \frac{|\psi(\mathbf{x})|^2}{\sum_{\mathbf{x}} |\psi(\mathbf{x})|^2}$. Specifically, the mean energy $\langle E \rangle \coloneqq \frac{\langle \psi | \hat{H} | \psi \rangle}{\langle \psi | \psi \rangle}$ can be represented as an expectation value as follows [11]:

$$\langle E \rangle = \mathbb{E}(E_{\text{loc}}(\mathbf{x})) = \sum_{\mathbf{x}} E_{\text{loc}}(\mathbf{x}) p(\mathbf{x}), \tag{1.2}$$

where $E_{\text{loc}}(\mathbf{x})$ is the so-called *local energy* estimator:

$$E_{\text{loc}}(\mathbf{x}) \coloneqq \sum_{\mathbf{x}' : H_{\mathbf{x}\mathbf{x}'} \neq 0} \frac{H_{\mathbf{x}\mathbf{x}'} \psi(\mathbf{x}')}{\psi(\mathbf{x})}. \tag{1.3}$$

Such expectation values can be estimated by obtaining a batch $\mathcal{B}$ of N_{s} samples from the distribution $p(\mathbf{x})$. Let $\mathcal{U} \coloneqq \text{Unique}(\mathcal{B})$ be the set of unique samples in $\mathcal{B}$ and let $n(\mathbf{x})$ be the occurrence number for the sample $\mathbf{x}$ so that $\sum_{\mathbf{x} \in \mathcal{U}} n(\mathbf{x}) = N_{\text{s}}$. We call the set of pairs $\{ \, (\mathbf{x}, n(\mathbf{x})) \mid \mathbf{x} \in \mathcal{U} \, \}$ *the sampling statistics*. Using the sampling statistics, one estimates the energy of the state as a Monte Carlo expectation value as follows:

$$\langle E \rangle \approx \sum_{\mathbf{x} \in \mathcal{U}} E_{\text{loc}}(\mathbf{x}) \cdot \frac{n(\mathbf{x})}{N_{\text{s}}}. \tag{1.4}$$

Importantly, in real-world Hamiltonians there are only polynomially many $\mathbf{x}'$ such that $H_{\mathbf{x}\mathbf{x}'} \neq 0$ for any given $\mathbf{x}$. Thus, for a fixed N_s the value given by Eq. 1.4 can be evaluated efficiently.

When a quantum state is represented with an ansatz, the energy gradient with respect to ansatz parameters can be estimated as a Monte Carlo expectation value too [11]:

$$\begin{aligned}\nabla_\theta \langle E \rangle = 2\,\mathrm{Re}\,\{&\mathbb{E}\left[E_\mathrm{loc}(\mathbf{x}) \cdot \nabla_\theta \ln \psi^*_\theta(\mathbf{x})\right] \\ &- \mathbb{E}\left[E_\mathrm{loc}(\mathbf{x})\right] \cdot \mathbb{E}\left[\nabla_\theta \ln \psi^*_\theta(\mathbf{x})\right]\}\,.\end{aligned} \tag{1.5}$$

This estimate is employed to update the ansatz parameters following the gradient descent rule or any advanced modification thereof [11].

Importantly, the described techniques are applicable not only to qubit systems, but rather to any quantum system. Moreover, an ability to calculate unnormalised amplitudes $\psi_\theta(\mathbf{x})$ is in principle enough to run a VMC optimisation. If $\psi_\theta(\mathbf{x})$ can be calculated efficiently, then it is possible to approximately calculate $\nabla_\theta \psi_\theta(\mathbf{x})$ with the finite difference method. Alternatively, if the calculation of $\psi_\theta(\mathbf{x})$ forms a valid differentiable computational graph, one can use the automatic differentiation algorithm. At the same time, sampling from $p(\mathbf{x})$ can be achieved by feeding the unnormalised probabilities $|\psi_\theta(\mathbf{x})|^2$ into any Markov chain Monte Carlo (MCMC) algorithm, with the celebrated Metropolis-Hastings algorithm being the most common choice. We introduce Metropolis-Hastings sampling and automatic differentiation in more detail later in the text.

This makes VMC a powerful tool capable to address a wide range of quantum many-body problems with ansatzes of the most generic nature. This is in contrast with, for example, *matrix product states (MPS)*, which are a celebrated family of ansatzes with a *specific* functional form. This form is particularly well-suited to representing the ground states of one-dimensional interacting spin systems and allows for a highly efficient MPS optimisation, which made them the ansatz of choice for studying such systems. However, it is challenging to optimise MPS applied to systems in higher dimensions. Equally important, MPS are *known* to

represent only a restricted class of quantum states inhabiting the Hilbert space (see Section 1.3.1) [9, 12, 13].

At the same time, while VMC optimisation enjoys much wider generality, it is no easy feat too. The crucial concerns are (i) picking an expressive enough ansatz; (ii) having an efficient sampling procedure for it (as discussed further, the Metropolis-Hastings algorithm is not without its shortcomings) and (iii) actually converging to the optimal set of parameters. Neural quantum states hold a particular appeal with regard to the first issue, since neural networks are known to be universal approximators [14, 15], as further discussed in Section 1.2. In Chapter 3 we show that the second issue can be addressed with *autoregressive* neural quantum states. Finally, in Section 1.1.2 we discuss how to improve the convergence of VMC optimisation using the stochastic reconfiguration gradient postprocessing technique.

1.1.1 Metropolis-Hastings sampling

The Metropolis-Hastings algorithm allows one to sample from an unnormalised probability distribution $p(\mathbf{x})$. We do not employ this algorithm in any of our numerical experiments, since we resort to *autoregressive* sampling, devoid of many drawbacks of the Metropolis-Hastings algorithm. Hence, in this section we cover the Metropolis-Hastings only superficially to put ANQS into context.

Markov chain Monte Carlo

The Metropolis-Hastings algorithm belongs to a broader family of Markov chain Monte Carlo (MCMC) methods. Markov chain is a stochastic process which represents a discrete time evolution of a probability distribution defined on some finite state space.[1] Without loss of generality let us consider an abstract system with the state space corresponding to N-bit vectors. Any probability distribution $p_0(\mathbf{x})$ describing a chance to find the system in one of possible states is a vector with 2^N non-negative values summing up to one. To define a Markov chain, one

[1] There exist straightforward generalisations of Markov chains to continuous time evolutions and/or infinite state spaces. However, for the purpose of explanation we consider Markov chains with finite state spaces which evolve in discrete time.

defines a so-called $2^N \times 2^N$ *transition matrix* $T(\mathbf{x}'|\mathbf{x})$, which describes a single step of the discrete time evolution. Each element of T is a non-negative probability to find the system after evolution in a state $\mathbf{x}'$, given that initially it was in a state $\mathbf{x}$. We also require that columns of T are normalised, i.e. $\sum'_{\mathbf{x}} T(\mathbf{x}'|\mathbf{x}) = 1$. Under such definition the evolved probability distribution can be obtained by multiplying $p_0(\mathbf{x})$ by $T(\mathbf{x}'|\mathbf{x})$ from the left, i.e. $p_1(\mathbf{x}') \coloneqq \sum_{\mathbf{x}} T(\mathbf{x}'|\mathbf{x}) p_0(\mathbf{x})$.[2]

It can be shown that under rather mild conditions the Markov chain has a stationary probability distribution $p_\infty(\mathbf{x})$ such that $p_\infty(\mathbf{x}') = \sum_{\mathbf{x}} T(\mathbf{x}'|\mathbf{x}) p_\infty(\mathbf{x})$. Equally as important, in the limit of infinitely many evolution steps *any* starting probability distribution p_0 converges to the stationary one, i.e. $\forall p_0 \ \lim_{n\to\infty} T^n p_0 = p_\infty$ (we drop $\mathbf{x}$ in parentheses for brevity). If the Markov chain has the latter property it is referred to as *ergodic* [16].

MCMC methods employ this property in the following way. First, one designs such transition matrix T that its stationary distribution $p_\infty(\mathbf{x})$ is the desired $p(\mathbf{x})$. Then, one initialises the system in a random state $\mathbf{x}_0$ and probabilistically transitions to another state $\mathbf{x}'$ with the probability given by $T(\mathbf{x}'|\mathbf{x}_0)$. After that, one takes $\mathbf{x}'$ as the new starting state and repeats the transition process again. Thus, one forms a sequence of "visited" states $\mathbf{x}_1, \mathbf{x}_2, \ldots$. In the limit of large number of discrete time steps the numbers of occurrences for the visited states become distributed according to $p_\infty(\mathbf{x})$.

The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm provides a specific recipe for the discrete time evolution. Each discrete time step proceeds in two stages [16]:

1. *Propose.* Given a current state $\mathbf{x}_t$ of the Markov chain, one *proposes* a new state $\mathbf{x}'$ according to a *proposal* distribution $q(\mathbf{x}'|\mathbf{x})$ chosen before the algorithm starts.

[2] It is easy to check that normalised columns of $T(\mathbf{x}'|\mathbf{x})$ imply that $p_1(\mathbf{x})$ is a valid probability distribution.

2. *Accept/reject.* One decides whether to move the Markov chain in the proposed state, or whether to leave it in the current one. To that end, one calculates the *acceptance* probability defined as follows:

$$A(\mathbf{x}'|\mathbf{x}) = \min\left(1, \frac{p(\mathbf{x}')q(\mathbf{x}|\mathbf{x}')}{p(\mathbf{x})q(\mathbf{x}'|\mathbf{x})}\right). \tag{1.6}$$

Then, one samples a random variable $u \sim \textsc{Uniform}(0, 1)$. If $u \leq A(\mathbf{x}'|\mathbf{x})$, one *accepts* the new state, i.e. sets $\mathbf{x}_{t+1}$ to $\mathbf{x}'$. Otherwise, one *rejects* it and thus $\mathbf{x}_{t+1} = \mathbf{x}_t$.

This amounts to such transition matrix $T(\mathbf{x}'|\mathbf{x}) = A(\mathbf{x}'|\mathbf{x})q(\mathbf{x}'|\mathbf{x})$ that the desired $p(\mathbf{x})$ satisfies the *detailed balance* condition, i.e. $T(\mathbf{x}'|\mathbf{x})p(\mathbf{x}) = T(\mathbf{x}|\mathbf{x}')p(\mathbf{x}')$. It can be shown that under rather mild conditions on the proposal distribution $q(\mathbf{x}'|\mathbf{x})$ (e.g. it allows to explore all possible $\mathbf{x}'$ regardless of the starting $\mathbf{x}$) the resulting Markov chain is also ergodic. Ergodicity and the detailed balance condition ensure that the Markov chain converges to $p(\mathbf{x})$ [16].

Importantly, calculating the acceptance probability does not require *normalised* $p(\mathbf{x})$. Instead, one needs to evaluate only the *ratios* between $p(\mathbf{x})$ and $p(\mathbf{x}')$. As a result one can employ generic ansatzes without worrying about the normalisation of their amplitudes.

However, Metropolis-Hastings sampling is not without its drawbacks. To name a few, depending on the target probability distribution the Markov chain might require a prohibitively large number of steps to converge. In addition, even when the Markov chain is converged, its consecutive states are correlated. To avoid high correlation one usually has to run sampling for longer and *thin* the chain by selecting the states separated by several time steps. Finally, Metropolis-Hastings sampling might struggle to represent a multimodal target probability distribution, i.e. the one having clusters of high probability located far from each other in the state space. Addressing this issue requires a careful choice of the proposal distribution which should be capable to efficiently explore the full state space [16].

1.1.2 Stochastic Reconfiguration

Another crucial part of VMC optimisation is gradient postprocessing: there is a compelling evidence that much better variational energies might be achieved if one resorts to *stochastic reconfiguration (SR)*, also known as *quantum natural gradient* [17]. Standard gradient descent performs poorly in curved parameter spaces, where small changes in ansatz *parameters* do not necessarily correspond to small changes in the ansatz *state*. Thus, the loss function gradient correctly indicates the steepest descent direction only for infinitesimal vicinity of the current parameter values. With a *finite* gradient descent step one can make a suboptimal step and a overshoot a narrow ravine in the parameter manifold.

Stochastic reconfiguration modifies the energy gradient at every iteration to account for the curvature of the underlying variational manifold. To that end, for an ansatz with N_p parameters $\{\theta_p\}$ one evaluates the so-called $N_\mathrm{p} \times N_\mathrm{p}$ *quantum geometric tensor (QGT)* which captures the local curvature of the parameter space [18, 19]:

$$S_{pq} \coloneqq \langle \partial_p \psi | \partial_q \psi \rangle - \langle \partial_p \psi | \psi \rangle \langle \psi | \partial_q \psi \rangle \,;\; p, q \in 1..N_\mathrm{p}, \tag{1.7}$$

where $|\partial_p \psi\rangle \coloneqq \frac{\partial |\psi\rangle}{\partial \theta_p}$. One can "flatten" the curvature of the parameter space by multiplying the energy gradient with an inverse of QGT: $\nabla E \to S^{-1} \nabla E$.

During the VMC optimisation one estimates S stochastically after having sampled N_s samples, of which N_unq are unique. First, one evaluates the Jacobian matrix $J_{\mathbf{x}p} \coloneqq \frac{\partial \log \psi(\mathbf{x})}{\partial \theta_p}$, $\mathbf{x} \in \mathcal{U}$ of size $N_\mathrm{unq} \times N_\mathrm{p}$. Here we index the rows of Jacobian with unique basis vectors. Second, one centers and normalises the Jacobian to obtain the following matrix (underneath each matrix we write its dimensions):

$$\underset{N_\mathrm{unq} \times N_\mathrm{p}}{O_{\mathbf{x}p}} \coloneqq \frac{1}{\sqrt{N_\mathrm{s}}} \left(J_{\mathbf{x}p} - \sum_{\mathbf{x}' \in \mathcal{U}}^{N_\mathrm{s}} J_{\mathbf{x}'p} \frac{n(\mathbf{x}')}{N_\mathrm{s}} \right). \tag{1.8}$$

Finally, one evaluates S as follows [11, 20]:

$$\underset{N_\mathrm{p} \times N_\mathrm{p}}{S} = \underset{N_\mathrm{p} \times N_\mathrm{unq}}{O^\dagger} \cdot \underset{N_\mathrm{unq} \times N_\mathrm{p}}{O}. \tag{1.9}$$

A naïve implementation of the SR transformation requires the inversion of explicitly constructed S in $\Theta(N_\mathrm{p}^3)$ operations. This is feasible only for ansatzes with up to few thousands of parameters. Fortunately, in Ref. [20] the authors showed how to use the product structure of S to reduce the total number of operations to $\Theta(N_\mathrm{p} N_\mathrm{unq}^2)$. Since typical VMC optimisation involves $N_\mathrm{s} \sim 10^3 \div 10^4$, this enabled scaling ansatzes to previously inaccessible numbers of parameters, easily surpassing hundreds of thousands.

Another important contribution was made in Ref. [21], where the authors showed how to efficiently implement *regularised* SR, i.e. the one where the matrix S is supplemented with a constant diagonal shift ε: $S \to S + \varepsilon I$. Regularising SR was known to alleviate the instabilities arising from stochastic evaluation of S, and thus a computationally efficient implementation allowed the authors to achieve state-of-the-art variational energies when studying the J_1–J_2 model, which is a paradigmatic frustrated magnetism model notably difficult for classical computational methods.

1.2 Neural networks

Neural networks are parameterised function approximators which are optimised to match a target function as accurately as possible. The degree of accuracy is usually estimated via the so-called *loss function* $\mathcal{L}$, whose gradient is used to optimise the network parameters. Specific to neural networks is the *functional form* of the approximation. In particular, one applies to the input vector $\mathbf{x}$ a sequence of L nested *nonlinear* transformations $f^{(l)}_{\theta^{(l)}}$, $l \in 1, \ldots, L$, each parameterised with a set of parameters θ_l, as follows:

$$f^{\mathrm{NN}}_\theta(\mathbf{x}) = f^{(L)}_{\theta^{(L)}}\left(f^{(L-1)}_{\theta^{(L-1)}}\left(\ldots f^{(1)}_{\theta^{(1)}}(\mathbf{x})\right)\right) \equiv f^{(L)}_{\theta^{(L)}} \circ f^{(L-1)}_{\theta^{(L-1)}} \circ \ldots \circ f^{(1)}_{\theta^{(1)}}(\mathbf{x}), \tag{1.10}$$

where the output of $f^{\mathrm{NN}}_\theta(\mathbf{x})$ can be of any dimension. In the context of this thesis a naturally arising example of a loss function is the energy of a quantum state, while the neural network aims to approximate the ground state wave function amplitudes, i.e. it has an output of dimension one: $f^{\mathrm{NN}}_\theta(\mathbf{x}) \equiv \psi(\mathbf{x})$.

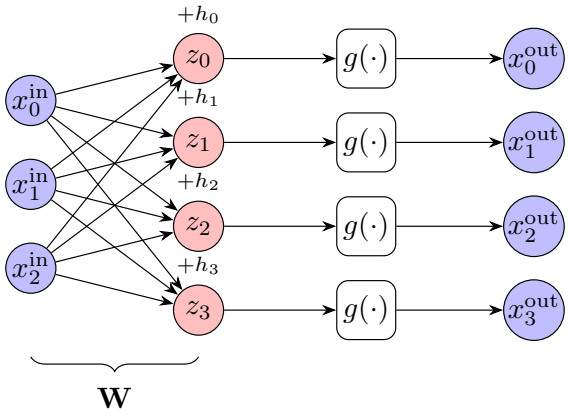


Figure 1.1: A feedforward layer.

The nonlinear transformations of a neural network are usually referred to as *layers*. While each layer can be of rather generic form, let us consider as an example a specific type of layer called *feedforward* or *dense*.

1.2.1 Feedforward neural networks

Suppose an input vector $\mathbf{x}$ has dimension N_{in}. First, one multiplies it by a so-called *weights* matrix $\mathbf{W}$ with dimensions $N_{\text{out}} \times N_{\text{in}}$ and then adds a so-called *bias* vector of size N_{out}. This produces an intermediate vector $\mathbf{z} \coloneqq \mathbf{W}\mathbf{x} + \mathbf{b}$ of dimension N_{out}. Second, one applies to $\mathbf{z}$ am elementwise nonlinear function g known as the *activation* function. As a result, the feedforward layer transforms the input as follows:

$$f^{\text{FF}}_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = g(\mathbf{z}) = g(\mathbf{W}\mathbf{x} + \mathbf{b}). \tag{1.11}$$

Here the weight matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$ are the trainable parameters of this layer. A pictorial representation of this layer is given in Fig. 1.1. Vectors are represented as columns of nodes known as *neurons*. A connection between the neurons i and j of vectors $\mathbf{z}$ and $\mathbf{x}$ correspondingly imply that $W_{ij} \neq 0$.

A neural network consisting of several feedforward layers is called a *multilayer perceptron (MLP)* or simply a *feedforward neural network (FNN)*. In this case the neurons representing the vectors $\mathbf{x}$ and $\mathbf{x}^{(L)} \coloneqq f^{\text{NN}}_{\theta}(\mathbf{x})$ are called *visible*, while the remaining neurons corresponding to outputs of every layer but last are called *hidden*.

Box 1.1: A historical note on neural networks naming

The historical reasons behind calling nodes neurons become clear if one considers the operation of a single node in Fig. 1.1. Suppose the nonlinear function $g(x)$ is the Heaviside step function $H(x) \coloneqq \begin{cases} 1, & x \geq 0; \\ 0, & x < 0. \end{cases}$ Then, the output node produces a non-zero value (or *activates*) only if the weighted sum of the connected node values is larger than some bias value. This is similar to what happens in actual neurons, which produce a voltage signal only when a weighted sum of signals from connected neurons is larger than a certain threshold voltage.

The number of layers in a neural networks is usually referred to as its *depth*, while the number of output neurons for a given layer is referred to as its *width*.

The appeal of FNNs stems from the so-called *universal approximation theorems*. The two most known of them claim that (i) a single layer FNN can approximate an arbitrary function provided that its width is sufficiently large; (ii) an FNN of a fixed width can approximate an arbitrary function provided that its depth is sufficiently large. In practice it has been empirically found that given the same parameter count deeper networks perform better than the wider ones. This shifted the research focus to deep neural networks, which explains the origin of term *deep learning*.

1.2.2 Backpropagation algorithm and automatic differentiation

To evaluate the gradient of a loss function during neural network optimisation one employs the so-called backpropagation algorithm. This algorithm is a special case of the so-called *automatic differentiation*, which allows one to numerically evaluate the Jacobian of a multi-dimensional function with respect to its arguments, provided that the function is specified as a sequence of differentiable operations also known as a *computational graph*. By design, neural networks belong to this class of functions.

The main idea behind automatic differentiation is to consider the result of each differentiable operation as an intermediate variable and repeatedly use the chain derivative rule to compose the final Jacobian as a product of intermediate Jacobians. As an example, let us consider the steps required to evaluate the Jacobian of a

simple *scalar* function, i.e. its gradient. Suppose one wants to evaluate the gradient of function $f(x_1, x_2) = \sin^2(x_1 + x_2)$ at the point $(x_1, x_2) = (-\frac{\pi}{2}, \frac{\pi}{4})$. One splits the calculation into steps as follows:

1. $x_1 + x_2 =: z_1 = -\frac{\pi}{4}$;

2. $\sin(z_1) =: z_2 = -\frac{1}{\sqrt{2}}$;

3. $z_2^2 =: f(x_1, x_2) = \frac{1}{2}$.

To evaluate $\frac{\partial f}{\partial x_1}\Big|_{x_1=-\frac{\pi}{2}}$, one uses the chain rule:

$$\frac{\partial f}{\partial x_1}\bigg|_{x_1=-\frac{\pi}{2}} = \frac{\partial f}{\partial z_2}\bigg|_{z_2=-\frac{1}{\sqrt{2}}} \cdot \frac{\partial z_2}{\partial z_1}\bigg|_{z_1=-\frac{\pi}{4}} \cdot \frac{\partial z_1}{\partial x_1}\bigg|_{x_1=-\frac{\pi}{2}}. \tag{1.12}$$

Evaluating each derivative gives:

$$\begin{gathered}\frac{\partial f}{\partial z_2}\bigg|_{z_2=-\frac{1}{\sqrt{2}}} = 2z_2|_{-\frac{1}{\sqrt{2}}} = -\sqrt{2};\ \frac{\partial z_2}{\partial z_1}\bigg|_{z_1=-\frac{\pi}{4}} = \cos(z_1)|_{z_1=-\frac{\pi}{4}} = \frac{1}{\sqrt{2}};\\ \frac{\partial z_1}{\partial x_1}\bigg|_{x_1=-\frac{\pi}{2}} = 1|_{x_1=-\frac{\pi}{2}} = 1;\end{gathered} \tag{1.13}$$

Thus:

$$\frac{\partial f}{\partial x_1}\bigg|_{x_1=-\frac{\pi}{2}} = -\sqrt{2} \cdot \frac{1}{\sqrt{2}} \cdot 1 = -1. \tag{1.14}$$

It can be shown in a similar way that $\frac{\partial f}{\partial x_2}\Big|_{x_2=\frac{\pi}{4}} = -1$, which thus completes the calculation of the gradient.

It is straightforward to generalise this example to the case of more sophisticated functions with multiple arguments. Importantly, the key deep learning software libraries such as PyTorch [22], TensorFlow [23] and Jax [24] have automatic differentiation engines at their core, which implement Jacobian functions for elementary differentiable operations, keep track of the computation graph and perform multiplication of chained Jacobians in an automated way.

Box 1.2: Backpropagation algorithm

The name *backpropagation* refers to evaluating the gradients of the loss function with respect to the neural network layers in *reverse order*, starting from the last (L-th) layer and ending with the first layer.
Suppose a neural network aims to approximate a function whose values $\mathbf{y}$ are given for a set $\mathcal{D}_{\text{train}}$ of input vectors $\mathbf{x}^{(0)}$. Suppose we calculate the l_2-error loss function between the network predictions $\mathbf{x}^{(L)} \coloneqq f_\theta^{\text{NN}}(\mathbf{x}^{(0)})$ and the actual function values $\mathbf{y}$. One starts with evaluating the gradient of the loss function with respect to the network output $\mathbf{x}^{(L)}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} = -2 \sum_{(\mathbf{x}^{(0)},\mathbf{y})\in\mathcal{D}_{\text{train}}} \left(\mathbf{y} - \mathbf{x}^{(L)}\right). \tag{1.15}$$

Then, one recursively evaluates the gradient of the loss with respect to the *output* of layer l given the gradient of the loss with respect to the output of layer $l+1$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l+1)}} \cdot \frac{\partial \mathbf{x}^{(l+1)}}{\partial \mathbf{x}^{(l)}}. \tag{1.16}$$

Here $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l+1)}}$ is already known and $\frac{\partial \mathbf{x}^{(l+1)}}{\partial \mathbf{x}^{(l)}}$ can be obtained by automatically differentiating the function $f^{(l+1)}(\mathbf{x}^{(l)}) \equiv \mathbf{x}^{(l+1)}$. Finally, the gradient of the loss function with respect to the l-th layer *parameters* is obtained using the chain rule too:

$$\frac{\partial \mathcal{L}}{\partial \theta^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial \mathbf{x}^{(l)}}{\partial \theta^{(l)}}. \tag{1.17}$$

Similarly to Eq. (1.16), here $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$ is known and $\frac{\partial \mathbf{x}^{(l)}}{\partial \theta^{(l)}}$ is obtained using the automatic differentiation of $f^{(l)}_{\theta^{(l)}}(\mathbf{x}^{(l-1)}) \equiv \mathbf{x}^{(l)}$.

1.2.3 Further advances

In the past decade, the field of deep learning has experienced tremendous growth, with state-of-the-art neural networks excelling in tasks traditionally associated with human intelligence, such as the generation and recognition of text and audiovisual information.

Part of the deep learning progress can be attributed to technical innovations, most notably the use of accelerated calculations leveraging the inherent parallelism of graphical processing units (GPUs). Another crucial component is developing better optimisation techniques, which aim to address the issues of the conventional gradient descent mentioned in Section 1.1.2. One of the most notable example

is the Adam optimiser, which adaptively calculates the learning rate for each parameter based on the gradient values at the previous iterations. This allows to account for the curvature of the loss landscape and significantly improve the training convergence [25].

The largest part of the deep learning success can be arguably associated with the development of new neural network architectures. While feedforward neural networks are universally expressive, it might be challenging to train them either due to the high computational cost of dense layers or due to a lack of useful *inductive bias*. The latter term refers to situations where a neural network architecture is tailored to certain aspects of the considered data set, which enhances optimisation convergence.

A paradigmatic example of a successful architectural choice are *convolutional* neural networks used in image recognition tasks [26]. These networks replace feedforward layers with more restricted trainable convolution operations, which are substantially less computationally demanding than matrix-vector multiplications. In addition, they provide a necessary inductive bias, where small kernels of the first layers capture local features of an image, while deeper layers hierarchically aggregate these into abstract representations. As a result, the majority of deep learning image recognition pipelines use large pretrained convolutional neural networks as their first stage.

Another important architecture is that of *recurrent* neural networks [27]. These networks are used to work with sequential data, for example, in translation tasks. One maps all words to vectors of the same dimension and processes these vectors one-by-one with *the same* neural network known as a *recurrent cell*. To capture the information about the already processed elements of sequence, one introduces a *context vector* which is updated during each recurrent cell application. The further development of ideas behind recurrent neural networks lead to the Transformer architecture, which is the main component of large language models such as generative pretrained Transformers (GPT) [28].

1.3 Neural quantum states

In 2016 Carleo and Troyer used a neural network called *restricted Boltzmann machine (RBM)* as a variational Monte Carlo ansatz and thus pioneered the field of neural quantum states [2] (see Box 1.3 for a brief introduction to RBMs). They demonstrated that after Metropolis-Hastings optimisation an RBM could closely describe the ground states of systems governed by the transverse field Ising and antiferromagnetic Heisenberg Hamiltonians, as well as track their dynamics in time. This success jumpstarted a line of research exploring the application of neural networks to the study of quantum many-body systems.

Since its conception the field flourished considerably and now neural quantum states are among state-of-the-art methods of computational quantum many-body physics. NQS are capable to tackle various problems across a multitude of physical systems such as interacting spins [29–33], bosons [34–36], nuclei [37–39] and fermions in first and second quantisation [40–50]. The range of NQS applications includes, but is not limited to, calculation of ground and excited states [5, 29, 30, 33, 34, 51], real-time dynamics [4, 52–54], study of open quantum systems [3, 55–57], quantum state tomography [6, 58–60] and classical simulation of quantum computing [61, 62].

Let us provide a glance at the two key features of neural quantum states which make them stand out compared to other variational ansatzes: their high expressibility and high flexibility. For a more methodical overview of the field we refer the reader to two recent reviews by Medvidović and Moreno [63] and by Lange *et al.* [64].

1.3.1 Neural quantum states are expressive

A body of works established what classes of quantum states can be efficiently represented with NQS. First, Deng et al. [66] have shown that RBMs are able to efficiently represent states with volume law entanglement (see Box 1.4 for a brief explanation of entanglement entropy scaling laws). This demonstrates that even a very simple NQS might be already superior to such celebrated ansatzes as matrix product states. However, the numerical study provided in Ref. [66] revealed that the average entanglement entropy of random RBM states is noticeably smaller than

Box 1.3: Restricted Boltzmann Machines

Restricted Boltzmann machines are generative networks, i.e. they aim to represent probability distributions over the bit vectors of length N. They consist of two neuron layers — a *visible* one of N neurons and a *hidden* one of M neurons. Network parameters are bias vectors $\mathbf{v} \in \mathbb{C}^N$ and $\mathbf{h} \in \mathbb{C}^M$ (for the visible and hidden layers correspondingly) and the weight matrix $\mathbf{w} \in \mathbb{C}^M \times \mathbb{C}^N$. The RBM parameterisation is *different* from that of a feedforward layer and is as follows:

$$\psi_{\mathbf{v},\mathbf{h},\mathbf{w}}(\mathbf{x}) = \prod_{i=1}^{M} \mathrm{e}^{v_i x_i} \prod_{j=1}^{M} \left(1 + e^{h_j + \sum_{i=1}^{N} w_{ji} x_i}\right) |\mathbf{x}\rangle \,, \tag{1.18}$$

Thanks to representability theorems [65], RBMs could approximate any probability distribution arbitrary well in the limit of infinitely growing network size. In Ref. [2] the authors noted that if RBM parameters are complex-valued, then by analogy the network can represent arbitrary N-qubit quantum states.

the average entropy of random quantum states, which indicates that the former belong to a rather restricted part of the Hilbert space.

This statement was further supported in Ref. [67], where the authors showed that RBMs cannot efficiently represent arbitrary states which either are generated by a polynomial-size quantum circuit or are ground states of gapped Hamiltonians. At the same time, they proved that this class of states can be efficiently represented with a more general NQS ansatz, namely the one based on the deep Boltzmann machine (DBM) with one visible, one hidden and one deep layer of neurons. The authors of Ref. [68] provided another evidence of high expressive power of DBMs — they derived explicit rules to modify the shape and parameters of an initial DBM during each step of the imaginary time evolution under the transverse field Ising, antiferromagnetic Heisenberg or J_1–J_2 Hamiltonians.

Other architectures proved to be more expressive than RBMs too: the Ref. [69] demonstrated that NQS built upon convolutional neural networks can support volume-law entanglement polynomially more efficient than RBMs, while recurrent neural networks can represent logarithmic corrections to the area law entanglement scaling in 1D, a feat inaccessible for MPS. Finally, the authors of Ref. [70] showed that the set of quantum states efficiently representable with NQS is strictly larger that

the set of states representable with tensor network ansatzes which are equipped with a deterministic polynomial-time procedure to calculate the observables of interest.

Box 1.4: Entanglement entropy scaling

Suppose the state of a multipartite quantum system is described with a density matrix $\hat{\rho}$. To calculate its entanglement entropy one partitions the system into two subsystems A and B, and calculates the reduced density matrix for one of them: $\hat{\rho}_\mathrm{A} = \mathrm{Tr}_\mathrm{B}\,\hat{\rho}$. Then, one calculates the entanglement entropy as $S := -\,\mathrm{Tr}\,\hat{\rho}_\mathrm{A} \ln \hat{\rho}_\mathrm{A}$.[a] The states whose entanglement entropy scales with the size of the *boundary* between A and B are said to follow the *area law*. At the same time, states whose whose entanglement entropy scales with the size of A and B themselves are said to satisfy the *volume law*. It is known that certain classes of variational quantum states — e.g. matrix product states — can support only area law entanglement scaling [9, 12, 13].

[a] It can be easily shown that the value of the entanglement entropy does not depend on whether one considers the reduced density matrix for subsystem A or B, but it *does* depend on the chosen partition.

1.3.2 Neural quantum states are flexible

Another advantage of NQS is their remarkable flexibility in the choice of underlying neural network architecture. This allows one to incorporate into the ansatz a useful inductive bias related to the physical properties of the system at consideration. For example, ansatzes built upon convolutional [30, 32], recurrent [29] or Transformer [33, 71] architectures proved to be extremely well suited to describe the systems of interacting spins on a lattice. This can be attributed to the fact that such neural networks respect the translational symmetry of lattice. Other, more complicated symmetries, can be incorporated with the so-called *group convolutional* neural networks or various symmetrisation techniques [5, 72].

Apart from adopting the existing solutions, a significant effort was directed towards the design of custom architectures. In first quantisation, custom architectures were developed to represent the real space wave functions of either bosons [36] or fermions [40–42] and correctly account for the exchange statistics of particles. In second quantisation the most notable examples are neural network backflow states (NNBF) and hidden fermion determinant states (HFDS) aimed at representing the

ground states of fermionic systems with notoriously complicated sign structure. Both NNBF and HFDS have been shown to significantly improve the energies achieved during the variational optimisation.

Finally, to address the disadvantages of the Metropolis-Hastings sampling, Ref. [73] introduced *autoregressive neural quantum states (ANQS)*. ANQS allow direct sampling from the underlying probability distribution with low computational cost. They proved to be especially successful when applied to the problem of quantum chemistry. We overview this problem in Chapter 2, while ANQS are considered in detail in Chapter 3.

2

Ab initio quantum chemistry

Contents

While Chapter 1 discussed the application of variational quantum Monte Carlo and neural quantum states to study of *arbitrary* quantum many-body systems, in this chapter we introduce a *specific* class of systems that is the focus of this thesis. In particular, we consider the problem of *ab initio* quantum chemistry, i.e. of finding the ground state of a molecule, also often referred to as its *electronic structure*.

We start with a short explanation of how a non-relativistic Schrödinger equation is mapped to a second quantised form, typically used in electronic structure calculations. Then, we overview the conventional quantum chemistry methods, including both well-established techniques and recent advancements in the field.

Finally, we discuss the existing applications of NQS to quantum chemistry as well as the outstanding challenges addressed in the subsequent chapters.

2.1 Molecular Hamiltonian

2.1.1 First quantisation

Many physical and chemical properties of a molecule are defined by its ground state, and thus solving the non-relativistic Schrödinger equation for a given molecule was recognised among the most important challenges of *ab initio* quantum chemistry since its dawn. However, as was mentioned in Introduction, finding the ground state of a quantum many-body Hamiltonian is a gargantuan task, and therefore one usually resorts to a set of established approximations.[1]

The most common approach is to take into account that nuclei are three orders of magnitude heavier than electrons. Thus, one might treat the former as motionless (i.e. fix their positions at a certain geometry) and consider a purely electronic Hamiltonian. This approximation is known as *Born-Oppenheimer* and it produces the following Hamiltonian:

$$\hat{H}_{\mathrm{BO}} = -\sum_i \frac{\nabla_i^2}{2} + \sum_{\substack{i,j \\ i<j}} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} - \sum_{i,A} \frac{Z_A}{|\mathbf{r}_i - \mathbf{R}_A|}. \tag{2.1}$$

Here $\mathbf{r}_i$ are the positions of the electrons, and $\mathbf{R}_A$ and Z_A are the fixed positions and charges of the nuclei respectively. We assume that electrons have unit mass and charge.

2.1.2 Second quantisation

In principle, one may seek the solution of the resulting Schrödinger equation as a position-space wave function, but in practice second quantisation methods are

[1] In fact, finding the ground state of a Hamiltonian is known to be QMA-complete, i.e. it is highly unlikely that there exist a polynomial time algorithm to find the ground state of an arbitrary Hamiltonian *even if* one has access to a quantum computer. Whether molecular Hamiltonians constitute a subclass of "simpler" Hamiltonians (i.e. "the ones found in Nature"), which are amenable to a polynomial-time treatment on a quantum computer, remains unknown. Nevertheless, in the case of classical computations (which is the subject of this thesis) it is widely believed that there does not exist a polynomial-time algorithm to find the ground states regardless of whether the considered Hamiltonian is molecular or not.

used more often. Under the second quantised formalism finding the ground state is equivalent to diagonalising the Hamiltonian matrix, which is substantially more straightforward than solving a partial differential equation corresponding to Eq. (2.1). To write down the second quantised Hamiltonian for a molecule with n_e electrons one proceeds with the two following steps. First, one picks a basis set of $N \geq n_\mathrm{e}$ single-electron wave functions (also known as spin-orbitals). Second, one projects the system Hamiltonian onto the basis of spin-orbitals, and obtains a conventional fermionic Hamiltonian describing one- and two-body interactions:

$$\hat{H}_\mathrm{SQ} = \sum_{ij} h_{ij} \hat{c}_i^\dagger \hat{c}_j + \sum_{ijkl} h_{ijkl} \hat{c}_i^\dagger \hat{c}_j^\dagger \hat{c}_k \hat{c}_l. \tag{2.2}$$

Any valid state of a molecule is thus represented as a linear combination of Slater determinants (SDs) comprised of n_e spin-orbitals out of N. The larger N is, the better such approach approximates the ground state, and hence much effort in the field of quantum chemistry is put into the design of finite basis sets expressive enough to represent real molecular wave functions.

2.1.3 Hartree-Fock procedure

In practice one starts with a basis set of *atomic* spin-orbitals, which approximate single-electron wave functions of each atom, and then hybdridises them with the so-called Hartree-Fock (HF) procedure to obtain the molecular spin-orbitals. This procedure produces spin-orbitals which are eigenstates of an effective Hamiltonian representing the motion of a single electron in the mean field of other electrons. In this case each orbital corresponds to a particular energy eigenvalue, which implies a natural ordering between them. More importantly, in the basis of Hartree-Fock orbitals the mean-field solution is represented with a single Slater determinant $|\mathbf{x}_\mathrm{HF}\rangle \coloneqq |\underbrace{11\ldots1}_{n_\mathrm{e}}\underbrace{0\ldots0}_{N-n_\mathrm{e}}\rangle$ where the first n_e lowest energy orbitals are filled and the rest are empty.

In many cases, a wave function consisting of a single Slater determinant $|\mathbf{x}_\mathrm{HF}\rangle$ can already provide a decent qualitative description of a molecule. However, as a mean field approach, the Hartree-Fock method is not expected to yield good

quantitative results. For example, it might not achieve the *chemical accuracy*, which is the ground state energy error of less than 1.6 millihartree (mHa). Such accuracy is required to predict the reaction rates at room temperature correctly to within an order of magnitude. Hence, a multitude of traditional quantum chemistry methods aims to refine the Hartree-Fock solution, and we overview a few of them relevant for benchmarking our ANQS quantum chemistry calculations in the following section.

2.2 Existing methods

The number of Slater determinants constituting the second quantised Hilbert space grows as $\binom{N}{n_\mathrm{e}}$, and thus, however tempting, exact diagonalisation of the Hamiltonian is possible only for small molecules. At the same time, with only a single Hartree-Fock determinant it is possible to capture a substantial part of the ground state physics. Hence, many conventional quantum chemistry methods consider various corrections to the HF state that can be evaluated in polynomial time [74].

2.2.1 Configuration interaction methods

The most conceptually straightforward is the *configuration interaction (CI)* family of methods. Their main idea is to diagonalise the Hamiltonian in a restricted Hilbert space spanned by excitations over $|\mathbf{x}_\mathrm{HF}\rangle$ up to a certain level. Specifically, let us introduce a sequence of excitation operators $\hat{T}_1, \hat{T}_2, \ldots, \hat{T}_{n_\mathrm{e}}$. Here the first operator $\hat{T}_1$, which is further referred to as a *single excitations operator*, is given as follows:

$$\hat{T}_1 \coloneqq \sum_{i \leq n_\mathrm{e}} \sum_{a > n_\mathrm{e}} t_i^a \hat{c}_a^\dagger \hat{c}_i. \tag{2.3}$$

This is an operator which represents all possible *single* excitations over the HF Slater determinant, i.e. such excitations where a single electron is taken from an occupied orbital (the one with an index $i \leq n_\mathrm{e}$) and is put to an unoccupied orbital (the one with an index $a > n_\mathrm{e}$). The values t_i^a are complex and are known as *excitation amplitudes*, since they define the amplitudes of the excited Slater determinants after $\hat{T}_1$ is applied to $|\mathbf{x}_\mathrm{HF}\rangle$.

The second operator $\hat{T}_2$ is the *double excitations operator* and is given as:

$$\hat{T}_2 \coloneqq \sum_{i,j\leq n_\mathrm{e}} \sum_{a,b>n_\mathrm{e}} t_{ij}^{ab} \hat{c}_a^\dagger \hat{c}_b^\dagger \hat{c}_i \hat{c}_j. \tag{2.4}$$

By analogy with $\hat{T}_1$, it is clear the $\hat{T}_2$ describes all possible excitations over the HF Slater determinant where *two* electrons are moved to unoccupied orbitals. In a similar way, $\hat{T}_3$ describes all triple excitations, and so on. Having defined excitation operators, one considers the *cluster operator*:

$$\hat{T} \coloneqq \sum_{k=1}^{n_\mathrm{e}} \hat{T}_k. \tag{2.5}$$

Finally, one introduces the CI ansatz as $|\psi_\mathrm{CI}\rangle \coloneqq \hat{T}\,|\mathbf{x}_\mathrm{HF}\rangle$ and seeks for such cluster operator amplitudes which minimise the variational energy:

$$\min_{t_i^a, t_{ij}^{ab}, \ldots} \frac{\left\langle \psi_\mathrm{CI} \middle| \hat{H} \middle| \psi_\mathrm{CI} \right\rangle}{\langle \psi_\mathrm{CI} | \psi_\mathrm{CI} \rangle}. \tag{2.6}$$

As defined, the cluster operator accounts for *all* possible excitations over the HF Slater determinant. Thus, it spans the whole Hilbert space and requires optimisation over factorially many cluster amplitudes. In other words, the full cluster operator does not restrict the Hilbert space *at all* and solving the minimisation problem (2.6) is equivalent to exact diagonalisation. Hence, in quantum chemistry the exact diagonalisation energies are more commonly referred to as *full configuration interaction (FCI)* energies.

To achieve polynomial scaling one truncates the cluster operator at a certain level of excitations. For example, in our numerical experiments we use the CISD method, i.e. configuration interaction *with singles and doubles*. This means that the truncated cluster operator is merely $\hat{T}_\mathrm{SD} \coloneqq \hat{T}_1 + \hat{T}_2$. Similarly to FCI, due to the generic nature of cluster amplitudes, CISD can be interpreted as diagonalising the Hamiltonian in the restricted Hilbert space including only the HF Slater determinant and all single and double excitations over it.

A simple estimate shows that for a maximum level of excitations N_CI there are $\binom{n_\mathrm{e}}{N_\mathrm{CI}}\binom{N-n_\mathrm{e}}{N_\mathrm{CI}} = \mathcal{O}(N^{N_\mathrm{CI}})$ Slater determinants in the CI space, which amounts to a vanishing fraction of the full Hilbert space as N grows and N_CI remains fixed.

Hence, despite its conceptual and numerical simplicity, CISD is not expected to be accurate for larger molecules. Thus, in our experiments we also use methods from the coupled cluster family, which are introduced below.

2.2.2 Coupled cluster methods

The coupled cluster family of methods aims to account for contributions of all excited determinants by considering the following ansatz:

$$|\psi_{\mathrm{CC}}\rangle \coloneqq e^{\hat{T}} |\mathbf{x}_{\mathrm{HF}}\rangle . \tag{2.7}$$

Here, similarly to CI methods, the cluster operator is usually truncated to a certain level of excitations. However, since the truncated $\hat{T}$ is exponentiated and not applied linearly, the coupled cluster state still includes *all* possible Slater determinants, which can be seen after Taylor expanding $e^{\hat{T}}$. This results in the following caveat: the amplitudes of different Slater determinants are not completely independent, and are instead non-linear combinations of the truncated cluster operator amplitudes.

Another important subtlety of coupled cluster methods is that they *do not* try to minimise the variational energy $\frac{\langle\psi_{\mathrm{CC}}|\hat{H}|\psi_{\mathrm{CC}}\rangle}{\langle\psi_{\mathrm{CC}}|\psi_{\mathrm{CC}}\rangle}$. Instead, they aim to find such cluster amplitudes that satisfy the Schrödinger equation projected to a subspace spanned by all excitations at the current level of the theory. To give an example, let us consider the *coupled cluster singles and doubles (CCSD)* method. The Schrödinger equation is $\hat{H} \underbrace{e^{\hat{T}} |\mathbf{x}_{\mathrm{HF}}\rangle}_{|\psi_{\mathrm{CC}}\rangle} = E_{\mathrm{CCSD}} \underbrace{e^{\hat{T}} |\mathbf{x}_{\mathrm{HF}}\rangle}_{|\psi_{\mathrm{CC}}\rangle}$. One usually multiplies it from the left by $e^{-\hat{T}}$ to simplify algebraic manipulations, and obtains the following equation:

$$e^{-\hat{T}} \hat{H} e^{\hat{T}} |\mathbf{x}_{\mathrm{HF}}\rangle = E_{\mathrm{CCSD}} e^{-\hat{T}} e^{\hat{T}} |\mathbf{x}_{\mathrm{HF}}\rangle = E_{\mathrm{CCSD}} |\mathbf{x}_{\mathrm{HF}}\rangle . \tag{2.8}$$

The space of all excitations at the current level of the theory includes $|\mathbf{x}_{\mathrm{HF}}\rangle$, all single excitations $\hat{c}_a^\dagger \hat{c}_i |\mathbf{x}_{\mathrm{HF}}\rangle$ and all double excitations $\hat{c}_a^\dagger \hat{c}_b^\dagger \hat{c}_i \hat{c}_j |\mathbf{x}_{\mathrm{HF}}\rangle$. Hence, after projecting the equation (2.8) to this space one obtains the following system of equations:

$$\begin{aligned}
\langle\mathbf{x}_{\mathrm{HF}}| \quad &\rightarrow \quad \langle\mathbf{x}_{\mathrm{HF}}|e^{-\hat{T}} \hat{H} e^{\hat{T}}|\mathbf{x}_{\mathrm{HF}}\rangle = E_{\mathrm{CCSD}} \langle\mathbf{x}_{\mathrm{HF}}|\mathbf{x}_{\mathrm{HF}}\rangle = E_{\mathrm{CCSD}} \\
\langle\mathbf{x}_{\mathrm{HF}}| \hat{c}_i^\dagger \hat{c}_a \quad &\rightarrow \quad \langle\mathbf{x}_{\mathrm{HF}}|\hat{c}_i^\dagger \hat{c}_a e^{-\hat{T}} \hat{H} e^{\hat{T}}|\mathbf{x}_{\mathrm{HF}}\rangle = E_{\mathrm{CCSD}} \langle\mathbf{x}_{\mathrm{HF}}|\hat{c}_i^\dagger \hat{c}_a|\mathbf{x}_{\mathrm{HF}}\rangle = 0 \\
\langle\mathbf{x}_{\mathrm{HF}}| \hat{c}_j^\dagger \hat{c}_i^\dagger \hat{c}_a \hat{c}_b \quad &\rightarrow \quad \langle\mathbf{x}_{\mathrm{HF}}|\hat{c}_j^\dagger \hat{c}_i^\dagger \hat{c}_a \hat{c}_b e^{-\hat{T}} \hat{H} e^{\hat{T}}|\mathbf{x}_{\mathrm{HF}}\rangle = E_{\mathrm{CCSD}} \langle\mathbf{x}_{\mathrm{HF}}|\hat{c}_j^\dagger \hat{c}_i^\dagger \hat{c}_a \hat{c}_b|\mathbf{x}_{\mathrm{HF}}\rangle = 0
\end{aligned} \tag{2.9}$$

The unknowns are the cluster amplitudes and the CCSD energy, and there are equally as many equations as there are unknowns. It is outside of the scope of this thesis to discuss how to solve these equations, and the relevant information can be found in the classic quantum chemistry textbooks such as, for example, Szabo and Ostlund [74]. We brought them up for a different purpose, namely to discuss the non-variational nature of CC methods.

It is known that E_{CCSD} obtained after solving the projected equations (2.9) might be lower than E_{FCI}. This raises the question of how adequate is a method which might produce "unphysical" answers. There are several reasons why this might not be as pressing of an issue as it seems. First, the system of coupled cluster equations (2.9) bears a strong flavour of perturbation theory, even though strictly speaking it is not an instance thereof. The latter is also known to produce unphysical results, e.g. only approximately normalised states, and yet it is rarely a concern. Second, in all practical calculations the only important quantities are energy differences. The coupled cluster methods successfully reproduce them, and thus are kept by the quantum chemistry community. Moreover, a specific guise of CCSD, known as CCSD(T), is considered a *de facto* golden standard among the quantum chemistry methods. The main idea of this method is to evaluate the amplitudes of *triply* excited Slater determinants from the CCSD wave function using the perturbation theory, and subsequently use them to calculate a perturbative correction to CCSD energy.

Finally, let us note that both CI and CC methods perform well mostly only within the domain of their applicability, i.e. for weakly correlated molecules where the Hartree-Fock solution serves as a good qualitative approximation to the ground state. Outside of this domain, post-HF methods fail to adequately describe the ground state, which necessitates the development of alternative approaches. Since NQS were conceived to tackle strongly correlated systems and they do not stand on perturbative footing (i.e. they are not built around the notion of a reference state and corrections to it), they were recognised as a promising tool for quantum chemistry calculations, and in the next section we overview their existing applications.

2.3 Existing NQS applications

2.3.1 Choo *et al.* [44]: Jordan-Wigner transformation and RBM

In Ref. [44] Choo *et al.* pioneered quantum chemistry calculations with NQS: they used complex-valued RBMs to represent wave functions of several small molecules. To map the second quantised fermionic Hamiltonian onto qubits they employed the Jordan-Wigner transformation — this approach became customary in the majority of subsequent works (see Box 2.1 for a brief overview of the Jordan-Wigner transformation). This produces a Hamiltonian of the following form:

$$\hat{H}_{\mathrm{JW}} = \sum_{l=0}^{N_{\mathrm{T}}-1} h_l \bigotimes_{i=0}^{N-1} \hat{P}_i^{(l)}, \tag{2.10}$$

where N_{T} is the number of terms and $\hat{P}_i^{(l)} \in \left\{\hat{I}_i, \hat{X}_i, \hat{Y}_i, \hat{Z}_i\right\}$ are Pauli operators acting on the i-th qubit.

Box 2.1: Jordan-Wigner transformation

The Jordan-Wigner transformation is perhaps the most common fermion-to-qubit mapping among many known in the literature [75]. It prescribes to replace the fermionic creation and annihilation operators with the following combinations of Pauli operators $\hat{X}$, $\hat{Y}$, $\hat{Z}$, ensuring that the fermionic anticommutation relations are satisfied:

$$\hat{c}_i = \frac{(\hat{\sigma}_i^X - i\hat{\sigma}_i^Y)}{2} \cdot \prod_{j<i} \hat{\sigma}_j^Z;$$
$$\hat{c}_i^\dagger = \frac{(\hat{\sigma}_i^X + i\hat{\sigma}_i^Y)}{2} \cdot \prod_{j<i} \hat{\sigma}_j^Z.$$

In this case, the computational basis vectors preserve the occupation number interpretation in that $|\mathbf{x}\rangle$ corresponds to a Slater determinant with $x_i = 1$ indicating the occupied orbitals.[a]

[a] In general, fermion-to-qubit mappings do not keep the correspondence between the bases of fermionic and qubit Hilbert spaces.

The authors reported energies within chemical accuracy to E_{FCI} and surpassing those of the CISD, CCSD, and CCSD(T) methods. However, the largest considered molecules corresponded to Hilbert spaces with at most $N = 20$ qubits and $\sim 10^5$

Slater determinants, still within the reach of exact diagonalisation. This highlighted the first challenge of NQS quantum chemistry calculations: many molecular ground state wave functions have a highly peaked structure, with a dominant amount of probability corresponding to the HF determinant. As a result, after a few initial optimisation iterations the majority of samples in the batch correspond to the HF determinant. Thus, large batch sizes are required to sample a non-trivial part of the Hilbert space and obtain accurate VMC estimates of the energy and its gradient. Subsequent works explored alternative sampling approaches addressing this issue.

2.3.2 Autoregressive approaches

Barrett *et al.* [45]: autoregressive neural quantum states

Ref. [45] offered a potent resolution for the sampling problem: it showed that autoregressive neural quantum states are well-suited for sampling from peaked probability distributions. First, ANQS produce *uncorrelated* and *unbiased* samples from the ansatz probability distribution, taking only one ansatz evaluation to produce one sample (see Chapter 3). This is in stark contrast with MCMC sampling, which requires some time to equilibrate and, as described in Chapter 1, requires multiple ansatz evaluations to thin the Markov chain and produce uncorrelated samples.

Second, Ref. [45] also developed an algorithm which samples the *statistics* corresponding to a batch of N_s samples without producing each sample individually. The complexity of this algorithm is dominated by evaluating amplitudes of only N_unq unique basis vectors, where $N_\mathrm{unq} \coloneqq |\mathcal{U}|$. This sampling algorithm opened the avenue to batch sizes N_s as big as 10^{12} (with $N_\mathrm{unq} \sim 10^3$–10^4) and became a *de facto* standard in the subsequent works on quantum chemistry calculations based on stochastic optimisation of an NQS.[2]

The improved sampling enabled optimising molecules up to Li_2O, which requires 30 qubits and is at the brink of exact diagonalisation. The best achieved energy for Li_2O came close to chemical accuracy, while the energies for smaller molecules were better than those achieved with CISD, CCSD and CCSD(T) methods.

[2]There also exist works where an NQS is optimised without sampling, which we shall overview shortly.

The further progress was hindered by the second scaling barrier typical for VMC quantum chemistry — namely, that of local energy calculation. This bottleneck stems from the fact that molecular Hamiltonians have large numbers of terms $N_{\mathrm{T}} = \Theta(N^4)$. As a result, to calculate the local energy for a given $\mathbf{x}$ according to Eq. (1.3) one has to evaluate the amplitudes of all basis vectors coupled to $\mathbf{x}$ via the Hamiltonian (i.e. of all such $\mathbf{x}'$ that $H_{\mathbf{x}\mathbf{x}'} \coloneqq \langle \mathbf{x}|\hat{H}|\mathbf{x}'\rangle \neq 0$). Hence, the calculation of local energies for all sampled basis vectors requires $\Theta(N_{\mathrm{unq}} N_{\mathrm{T}})$ additional ansatz evaluations, which quickly becomes unwieldy as the system size grows. For example, the authors reported the total running time of 1.9 days for the Li_2O optimisation, while the conventional quantum chemistry methods require at most several minutes to run. The subsequent works aimed to alleviate this issue.

Zhao *et al.* [46]: faster amplitude evaluation and parallelisation

In Ref. [46] Zhao *et al.* offered two possible ways to improve the computational efficiency of ANQS quantum chemistry. First, they considered an ansatz architecture different to that employed in Ref. [45], which allows cheaper evaluation of ansatz amplitudes.[3] Second, they parallelised the local energy evaluation over several GPUs. As a result, they managed to run quantum chemistry calculations for Na_2CO_3 molecule, requiring 76 qubits, and narrowly outperformed the CCSD method. In addition, on a set of small molecules their ansatz required less time to achieve the CCSD level of accuracy as compared to the ansatz presented in Ref. [45]. However, it remained unclear how demanding the calculations for larger molecules were, i.e. how much compute, both in terms of nodes and wall time, was required. Thus, it is difficult to assess how well this approach scales to even larger molecules.

Wu *et al.* [47]: simplified local energy calculations

In Ref. [47] Wu *et al.* adapted the Transformer architecture for an ANQS. They also introduced a computationally cheaper procedure to calculate an approximation to the local energy. This procedure assigns zero amplitudes to unsampled basis vectors, and thus avoids an excessive amount of ansatz evaluations; we discuss this

[3] We overview both architectures considered in Ref. [45] and Ref. [46] in Chapter 3.

strategy in more details in Chapter 6. The authors achieved chemical accuracy for H_2 molecule in cc-pVTZ and aug-cc-pVTZ basis sets requiring 56 and 92 qubits correspondingly. However, due to the simplicity of the molecule and multiple present symmetries the corresponding Hilbert spaces contained $\sim 10^5$ Slater determinants, and thus were still amenable to FCI. The authors also reported timing benchmarks for molecules with up to 120 qubits, however, no energy optimisation was performed for these molecules. In addition, it remained unclear to what extent the Transformer architecture was responsible for the improved accuracy of the method as opposed to, for example, extended duration of optimisation.

As a result, while ANQS hold a promise to become a powerful ansatz for quantum chemistry calculations, at its current state the field lacks a compelling evidence suggesting that NQS can scale up to challenging system sizes — i.e. large qubit counts *and* large Hilbert space sizes — and achieve state-of-the-art energies there. In this thesis we aimed to bridge precisely this gap.

2.3.3 Deterministic approaches

Let us also mention works which deal with the sampling bottleneck in a different way, namely, by giving sampling up altogether.

Li *et al.* [76]. Deterministic optimisation of an RBM

In Ref. [76] Li *et al.* employ an RBM to represent a molecular wave function and optimise it deterministically. They start with a predetermined set of unique samples $\mathcal{U}$ (for example, consisting of only $|\mathbf{x}_{\text{HF}}\rangle$) and update it at each iteration. First, they supplement $\mathcal{U}$ with Slater determinants which are single or double excitations of the determinants already contained in $\mathcal{U}$. Second, they evaluate the normalised probabilities $p(\mathbf{x}) = \frac{|\psi(\mathbf{x})|^2}{\sum_{\mathbf{x}\in\mathcal{U}}|\psi(\mathbf{x})|^2}$ for the considered determinants. Finally, they keep only those determinants whose probabilities are larger than a user specified threshold. The energy and gradient calculations are performed as if the core space determinants were truly sampled, with the renormalised probabilities used instead of

the ocurrence numbers. By choosing the threshold value one balances the size of the considered Hilbert space fraction against the computational efficiency of the method.

The authors reported orders of magnitude speedup as compared to the traditional MCMC optimisation, yet the largest considered molecule was still Li_2O, where they achieved chemical accuracy with respect to the FCI method.

Li *et al.* [77]. Simplified local energy calculations

In Ref. [77] the same group of authors significantly improved the system sizes available to deterministic optimisation of Ref. [76]. Similarly to Ref. [47], they used a simplified expression for local energy calculations, excluding contributions from unsampled determinants. They also simplified stochastic reconfiguration by dividing the geometric tensor into blocks smaller blocks and inverting them independently.

The authors reported dissociation curves for H_2O and N_2 molecules using the cc-pVDZ basis set, achieving energies better than CCSD for some distances, though not reaching chemical accuracy compared to the FCI method. For the Cr_2 molecule at a bond length of 1.5Å using the Ahlrichs SV basis set, they surpassed CCSD(T) energy (no FCI energy is available for this molecule). However, to surpass CCSD(T), the authors calculated a perturbative second-order correction to the variational energy evaluated from an RBM; the variational energy itself was of lower quality than CCSD(T).

Liu and Clark [50]. Neural network backflow ansatz

Finally, Ref. [50] considered another NQS architecture — neural network backflow (NNBF, see Box 2.2) — within the context of deterministic optimisation. They reported variational energies coming as close to E_{FCI} as 0.1 mHa for the Li_2O molecule, which is the best result reported in the literature. The authors attributed this success to the NNBF architecture which is believed to better suited for the study of fermionic systems. They also obtained energies better than those of the CCSD method for the C_2H_4O molecule requiring $N = 38$ qubits. Yet, scaling to larger system sizes was left for the future work.

Box 2.2: Neural network backflow

Neural network backflow quantum states consist of two components. The first component is an $n_\mathrm{e} \times N$ complex-valued matrix F of "single electron orbitals". It is said that these orbitals encode the "mean-field" wave function amplitudes $\psi_\mathrm{MF}(\mathbf{x}) \coloneqq \det F[\mathbf{x}]$. Here $F[\mathbf{x}]$ is an $n_\mathrm{e} \times n_\mathrm{e}$ matrix obtained from F by keeping only n_e columns corresponding to $x_i = 1$. This is in direct analogy with constructing a mean field fermionic wave function out of single particle wave functions using Slater determinants.
The second component is a neural network which takes $\mathbf{x}$ as input and produces an $n_\mathrm{e} \times n_\mathrm{e}$ matrix $B(\mathbf{x})$ as output. This matrix is used as an $\mathbf{x}$-dependent correction to F, which results in the ansatz amplitudes given by $\psi_\mathrm{NNBF}(\mathbf{x}) = \det\left(F[\mathbf{x}] + B(\mathbf{x})\right)$. Such construction allows introducing rather generic correlations on top of the mean field picture, crucial for successfully approaching strongly correlated systems [78, 79]

To conclude, let us note that on the one hand deterministic optimisation of NQS is a promising approach to circumvent the sampling bottleneck of quantum chemistry approximation. On the other hand, its computational performance strongly depends on the number of determinants within the core space, which is controlled only indirectly by the acceptance threshold. To keep the core space size reasonable, one has to actively schedule this threshold, as well as select it according to some physical knowledge about the system. This calls for intricate control techniques and makes scaling to the bigger systems sizes challenging. On the contrary, in Chapter 6 we develop a procedure for ANQS quantum chemistry calculations which is devoid of both sampling and local energy bottlenecks and provides the user with a full control over the number of unique determinants in the computational space.

3

Autoregressive neural quantum states

Contents

In this chapter we explore autoregressive neural quantum states, which are the ansatz of choice for the remainder of this thesis. We begin by showing that if a so-called *autoregressive decomposition* is known for a probability distribution, then it is possible to perform fast and unbiased sampling from this distribution. We then discuss how to construct an autoregressive neural quantum state, i.e. an ansatz with a known autoregressive decomposition for its Born distribution. This enables ansatz sampling devoid of Markov chain Monte Carlo drawbacks.

Next, we overview two important modifications to the ANQS sampling procedure. First, we discuss an algorithm for autoregressive statistics sampling, which allows one to obtain the *statistics* corresponding to a batch of N_s samples *without* sampling

the batch itself. Instead, the ansatz is invoked to evaluate only N_{unq} wave function amplitudes, where $N_{\text{unq}} \coloneqq |\mathcal{U}|$. Yet, the number of obtained unique samples N_{unq} cannot be set in advance; it emerges during sampling. Thus, we also cover an algorithm for autoregressive *sampling without replacement*, which produces the desired number of unique samples from the ansatz Born distribution.

3.1 Autoregressive neural quantum states

3.1.1 Autoregressive decomposition

As discussed in Chapter 1, the Metropolis-Hastings algorithm produces unbiased samples from the target probability distribution only when the underlying Markov chain process has converged, which in practice results in long sampling times. In addition, it struggles to adequately sample multimodal distributions, and in many cases the success of sampling hinges on a lucky initialisation of the underlying Markov chain. Hence, in Ref. [73] Sharir et al. adopted the idea of *autoregressive sampling* developed in the deep learning community as an alternative to Metropolis-Hastings sampling, guaranteed to produce unbiased samples with less computational burden.

The main idea of autoregressive sampling is to replace the strenuous sampling of an N-dimensional probability distribution with the consecutive sampling of N one-dimensional probability distributions. This requires decomposing the target probability distribution $p(\mathbf{x})$ into a product of single-variable *conditional* distributions according to the chain rule:

$$p(\mathbf{x}) = \prod_{i=0}^{N-1} p_i(x_i|\mathbf{x}_{<i}), \tag{3.1}$$

where $\mathbf{x}_{<i} \coloneqq (x_0, x_2, \ldots, x_{i-1})$. In our case all variables are binary, and therefore each conditional probability distribution corresponds to a Bernoulli random variable.

If all conditional probability distributions $p_i(x_i|\mathbf{x}_{<i})$ are known, the sampling process forms a tree as depicted in Fig. 3.1, where each node represents a conditional probability distribution. One starts at the root with an empty vector $\mathbf{x}_{<0} = \varnothing$ and samples the value of the first variable x_0 according to the *unconditional* probability distribution $p_0(x_0)$. Based on the obtained value, one forms a partially sampled

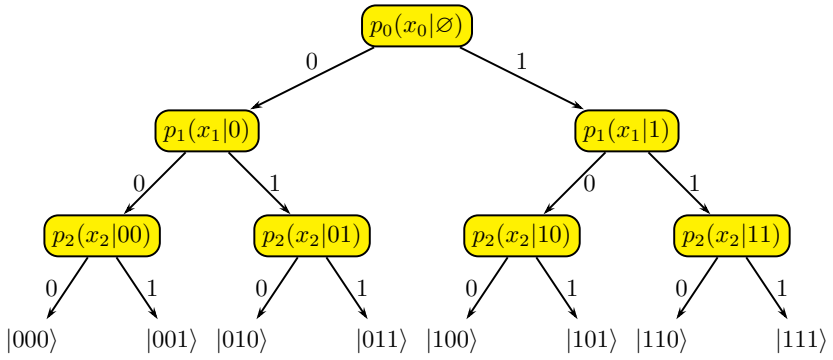


Figure 3.1: Autoregressive sampling tree.

bit vector $\mathbf{x}_{<1} = x_0$ and proceeds to sample the respective conditional probability distribution $p_1(x_1|\mathbf{x}_{<1})$. In essence, one chooses a second conditional distribution given the result of the first sampling, which diagrammatically corresponds to moving to one of the root node's child subtrees. Then, the whole process is recurrently repeated until the full bit vector $\mathbf{x} = x_0 x_1 \dots x_{N-1}$ is sampled. An unbiased sample from $p(\mathbf{x})$ is produced after each transversal of the sampling tree as soon as it is possible to produce unbiased samples from each single-variable probability distribution. This is in stark contrast to Metropolis-Hastings sampling which might require multiple evaluations of unnormalised probability necessary for Markov chain thinning.

3.1.2 Autoregressive models

To define a sampling tree one needs to specify $2^N - 1$ single bit conditional distributions.[1] Hence, at first glance autoregressive sampling appears to be a conceptually simpler yet unfeasible procedure, since it requires an exponential amount of memory to store the conditional distributions. A key to making it practical is to give up *storing* the conditional probabilities, and instead (i) *parameterise* them and (ii) calculate *on demand*. In other words, one has to define an autoregressive model which takes the vector $\mathbf{x}_{<i}$ as input and produces as output positive values of $p_{i,\theta}(x_i|\mathbf{x}_{<i})$. As long as each $p_{i,\theta}(x_i|\mathbf{x}_{<i})$ is a normalised probability distribution, i.e.

[1] Unsurprisingly, this equals the number of independent probabilities in a distribution over bit vectors of length N.

$\sum_{x_i} p_{i,\theta}(x_i|\mathbf{x}_{<i}) = 1 \; \forall i, \; \forall \mathbf{x}_{<i}$, the autoregressive sampling will produce unbiased samples from the overall probability distribution $p_\theta(\mathbf{x}) = \prod_{i=0}^{N-1} p_{i,\theta}(x_i|\mathbf{x}_{<i})$.

Consequently, to sample from a *target* probability distribution $p(\mathbf{x})$, one can seek for such set of parameters θ that $p_\theta(\mathbf{x})$ mimics $p(\mathbf{x})$ as closely as possible. Accurate approximations of this kind require that the chosen parameterisation has high expressive power. Therefore, a range of works explored using deep neural networks as autoregressive models. It turns out that a vast number of interesting distributions can be represented to the desired accuracy in such a way [80–82].

3.1.3 Autoregressive wave functions

While so far the discussion concerned probability distributions, Sharir *et al.* showed that the same line of reasoning can be applied to wave functions to comprise autoregressive neural *quantum states* [73]. In this case, instead of conditional probability distributions, one considers normalised conditional wave functions, i.e. $\psi_i(x_i|\mathbf{x}_{<i})$ such that $\sum_{x_i} |\psi_i(x_i|\mathbf{x}_{<i})|^2 = 1$. As a result, the total amplitudes $\psi(\mathbf{x})$ calculated as $\prod_{i=0}^{N-1} \psi_i(x_i|\mathbf{x}_{<i})$ are normalised by construction, i.e. $\sum_{\mathbf{x}} \left|\prod_{i=0}^{N-1} \psi_i(x_i|\mathbf{x}_{<i})\right|^2 = 1$. Fig. 3.2A illustrates how an ANQS can be used to calculate the amplitude $\psi(\mathbf{x})$ of a basis vector $\mathbf{x}$, while Fig. 3.2B depicts how an ANQS can be used to sample from the Born distribution $p(\mathbf{x})$ given by $|\psi(\mathbf{x})|^2$.

The conditional wave functions of an ANQS are represented in Fig. 3.2 with separate subnetworks having different numbers of inputs. This corresponds to a so-called *neural autoregressive density estimator (NADE)* architecture, where to evaluate a full wave function amplitude one needs to explicitly iterate over N conditional subnetworks. There exist other architectures for autoregressive models, e.g. *masked autoregressive density estimator (MADE)*. In this architecture all conditional probabilities are obtained with a *single* MLP with N outputs. It takes $\mathbf{x}$ as input, and produces as output all conditional probabilities $p(x_i|\mathbf{x}_{<i})$ in parallel, which substantially speeds up the amplitude evaluation. The autoregressive property is ensured via *masking* the weights of the fully connected layers inside the MLP. This means that certain weights are set to zero so that $\psi_i(x_i|\mathbf{x}_{<i})$ does

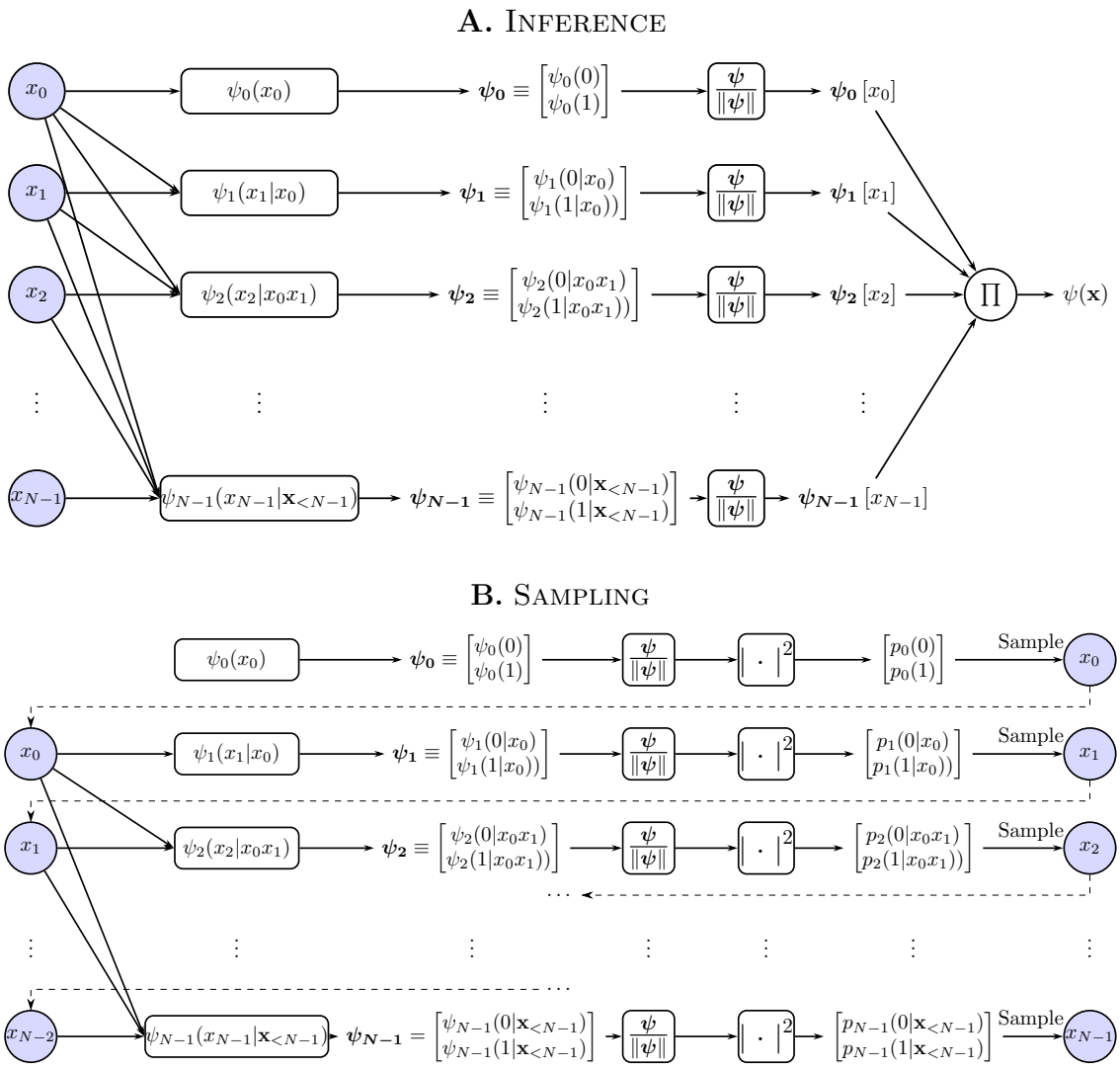


Figure 3.2: An autoregressive neural quantum state.

not depend on $\mathbf{x}_{\geq i}$ [46]. However, in our preliminary studies we did not observe a significant benefit in using the MADE architecture, and thus all ansatzes employed in this thesis followed the NADE architecture.

3.2 Autoregressive statistics sampling

As described, the autoregressive sampling is already capable to provide a substantial speed up over the Metropolis-Hastings sampling. However, in the context of VMC calculations it is possible to make ANQS even more performant. Specifically, Ref. [45] proposed an algorithm for autoregressive *statistics* sampling.

Suppose one wants to obtains statistics for a batch of N_s samples. A naïve approach is to traverse the sampling tree N_s times. In this case, the random Bernoulli variable corresponding to the tree root is repeatedly sampled N_s times. However, this provides redundant information on how the outcomes are distributed in the resulting sequence of samples. Instead, the only important information is how many times one has to proceed to the left and to the right child trees, which we denote as n_L and $n_\mathrm{R} = N_\mathrm{s} - n_\mathrm{L}$ correspondingly.

Another way to obtain the values of n_L and n_R is to sample a random *binomial* variable $B\left(N_\mathrm{s}, p_0(x_0|\varnothing)\right)$ corresponding to N_s trials of the underlying Bernoulli distribution $p_0(x_0|\varnothing)$. In practice this can be done substantially faster than sampling N_s individual Bernoulli random variables. One can apply the same trick to the second level of the sampling tree, and sample two random binomial variables $B\left(n_\mathrm{L}, p_1(x_1|0)\right)$ and $B\left(n_\mathrm{R}, p_1(x_1|1)\right)$ to figure out how many times each conditional distribution at the third level of the sampling tree should be sampled. This process is repeated recursively until one reaches the N-th level of the sampling tree. If for some node the sampled occurrence number is zero, it is discarded from further sampling, thus preventing exponential complexity growth.

We represent this process pictorially in Fig. 3.3 as a modification of the sampling tree, where along each edge "flows" the number of times each partial vector $\mathbf{x}_{<i}$ was sampled, which in turn tells how many times the conditional probability distribution $p_i(x_i|\mathbf{x}_{<i})$ should be sampled.

As a result, the neural network is evaluated on only N_unq unique configurations, as opposed to the direct sampling where the neural network is invoked N_s times. This proved to be beneficial for the electronic structure calculations with highly peaked ground state wave functions. In particular, Ref. [45] reported emulating sampling statistics for N_s as big as 10^{12}, which is many orders of magnitude larger than batch sizes typical for standard NQS calculations.

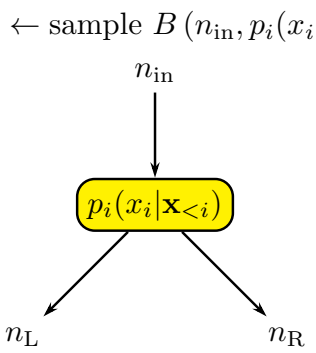


Figure 3.3: Information flow during the autoregressive statistics sampling. As described in the main text, the node receives a partially sampled vector $\mathbf{x}_{<i}$ and its corresponding number of occurrences n_in. One samples the corresponding binomial variable and sends updated partial bit strings $\mathbf{x}_{<i}0$ and $\mathbf{x}_{<i}1$ together with their occurrence numbers n_L and n_R down the sampling tree.

3.3 Autoregressive sampling without replacement

Both Metropolis-Hastings and conventional autoregressive sampling are examples of so-called sampling *with* replacement. Such sampling is *memoryless*, i.e. if $\mathbf{x}$ was obtained as the k-th sample ($k < N_\mathrm{s}$), it can still be produced by the ansatz as *the*k'-th sample, where $k' > k$. As a result, the number of obtained unique samples N_unq cannot be set in advance; it emerges during sampling. One controls N_unq only indirectly by changing the batch size N_s, and the same value of N_s can produce vastly different N_unq depending on the probability distribution encoded by the ansatz. The autoregressive statistics sampling of Ref. [45], while being more efficient, also emulates sampling with replacement and hence suffers from the same shortcoming.

It is however desirable to have direct control over N_unq. First, N_unq defines the computational complexity of both autoregressive statistics sampling and local energy calculations, while N_s does not. Second, N_unq serves as a direct measure for the Hilbert space portion explored by an ANQS.

In this section we introduce an approach addressing this issue, which is to sample from $p(\mathbf{x})$ *without* replacement. Such sampling is *memoryful*, i.e. if $\mathbf{x}$ was obtained as the k-th sample, then (i) it can never be obtained in any of subsequent samplings; (ii) the remaining samples are produced from a renormalised probability distribution in which the samples that have already occurred are assigned zero

probability. As a result, one obtains a batch of *precisely* N_{unq} unique samples. This sampling procedure, based on the algorithm by Kool *et al.* [83], invokes the ancestral Gumbel top-K trick [84, 85] which is explained shortly.[2]

3.3.1 Sequential sampling without replacement

A conventional way to obtain K samples without replacement from $p(\mathbf{x})$ is to produce samples one-by-one and adaptively change the sampled distribution in the following way. Suppose one has sampled without replacement $k-1$ unique samples constituting the set $\mathcal{U}^{(k-1)}$. Then, one obtains the k-th unique sample by sampling from the following probability distribution:

$$p^{(k)}(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in \mathcal{U}^{(k-1)}, \\ \frac{p(\mathbf{x})}{\sum_{\mathbf{x}' \notin \mathcal{U}^{(k-1)}} p(\mathbf{x})} \equiv \frac{p(\mathbf{x})}{1-\sum_{\mathbf{x}' \in \mathcal{U}^{(k-1)}} p(\mathbf{x})} & \text{otherwise.} \end{cases} \tag{3.2}$$

In other words, one manually removes the probability mass associated with already produced samples and then renormalises the remaining probabilities.

3.3.2 Gumbel top-K trick

The Gumbel top-K trick replaces sequential sampling from $p(\mathbf{x})$ with multiple samplings from a standard distribution, performed in a parallel fashion. It proceeds as follows [83]:

1. For each $\mathbf{x}$ one obtains a sample of *Gumbel noise* $G_{\mathbf{x}}$. All $G_{\mathbf{x}}$ are i.i.d. random variables with the CDF $F_{\text{G}}(y) = e^{-e^{-y}}$. This can be achieved by inverse transform sampling as $G = F_{\text{G}}^{-1}(U) = -\log(-\log U)$, where $U \sim \text{Uniform}(0,1)$.

2. One evaluates the *log*-probabilities corresponding to each possible outcome $l_{\mathbf{x}} \coloneqq \log p(\mathbf{x})$, which we further refer to as *unperturbed* log-probabilities.

3. One calculates the so-called *perturbed* log-probabilities by adding a Gumbel noise sample to each unperturbed log-probability: $g_{\mathbf{x}} \coloneqq l_{\mathbf{x}} + G_{\mathbf{x}}$.

[2] Emil Julius Gumbel (1891–1966) was not only a statistician, but also an outspoken pacifist and a staunch anti-fascist publicist, who was denaturalised by Nazi Germany, whose books got burned there, and who had to emigrate to the United States.

4. One sorts the perturbed log-probabilities in the descending order and takes first K $\mathbf{x}$'s corresponding to the highest perturbed log-probabilities as an output of the sampling without replacement algorithm.

To intuitively understand the Gumbel top-K trick, let us suppose we have already selected $k < K$ basis vectors and are now seeking to add the $(k+1)$-st. Let $\mathbf{x}$ and $\mathbf{x}'$ be two basis vectors that are not among the k already selected. Let us evaluate the probability of the event that $g_{\mathbf{x}} < g_{\mathbf{x}'}$, which equals to the probability that $\mathbf{x}'$ is selected in this step *conditioned* on the event that either $\mathbf{x}$ or $\mathbf{x}'$ is selected.

This probability is $\Pr(g_{\mathbf{x}} < g_{\mathbf{x}'}) = \Pr(l_{\mathbf{x}} + G_{\mathbf{x}} < l_{\mathbf{x}'} + G_{\mathbf{x}'}) = \Pr(G_{\mathbf{x}} - G_{\mathbf{x}'} < l_{\mathbf{x}'} - l_{\mathbf{x}})$. Thus, one seeks the probability that the difference $\Delta G \coloneqq G_{\mathbf{x}} - G_{\mathbf{x}'}$ of two independent Gumbel variables to be less than the given value $l_{\mathbf{x}'} - l_{\mathbf{x}}$. This probability is given by the value of the CDF $F_{\Delta G}(l_{\mathbf{x}'} - l_{\mathbf{x}})$. It can be calculated to be $F_{\Delta G}(l_{\mathbf{x}'} - l_{\mathbf{x}}) = \frac{1}{1+e^{-(l_{\mathbf{x}'} - l_{\mathbf{x}})}}$. Since $e^{-(l_{\mathbf{x}'} - l_{\mathbf{x}})} = \frac{p(\mathbf{x})}{p(\mathbf{x}')}$, we obtain that $\Pr(g_{\mathbf{x}} < g_{\mathbf{x}'}) = \frac{p(\mathbf{x}')}{p(\mathbf{x})+p(\mathbf{x}')}$. Thus, conditioned on the assumption that either $\mathbf{x}$ or $\mathbf{x}'$ is selected, $\mathbf{x}'$ is selected with the probability $\frac{p(\mathbf{x}')}{p(\mathbf{x})+p(\mathbf{x}')}$, whereas $\mathbf{x}$ is selected with the probability $\frac{p(\mathbf{x})}{p(\mathbf{x})+p(\mathbf{x}')}$, which is precisely what one expects from sampling without replacement as per Eq. (3.2).

3.3.3 Autoregressive Gumbel top-K trick

To explain the *autoregressive* Gumbel top-K sampling let us focus on $K = 1$ for simplicity. In this case the ordinary Gumbel top-K trick prescribes to evaluate perturbed log-probabilities *for all* $\mathbf{x}$ and find the single maximum among them. For N qubits this amounts to 2^N perturbed log-probabilities and is clearly unfeasible. Hence, the approach of Ref. [83] effectively resorts to a binary search to find the $\mathbf{x}$ corresponding to the maximum of the perturbed log-probabilities.

The authors show that given a vector $\mathbf{x}_{<i}$ the maximum of perturbed log-probabilities of its "children", i.e. $\max_{\mathbf{x}_{\geq i}} \left(\log p(\mathbf{x}_{<i}\mathbf{x}_{\geq i}) + G_{\mathbf{x}_{\geq i}}\right)$ is distributed as $\log p(\mathbf{x}_{<i}) + G_{\mathbf{x}_{<i}} =: g_{\mathbf{x}_{<i}}$. In other words, the *perturbed* log-probability of $\mathbf{x}_{<i}$ can serve as an upper bound for the perturbed log-probabilities of its children. Consequently, given $\mathbf{x}_{<i}$ one might estimate the perturbed log-probabilities of

$\mathbf{x}_{<i}0$ and $\mathbf{x}_{<i}1$ and keep only that partially sampled vector which has the larger perturbed log-probability. By starting this process with an empty vector $\mathbf{x}_{<0} \equiv \varnothing$ and repeating it iteratively, one obtains the full $\mathbf{x}$ corresponding to the highest perturbed log-probability without exponential growth of complexity.

There is, however, a subtlety. Since the perturbed log-probability of $\mathbf{x}_{<i}$ bounds the perturbed log-probabilities of $\mathbf{x}_{<i}0$ and $\mathbf{x}_{<i}1$ from above, the latter, being sampled at a *later* stage, should not exceed the value $g_{\mathbf{x}_{<i}}$ obtained at an *earlier* stage. In other words, sampling of $g_{\mathbf{x}_{<i}0}$ and $g_{\mathbf{x}_{<i}1}$ should be *conditioned* on the value of $g_{\mathbf{x}_{<i}}$. The authors of Ref. [83] show that the correct conditioned values denoted as $\tilde{g}_{\mathbf{x}_{<i}0}$ and $\tilde{g}_{\mathbf{x}_{<i}1}$ can be obtained by starting with the unconditioned values, finding their maximum $Z \coloneqq \max\{g_{\mathbf{x}_{<i}0}, g_{\mathbf{x}_{<i}1}\}$ and calculating the conditioned ones as $\tilde{g}_{\mathbf{x}_{<i}x_i} = -\log\left(\mathrm{e}^{-\tilde{g}_{\mathbf{x}_{<i}}} - \mathrm{e}^{-Z} + \mathrm{e}^{g_{\mathbf{x}_{<i}x_i}}\right)$. As a result, the autoregressive sampling without replacement is described with the following pseudocode:

Algorithm 1 Autoregressive Gumbel top-K sampling

function ARGumbelTopK($p_i(x_i|\mathbf{x}_{<i})$; K)
  `# Initialise the set of partially sampled vectors and their (perturbed) log-probabilities.`
  $\mathcal{UG} \coloneqq [(\varnothing, 0, 0)]$
  for i **from** 0 **to** $N-1$ **do**
    $\mathcal{UG}' \coloneqq [\,]$
    for $(\mathbf{x}_{<i}, l_{\mathbf{x}_{<i}}, \tilde{g}_{\mathbf{x}_{<i}})$ **in** $\mathcal{UG}$ **do**
      `# Evaluate the unconditioned log-probs`
      for x_i **in** $\{0, 1\}$ **do**
        $l_{\mathbf{x}_{<i}x_i} \coloneqq l_{\mathbf{x}_{<i}} + \log p_i(x_i|\mathbf{x}_{<i})$
        $g_{\mathbf{x}_{<i}x_i} \coloneqq l_{\mathbf{x}_{<i}x_i} + \text{Gumbel}(0)$
      $Z \coloneqq \max\{g_{\mathbf{x}_{<i}0}, g_{\mathbf{x}_{<i}1}\}$
      `# Evaluate the conditioned log-probs`
      for x_i **in** $\{0, 1\}$ **do**
        $\tilde{g}_{\mathbf{x}_{<i}x_i} \coloneqq -\log\left(\mathrm{e}^{-\tilde{g}_{\mathbf{x}_{<i}}} - \mathrm{e}^{-Z} + \mathrm{e}^{g_{\mathbf{x}_{<i}x_i}}\right)$
        $\mathcal{UG}'$`.append`$((\mathbf{x}_{<i}x_i, l_{\mathbf{x}_{<i}x_i}, \tilde{g}_{\mathbf{x}_{<i}x_i}))$
    Sort $\mathcal{UG}'$ in the descending order of $\tilde{g}_{\mathbf{x}_{<i+1}}$.
    $\mathcal{UG} \coloneqq \mathcal{UG}'[1:K]$.
  $\mathcal{U} \coloneqq$ every $\mathbf{x}$ in $\mathcal{UG}$.
  return $\mathcal{U}$

A generalisation to $K > 1$ is achieved by retaining top-K (perturbed) log-

probabilities during sampling of each qubit, see Ref. [83] for more details. The complexity of Algorithm 1 is $\mathcal{O}\left(N_{\text{unq}}\mathcal{C}_{\text{NQS}}\right)$, where $\mathcal{C}_{\text{NQS}}$ is the complexity of one ansatz evaluation. Importantly, the ansatz evaluations can be parallelised over all $\mathbf{x}_{<i}$ for a given i using batched GPU operations, and the only required loop is over the qubits. As a result, the total complexity of producing N_{unq} samples is comparable to that of evaluating their amplitudes, which is a hallmark of autoregressive ansatzes.

Part II

Symmetries

4

Incorporating quantum number symmetries

Contents

A textbook theorem of quantum physics states that if a Hamiltonian $\hat{H}$ commutes with a symmetry operator $\hat{S}$, then they are simultaneously diagonalisible. For different types of symmetries this has different implications. For example, if the Hamiltonian commutes with the operator of discrete translations, then, according to Bloch's theorem, its eigenstates have a specific functional form in the positional representation and are known as Bloch states. Another example are symmetries diagonal in the computational basis. In this case each basis vector carries a quantum number associated with it. Therefore, the Hilbert space becomes split into symmetry sectors labelled by quantum numbers and each Hamiltonian eigenstate belongs to one of those.

It is of benefit to incorporate symmetries into variational optimisation: for example, in the case of translational symmetry one might enforce the correct functional form on the ansatz wave function, which usually alleviates training. At the same time, in the case of quantum number symmetries, one might discard basis vectors belonging to the wrong symmetry sectors instead of making an ansatz "learn" that they should have zero amplitudes.

In this part we explore how multiple quantum number symmetries can be incorporated into ANQS optimisation. First, we formally introduce quantum number symmetries and the problem of symmetry-aware sampling, i.e. such sampling which does not produce the basis vectors outside of the correct symmetry sector. The existing works which considered quantum number symmetries in the context of ANQS optimisation either relied on *ad hoc* solutions to address a small number of specific symmetries [29, 45, 46], or considered quantum number symmetries of less generic nature [86]. Second, we present an algorithm which enables symmetry-aware ANQS sampling for an arbitrary number of quantum number symmetries. Finally, in Chapter 5 we showcase the advantage of such algorithm in application to quantum chemistry calculations, typically characterised by multiple quantum number symmetries.

4.1 Quantum number symmetries

Suppose $\{\hat{S}_m\}_{m=1}^{N_{\text{sym}}}$ is a set of N_{sym} operators commuting with the Hamiltonian such that each operator is also diagonal in the computational basis, i.e., $\forall m\ \hat{S}_m = \sum_{\mathbf{x}} s_m(\mathbf{x}) \left|\mathbf{x}\right\rangle \left\langle\mathbf{x}\right|$, where $s_m(\mathbf{x}) \in \mathbb{C}$ is the corresponding eigenvalue of the basis vector $\left|\mathbf{x}\right\rangle$. We refer to such set of operators as to the set of Hamiltonian quantum number symmetries, and provide examples of the most common quantum number symmetries in Section 4.1.1. Since operators $\hat{S}_m$ are diagonal in the computational basis, they also commute pairwise, and therefore any Hamiltonian eigenstate will be an eigenstate of *all* symmetry operators. In this case, the associated set of eigenvalues $\mathbf{s} = \left[s_1, s_2, \ldots, s_{N_{\text{sym}}}\right]$ specifying the symmetry sector of the Hamiltonian. An important consequence is that the Hamiltonian ground state $\left|\psi_{\text{GS}}\right\rangle$ has the

set $\mathbf{s}_{\text{ref}}$ of eigenvalues associated with it, and only computational basis vectors corresponding to the same symmetry sector can contribute to it, i.e., if $\mathbf{s}(\mathbf{x}) \coloneqq \left[s_1(\mathbf{x}), s_2(\mathbf{x}), \ldots, s_{N_{\text{sym}}}(\mathbf{x})\right] \neq \mathbf{s}_{\text{ref}}$, then $\langle \mathbf{x}|\psi_{\text{GS}}\rangle = 0$.

4.1.1 Common quantum number symmetries

1. *Total number of particles* [45, 46]. Suppose the computational basis vectors bear the meaning of occupation strings, i.e. $x_i = 0, 1, \ldots$ indicates the number of particles in the i-th subsystem. Then, the operator for the total number of particles has eigenvalues calculated as $s(\mathbf{x}) = \sum_{i=0}^{N-1} x_i$. In the context of quantum chemistry particles are electrons and subsystems are spin-orbitals.

2. *Total spin projection* [45, 46]. In quantum chemistry calculations spin-orbitals are usually ordered so that the odd orbitals have spin projection $s_z = \frac{1}{2}$ and the even ones have $s_z = -\frac{1}{2}$. The total spin projection of the ensemble of electrons represented by a computational basis vector is calculated as a sum of the spin projections of occupied orbitals. Correspondingly, the eigenvalues of this symmetry are calculated as $s(\mathbf{x}) = \frac{1}{2}\sum_{i=0}^{N-1}(-1)^{(i-1)}x_i$.

3. *Total magnetisation* [29]. A similar symmetry arises in the context of interacting spin systems, where each x_i corresponds to the spin projection of the i-th subsystem. For example, in the case of spin-$\frac{1}{2}$ systems $|0\rangle$ can denote $|\uparrow\rangle$ and $|1\rangle$ can correspond to $|\downarrow\rangle$. In this case, the total magnetisation is calculated as $s(\mathbf{x}) = \sum_{i=0}^{N-1}\left(\frac{1}{2} - x_i\right)$.

4. $\mathbb{Z}_2$ *symmetry* [87]. A $\mathbb{Z}_2$ symmetry is an operator $\hat{S}$ which commutes with $\hat{H}$ and is an N-fold product of either Pauli $\hat{I}$ or $\hat{Z}$ matrices. For example, if $\hat{H} = \hat{X}_1\hat{X}_2 + 0.5\hat{Z}_1\hat{Z}_2$, then $\hat{Z}_1\hat{Z}_2$ is its $\mathbb{Z}_2$ symmetry, while $\hat{I}_1\hat{Z}_2$ and $\hat{Z}_1\hat{I}_2$ are not. Such operators have only two unique eigenvalues ± 1 which are calculated as follows. Let $\mathcal{Z}$ be the set of indices specifying the positions of Pauli $\hat{Z}$ operators constituting a $\mathbb{Z}_2$ symmetry, i.e. $\hat{S} = \otimes_{i\in\mathcal{Z}}\hat{Z}_i$. For example, if $N = 4$ and $\hat{S} = \hat{Z}_1\hat{Z}_3$, then $\mathcal{Z} = \{1, 3\}$. In this case $s(\mathbf{x}) = \prod_{i\in\mathcal{Z}}(-1)^{x_i}$. In Chapter 5 we discuss how such symmetries arise in quantum chemistry calculations.

4.2 Symmetry-aware sampling

The VMC calculations can accommodate quantum number symmetries if one postselects only the samples in the correct symmetry sector and renormalises their occurrence numbers. However, this wastes computational power since samples outside the correct symmetry sector are produced too. During the initial iterations of optimisation, the fraction of relevant samples can be extremely poor, making this approach significantly inefficient. Hence, we are interested in designing a sampling procedure which automatically produces only basis vectors from the correct symmetry sector. We call such sampling procedures *symmetry-aware*.

4.2.1 Pruning of autoregressive sampling tree

MCMC sampling can be easily made symmetry-aware with regard to the quantum number symmetries: one can modify the proposal step of the Metropolis-Hastings algorithm so that the basis vectors outside of the correct symmetry sector are never proposed [44]. Unfortunately, the same line of reasoning does not apply to autoregressive sampling since the latter does not have a proposal step and any of 2^N basis vectors can be sampled.

One can make the autoregressive sampling symmetry-aware by pruning the sampling tree on-the-fly — that is, avoiding "unphysical" subtrees that have no leaves in the correct symmetry sector, as illustrated in Fig. 4.1. In other words, a partially sampled vector $\mathbf{x}_{<i}$ is discarded as soon as it becomes apparent that it has no physical continuations.

A naïve approach to determine physicality of a subtree $\mathbf{x}_{<i}$ is to check whether at least one if its leaves is physical, which is, however, exponentially costly. For some symmetries, there exist *ad hoc* ways to circumvent this hurdle: e.g. for the particle number symmetry with the target eigenvalue n_e, the subtree $\mathbf{x}_{<i}$ is unphysical if $\sum_{j=0}^{i} x_j > n_\mathrm{e}$. Such techniques are however not readily generalisable. An additional complication is that, even if a simple rule exists for each individual symmetry, testing them one-by-one might miss a node that is unphysical because its left and right subtrees are rendered unphysical by different symmetries (see Box 4.1

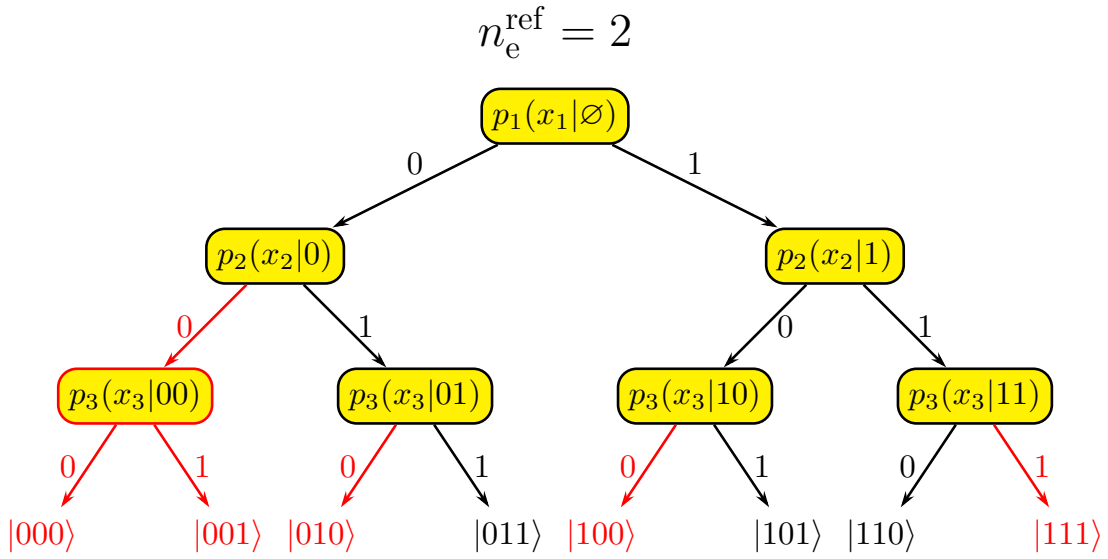


Figure 4.1: An example of an autoregressive sampling tree with labelled unphysical leaves and subtrees. The considered symmetry is particle conservation symmetry $\hat{n}_e$ with the target eigenvalue $n_e^{ref} = 2$. We depict in red the sampling tree leaves violating this symmetry, as well as the corresponding unphysical subtrees.

for an example). Hence it is desirable to find a physicality check algorithm that (a) has polynomial complexity with respect to N and (b) checks all symmetries in combination.

Box 4.1: A node can be unphysical under a set of symmetries even if it is physical under each individual symmetry

Let us consider a system of size $N = 3$ which has two quantum number symmetries: $\hat{S}_1 = \hat{Z}_0\hat{Z}_2$ and $\hat{S}_2 = \hat{Z}_1\hat{Z}_2$ with the reference eigenvalues $s_1^{ref} = 1$ and $s_2^{ref} = 1$ respectively. In Fig. 4.2A and Fig. 4.2B we paint in red all unphysical subtrees specified by $\hat{S}_1$ and $\hat{S}_2$ individually. For both symmetries the node $p_2(x_2|01)$ has at least one physical leaf, and thus the subtree 01 is physical. However, if one considers both symmetries *simultaneously*, then the node $p_2(x_2|01)$ turns out to be unphysical, as depicted in Fig. 4.2C. Consequently, it is not enough to have an individual physicality evaluation algorithm for each of the symmetries, and for every set of quantum number symmetries a composite algorithm should be devised.

4.2.2 Physicality evaluation algorithm

In this section we provide such an algorithm for an arbitrary number of symmetries as long as each of them satisfies the following two requirements:

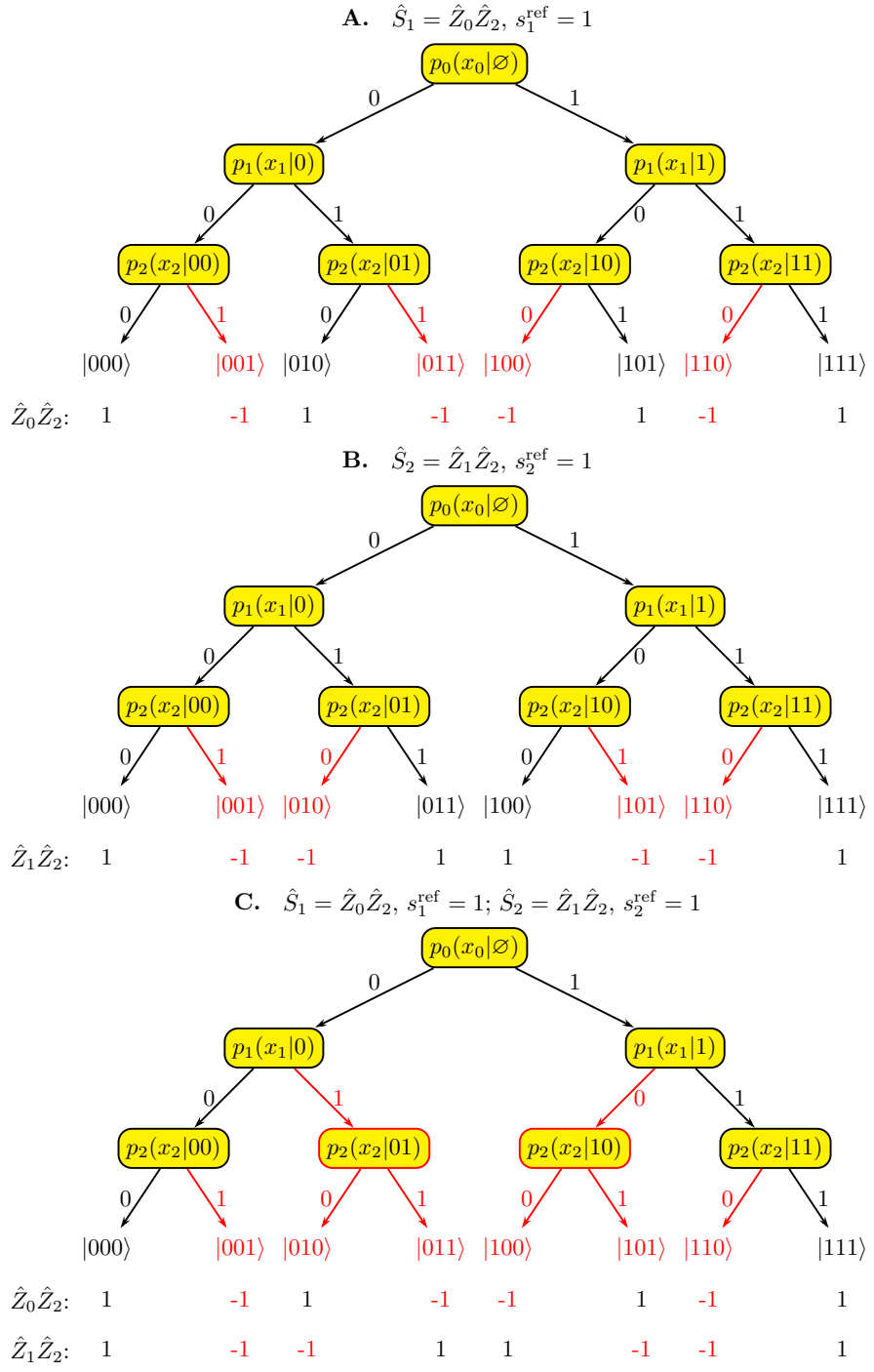


Figure 4.2: An illustration on how for a set of quantum number symmetries the resulting physicality evaluation rule is *not* just the union of the individual physicality evaluation rules. The nodes $p_2(x_2|01)$ and $p_2(x_2|10)$ are physical when each symmetry is considered separately. However, they become unphysical if both symmetries are considered jointly.

1. *Local decomposability.* The eigenvalues of $\hat{S}$ can be obtained as a composition of local eigenvalues calculated on individual qubits, i.e., $s(\mathbf{x}) = \odot_{i=0}^{N-1} s_i(x_i)$, where $\odot$ is some binary composition operation. For example, in the case of the particle number symmetry, $s(\mathbf{x}) = \sum_i x_i$, and therefore $\forall i \; s_i(x) \equiv x$ and $\odot$ is merely the addition operation. For symmetries with this property, we can define partial eigenvalues as $s_{<i} \coloneqq s(\mathbf{x}_{<i}) = \odot_{j=0}^{i} s_j(x_j)$.

2. *Polynomially-sized spectrum.* The number of unique eigenvalues of $\hat{S}$, either partial or not, is bounded from above by $\mathcal{O}(\text{poly}(N))$.

It is easy to see that the already discussed quantum number symmetries do satisfy both of these requirements. For the total number of particles, total magnetisation and total spin projection $\odot$ is just the binary addition and the corresponding $s_i(x_i)$ are given by the expression under the summation sign. For a system with N qubits each of these symmetries has at most $N+1$ different eigenvalues. In the case of s $\mathbb{Z}_2$ symmetry, $\odot$ is the binary multiplication and $s_i(x_i)$ can be defined as follows:

$$s_i(x_i) = \begin{cases} (-1)^{x_i} \text{ if } i \in \mathcal{Z}, \\ 1 \text{ otherwise.} \end{cases}$$

Regardless of N, this symmetry has only two different eigenvalues.

An important consequence of the local decomposability property is the following lemma:

Proposition 1. *Consider two sampling subtrees defined by the partial basis vectors* $\mathbf{x}_{<i}$ *and* $\mathbf{x}'_{<i}$*. If their vectors of partial eigenvalues are equal, i.e.,* $\mathbf{s}_{<i}(\mathbf{x}_{<i}) = \mathbf{s}_{<i}(\mathbf{x}'_{<i})$*, then the subtrees are either both physical or not.*

Proof. Suppose $\mathbf{x}_{<i}$ is physical. It means that $\exists \mathbf{x}_{\geq i} : \mathbf{s}(\mathbf{x}_{<i}\mathbf{x}_{\geq i}) = \mathbf{s}_{<i}(\mathbf{x}_{<i}) \odot \mathbf{s}_{\geq i}(\mathbf{x}_{\geq i}) = \mathbf{s}_{\text{GS}}$. In other words, the subtree $\mathbf{x}_{<i}$ has a physical leaf $\mathbf{x}_{<i}\mathbf{x}_{\geq i}$. But then the same continuation path $\mathbf{x}_{\geq i}$ will lead to a physical leaf of $\mathbf{x}'_{<i}$. Indeed, thanks to the local decomposability $\mathbf{s}(\mathbf{x}'_{<i}\mathbf{x}_{\geq i}) = \mathbf{s}_{<i}(\mathbf{x}'_{<i}) \odot \mathbf{s}_{\geq i}(\mathbf{x}_{\geq i}) = \mathbf{s}_{<i}(\mathbf{x}_{<i}) \odot \mathbf{s}_{\geq i}(\mathbf{x}_{\geq i}) = \mathbf{s}_{\text{GS}}$. Hence $\mathbf{x}'_{<i}$ is physical too. □

Algorithm 1 Physicality evaluation

Input:
i: the index of the current sampling tree level;
$\mathbf{s}_{<i}$: a vector of partial eigenvalues.

Output:
TRUE if $\mathbf{s}_{<i}$ is physical, FALSE otherwise.

```
1: function IsPhys(i; s_{<i})
2:     if (i, s_{<i}) is not in Lookup then
3:         if i = N then
4:             if s_{<i} = s_ref then
5:                 Lookup(i, s_{<i}) := True
6:             else
7:                 Lookup(i, s_{<i}) := False
8:         else
9:             Lookup(i, s_{<i}) := OR [ IsPhys(i + 1; s_{<i} ⊙ s_i(0))
                                       [ IsPhys(i + 1; s_{<i} ⊙ s_i(1)).
10:    return Lookup(i, s_{<i})
```

Therefore, for a given $\mathbf{x}_{<i}$ it is enough to enquire only about the physicality of the corresponding $\mathbf{s}_{<i}$. Let us define a function $\textsc{IsPhys}(i; \mathbf{s}_{<i} \coloneqq \mathbf{s}(\mathbf{x}_{<i}))$ which returns TRUE if $\mathbf{s}_{<i}$ is physical, and FALSE otherwise. This function can be calculated recursively using the following considerations. First, a subtree is physical as soon as at least one of its child subtrees is physical. Second, the partial eigenvalue vectors for the left and right child subtrees are $\mathbf{s}_{<i} \odot \mathbf{s}_i(0)$ and $\mathbf{s}_{<i} \odot \mathbf{s}_i(1)$ correspondingly. Hence, IsPhys satisfies the following recurrent relation:

$$\textsc{IsPhys}(i; \mathbf{s}_{<i}) = \mathrm{OR} \left[\begin{array}{l} \textsc{IsPhys}(i+1; \mathbf{s}_{<i} \odot \mathbf{s}_i(0)) \\ \textsc{IsPhys}\,(i+1; \mathbf{s}_{<i} \odot \mathbf{s}_i(1))\,. \end{array} \right. \tag{4.1}$$

The recursion should terminate at $i = N$ with $\textsc{IsPhys}(N, \mathbf{s}_{<N}) = \textsc{True}$ when $\mathbf{s}_{<N} = \mathbf{s}_{\mathrm{GS}}$ and FALSE otherwise. We provide the corresponding pseudocode in Algorithm 1.

It might seem that due to binary branching at each recursion level the runtime of the algorithm is exponential; however, one can resort to caching — that is, store each calculated value of $\textsc{IsPhys}(\cdot)$ in a lookup table. Since the spectra of the considered symmetries are polynomially-sized, at each level of the sampling tree

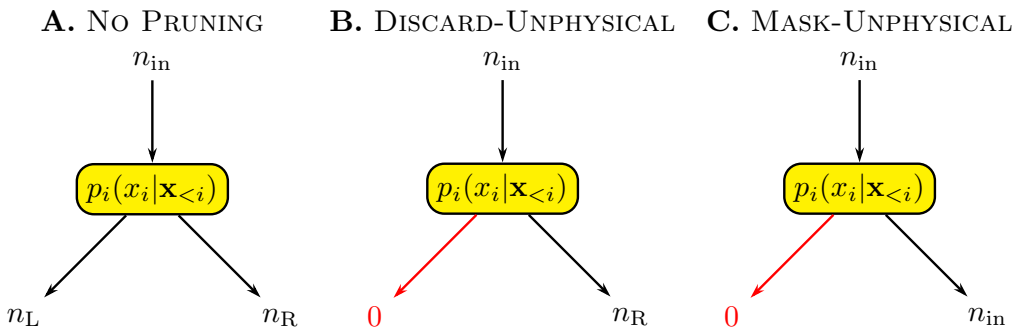


Figure 4.3: Information flow imposed by different pruning strategies in the case when the left child subtree of a node is unphysical. We omit partially sampled vectors flowing along the edges of the sampling tree and focus only on how their occurrence numbers are calculated. Note that a situation when $n_{\text{in}} \neq 0$ and *both* subtrees of the node are unphysical is impossible: in this case, the node is unphysical too, and therefore it should have been omitted earlier in the sampling.

we can have at most $\mathcal{O}\left((\text{poly}(N))^{N_{\text{sym}}}\right) = \mathcal{O}\left(\text{poly}(N)\right)$ different $\mathbf{s}_{<i}$, in contrast to the exponential number of subtrees. As a result, the domain of IsPhys$(\cdot)$ is polynomially-sized, and so is the runtime of the algorithm. Finally, let us note that the proposed algorithm allows targeting *any* symmetry sector of the Hamiltonian — one just has to substitute $\mathbf{s}_{\text{GS}}$ with the desired vector of eigenvalues $\mathbf{s}_{\text{ref}}$ in the termination condition for IsPhys.

4.2.3 Pruning strategies

Suppose n_{in} samples enter the node containing the local probability distribution $p_i(x_i|\mathbf{x}_{<i})$ during autoregressive statistics sampling. We obtain two numbers n_{L} and n_{R} (with $n_{\text{L}} + n_{\text{R}} = n_{\text{in}}$), which should be passed to the left and right child subtrees, respectively (4.3A). Suppose, however, that IsPhys shows the left child subtree to be unphysical. One can think of two strategies to handle this situation. The first approach would be to pass n_{R} to the right subtree as planned, and pass no samples to the left subtree. We refer to such strategy as Discard-Unphysical (DU, Fig. 4.3B). Its shortcoming is loss of samples: the occurrence numbers of final samples $n(\mathbf{x})$, $\mathbf{x} \in \mathcal{U}$ might not add up to N_{s}.

An alternative strategy was adopted in Ref. [45] and Ref. [29]; we refer to it as Mask-Unphysical (MU, Fig. 4.3C). This strategy prescribes passing all

n_{in} "virtual" samples to the right subtree so that no samples are ever lost at any level of the sampling tree. However, this biases the sampling and the empiric probabilities $\frac{n(\mathbf{x})}{N_{\text{s}}}$ do not correspond to the probability distribution $|\psi(\mathbf{x})|^2$ yielded by the ansatz. To restore the correspondence, one has to modify the conditional wave functions too: if a subtree $\mathbf{x}_i 0$ of the node $\mathbf{x}_{<i}$ is unphysical, the conditional wave function must be manually modified (or *masked*) so that $\psi_i^{\text{MU}}(0|\mathbf{x}_{<i}) = 0$ and $\psi_i^{\text{MU}}(1|\mathbf{x}_{<i}) = 1$ regardless of their initial values. These modified amplitudes are then used in all subsequent calculations.

Unfortunately, masking might bring unforeseen side effects affecting the ansatz expressibility. For example, for a particle number symmetry the value of x_{N-1} is completely defined by the sampled values of $\mathbf{x}_{<N-1}$, and therefore for *all* $\mathbf{x}_{<N-1}$ the conditional wave function $\psi_{N-1}(x_{N-1}|\mathbf{x}_{<N-1})$ will be masked. As a result, the subnetwork encoding $\psi_N(x_N|\mathbf{x}_{<N})$ is rendered useless. More generally masking reduces the number of independent conditional wave functions, thus limiting the variational freedom of the model. A possible remedy is to apply MU to the first $N - d$ levels of the tree, and then use DU for the last d levels (where d is a hyperparameter). We denote such family of strategies as MU-d. The purpose of MU-d strategies is to maintain a higher level of variational freedom at the cost of sample loss in a controllable way.

5 Numerical results

Contents

In this chapter we apply the Algorithm 1 developed in the previous chapter to ANQS quantum chemistry. First, we identify a class of symmetries that had previously not been considered in NQS quantum chemistry calculations — the $\mathbb{Z}_2$ symmetries encoding the spatial symmetries of a molecule. Second, we perform ablation studies to identify the most advantageous local pruning strategy. Finally, we run calculations employing electron number, spin projection and $\mathbb{Z}_2$ symmetries for a range of small to medium size molecules. We observe dramatically improved accuracy and computational efficiency of the variational optimisation. In particular, we reach chemical accuracy for all studied molecules and achieve this with an order of magnitude speedup compared to previous ANQS quantum chemistry calculations.

5.1 Molecular symmetries

We consider three types of quantum number symmetries inherent to molecules. First, molecules have a fixed number of electrons n_e. Second, molecules have a fixed value of their total spin projection. In all our experiments we study molecules with the total spin projection equal to 0, also known as *singlet* molecules. These two symmetries were already incorporated in previous research [45–47]. The third type are $\mathbb{Z}_2$ symmetries encoding the spatial symmetries of a molecule; they are considered for the first time in the context of NQS quantum chemistry calculations.

One usually accounts for the spatial symmetries of a molecule by considering irreducible representations (irreps) of its point group, which is a group of spatial transformations preserving the nuclear positions. It is known that the ground state of a molecule belongs to one of the irreps of the largest Abelian subgroup of the point group. At the same time, each Slater determinant belongs to one of the irreps too, and therefore only the SDs within the same irrep as the ground state contribute to the latter.

To include this selection rule into ANQS optimisation, we build on the observation made by Setia *et al.* [87]. It states that after the Jordan-Wigner transform every spatial symmetry in the largest Abelian subgroup of the point group would translate into a Hamiltonian $\mathbb{Z}_2$ symmetry. All $\mathbb{Z}_2$ symmetries of a Hamiltonian could be found in an automated way using the algorithm outlined by Bravyi *et al.* [88].

An important consequence is that a basis vector belongs to the ground state irrep only if for each $\mathbb{Z}_2$ symmetry it has the same eigenvalue as the ground state. However, at the start of calculations the ground state and its symmetry sector are unknown. Therefore, to fix the symmetry sector we assume that the Hartree-Fock Slater determinant will necessarily contribute to the ground state, which is a valid premise for a vast number of molecules. Consequently, for the purpose of running the IsPhys algorithm, we fix $\mathbf{s}_{\mathrm{GS}}$ to be equal to $\mathbf{s}(\mathbf{x}_{\mathrm{HF}})$.

5.2 Experiment particulars

5.2.1 ANQS architecture

The autoregressive ansatz used in our numerical experiments is comprised of N subnetworks following the NADE architecture described in Chapter 3. For the purpose of numerical stability, each subnetwork computes the logarithm of amplitudes $\log \psi_i(x_i|\mathbf{x}_{<i})$. Each subnetwork is a fully-connected multilayer perceptron of depth 3 and width 64 equipped with complex weights. Each MLP takes $i-1$ bits as input and produces two complex numbers $\log \psi_i(0|\mathbf{x}_{<i})$ and $\log \psi_i(1|\mathbf{x}_{<i})$ as output. The first MLP corresponds to the unconditional wave function $\psi_0(\cdot)$ and therefore has no input. In our preliminary experiments we found out that the following sequence of nonlinear activations allows achieving the best energies:

$$\begin{aligned}
&\text{Layer 1: } f_1(z) = \tanh z; \\
&\text{Layer 2: } f_2(z) = \mathrm{LeReLU}(\operatorname{Re} z) + \mathrm{i}\,\mathrm{LeReLU}(\operatorname{Im} z) \\
&\text{Layer 3: } f_3(\mathbf{z}) = \mathbf{z} - \frac{1}{2} \cdot \mathrm{LogSumExp}\left(2 \cdot \operatorname{Re} \mathbf{z}\right).
\end{aligned}$$

Here LeReLU is the real-valued leaky ReLU function:

$$\mathrm{LeReLU}(x) = \begin{cases} x, \text{ if } x \geq 0, \\ -0.01x \text{ otherwise.} \end{cases}$$

The first two activations are applied elementwise, while the third activation is applied across the whole two-dimensional output vector $\mathbf{z}$ to normalise the probability $\exp(\operatorname{Re} f_3(z_1))^2 + \exp(\operatorname{Re} f_3(z_2))^2$ to unity.

5.2.2 Batch size schedule

In our calculations we optimise ANQS using autoregressive statistics sampling. To reduce the computational burden of the first iterations, when many basis vectors might be sampled, we resort to the following batch size schedule (here t stands for the optimisation iteration index):

$$N_\mathrm{s} = \begin{cases} 10^5, \ 1 \leq t \leq 100; \\ 10^6, \ 101 \leq t \leq 200; \\ 10^7, \ 201 \leq t \leq 1000; \\ 10^8, \ t > 1000. \end{cases}$$

5.2.3 Computing system specifications

The calculations are mainly run on NVIDIA RTX A5000 GPU with 24GB of RAM. The only routine held on a CPU is vectorised sampling of individual binomial distributions. To that end we use 16 Intel(R) Core(TM) i9-11900K CPU cores working at the clock frequency of 3.50GHz.

5.3 Results

5.3.1 Ablation studies

In the ablation studies we test how our innovations affect the optimisation performance. In particular, we (i) demonstrate the effect of taking into account $\mathbb{Z}_2$ symmetries and (ii) compare the pruning strategies described in Sec. 4.2.3. We run electronic structure calculations for a set of small molecules with geometries taken from Ref. [44] in the basis set STO-3G [75]. The $\mathbb{Z}_2$ symmetries were calculated for each qubit Hamiltonian using the implementation of the Bravyi *et al.* algorithm [88] provided by the PennyLane software library [89, 90].

For each molecule we employ five randomly initialised instances of ANQS corresponding to different seeds of the underlying pseudorandom number generator. Each optimisation lasts for 3×10^4 iterations. After the gradient calculation, the ANQS parameters are updated with the Adam optimiser in the default configuration of hyperparameters.

The ablation results are presented in Fig. 5.1, where we measure the difference between the minimum energy achieved in the course of optimisation and the full configuration interaction energy E_{FCI}. We also plot reference energy difference values obtained with such conventional quantum chemistry methods as CISD, CCSD and CCSD(T).

We observe that taking $\mathbb{Z}_2$ symmetries into account universally results in better median variational energies. Particularly striking are the results for highly symmetric molecules such as N_2 and C_2: the accuracy improvement reaches two orders of magnitude for MU-2 and DU strategies and allows us to achieve the chemical

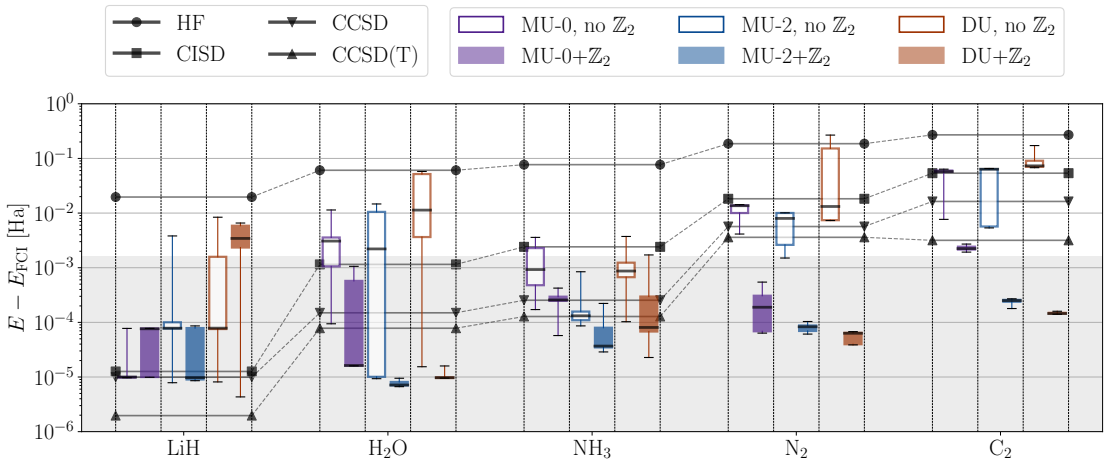


Figure 5.1: Ablation studies of various local pruning strategies and symmetry sets used for ANQS optimisation. Box plots represent five differently initialised ANQS. The black bold line on the box bodies corresponds to the median value and whiskers stretch from the minimum to maximum value in the distribution of results. The shadowed area spans energies below the chemical accuracy benchmark. For better visibility we plot the reference energies of different methods as continuous curves, even though they belong to different molecules and are not related.

accuracy of 1.6 mHa. As expected, the MU-2 strategy outperforms MU. On the other hand, one can see that the MU-2 strategy performs on par with DU. Hence, in the subsequent experiments we employed only MU-2 and DU pruning strategies.

5.3.2 General results

In the main set of experiments we test our ANQS on a large set of molecules. The reference information for the molecules studied, including the number of $\mathbb{Z}_2$ symmetries and the corresponding computational space sizes is presented in Table 5.1. It can be seen that molecules with two $\mathbb{Z}_2$ symmetries have computational space sizes equivalent to those when only electron number and total spin projection symmetries are taken into account. At the same time, each additional $\mathbb{Z}_2$ symmetry beyond the first two roughly halves the computational space size.

Each optimisation was run in a manner described in Section 5.3.1. The results are given in Fig. 5.2. We compare our results with the minimum energies obtained in the existing works [45] and [46] (which did not account for $\mathbb{Z}_2$ symmetries and used the MU strategy). Another difference is that both of the above references

Molecule	N	n_e	Number of $\mathbb{Z}_2$ symmetries	Computational space size (SDs)	
				With $\mathbb{Z}_2$	Without $\mathbb{Z}_2$
LiH	12	4	4	69	225
H_2O	14	10	3	261	441
NH_3*	16	10	2 (3)	3,136 (1,576)	3,136
CH_4*	18	10	2 (4)	15,876 (4,076)	15,876
N_2	20	14	5	1,824	14,400
C_2	20	12	5	5,612	44,100
LiF	20	12	4	11,124	44,100
PH_3*	24	18	2 (3)	48,400 (24,202)	48,400
LiCl	28	20	4	250,581	1,002,001
Li_2O*	30	14	4 (5)	10,355,569 (5,179,569)	41,409,225
NaCl	36	28	4	2,341,648	9,363,600

Table 5.1: Reference information for studied molecules. The asterisk indicates molecules which exist in geometries with larger number of spatial symmetries compared to the geometries provided by the PubChem database [91] and studied in Ref. [45] and [46]. For such molecules we provide in parentheses figures corresponding to the most symmetric geometries.

aimed to keep the final N_{unq} within some range and thus the initial batch size N_s was adaptively chosen at each iteration after multiple possible resamplings. Their motivation was to sample not too few unique basis vectors — in which case the gradient becomes too noisy — but also not too many, since the local energy calculations might become prohibitively expensive. In our experience, there is no need for adaptive batch sizes so long as N_{s} is large enough.

For all molecules, our method with both pruning strategies demonstrates median errors below the chemical accuracy of 1.6 mHa — this is a milestone unreported in either of Refs. [45] and [46]. We also observe that the largest gain in accuracy (up to an order of magnitude compared with previous research) is achieved for molecules with higher spatial symmetry such as LiCl, N_2, LiF, Li_2O and C_2. We note that our method appears to underperform with respect to Ref. [45] on LiCl molecule. However, the experimental point shown for Ref. [45] in Fig. 5.2 corresponds to the best performance amongst multiple seeds; the average energy of the Ref. [45] optimisation on LiCl is above the median value obtained in our experiments.

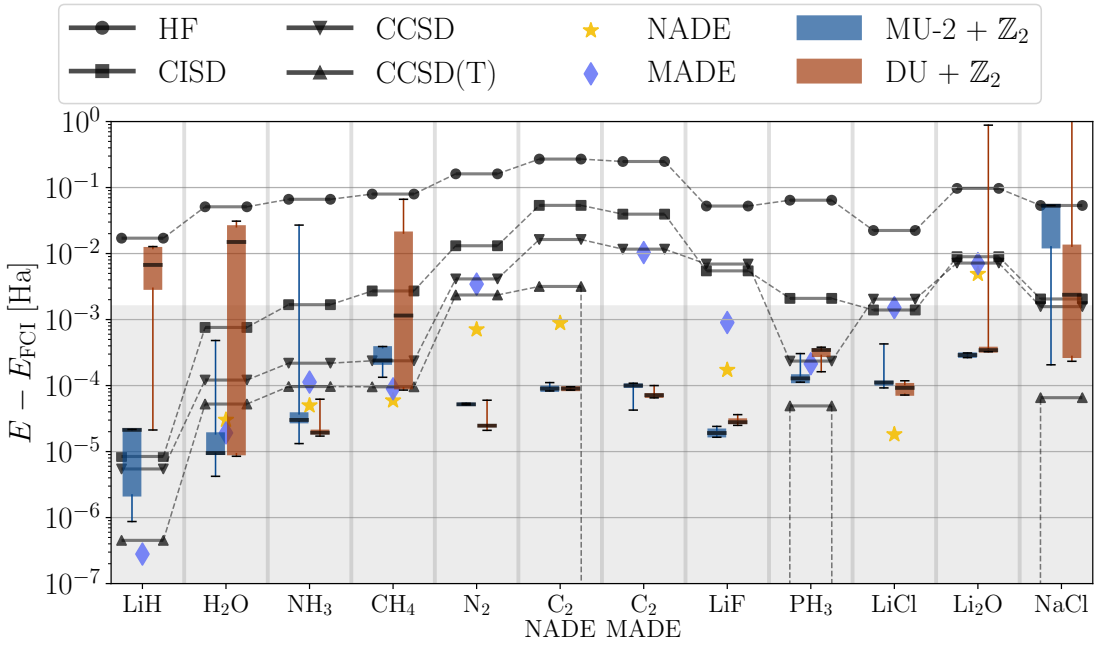


Figure 5.2: Comparison of the variational energies achieved by our symmetry-aware ANQS optimisation and the existing ANQS variants studied in the literature [45, 46] (denoted as "NADE" and "MADE" respectively). Ref. [45] and Ref. [46] studied the C_2 molecule at two different geometries, we present results for both of them. The black bold line on the box bodies corresponds to the median value and whiskers stretch from the minimum to maximum value in the distribution of results. The shadowed area spans energies below the chemical accuracy benchmark. For better visibility we plot the reference energies of different methods as continuous curves, even though they belong to different molecules and are not related. For C_2 (MADE), LiF, LiCl and Li_2O molecules CCSD(T) energies are below the corresponding FCI energies, and therefore the CCSD(T) curve "dips" due to logarithmic scale of the energy error axis.

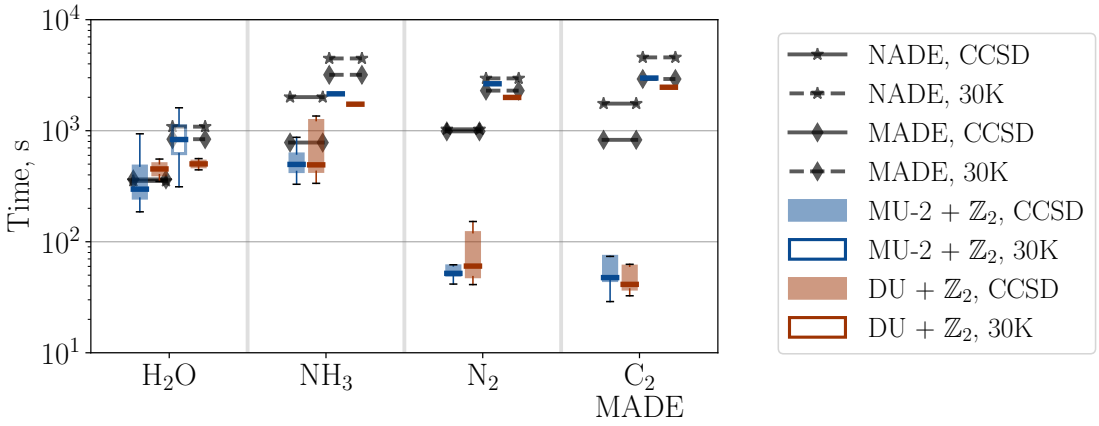


Figure 5.3: Comparison of the computational performance of the proposed symmetry-aware ANQS optimisation and the existing ANQS variants. Data points labelled "CCSD" correspond to the time required to achieve the CCSD level of accuracy; data points labelled "30K" correspond to the time spent on 3×10^4 variational optimisation iterations.

5.3.3 Time to CCSD

While the final accuracy achieved during the variational optimisation is deemed to be the primary figure of merit, it is also important to analyse the computational cost of the method. To that end, we follow Ref. [46] and extract two additional metrics from the experiments held in the previous section — the time required to achieve the CCSD level of accuracy and the total time spent on 3×10^4 iterations; we present the results in Fig. 5.3 together with the numbers from Ref. [46].

As it can be seen, the per-iteration runtime of our method is similar to that of previous approaches. However, for highly symmetric N_2 and C_2 molecules our method converges to the desired level of accuracy much faster than the existing ones: we observe a speedup of more than an order of magnitude. Another example is Li_2O: the ANQS of Ref. [45] required 45.6 hours to achieve the accuracy of $1.8 \cdot 10^{-3}$ Ha, which is above the chemical accuracy (not shown on the plot, we quote the figure given in Ref. [45]). In contrast, our method required 5.1 hours on average to reach chemical accuracy.

5.3.4 Loss of samples

In the experiments described so far, both MU-2 and DU strategies performed on par. To reveal the difference between them, we investigate how the total number of samples and the number of unique samples produced per iteration changes in the course of optimisation.

In Fig. 5.4 we take N_2 as a model molecule and show how typical optimisation unravels for both strategies. At the very start, the ANQS is equally likely to sample any physical basis vector, and therefore first iterations feature large N_{unq}. In addition, a substantial number of samples is lost by both strategies since the probability mass assigned to the correct symmetry sector is roughly equal to the fraction of the unmasked Hilbert space it occupies. Yet, the ansatz quickly learns the peaked structure of the molecular wave function, and as the variational energy passes the Hartree-Fock reference value (which can be achieved with a single basis vector contributing to the state), N_{unq} reaches minimum. At this stage, a large

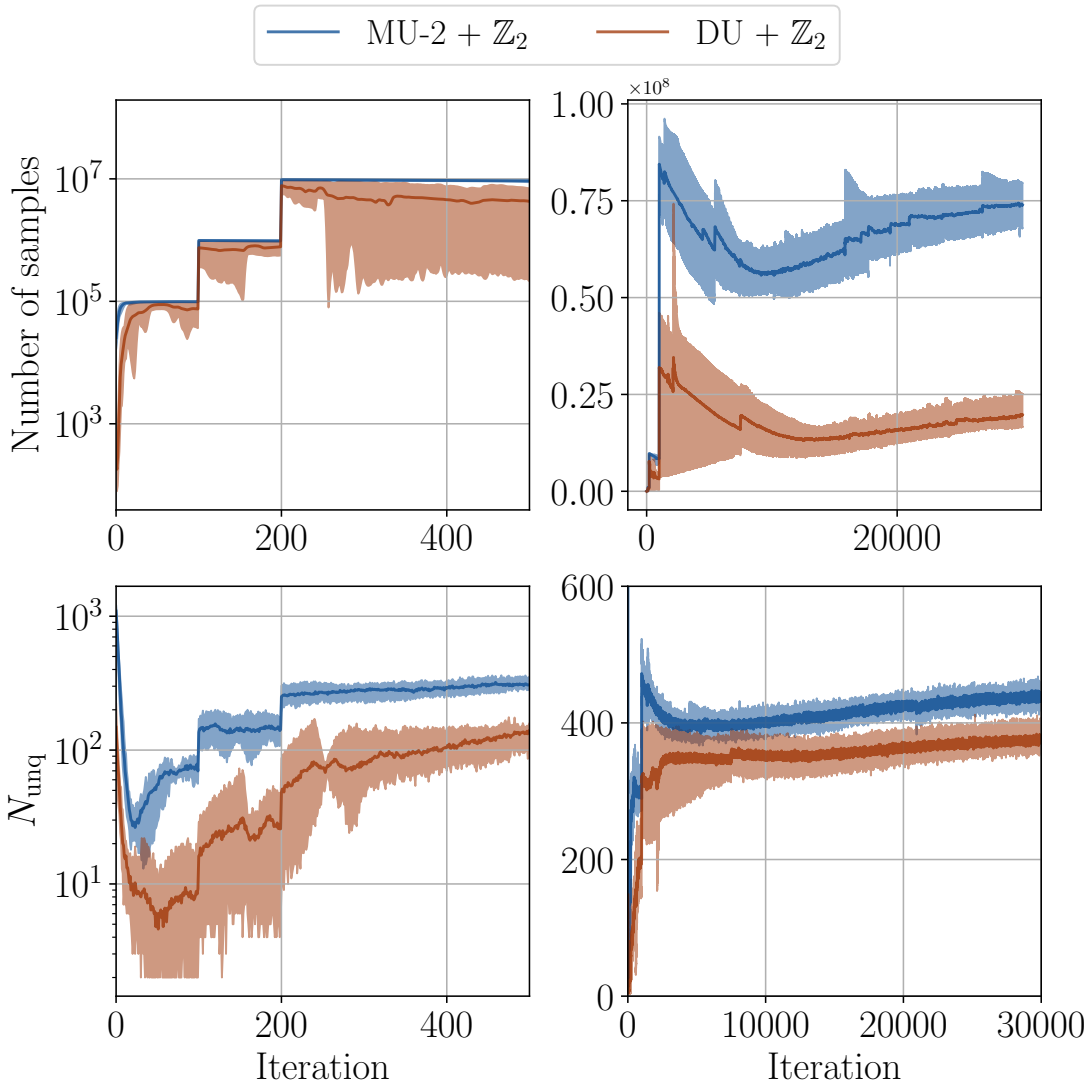


Figure 5.4: Comparison of the MU-2 and DU pruning strategies with respect to the total number of samples and the number of unique samples produced at every iteration. The solid lines represent the median values obtained during five runs of a randomly initialised ANQS, while shaded regions span from minimum to maximum values. The plots in the left column focus on the first 500 iterations when the variational energy passes the HF reference value. The plots in the right column show the performance of both strategies over the whole course of iteration. Overall, the MU-2 strategy loses fewer samples and produces more unique samples at each iteration.

portion of the probability mass is located inside the correct symmetry sector, and therefore few samples are lost. Finally, as the optimisation proceeds, N_{unq} gradually increases, which reflects how the ANQS seeks to decrease the energy by adding more and more basis vectors to the quantum state.

While both strategies perform qualitatively similarly, there is a quantitative difference between them: the DU strategy is more likely to lose samples, whether unique or not — sometimes it produces as few unique samples as one. One might not consider this as a major drawback: neither the total number of samples nor the number of unique samples are a primary figure of merit for the variational optimisation; in the end, DU achieves lower variational energies on N_2 molecule than MU-2. However, we see this as a disadvantage of *practical* importance: for bigger molecules the DU strategy is more likely to produce no samples in the correct symmetry sector at early stages of optimisation, and thus stall the whole process. Even though this can be mitigated by carefully scheduling N_{s}, we believe this puts MU-2 forward as a more robust and practical pruning strategy.

Part III

Computational performance

6

Blessing of peaked structure

Contents

We proceed to the third part of the thesis, which considers a set of advances of both conceptual and practical nature aimed at reducing the compute time required for ANQS quantum chemistry calculations. There are two outstanding challenges inherent to such calculations. First, the molecular ground state wave functions are often of peaked structure, i.e. dominated by a few high amplitude components in the computational basis, which constitutes a major obstacle for sampling the Born distribution. Second, molecular Hamiltonians have an excessive number of terms, which renders the stochastic estimation of energy highly expensive and poses a formidable challenge of computational nature.

In this chapter we show that by carefully revisiting the optimisation procedure

complexity one can substantially reduce the computational cost of NQS calculations. Specifically, we argue that, paradoxically, the previously unwelcome peakedness of the wave function might substantially decrease the computational demands of local energy calculations. Our approach includes four key components (i) autoregressive sampling without replacement; (ii) two novel approaches to local energy evaluation with improved asymptotic complexity; (iii) a modification of the stochastic reconfiguration (SR) technique tailored to ANQS; (iv) compressed data representation enabling memory and compute efficient GPU implementation. We start this chapter with a high-level overview of each component. In the second part of this chapter we provide a technical consideration of the streamlined local energy calculations, while Chapter 7 covers our GPU implementation.

6.1 Our method at a glance

6.1.1 Autoregressive sampling without replacement

As discussed in Chapter 3, the autoregressive statistics sampling addresses the sampling bottleneck only to an extent. Specifically, it requires careful batch size scheduling as corroborated by the results of Chapter 5. There, achieving chemical accuracy for the studied systems required $N_\mathrm{s} \approx 10^8$. However, we had to keep N_s equal to 10^6 early in optimisation to ensure that the number of unique samples is $\approx 10^3$ and does not cause out-of-memory errors. Such batch scheduling introduces an additional degree of complexity to the variational optimisation and it is highly desirable to avoid it. Thus, to obtain the direct control over the number of unique samples we resort to autoregressive Gumbel top-K sampling described in Chapter 3.

6.1.2 Streamlined local energy calculations

The calculation of local energies constitutes the central computational bottleneck of NQS quantum chemistry calculations since the number of terms in molecular Hamiltonians grows quartically with the number of orbitals, $N_\mathrm{T} = \mathcal{O}(N^4)$ [75]. As discussed in Chapter 2, this implies that to calculate the local energy according to Eq. (1.3) one has to evaluate $\mathcal{O}(N_\mathrm{unq} N_\mathrm{T})$ additional ansatz amplitudes. We

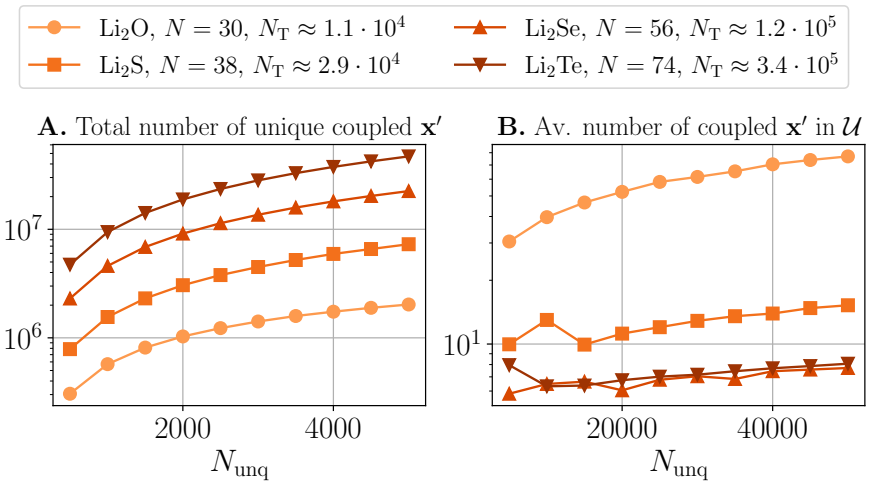


Figure 6.1: Computational scaling of local energy calculations. All molecules are considered in the minimal STO-3G basis set. **A.** The number of additional ansatz evaluations required per iteration for local energy calculation. **B.** The average number of $\mathbf{x}'$ coupled to a given $\mathbf{x}$ among all sampled basis vectors.

illustrate a prohibitive incurred cost of such calculation in Fig. 6.1A, where we consider four molecules of increasing size. We vary the number of unique samples N_{unq} and plot how many different $\mathbf{x}'$ are coupled to all $\mathbf{x}$ in $\mathcal{U}$. One can see that even for the smallest molecule Li_2O with 30 qubits, setting N_{unq} to 5000 requires an additional million of ansatz evaluations; this number climbs up to 50 millions for the largest considered molecule Li_2Te, which proves to be extremely demanding both in terms of required compute and memory.

To mitigate this problem, we build upon the proposal of Wu *et al.* [47] and replace the local energy calculation (1.3) with its computationally cheaper surrogate $E_{\text{loc}}^{\text{var}}(\mathbf{x})$ which contains contributions only from those $\mathbf{x}'$ which belong to the set of unique sampled basis vectors. Hence, for the full energy calculation one requires only the amplitudes of basis vectors in $\mathcal{U}$, which can be computed and stored in memory with $\mathcal{O}(N_{\text{unq}})$ ansatz evaluations immediately after $\mathcal{U}$ is obtained.

In addition, instead of evaluating the state energy as a Monte Carlo expectation (1.4) of the local energy, we directly calculate the variational energy of the instantly sampled state:

$$E^{\mathrm{var}} = \sum_{\mathbf{x}\in\mathcal{U}} E^{\mathrm{var}}_{\mathrm{loc}}(\mathbf{x}) \cdot \frac{p(\mathbf{x})}{\mathcal{N}}, \text{where } \mathcal{N} \coloneqq \sum_{\mathbf{x}\in\mathcal{U}} p(\mathbf{x}). \tag{6.1}$$

Here we weigh local energy surrogate values by the renormalised probabilities obtained *directly from the ansatz*, as opposed to weighing them by empirical frequencies $\frac{n(\mathbf{x})}{N_{\mathrm{s}}}$. The reason for this is twofold. First, sampling without replacement does not provide any empirical frequencies, but only the unique samples themselves, and thus one cannot use Eq. (1.4) directly. Second, under such definition, the value calculated via Eq. (6.1) becomes variational, in that it is always an upper bound for the ground state energy since it corresponds to a physical state spanned by the vectors in $\mathcal{U}$. Hence, in what follows we refer to the value given by Eq. (6.1) as the variational energy, and to the values of $E^{\mathrm{var}}_{\mathrm{loc}}(\mathbf{x})$ as the variational (proxy of) local energy.

Improved asymptotic complexity

Although the approach of Ref. [47] reduces the number of *ansatz evaluations* significantly below $\mathcal{O}(N_{\mathrm{unq}}N_{\mathrm{T}})$, the number of *computational operations* required to calculate E^{var} still scales with N_{T}. This is because there are N_{unq} values of $E^{\mathrm{var}}_{\mathrm{loc}}(\mathbf{x})$ to be calculated; to obtain each of them, the authors iterate over N_{T} Hamiltonian terms and calculate the corresponding coupled candidate $\mathbf{x}'$. If $\mathbf{x}'$ belongs to $\mathcal{U}$, they add the relevant term to $E^{\mathrm{var}}_{\mathrm{loc}}(\mathbf{x})$. In Ref. [47] checking whether $\mathbf{x}'$ belongs to $\mathcal{U}$ is implemented with binary search in $\mathcal{O}(\log N_{\mathrm{unq}})$ operations, and thus in total one performs $\tilde{\mathcal{O}}(N_{\mathrm{unq}}N_{\mathrm{T}})$ operations. Here $\tilde{\mathcal{O}}$ indicates the asymptotic complexity with omitted logarithmic scaling factors.

In this work we propose two novel methods to evaluate E^{var}, which have asymptotic complexity that does not depend on N_{T}. Specifically, we split the calculation of E^{var} into two steps. First, we obtain all pairs $(\mathbf{x}, \mathbf{x}') \in \mathcal{U} \times \mathcal{U}$ which are coupled via the Hamiltonian. We refer to this stage as the FindCoupledPairs procedure. Second, we evaluate the matrix elements $H_{\mathbf{x}\mathbf{x}'}$ and add the corresponding factors to $E^{\mathrm{var}}_{\mathrm{loc}}(\mathbf{x})$ and $E^{\mathrm{var}}_{\mathrm{loc}}(\mathbf{x}')$. We refer to this stage as the MatrixElement procedure. Our methods focus on improving the asymptotic complexity of FindCoupledPairs.

The benchmark implementation we compare with is that of Ref. [47], to which we refer as LoopOverTerms.

When $N_{\text{unq}} \ll N_{\text{T}}$ the majority of candidate $\mathbf{x}'$ will not belong to $\mathcal{U}$. To take advantage of this, our first approach, to which we refer as LoopOverBatch, iterates over each pair $(\mathbf{x}, \mathbf{x}') \in \mathcal{U} \times \mathcal{U}$ and checks whether there exists a Hamiltonian term coupling $\mathbf{x}$ to $\mathbf{x}'$. This approach brings the complexity of FindCoupledPairs down to $\tilde{\mathcal{O}}(N_{\text{unq}}^2)$.

Our second approach employs an empirical observation that in practice very few pairs $(\mathbf{x}, \mathbf{x}')$ are actually coupled via the Hamiltonian (see Fig. 6.1B). Thus, iterating over each element of $\mathcal{U} \times \mathcal{U}$ results in unnecessary computations too. We address this issue by preprocessing $\mathcal{U}$ and reorganising it into a data structure known as *prefix tree* or simply *trie*, which we describe in more details in Section 6.2. This data structure allows one to exploit the sparsity of couplings within $\mathcal{U} \times \mathcal{U}$ and substantially speed up FindCoupledPairs. We refer to this implementation of FindCoupledPairs as LoopOverTrie.

6.1.3 ANQS-tailored stochastic reconfiguration

We also harness the improved convergence of the stochastic reconfiguration procedure described in Chapter 1. We introduce two modifications specifically tailored to the use of an autoregressive ansatz. First, we evaluate stochastic averages using renormalised probabilities $\frac{p(\mathbf{x})}{\mathcal{N}}$ of unique samples, as opposed to their occurrence numbers. Second, evaluating the Jacobian matrix $J_{\mathbf{x}p} = \frac{\partial \psi(\mathbf{x})}{\partial \theta_p}$ for each $\mathbf{x}$ in $\mathcal{U}$ is computationally heavy, and thus we restrict the set of unique samples used to evaluate S. Specifically, we take only first $N_{\text{unq}}^{\text{SR}} \ll N_{\text{unq}}$ unique samples corresponding to the highest probabilities. In all experiments we use $N_{\text{unq}}^{\text{SR}} = 100$ chosen after a study on how $N_{\text{unq}}^{\text{SR}}$ impacts the achieved variational energies (see Chapter 8). Apart from the computational benefit, this also seems to numerically stabilise the inversion of S, which is performed following the recipes of Refs. [20] and [21].

We use the SR-transformed gradients jointly with the Adam optimiser. We observe that SR substantially boosts the optimisation convergence, i.e. allows

achieving lower energies earlier in the optimisation. This observation contrasts with previous reports that cast doubt on the merit of SR for ANQS optimisation [34, 53].

6.1.4 GPU implementation

Apart from improving the asymptotic complexity of optimisation, we also focus on the practical aspects of its implementation, namely on fully leveraging the inherent massive parallelism of GPUs. We adapt the approach of Wu *et al.* [47] to store each basis vector $\mathbf{x}$ in a compressed form as a tuple of integers. This leads to an eightfold reduction in memory requirement compared to previous implementations, in which each bit was stored as a one-byte integer. This is critical given our algorithms require simultaneous handling of data structures containing $\mathcal{O}({N_{\text{unq}}}^2) \gtrsim 10^9$ bit vectors of length N (see Chapter 7).

We advance this method further by implementing *all* steps of local energy calculation as a sequence of *bitwise* operations on $\mathbf{x}$ stored in such compressed format. This allows us to replace looping over qubits with vectorised bitwise processor operations, leading to highly accelerated calculations. Specifically, encoding the basis vectors in 64-bit integers speeds up the calculation by a factor of up to 64. As a result, experiments for systems with up to $3 \cdot 10^6$ Hamiltonian terms required only a single GPU with 24GB of RAM and barely over a second per iteration. This is a substantial reduction in comparison to Refs. [46] and [47] which relied on several GPUs and took dozens of seconds per iteration. In addition, our implementation employs standard tensor algebra routines of PyTorch software library [22] and may be run without any custom CUDA code. Chapter 7 overviews the key aspects of our code.

6.2 Technical details

6.2.1 Hamiltonian arithmetic

We start the technical consideration of our approach by describing how each Hamiltonian term in Eq. (2.10) can be represented as a tuple of N-bit vectors.

We define the bit vector $\mathbf{X}^{(l)}$ encoding the information about the positions of $\hat{X}$ operators in the Hamiltonian term $\hat{P}^{(l)} \coloneqq \bigotimes_{i=0}^{N-1} \hat{P}_i^{(l)}$ as follows:

$$X_i^{(l)} \coloneqq \begin{cases} 1, \text{ if } \hat{P}_i^{(l)} = \hat{X}; \\ 0, \text{ otherwise.} \end{cases}$$

The bit vectors $\mathbf{Y}^{(l)}$ and $\mathbf{Z}^{(l)}$ are defined in analogous way. Thus, one can store the information about each term as a tuple of four items $\left(h_l, \mathbf{X}^{(l)}, \mathbf{Y}^{(l)}, \mathbf{Z}^{(l)}\right)$.

This representation gives rise to *Hamiltonian arithmetic*: a set of rules to evaluate $\langle \mathbf{x} | \hat{P}^{(l)} | \mathbf{x}' \rangle$ as a sequence of bitwise operations on N-bit vectors representing the term and basis vectors:

$$\langle \mathbf{x} | \hat{P}^{(l)} | \mathbf{x}' \rangle = \begin{cases} 0, \text{ if } \mathbf{x} \otimes \mathbf{x}' \neq \underline{\mathbf{XY}}^{(l)}; \\ \mathrm{e}^{i\phi\left(\mathbf{x}', \hat{P}^{(l)}\right)}, \text{ otherwise,} \end{cases} \tag{6.2}$$

$$\text{where } \phi\left(\mathbf{x}', \hat{P}^{(l)}\right) \coloneqq \frac{\pi}{2} \left| \mathbf{Y}^{(l)} \right| + \pi \left| \mathbf{x}' \odot \underline{\mathbf{YZ}}^{(l)} \right|,$$

where $\oplus$, $\odot$ and $\otimes$ denote the bitwise OR, AND and XOR operations, respectively; $|\mathbf{x}| \coloneqq \sum_{i=0}^{N-1} x_i$ is the Hamming weight of a $\mathbf{x}$; $\underline{\mathbf{XY}}^{(l)} \coloneqq \mathbf{X}^{(l)} \oplus \mathbf{Y}^{(l)}$ and $\underline{\mathbf{YZ}}^{(l)} \coloneqq \mathbf{Y}^{(l)} \oplus \mathbf{Z}^{(l)}$.

We see that each $\hat{P}^{(l)}$ couples $\mathbf{x}$ to a single $\mathbf{x}'$ which can be calculated as $\mathbf{x}' = \mathbf{x} \otimes \underline{\mathbf{XY}}^{(l)}$. In other words, a pair $(\mathbf{x}, \mathbf{x}')$ is coupled by a given Hamiltonian term $\hat{P}^{(l)}$ if $\underline{\mathbf{XY}}^{(l)}$ equals $\mathbf{x} \otimes \mathbf{x}'$. The matrix element itself is just a complex exponent with the phase $\phi\left(\mathbf{x}', \hat{P}^{(l)}\right)$.

Unique $\underline{\mathbf{XY}}$

Note, however, that in realistic molecular Hamiltonians *several* $\hat{P}^{(l)}$ might have the same $\underline{\mathbf{XY}}$ representation, i.e. the number of *unique* $\underline{\mathbf{XY}}^{(l)}$ is less than N_T. Let us denote the set of all unique $\underline{\mathbf{XY}}^{(l)}$ as $\mathcal{XY}$; the size $|\mathcal{XY}|$ of this set is still $\mathcal{O}(N_\mathrm{T}) = \mathcal{O}(N^4)$. For a given $\mathbf{x}$, every implementation of FindCoupledPairs seeks to find a set of $\mathbf{x}'$ such that $\mathbf{x} \otimes \mathbf{x}' \in \mathcal{XY}$. We denote such set as $\mathcal{SC}_\mathbf{x} \coloneqq \{\, \mathbf{x}' \mid \mathbf{x} \otimes \mathbf{x}' \in \mathcal{XY} \,\}$.

6.2.2 Implementations of FindCoupledPairs

Let us discuss three potential implementations of the FindCoupledPairs subroutine, each having different computational complexity; we point out the preferred use cases for each of them.

LoopOverTerms

The LoopOverTerms implementation can be described with the following pseudocode:

Algorithm 2A Find $\mathcal{SC}_{\mathbf{x}}$ via looping over $\mathcal{XY}$ $\underline{\mathbf{XY}}^{(l)}$.

```
1: function LoopOverTerms(x)
2:     SC_x := [ ]
3:     for XY in XY do
4:         x' := x ⊗ XY
5:         if x' in U then
6:             SC_x.append(x')
7:     return SC_x
```

Invoking LoopOverTerms for all $\mathbf{x}$ in $\mathcal{U}$ results in the time complexity $\mathcal{O}\left(N_{\text{unq}} \cdot N_{\text{T}} \cdot (\mathcal{C}_{\text{bitop}} + \mathcal{C}_{\in\mathcal{U}})\right)$. Here $\mathcal{C}_{\text{bitop}}$ is the complexity of a bitwise operation on a pair of basis vectors, which, in principle, is $\mathcal{O}(N)$. However, with the compressed storing of N-bit vectors discussed in Section 6.1.4 it is possible to reduce the scaling constant by the integer bit depth (64 in our case). At the same time, $\mathcal{C}_{\in\mathcal{U}}$ is the complexity of checking whether a candidate vector $\mathbf{x}'$ belongs to the batch of sampled unique vectors. In Ref. [47] $\mathcal{U}$ is stored as an ordered table and the checking is performed via binary search; hence $\mathcal{C}_{\in\mathcal{U}} = \mathcal{O}(\log N_{\text{unq}})$. However, implementing binary search posed a significant challenge for our GPU-based implementation. Hence, in Chapter 7 we propose an alternative approach with similar asymptotic complexity.

LoopOverBatch

The pseudocode for LoopOverBatch is as follows:

Algorithm 2B Find $\mathcal{SC}_{\mathbf{x}}$ via looping over $\mathcal{U}$.

```
1: function LoopOverBatch(x)
2:     SC_x := [ ]
3:     for x' in U do
4:         XY := x ⊗ x'
5:         if XY in XY then
6:             SC_x.append(x')
7:     return SC_x
```

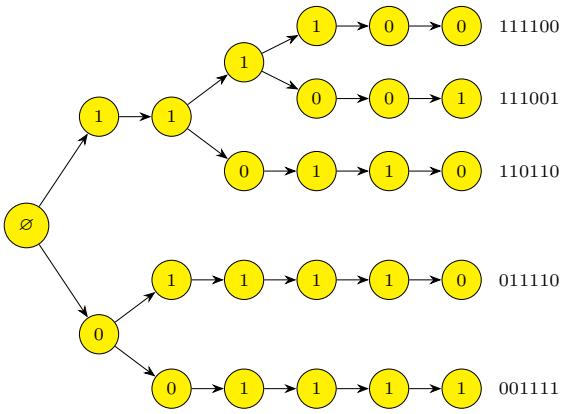


Figure 6.2: Prefix tree for a set of five bit vectors $\{001111, 011110, 110110, 111001, 111100\}$. One can see that, for example, the strings 110110 and 111001 are encoded by the same first two levels of the prefix tree, while the remaining levels are different for them.

Invoking LoopOverTerms for all $\mathbf{x}$ in $\mathcal{U}$ results in the time complexity $\mathcal{O}\left(N_{\text{unq}}^2(\mathcal{C}_{\text{bitop}} + \mathcal{C}_{\in\mathcal{XY}})\right)$. Similarly to LoopOverTerms $\mathcal{C}_{\in\mathcal{XY}} = \mathcal{O}\left(\log N_{\text{T}}\right)$ is the complexity of checking whether each calculated $\underline{\mathbf{XY}}$ belongs to $\mathcal{XY}$. To implement this check in $\ll \mathcal{O}(N_{\text{T}})$ operations for each pair, we create an ordered data structure that lists both the $(\mathbf{x}, \mathbf{x}')$ pairs and the Hamiltonian terms encoded as bit vectors in a way enabling rapid lookup, as further explained in the Chapter 7.

Prefix tree

Algorithm 2B evaluates $\mathbf{x} \otimes \mathbf{x}'$ for each pair of unique basis vectors and then checks whether the result belongs to $\mathcal{XY}$. This requires looping over *every* bit in $\mathbf{x}$ and $\mathbf{x}'$. Importantly, during this loop, Algorithm 2B does not try to check whether a *partial* XOR of $\mathbf{x}$ and $\mathbf{x}'$ is valid, i.e. whether it might be a prefix of any $\underline{\mathbf{XY}}$ in $\mathcal{XY}$. Such approach results in unnecessary calculations, since it might become apparent early in the loop that $\mathbf{x}$ and $\mathbf{x}'$ cannot be coupled by any term in $\hat{H}$. We address this issue by storing *both* $\mathcal{U}$ and $\mathcal{XY}$ as data structures known as *prefix trees*. As will be discussed shortly, this allows removing "futile" pairs $(\mathbf{x}_{<i}, \mathbf{x}'_{<i})$ from consideration as early as possible.

Prefix tree (also known as *trie*) is a data structure used to store a set of bit vectors (or more generally strings over some alphabet) in a compressed way. Suppose

one has to store two strings *abcd* and *abce*. Both strings share the same prefix *abc*. To avoid keeping redundant information, one might just store the prefix *abc*, two possible endings *d* and *e* and the way the endings are connected to the prefix. The prefix tree develops this idea further and applies it to *all* possible prefixes in a set of strings as illustrated in Fig. 6.2. From a more formal perspective, prefix tree is a directed tree, where the root node corresponds to the "start of a string" symbol, ordinary nodes contain string symbols, and every path in the tree from the root to a leaf represents a particular string in the set.

Prefix tree construction

Algorithm 3 presents a pseudocode to construct a prefix tree from a set of unique bit vectors. We store the tree $\mathcal{T}$ as a list of nodes representing unique prefixes. We presume that each node is a data structure with three fields: `value`, storing x_i; `parent`, storing the reference to a parent node at the previous level; and array of two references `children` to children nodes at the next level. By default, the references are set to `None` indicating that the link between the node and its parent/children has not been established yet. The prefix $\mathbf{x}_{<i}$ of a node can be reconstructed by traversing the prefix tree up via `parent` references.

Algorithm 3 Constrution of prefix tree

```
1: function ConstructPrefixTree(𝒰)
2:     𝒯 ≔ [ ]
3:     𝒯.append(Node(value = ∅))
4:     for x in 𝒰 do
5:         current_node ≔ 𝒯[0]
6:         for i from 0 to N − 1 do
7:             if current_node.children[x_i] = None then
8:                 new_node ≔ Node(value=x_i)
9:                 new_node.parent ≔ current_node
10:                current_node.children[x_i] ≔ new_node
11:                𝒯.append(new_node)
12:            current_node ≔ current_node.children[x_i]
13:    return 𝒯
```

LoopOverTrie

In the LoopOverTrie implementation of the FindCoupledPairs procedure we store both $\mathcal{U}$ and $\mathcal{XY}$ as prefix trees, which we denote as $\mathcal{T}^{\mathcal{U}}$ and $\mathcal{T}^{\mathcal{XY}}$ respectively. For a given $\mathbf{x}$, we traverse the prefix tree $\mathcal{T}^{\mathcal{U}}$ level-by-level and keep only those paths which correspond to a valid prefix of the same length in $\mathcal{T}^{\mathcal{XY}}$. Note that checking whether $\underline{\mathbf{XY}}_{<i}$ belongs to $\mathcal{T}^{\mathcal{XY}}$ does not require any search operations. Instead, since each prefix $\underline{\mathbf{XY}}_{<i}$ is built sequentially starting from the root, one only needs to examine whether the node corresponding to $\underline{\mathbf{XY}}_{<i-1}$ has a child corresponding to $x_i \otimes x'_i$. The pseudocode 2C provides a more rigorous description of this procedure.

Algorithm 2C Find $\mathcal{SC}_{\mathbf{x}}$ via prefix tree.

```
function LoopOverTrie(x)
    T^U := ConstructPrefixTree(U)
    T^XY := ConstructPrefixTree(XY)
    # Initialise the set of coupled x'_{<i} and corresponding XY_{<i}.
    SCXY_x := [(T^U[0], T^XY[0])]

    for i from 0 to N − 1 do
        SCXY'_x := [ ]

        for (u_node, xy_node) in SCXY_x do
            for x'_i in {0, 1} do
                new_u_node := u_node.children[x'_i]
                new_xy_node := xy_node.children[x_i ⊗ x'_i]

                if new_u_node ≠ None and new_xy_node ≠ None then
                    SCXY'_x.append((new_u_node, new_xy_node))
                else
                    continue
        SCXY_x := SCXY'_x
    Reconstruct SC_x from every u_node in SCXY_x.
    return SC_x
```

LoopOverTrie complexity

To conclude the discussion of LoopOverTrie, let us consider its worst case computational complexity. The worst case corresponds to the situation when *all* $\mathbf{x}$ in $\mathcal{U}$ are coupled to each other and *no* paths are dropped during the traversal

of $\mathcal{T}^{\mathcal{U}}$. Hence, starting from some level we have N_{unq} paths. Consequently, the number of operations required to traverse through N levels of $\mathcal{T}^{\mathcal{U}}$ is bounded from above by $\mathcal{O}(N \cdot N_{\text{unq}})$. Since we invoke LoopOverTrie for each $\mathbf{x}$ in $\mathcal{U}$, the total complexity of FindCoupledPairs becomes $\mathcal{O}(N \cdot N_{\text{unq}}^2)$. Thus, we recover the complexity of all-to-all coupling Algorithm 2B, assuming that $\mathcal{C}_{\text{bitop}} = \mathcal{O}(N)$.

The above discussion leads us to expect LoopOverTrie to be at least as fast as LoopOverBatch. However, this is not always the case in practice, as will be demonstrated in Chapter 8. This is because LoopOverTrie requires explicit looping over N levels of prefix trees, and thus can not be fully vectorised. Nevertheless, for large systems efficient exploitation of coupling sparsity outweighs this disadvantage, and LoopOverTrie does demonstrate practical speedups.

6.2.3 Calculating the Hamiltonian matrix element

To calculate the Hamiltonian matrix element between two basis vectors, one needs to sum the values $h_l \langle \mathbf{x}|\hat{P}^{(l)}|\mathbf{x}'\rangle$ over the set of all $\hat{P}^{(l)}$ corresponding to the same $\underline{\mathbf{XY}}$. We define this set as follows:

$$\mathcal{P}_{\underline{\mathbf{XY}}} \coloneqq \left\{ \hat{P}^{(l)} \equiv \left(h_l, \mathbf{X}^{(l)}, \mathbf{Y}^{(l)}, \mathbf{Z}^{(l)}\right) \;\middle|\; \underline{\mathbf{XY}}^{(l)} = \underline{\mathbf{XY}} \right\}. \tag{6.3}$$

In this case, the matrix element $H_{\mathbf{x}\mathbf{x}'}$ can be calculated using the following algorithm:

Algorithm 3 Calculating the Hamiltonian matrix element

```
function MatrixElement(x, x')
    H_xx' := 0
    XY := x ⊗ x'
    for (h_l, X^(l), Y^(l), Z^(l)) in P_XY do
        φ := (π/2) |Y^(l)| + π |x' ⊙ YZ^(l)|
        H_xx' += h_l · e^(iφ)
    return H_xx'
```

The set $\mathcal{P}_{\underline{\mathbf{XY}}}$ for every unique $\underline{\mathbf{XY}}$ can be precalculated at the start of optimisation and accessed from memory on demand. Thus, for a given $\underline{\mathbf{XY}}$, the complexity of Algorithm 3 mainly depends on the cardinality of $\mathcal{P}_{\underline{\mathbf{XY}}}$, which we

denote as $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$. Realistic molecular Hamiltonians contain different $\underline{\mathbf{XY}}$ which correspond to $\mathcal{P}_{\underline{\mathbf{XY}}}$ of different sizes. Therefore, it only makes sense to consider some average *empiric* complexity of MatrixElement, which depends on $\underline{\mathbf{XY}}$ vectors encountered during the optimisation and their corresponding $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$. Let us provide a brief glance at what this complexity is.

	N_{T}	$\lvert\mathcal{XY}\rvert$	Average $\left\lvert\mathcal{P}_{\underline{\mathbf{XY}}}\right\rvert$	$\left\lvert\mathcal{P}_{\underline{\mathbf{XY}}=0}\right\rvert$	Average $\left\lvert\mathcal{P}_{\underline{\mathbf{XY}}\neq 0}\right\rvert$
Li_2O	11434	2074	5.51	466	5.29
Li_2S	28862	5430	5.32	742	5.18
Li_2Se	118041	23497	5.02	1597	4.96
Li_2Te	344248	70471	4.88	2776	4.85

Table 6.1: Quantities defining the MatrixElement complexity.

In Table 6.1 we list the information about the Hamiltonians for already considered molecules Li_2O, Li_2S, Li_2Se and Li_2Te. In particular, we provide figures for the number of Hamiltonian terms N_{T}, number of unique $\underline{\mathbf{XY}}$, which we denote as $|\mathcal{XY}|$, and the average number of terms per unique $\underline{\mathbf{XY}}$, which we calculate as $\frac{N_{\mathrm{T}}}{|\mathcal{XY}|}$. It can be seen that across all four molecules there are on average ≈ 5 terms per unique $\underline{\mathbf{XY}}$. Hence, one might expect that MatrixElement is a rather inexpensive subroutine, which requires only few bitwise operations. Unfortunately, in Fig. 6.3 we display the data which indicates that this is *not* the case.

More specifically, in Fig. 6.3A we show the average $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ observed during the variational optimisation. It can be seen that for all molecules the observed values are substantially larger than what one might expect from the naïve calculation given in Table 6.1. The main reason for such discrepancy is that *every* $\mathcal{SC}_{\mathbf{x}}$ contains an $\underline{\mathbf{XY}}$ vector equal to 0. This vector corresponds to the self-energy $\hat{H}_{\mathbf{xx}}$ of a Slater determinant and there is a large number of terms corresponding to it. In Table 6.1 we provide the relevant numbers denoted as $\left|\mathcal{P}_{\underline{\mathbf{XY}}=0}\right|$; we found out empirically that $\left|\mathcal{P}_{\underline{\mathbf{XY}}=0}\right|$ equals $\frac{N^2+N}{2}+1$.[1] Since each Slater determinant is necessarily coupled to itself, calculations with the terms corresponding to $\underline{\mathbf{XY}}=0$ substantially increase the average observed $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$.

[1]This follows an intuitive idea that the Hamiltonian (2.2) couples a basis vector to itself only via terms $\hat{c}_i^\dagger\hat{c}_j^\dagger\hat{c}_k\hat{c}_l$ having $k=j, l=i$ or $l=j, k=i$. Clearly there are $\mathcal{O}(N^2)$ such terms.

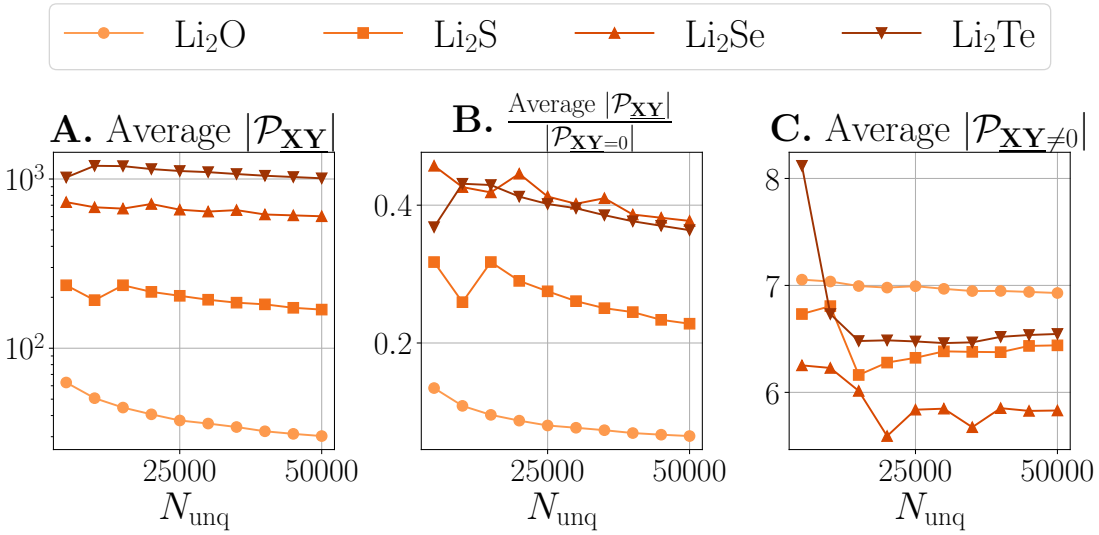


Figure 6.3: Empiric scaling of quantities defining the MatrixElement complexity. **A.** Empirically observed average $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$, i.e. the number of different $\underline{\mathbf{YZ}}$ corresponding to the same $\underline{\mathbf{XY}}$. **B.** Ratio of the empiric average $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ to the number of $\underline{\mathbf{YZ}}$ corresponding to $\underline{\mathbf{XY}} = 0$. **C.** Empirically observed average number of different $\underline{\mathbf{YZ}}$ per $\underline{\mathbf{XY}} \neq 0$.

To provide a more quantitative estimate, in Fig. 6.3B we plot the ratio between the average observed $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ and $\left|\mathcal{P}_{\underline{\mathbf{XY}}=0}\right|$. For all molecules the provided figure *decreases* as N_{unq} grows, since more $\underline{\mathbf{XY}}$ vectors with *fewer* corresponding terms $\hat{P}^{(l)}$ participate in the local energy calculation. Nevertheless, across all molecules and the whole range of considered N_{unq} this ratio remains considerable, and thus for the sake of complexity analysis we presume that average $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ is dominated by $\left|\mathcal{P}_{\underline{\mathbf{XY}}=0}\right|$, i.e. by $\mathcal{O}\left(N^2\right)$ operations. In Chapter 8 we study how critical this scaling is by measuring what part of computations is spent on the MatrixElement subroutine.

Finally, in Fig. 6.3C we plot the observed average $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ obtained after excluding $\underline{\mathbf{XY}} = 0$ from consideration, which we denote as "Average $\left|\mathcal{P}_{\underline{\mathbf{XY}}\neq 0}\right|$". The obtained values are indeed on the order of 10^1 and are close to the "naïve" values given in Table 6.1. Hence, the most expensive matrix element evaluations are those required to calculate the self-energy of a Slater determinant. We leave it as an open research question whether it is possible to speed up the self-energy calculations and improve the $\mathcal{O}\left(N^2\right)$ scaling.

Wake up, Neo... The Matrix has you...

— The Matrix. Directed by Lana and Lilly Wachowski,
Warner Bros., 1999.

7 GPU implementation

Contents

The main purpose of this chapter is to overview our GPU implementation of the functions FindCoupledPairs and MatrixElement introduced in Chapter 6. In Section 7.1 we introduce a toy example of local energy calculation, which we

l	h_l	$\hat{P}^{(l)}$	$\mathbf{Y}^{(l)}$		$\underline{\mathbf{XY}}^{(l)}$		$\underline{\mathbf{YZ}}^{(l)}$	
			Bin	Dec	Bin	Dec	Bin	Dec
0	0.9	$\hat{I}$	0000	**0**	0000	**0**	0000	**0**
1	0.1	$\hat{Z}_1\hat{Z}_2$	0000	**0**	0000	**0**	0110	**6**
2	-0.2	$\hat{X}_0\hat{X}_2$	0000	**0**	1010	**10**	0000	**0**
3	-0.2	$\hat{X}_1\hat{X}_3$	0000	**0**	0101	**5**	0000	**0**
4	0.3	$\hat{Y}_1\hat{Y}_2$	0110	**6**	0110	**6**	0110	**6**

Table 7.1: Decomposition of toy Hamiltonian terms into bit strings $\mathbf{Y}$, $\underline{\mathbf{XY}}$ and $\underline{\mathbf{YZ}}$. The columns Bin and Dec show binary and decimal representations of bit vectors correspondingly.

use as a reference for the detailed explanation of every considered subroutine. In Section 7.2 we discuss the general principles of our GPU implementation and the preparatory steps performed before the start of each simulation. In Section 7.3 we cover such subroutines as FindCoupledPairs and MatrixElement, while the prefix tree primitives are outlined in Section 7.4. Finally, in Section 7.5 we discuss implementation of several technical subroutines, not readily available in PyTorch software package [22] employed as the basis for our GPU calculations.

7.1 Model problem

7.1.1 Toy Hamiltonian

We consider the following "toy" Hamiltonian acting on the Hilbert space of four qubits:

$$\hat{H} = 0.9 \cdot \underbrace{\hat{I}}_{\hat{P}^{(0)}} + 0.1 \cdot \underbrace{\hat{Z}_1\hat{Z}_2}_{\hat{P}^{(1)}} - 0.2 \cdot \underbrace{\hat{X}_0\hat{X}_2}_{\hat{P}^{(2)}} - 0.2 \cdot \underbrace{\hat{X}_1\hat{X}_3}_{\hat{P}^{(4)}} + 0.3 \cdot \underbrace{\hat{Y}_1\hat{Y}_2}_{\hat{P}^{(4)}}. \tag{7.1}$$

This Hamiltonian does not correspond to any particular molecule; we select it exclusively for the purpose of illustration. Table 7.1 contains information about each of five Hamiltonian terms, including the bit vectors $\mathbf{Y}$, $\underline{\mathbf{XY}}$ and $\underline{\mathbf{YZ}}$ introduced in Chapter 6. For each bit vector we provide both its binary and decimal representations.

It can be seen that there are only *four* unique $\underline{\mathbf{XY}}$ in Table 7.1 which we denote as follows[1]:

$$\begin{gathered} \underline{\mathbf{XY}}_0 \coloneqq 0000; \quad \underline{\mathbf{XY}}_1 \coloneqq 1010; \quad \underline{\mathbf{XY}}_2 \coloneqq 0101; \quad \underline{\mathbf{XY}}_3 \coloneqq 0110; \\ \textsc{Dec}(\underline{\mathbf{XY}}_0) = \mathbf{0}; \quad \textsc{Dec}(\underline{\mathbf{XY}}_1) = \mathbf{10}; \quad \textsc{Dec}(\underline{\mathbf{XY}}_2) = \mathbf{5}; \quad \textsc{Dec}(\underline{\mathbf{XY}}_3) = \mathbf{6}; \end{gathered} \tag{7.2}$$

Thus, the set of unique $\underline{\mathbf{XY}}$ is as follows: $\mathcal{XY} = [\underline{\mathbf{XY}}_0, \underline{\mathbf{XY}}_1, \underline{\mathbf{XY}}_2, \underline{\mathbf{XY}}_3]$. Finally, the sets $\mathcal{P}_{\underline{\mathbf{XY}}}$ of Hamiltonian terms corresponding to the same $\underline{\mathbf{XY}}$ are as follows:

$$\mathcal{P}_{\underline{\mathbf{XY}}_0} = \left[\hat{P}^{(0)}, \hat{P}^{(1)}\right]; \quad \mathcal{P}_{\underline{\mathbf{XY}}_1} = \left[\hat{P}^{(2)}\right]; \quad \mathcal{P}_{\underline{\mathbf{XY}}_2} = \left[\hat{P}^{(3)}\right]; \quad \mathcal{P}_{\underline{\mathbf{XY}}_3} = \left[\hat{P}^{(4)}\right]. \tag{7.3}$$

In other words, there are *two* terms corresponding to $\underline{\mathbf{XY}}_0$, while the rest of $\underline{\mathbf{XY}}$ in $\mathcal{XY}$ are represented by single terms.

7.1.2 Unique batch

Suppose our batch of unique samples contains only three basis vectors $\mathcal{U} = [\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2]$ which are as follows:

$$\begin{gathered} \mathbf{x}_0 \coloneqq 1100; \quad \mathbf{x}_1 \coloneqq 1001; \quad \mathbf{x}_2 \coloneqq 0110; \\ \textsc{Dec}(\mathbf{x}_0) = \mathbf{12}; \quad \textsc{Dec}(\mathbf{x}_1) = \mathbf{9}; \quad \textsc{Dec}(\mathbf{x}_2) = \mathbf{6}. \end{gathered} \tag{7.4}$$

In addition, we assume that these basis vectors have the following (unnormalised) amplitudes:

$$\psi(\mathbf{x}_0) = 2; \quad \psi(\mathbf{x}_1) = 1; \quad \psi(\mathbf{x}_2) = -1. \tag{7.5}$$

The first step to evaluate local energies of the basis vectors is to find the coupled pairs. To that end, we compose the following tables:

	$\mathbf{x}_0$	$\mathbf{x}_1$	$\mathbf{x}_2$
$\mathbf{x}_0$	0000	0101	1010
$\mathbf{x}_1$	0101	0000	1111
$\mathbf{x}_2$	1010	1111	0000

$\equiv$

	$\mathbf{x}_0$	$\mathbf{x}_1$	$\mathbf{x}_2$
$\mathbf{x}_0$	**0**	**5**	**10**
$\mathbf{x}_1$	**5**	**0**	**15**
$\mathbf{x}_2$	**10**	**15**	**0**

$\equiv$

	$\mathbf{x}_0$	$\mathbf{x}_1$	$\mathbf{x}_2$
$\mathbf{x}_0$	$\underline{\mathbf{XY}}_0$	$\underline{\mathbf{XY}}_2$	$\underline{\mathbf{XY}}_1$
$\mathbf{x}_1$	$\underline{\mathbf{XY}}_2$	$\underline{\mathbf{XY}}_0$	—
$\mathbf{x}_2$	$\underline{\mathbf{XY}}_1$	—	$\underline{\mathbf{XY}}_0$

Here at the intersection of a row $\mathbf{x}_i$ with a column $\mathbf{x}_j$ we put the value of $\mathbf{x}_i \otimes \mathbf{x}_j$. In the first table we display its binary representation, in the second table we put

[1] One should be careful to distinguish between three similarly looking notations: (i) $\underline{\mathbf{XY}}^{(l)}$ denotes the $\underline{\mathbf{XY}}$ vector corresponding to l-th Hamiltonian term; (ii) $\underline{XY}_i$ denotes the i-th bit of a vector $\underline{\mathbf{XY}}$; (iii) finally, $\underline{\mathbf{XY}}_m$ is the m-th bit vector $\underline{\mathbf{XY}}$ in the set $\mathcal{XY}$.

its decimal representation and in the third table we specify the corresponding $\underline{\mathbf{XY}}$ in $\mathcal{XY}$. In our case the sets of coupled basis vectors are as follows:

$$\mathcal{SC}_{\mathbf{x}_0} = [\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2]\,; \quad \mathcal{SC}_{\mathbf{x}_1} = [\mathbf{x}_0, \mathbf{x}_1]\,; \quad \mathcal{SC}_{\mathbf{x}_2} = [\mathbf{x}_0, \mathbf{x}_2]\,. \tag{7.6}$$

In other words, each basis vector is coupled to itself and in addition the basis vector $\mathbf{x}_0$ is coupled to both $\mathbf{x}_1$ and $\mathbf{x}_2$.

7.1.3 Matrix elements

The corresponding matrix elements are obtained with straightforward algebraic calculations:

$$\hat{H}_{\mathbf{x}_0\mathbf{x}_0} = 0.9\,\langle \mathbf{x}_0|\hat{P}^{(0)}|\mathbf{x}_0\rangle + 0.1\,\langle \mathbf{x}_0|\hat{P}^{(1)}|\mathbf{x}_0\rangle = 0.9\underbrace{\langle 1100|1100\rangle}_{+1} + 0.1\underbrace{\langle 1100|\hat{Z}_1\hat{Z}_2|1100\rangle}_{-1} = \underline{0.8};$$

$$\hat{H}_{\mathbf{x}_0\mathbf{x}_1} = -0.2\,\langle \mathbf{x}_0|\hat{P}^{(3)}|\mathbf{x}_1\rangle = -0.2\underbrace{\langle 1100|\hat{X}_1\hat{X}_3|1001\rangle}_{1} = \underline{-0.2};$$

$$\hat{H}_{\mathbf{x}_0\mathbf{x}_2} = -0.2\,\langle \mathbf{x}_0|\hat{P}^{(2)}|\mathbf{x}_2\rangle = -0.2\underbrace{\langle 1100|\hat{X}_0\hat{X}_2|0110\rangle}_{1} = \underline{-0.2};$$

$$\hat{H}_{\mathbf{x}_1\mathbf{x}_1} = 0.9\,\langle \mathbf{x}_1|\hat{P}^{(0)}|\mathbf{x}_1\rangle + 0.1\,\langle \mathbf{x}_1|\hat{P}^{(1)}|\mathbf{x}_1\rangle = 0.9\underbrace{\langle 1001|1001\rangle}_{+1} + 0.1\underbrace{\langle 1001|\hat{Z}_1\hat{Z}_2|1001\rangle}_{+1} = \underline{1.0};$$

$$\hat{H}_{\mathbf{x}_1\mathbf{x}_0} = -0.2\,\langle \mathbf{x}_1|\hat{P}^{(3)}|\mathbf{x}_0\rangle = -0.2\underbrace{\langle 1001|\hat{X}_1\hat{X}_3|1100\rangle}_{1} = \underline{-0.2};$$

$$\hat{H}_{\mathbf{x}_2\mathbf{x}_2} = 0.9\,\langle \mathbf{x}_2|\hat{P}^{(0)}|\mathbf{x}_2\rangle + 0.1\,\langle \mathbf{x}_2|\hat{P}^{(1)}|\mathbf{x}_2\rangle = 0.9\underbrace{\langle 0110|0110\rangle}_{+1} + 0.1\underbrace{\langle 0110|\hat{Z}_1\hat{Z}_2|0110\rangle}_{1} = \underline{1.0};$$

$$\hat{H}_{\mathbf{x}_2\mathbf{x}_0} = -0.2\,\langle \mathbf{x}_2|\hat{P}^{(2)}|\mathbf{x}_0\rangle = -0.2\underbrace{\langle 0110|\hat{X}_0\hat{X}_2|1100\rangle}_{1} = \underline{-0.2};$$

7.1.4 Local energies

Finally, we bring together the computed matrix elements and the amplitudes produced by the ansatz to calculate the local energy values:

$$E^{\text{var}}_{\text{loc}}(\mathbf{x}_0) = \frac{1}{\psi(\mathbf{x}_0)}\left(\psi(\mathbf{x}_0)\hat{H}_{\mathbf{x}_0\mathbf{x}_0} + \psi(\mathbf{x}_1)\hat{H}_{\mathbf{x}_0\mathbf{x}_1} + \psi(\mathbf{x}_2)\hat{H}_{\mathbf{x}_0\mathbf{x}_2}\right) = 0.8 - \frac{1}{2}\cdot 0.2 + \frac{1}{2}\cdot 0.2 = \underline{0.8};$$

$$E^{\text{var}}_{\text{loc}}(\mathbf{x}_1) = \frac{1}{\psi(\mathbf{x}_1)}\left(\psi(\mathbf{x}_0)\hat{H}_{\mathbf{x}_1\mathbf{x}_0} + \psi(\mathbf{x}_1)\hat{H}_{\mathbf{x}_1\mathbf{x}_1}\right) = -2\cdot 0.2 + 1.0 = \underline{0.6};$$

$$E^{\text{var}}_{\text{loc}}(\mathbf{x}_2) = \frac{1}{\psi(\mathbf{x}_2)}\left(\psi(\mathbf{x}_0)\hat{H}_{\mathbf{x}_2\mathbf{x}_0} + \psi(\mathbf{x}_2)\hat{H}_{\mathbf{x}_2\mathbf{x}_2}\right) = 2\cdot 0.2 + 1.0 = \underline{1.4}.$$

7.2 General principles

At the core of every deep learning framework such as PyTorch [22], TensorFlow [23] and Jax [24] lies a library of GPU-accelerated tensor algebra primitives. These primitives operate on contiguous arrays of known size that store relevant numerical quantities (e.g. network parameters and real-world data). Such arrays are often manipulated in a vectorised (batched) manner so that the number of explicit loops is reduced to a minimum.

The toy example of previous section suggests that implementing our optimisation procedure using tensor algebra primitives is not straightforward. For example, the size of each $\mathcal{SC}_{\mathbf{x}}$ set is *dynamic* and depends on the batch $\mathcal{U}$ sampled at the current iteration. In addition, finding out coupled pairs requires a search operation to determine which of $\underline{\mathbf{XY}} = \mathbf{x} \otimes \mathbf{x}'$ corresponds to a Hamiltonian term.

In principle, one might still leverage GPU parallelism by writing a specialised CUDA code implementing the variational optimisation. However, this is beyond our ability. Instead, we implement key components of ANQS variational optimisation using only a stringent set of tensor algebra primitives, specifically that of PyTorch software library.

In this section we outline the main principles behind our implementation, while specific routines such as FindCoupledPairs and MatrixElement are covered in the subsequent sections. We illustrate every operation using the numerical values obtained for the model problem of Section 7.1.

7.2.1 Contiguous arrays

The first guiding principle of our implementation is storing information *contiguously*, as illustrated in Fig. 7.1A, which enables the use of vectorised PyTorch primitives. As discussed in Chapter 6, we represent N-bit basis vectors as tuples of $N_{\frac{\text{int}}{\text{vec}}} := \left\lceil \frac{N}{N_{\frac{\text{bits}}{\text{int}}}} \right\rceil$ integers, where $N_{\frac{\text{bits}}{\text{int}}}$ is the number of bits contained in a single integer number. In our case $N = 4$ and thus we assume only one integer is needed per basis vectors. As a result, the batch of three unique samples is stored as an integer array $\mathcal{U}$ of length 3, or, equivalently, of shape $[N_{\text{unq}}, 1]$. In our diagrams, the first and second

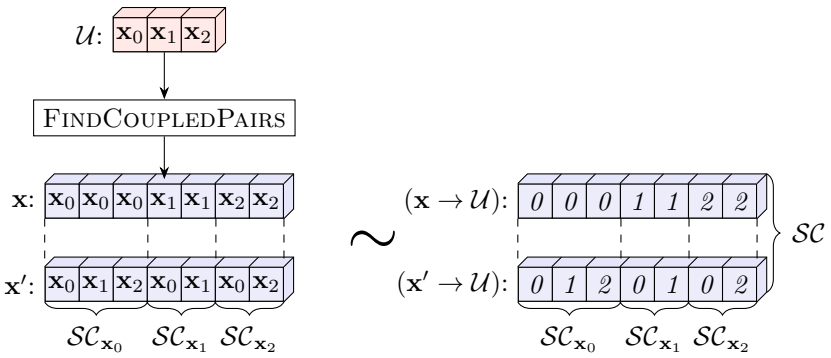


Figure 7.1: An example of storing the relevant data structures as contiguous pointers.

dimensions of each tensor are depicted horizontally and vertically, respectively. This representation is opposite to the conventional notion where a tensor with shape $[N, 1]$ is viewed as a column vector. If more than one integer is required to represent a basis vector, $\mathcal{U}$ becomes an array of shape $\left[N_{\text{unq}}, N_{\frac{\text{int}}{\text{vec}}}\right]$. In this case all procedures can be generalised to account for the extra array dimension (see our code for more details [92]).

In Fig. 7.1 we emphasise that all procedures return contiguous arrays too. For example, one can see that the FindCoupledPairs procedure takes an array $\mathcal{U}$ as input and outputs two arrays $\mathbf{x}$ and $\mathbf{x}'$. These arrays store concatenated first and second elements in each pair of every $\mathcal{SC}_{\mathbf{x}}$ respectively.

7.2.2 Pointer arithmetic

The second key aspect of our code is the use of pointer arithmetic. Suppose one has an array `A` of length $L_{\texttt{A}}$. We assume that an array $(\texttt{B} \to \texttt{A})$ is an array of pointers to `A` if $(\texttt{B} \to \texttt{A})$ is an array of integer numbers ranging from 0 to $L_{\texttt{A}} - 1$ inclusively. In this case we interpret the i-th element of $(\texttt{B} \to \texttt{A})$ as a *pointer* to the $(\texttt{B} \to \texttt{A})[i]$-th element of `A`. Arrays $(\mathbf{x} \to \mathcal{U})$ and $(\mathbf{x}' \to \mathcal{U})$ depicted in Fig. 7.1 are specific examples of pointer arrays. They represent arrays $\mathbf{x}$ and $\mathbf{x}'$ by storing not $\mathbf{x}_0, \mathbf{x}_1, \ldots$ themselves, but their indices instead. For example, $\mathbf{x}[3] = \mathbf{x}_1$ and thus $(\mathbf{x} \to \mathcal{U})[3] = 1$.

Pointer arithmetic is crucial for our implementation for two reasons. First, it allows one to avoid redundant computation, as illustrated in Fig. 7.2A. One obtains

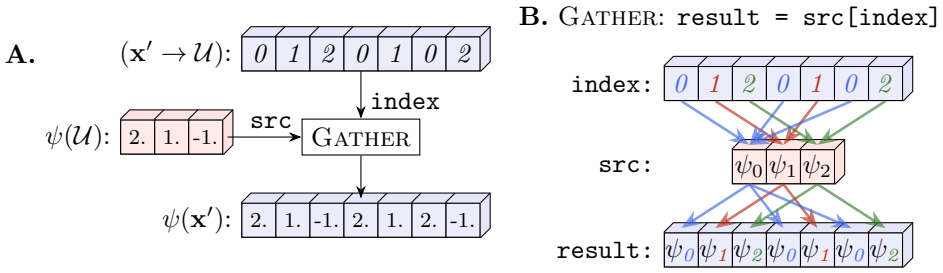


Figure 7.2: A. Efficient evaluation of ansatz amplitudes using GATHER routine. The amplitudes are calculated only for unique basis vectors and then distributed as necessary. **B.** An example GATHER operation.

the amplitudes for each $\mathbf{x}'$ in $\mathcal{SC}$ by reusing the known values of amplitudes for basis vectors in $\mathcal{U}$, rather than evaluating them directly. The substitution is performed with a standard tensor algebra routine GATHER depicted in Fig. 7.2B. As input, this routine takes a `src` array and an array `index` of pointers to `src`. As output, it produces the `result` array defined as `result`$[i]$ = `src`[`index`$[i]$]. We call this process *dereferencing* of `index` with respect to `src`. The same pointer array might be dereferenced with respect to different source arrays. For example, GATHER(`src` : $\psi(\mathcal{U})$, `index` : $(\mathbf{x}' \to \mathcal{U})) = \psi(\mathbf{x}')$, while GATHER(`src` : $\mathcal{U}$, `index` : $(\mathbf{x}' \to \mathcal{U})) = \mathbf{x}'$.

The second advantage of pointer arithmetic is that it enables summation over dynamically defined ranges of indices as illustrated in Fig. 7.3A. Suppose we employed MATRIXELEMENT function to obtain the array $H_{\mathbf{x}\mathbf{x}'}$ of Hamiltonian matrix elements. To obtain $E^{\text{var}}_{\text{loc}}(\mathbf{x})$ one has to perform three following steps:

1. Multiply it elementwise by an array $\psi(\mathbf{x}')$;

2. Divide it elementwise by an array $\psi(\mathbf{x})$;

3. Sum the values $H_{\mathbf{x}\mathbf{x}'}\frac{\psi(\mathbf{x}')}{\psi(\mathbf{x})}$ corresponding to the same unique $\mathbf{x}$.

The summation at the last stage can be performed with another tensor algebra routine known as SCATTERADD. This routine adds the i-th element of an array `src` to the `index`$[i]$-th element of array `result` as depicted in Fig. 7.3B.

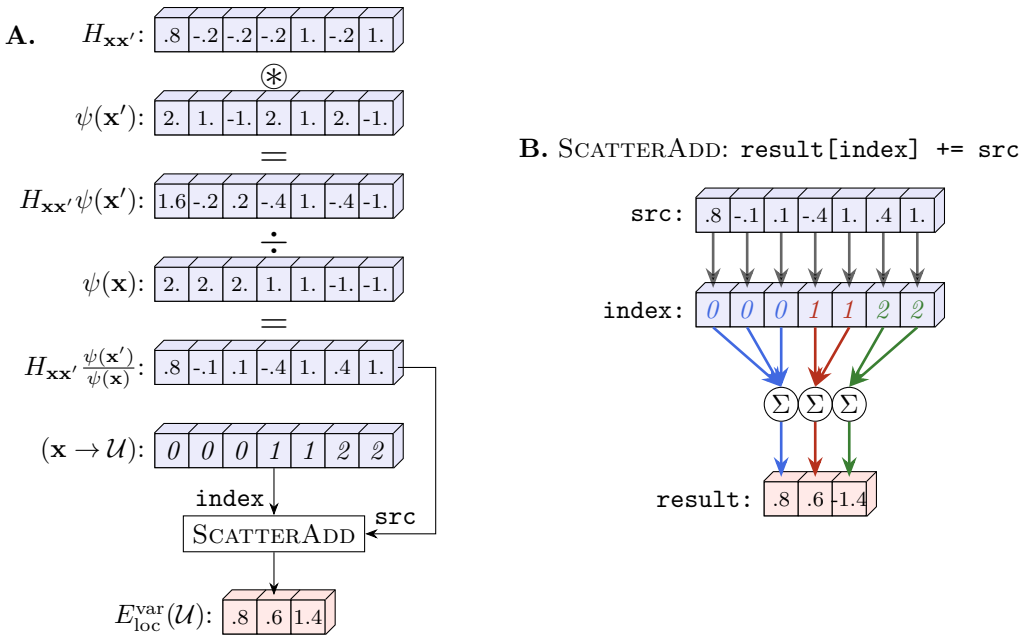


Figure 7.3: A. Efficient evaluation of $E^{\text{var}}_{\text{loc}}$ where the dynamic range summations are performed with ScatterAdd routine. **B.** An example ScatterAdd operation.

7.2.3 Notation remarks

Let us make two remarks regarding the notation and the diagrams representing GPU operations. First, the names of arrays which store pointers of any kind will necessarily include the symbol $\to$. Second, we use different font styling to display the content of arrays of different nature: (i) we depict **in bold** the content of arrays storing values corresponding to N-bit vectors, e.g. $\mathbf{x}$ and $\mathbf{x}'$; (ii) we display *in italics* the content of arrays storing pointers of any kind, e.g. $(\mathbf{x} \to \mathcal{U})$; (iii) we use normal styling for the content of arrays storing genuine floating point or integer values, e.g. $\psi(\mathbf{x})$ or $E^{\text{var}}_{\text{loc}}(\mathbf{x})$.

7.2.4 Preparation

Before the start of each simulation we preprocess the Hamiltonian to represent it as a set of tensors stored on a GPU as depicted in Fig. 7.4. For example, h_l is a contiguous array storing the weights of all Hamiltonian terms, i.e. $h_l[0] = h_0$. The arrays $\underline{\mathbf{XY}}^{(l)}$, $\mathbf{Y}^{(l)}$ and $\underline{\mathbf{YZ}}^{(l)}$ are constructed in a similar way.

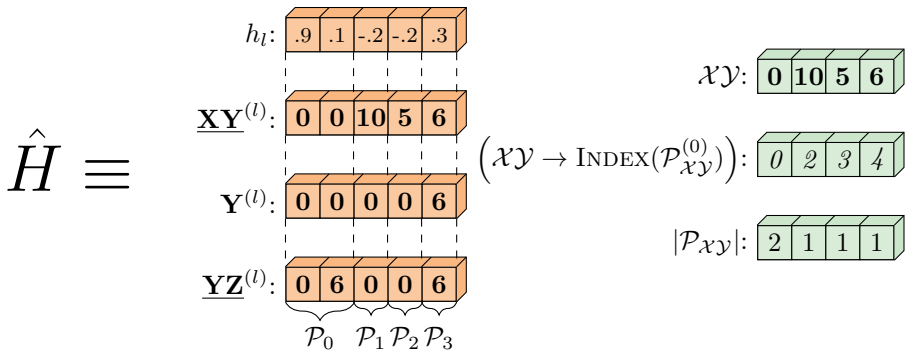


Figure 7.4: Storing the Hamiltonian as a set of tensors.

We also introduce three auxiliary arrays to represent the fact that each unique $\underline{\mathbf{XY}}^{(l)}$ might correspond to several terms constituting the set $\mathcal{P}_{\underline{\mathbf{XY}}}$. In Fig. 7.4 we show how elements of the arrays h_l, $\mathbf{Y}^{(l)}$ and $\underline{\mathbf{YZ}}^{(l)}$ are grouped into the corresponding sets. For example, the zeroth and first elements of these arrays correspond to $\mathcal{P}_{\underline{\mathbf{XY}}_0}$, which we denote for brevity as $\mathcal{P}_0$. The first auxiliary array $\mathcal{XY}$ is an array of unique $\underline{\mathbf{XY}}$ vectors themselves. Second, $\left(\mathcal{XY} \to \textsc{Index}(\mathcal{P}^{(0)}_{\mathcal{XY}})\right)$ stores pointers to the first elements in $\mathcal{P}_{\underline{\mathbf{XY}}}$ for each $\underline{\mathbf{XY}}$ in $\mathcal{XY}$. For example, $\left(\mathcal{XY} \to \textsc{Index}(\mathcal{P}^{(0)}_{\mathcal{XY}})\right)[1] = 2$ since the set $\mathcal{P}_1$ starts from the position 2 in arrays h_l, $\mathbf{Y}^{(l)}$ and $\underline{\mathbf{YZ}}^{(l)}$. Finally, the array $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ stores the number of terms in $\mathcal{P}_{\underline{\mathbf{XY}}}$ for each unique $\underline{\mathbf{XY}}$. These three arrays will prove instrumental for our implementation of MatrixElement discussed further.

7.3 Main functions

In this section we cover two key procedures of ANQS optimisation: LoopOverBatch and MatrixElement. The implementation of LoopOverTerms is similar to that of LoopOverBatch and can be found in our code [92].

7.3.1 LoopOverBatch implementation

We split GPU implementation of LoopOverBatch algorithm into three main stages.

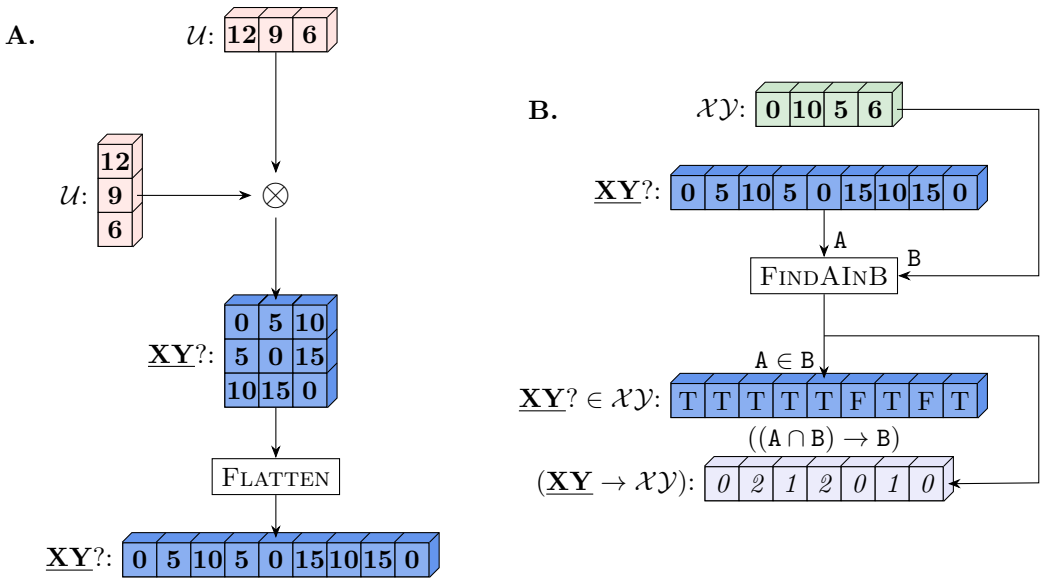


Figure 7.5: Two first stages of LOOPOVERBATCH **A.** One evaluates candidate $\underline{\mathbf{XY}}$ bit vectors in a vectorised manner. **B.** One invokes a custom FINDAINB function to obtain a boolean mask indicating whether a candidate $\underline{\mathbf{XY}}$ belongs to $\mathcal{XY}$. As well, one obtains an array of pointers from "successful" candidate $\underline{\mathbf{XY}}$ to $\mathcal{XY}$.

Stage 1: Find candidate $\underline{\mathbf{XY}}$

The first stage is illustrated in Fig. 7.5A. It calculates in a vectorised way all possible candidate vectors $\underline{\mathbf{XY}} = \mathbf{x} \otimes \mathbf{x}'$, $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{U}$; we denote an array of such candidates as $\underline{\mathbf{XY}}$?. To that end, one applies the bitwise XOR operation to two copies of $\mathcal{U}$, with one reshaped into a column vector. The broadcasting rules of PyTorch ensure that the resulting 2D array $\underline{\mathbf{XY}}$? is a matrix of size $N_{\text{unq}} \times N_{\text{unq}}$. At the end of this stage one flattens $\underline{\mathbf{XY}}$? into a 1D array by concatenating its rows.

Stage 2: Filter candidate $\underline{\mathbf{XY}}$

The second stage is shown in Fig. 7.5B. It invokes a custom function FINDAINB to find which of the candidate $\underline{\mathbf{XY}}$? correspond to bit vectors in $\mathcal{XY}$.

To explain FINDAINB functioning, suppose we feed it two integer arrays, `A` and `B`. We presume that there are no repeated values in the array `B`, while there might be some in `A`. FINDAINB produces two output arrays. The first output array `A` $\in$ `B` is a boolean mask indicating whether the elements of `A` can be found in `B`. In our

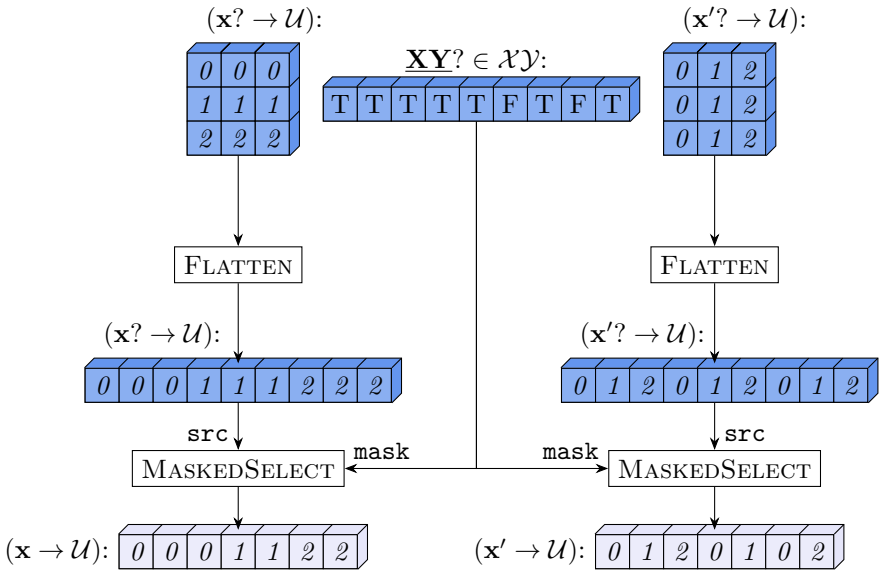


Figure 7.6: The last stage of LOOPOVERBATCH. One obtains $\mathcal{SC}$ by masking off the pointers to uncoupled pairs of $\mathbf{x}$ and $\mathbf{x}'$.

example $(\mathtt{A} \in \mathtt{B})[1] = \text{TRUE}$ since $\underline{\mathbf{XY}}?[1] = \mathbf{5}$ belongs to $\mathcal{XY}$. At the same time $(\mathtt{A} \in \mathtt{B})[5] = \text{FALSE}$ since $\underline{\mathbf{XY}}?[5] = \mathbf{15}$ *does not* belong to $\mathcal{XY}$.

The second output array $((\mathtt{A} \cap \mathtt{B}) \to \mathtt{B})$ contains the pointers from those elements of $\mathtt{A}$ which belong to $\mathtt{B}$ (i.e. $\mathtt{A} \cap \mathtt{B}$) to their corresponding positions in $\mathtt{B}$. Its length is equal to the number of elements in $\mathtt{A}$ which belong to $\mathtt{B}$ and the order between elements is the same as in the initial array $\mathtt{A}$. For example, $(((\mathtt{A} \cap \mathtt{B}) \to \mathtt{B}))[1] = 2$ because the second element in $\underline{\mathbf{XY}}?$ corresponds to the third element in $\mathcal{XY}$.

In what follows we refer to the array $\mathtt{A} \in \mathtt{B}$ as $\underline{\mathbf{XY}}? \in \mathcal{XY}$, and to the array $((\mathtt{A} \cap \mathtt{B}) \to \mathtt{B})$ as $(\underline{\mathbf{XY}} \to \mathcal{XY})$. Since $(\underline{\mathbf{XY}} \to \mathcal{XY})$ corresponds to all correctly coupled $\mathbf{x}$ and $\mathbf{x}'$, its length is equal to the length of $(\mathbf{x} \to \mathcal{U})$ and $(\mathbf{x}' \to \mathcal{U})$ representing $\mathcal{SC}$.

Stage 3: Obtain $\mathcal{SC}$

The third stage is depicted in Fig. 7.6. During this stage one uses the boolean mask $\underline{\mathbf{XY}}? \in \mathcal{XY}$ to obtain $(\mathbf{x} \to \mathcal{U})$ and $(\mathbf{x}' \to \mathcal{U})$. To that end, one forms a 2D array of candidate pointers from $\mathbf{x}$ to $\mathcal{U}$ which we denote as $(\mathbf{x}? \to \mathcal{U})$. The first row of this array contains only zeros since it corresponds to all $\underline{\mathbf{XY}}?$ formed with $\mathbf{x}_0$ being

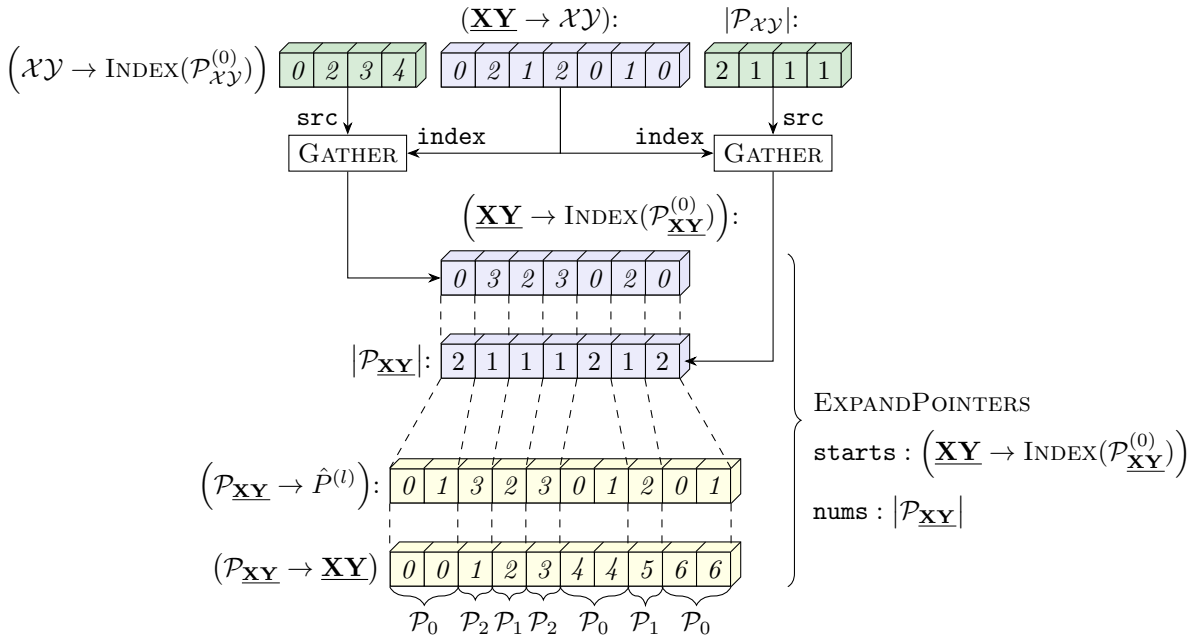


Figure 7.7: The first stage of MATRIXELEMENT. For each coupling $\underline{\mathbf{XY}}$ one substitutes its pointer to $\mathcal{XY}$ with a contiguous sequence of pointers $\hat{P}^{(l)}$ in $\mathcal{P}_{\underline{\mathbf{XY}}}$. To that end, one employs a custom EXPANDPOINTERS function described fin Section 7.5.

the first element in the pair. Correspondingly, the second row contains only ones, the third contains only twos and so on. The 2D array $(\mathbf{x}'? \to \mathcal{U})$ is constructed in a similar way, except for its *columns* consisting of the same values. One flattens $(\mathbf{x}? \to \mathcal{U})$ and $(\mathbf{x}'? \to \mathcal{U})$ and applies the boolean mask $\underline{\mathbf{XY}}? \in \mathcal{XY}$ to filter out only those pointers which correspond to the actually coupled pairs $(\mathbf{x}, \mathbf{x}')$.

7.3.2 MatrixElement implementation

The GPU implementation of MATRIXELEMENT consists of three consecutive stages.

Stage 1: Expand the pointers

The first stage is illustrated in Fig. 7.7. Since each $\underline{\mathbf{XY}}$ might correspond to several terms in Hamiltonian, this stage substitutes every pointer in $(\underline{\mathbf{XY}} \to \mathcal{XY})$ with a sequence of pointers to $\underline{\mathbf{YZ}}$ corresponding to the given $\underline{\mathbf{XY}}$. Let us provide an example. The first element of $(\underline{\mathbf{XY}} \to \mathcal{XY})$ is *0* and thus it points to the first element of $\mathcal{XY}$. This element $\mathcal{XY}[0]$ corresponds to $\underline{\mathbf{XY}} = \mathbf{0}$ which is represented

with Hamiltonian terms $\hat{P}_0$ and $\hat{P}_1$. Thus, $((\underline{\mathbf{XY}} \to \mathcal{XY}))[0]$ should be expanded to pointers *0* and *1*. We store these pointers contiguously in a larger array $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \hat{P}^{(l)}\right)$. In addition, we keep the reverse pointers from each element in $\mathcal{P}_{\underline{\mathbf{XY}}}$ to its parent $\underline{\mathbf{XY}}$ in an array $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \underline{\mathbf{XY}}\right)$ of the same size as $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \hat{P}^{(l)}\right)$.

In more details this stage proceeds as follows.

1. First, one uses the GATHER operation to obtain arrays $\left(\underline{\mathbf{XY}} \to \textsc{Index}(\mathcal{P}^{(0)}_{\underline{\mathbf{XY}}})\right)$ and $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ by dereferencing the pointers $(\underline{\mathbf{XY}} \to \mathcal{XY})$ with respect to the arrays $\left(\mathcal{XY} \to \textsc{Index}(\mathcal{P}^{(0)}_{\mathcal{XY}})\right)$ and $|\mathcal{P}_{\mathcal{XY}}|$. Thus, each element of $\left(\underline{\mathbf{XY}} \to \textsc{Index}(\mathcal{P}^{(0)}_{\underline{\mathbf{XY}}})\right)$ stores a pointer to the first Hamiltonian term in the set $\mathcal{P}_{\underline{\mathbf{XY}}}$. Similarly, the array $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ stores the *sizes* of the corresponding $\mathcal{P}_{\underline{\mathbf{XY}}}$ sets.

2. As a second step, one expands pointers to the first $\mathcal{P}_{\underline{\mathbf{XY}}}$ elements (which we further refer to as *starts*) and the sizes of $\mathcal{P}_{\underline{\mathbf{XY}}}$ into a contiguous array containing pointers to *all* elements in $\mathcal{P}_{\underline{\mathbf{XY}}}$. This is achieved with a custom function EXPANDPOINTERS, which is explained in more details in Section 7.5. The resulting pointers are stored in the array $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \hat{P}^{(l)}\right)$

3. Finally, one obtains the *reverse* pointers $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \underline{\mathbf{XY}}\right)$ by repeating the *index* of each $\underline{\mathbf{XY}}$ element $\left|\mathcal{P}_{\underline{\mathbf{XY}}}\right|$ times as illustrated in the last line of Fig. 7.7. This can be easily achieved with the standard REPEATINTERLEAVE routine of the PyTorch software library.

Stage 2: Calculate term expectation values

The second stage of MATRIXELEMENT calculation is shown in Fig. 7.8. First, one dereferences the pointers $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \hat{P}^{(l)}\right)$ with respect to the array $\underline{\mathbf{YZ}}^{(l)}$ to obtain an array of $\underline{\mathbf{YZ}}$ corresponding to each element in $\mathcal{P}_{\underline{\mathbf{XY}}}$. Second, one dereferences the pointers $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \underline{\mathbf{XY}}\right)$ with respect to the array $\mathbf{x}'$. Thus, one obtains an array $\mathbf{x}'(\mathcal{P}_{\underline{\mathbf{XY}}})$ which contains all $\mathbf{x}'$ required to evaluate $\langle \mathbf{x}|\hat{P}|\mathbf{x}'\rangle$. Finally, one feeds both $\underline{\mathbf{YZ}}$ and $\mathbf{x}'(\mathcal{P}_{\underline{\mathbf{XY}}})$ into a simple arithmetic function calculating $\mathrm{e}^{\mathrm{i}\pi|\mathbf{x}'\odot\underline{\mathbf{YZ}}|}$ elementwise. As a result, one obtains the array containing the values of $\langle \mathbf{x}|\hat{P}|\mathbf{x}'\rangle$.

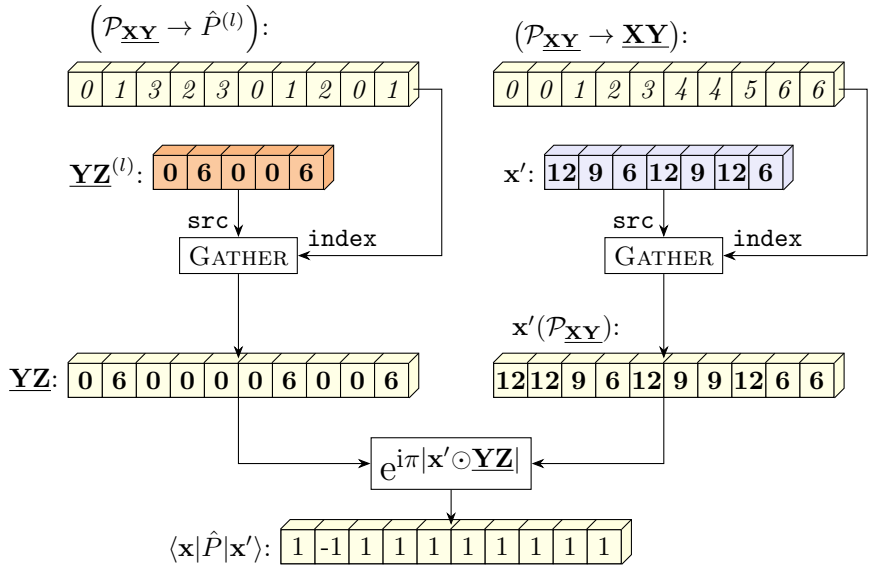


Figure 7.8: The second stage of MatrixElement. One employs the pointers to $\hat{P}^{(l)}$ to evaluate the matrix elements $\langle\mathbf{x}|\hat{P}|\mathbf{x}'\rangle$ for all $\hat{P}$ coupling each pair $(\mathbf{x}, \mathbf{x}')$.

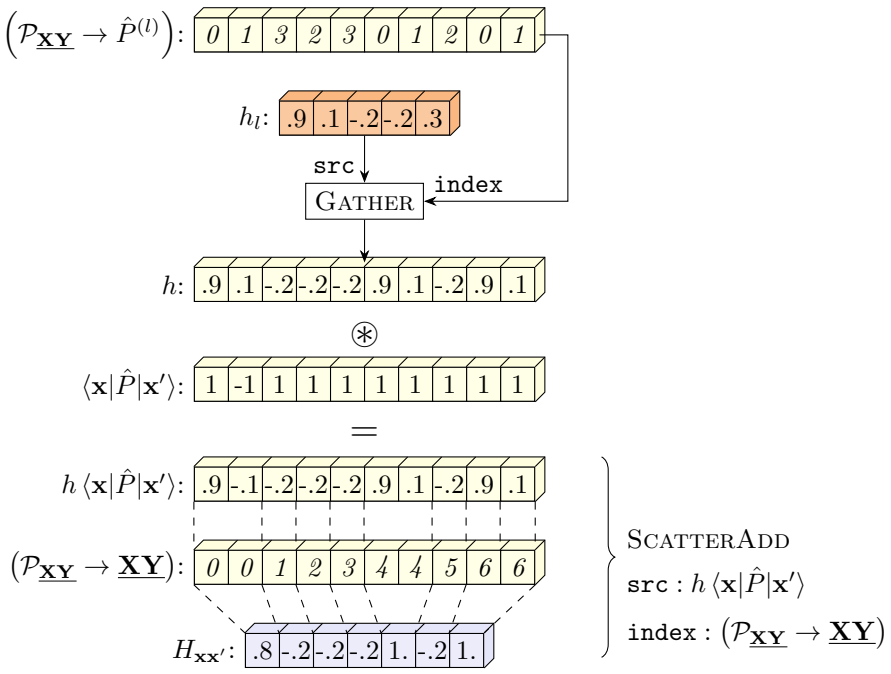


Figure 7.9: The third stage of MatrixElement. One employs the ScatterAdd function to sum all $h\,\langle\mathbf{x}|\hat{P}|\mathbf{x}'\rangle$ corresponding to each pair $(\mathbf{x}, \mathbf{x}')$ into $H_{\mathbf{xx}'}$.

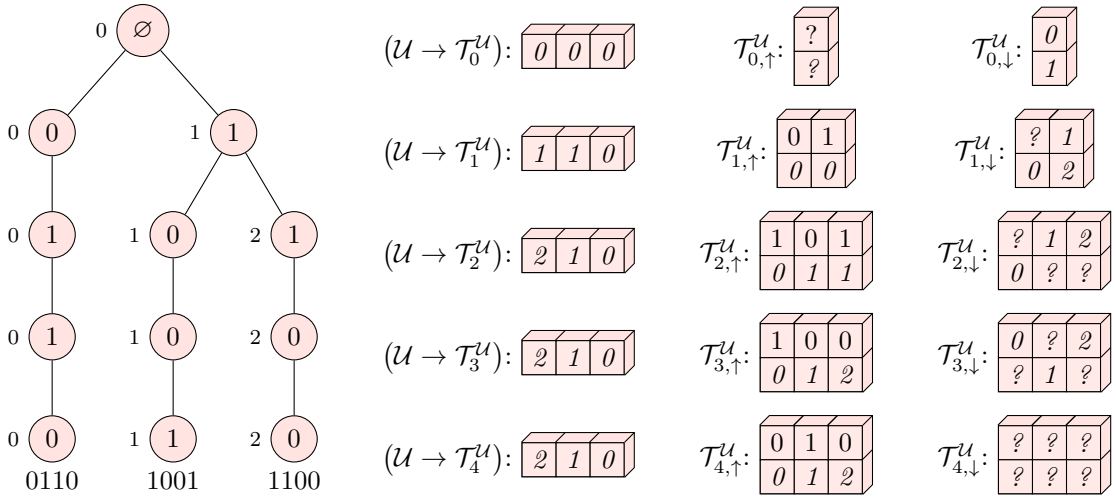


Figure 7.10: The prefix tree data structures for the batch of unique samples $\mathcal{U}$.

Stage 3: Scatter term expectation values

The third stage is depicted in Fig. 7.9. First, one dereferences the pointers $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \hat{P}^{(l)}\right)$ with respect to the array h_l. Thus, one fetches the weights of Hamiltonian terms for corresponding $\langle \mathbf{x}|\hat{P}|\mathbf{x}'\rangle$. Second, one multiplies h and $\langle \mathbf{x}|\hat{P}|\mathbf{x}'\rangle$ elementwise to obtain an array $h_l \langle \mathbf{x}|\hat{P}|\mathbf{x}'\rangle$. Finally, one employs the ScatterAdd routine to sum the values of $h_l \langle \mathbf{x}|\hat{P}|\mathbf{x}'\rangle$ corresponding to the same $\underline{\mathbf{XY}}$. During ScatterAdd the array $\left(\mathcal{P}_{\underline{\mathbf{XY}}} \to \underline{\mathbf{XY}}\right)$ serves as the `index` input argument.

7.4 Prefix tree operations

7.4.1 Prefix tree data structures

In our GPU implementation we store the prefix tree nodes as contiguous arrays, as opposed to storing them as individual data structures introduced in Section 2B. As an example, let us consider the batch of unique samples $\mathcal{U}$ and its prefix tree $\mathcal{T}^{\mathcal{U}}$. For the i-th level of the prefix tree we store *three* arrays denoted as $\left(\mathcal{U} \to \mathcal{T}_i^{\mathcal{U}}\right)$, $\mathcal{T}_{i,\uparrow}^{\mathcal{U}}$ and $\mathcal{T}_{i,\downarrow}^{\mathcal{U}}$ as depicted in Fig. 7.10.

1. The array $\left(\mathcal{U} \to \mathcal{T}_i^{\mathcal{U}}\right)$ has a shape $[N_{\text{unq}}, 1]$. Its k-th element points to the node at the i-th level of the prefix tree that corresponds to the k-th basis

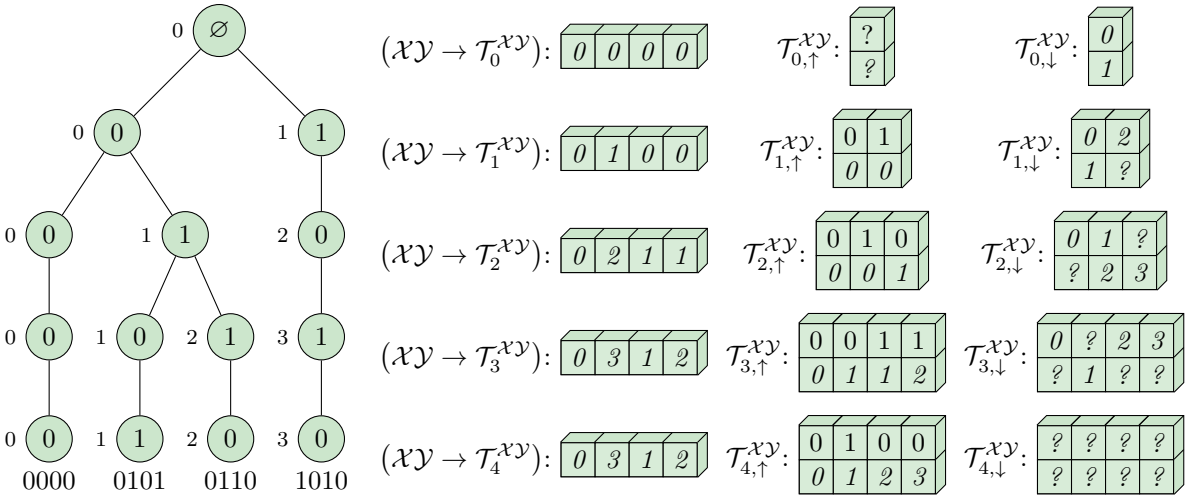


Figure 7.11: The prefix tree data structures for the set $\mathcal{XY}$ of unique **XY** strings representing Hamiltonian terms.

vector in $\mathcal{U}$. For example, at the level 2 of the prefix tree $\left(\mathcal{U} \to \mathcal{T}_2^{\mathcal{U}}\right)[0] = 2$ since the first two symbols of $\mathbf{x}_0$ are 11, and this prefix path leads to the node 2 of level 2. At the same time, $\left(\mathcal{U} \to \mathcal{T}_0^{\mathcal{U}}\right)[k] = 0\ \forall k$ since *all* vectors in $\mathcal{U}$ start from an empty string $\varnothing$.

2. The array $\mathcal{T}_{i,\uparrow}^{\mathcal{U}}$ has a shape $\left[\left|\mathcal{T}_i^{\mathcal{U}}\right|, 2\right]$, where $\left|\mathcal{T}_i^{\mathcal{U}}\right|$ is the number of nodes at the i-th prefix tree level. It stores in a contiguous way the fields `value` and `parent` of the NODE data structure introduced in the Chapter 6. Specifically, $\mathcal{T}_{i,\uparrow}^{\mathcal{U}}[l, 0]$ stores x_i, while $\mathcal{T}_{i,\uparrow}^{\mathcal{U}}[l, 1]$ keeps the pointer to the parent node at the previous level. For example, the bit value of node 2 at level 2 is equal to 1, and thus $\mathcal{T}_{2,\uparrow}^{\mathcal{U}}[2, 0] = 1$. At the same time, its parent is node 1 at level 1, and thus $\mathcal{T}_{2,\uparrow}^{\mathcal{U}}[2, 1] = 1$. Let us note that $\mathcal{T}_{0,\uparrow}^{\mathcal{U}}$ always has a shape $[1, 2]$ and we fill it with NANs, since this node does not have a definite bit value and does not have a parent node.

3. The array $\mathcal{T}_{i,\downarrow}^{\mathcal{U}}$ has a shape $\left[\left|\mathcal{T}_i^{\mathcal{U}}\right|, 2\right]$ too. It stores in a contiguous way the `children` field of the NODE data structure. In particular, $\mathcal{T}_{i,\downarrow}^{\mathcal{U}}[l, 0]$ stores the pointer to a child with the value of 0 of the node l . Similarly, $\mathcal{T}_{i,\downarrow}^{\mathcal{U}}[l, 1]$ stores the pointer to a child with the value 1. In our diagrams these children nodes

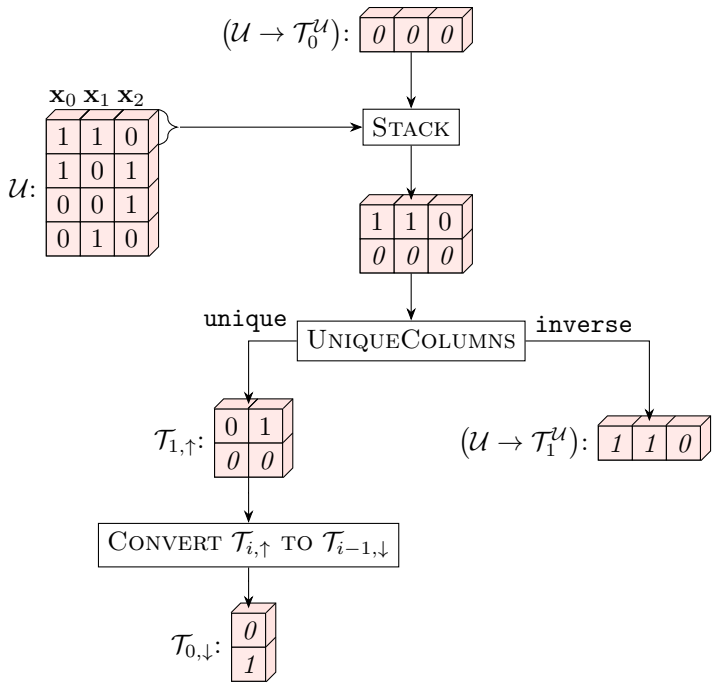


Figure 7.12: An iteration of the ConstructPrefixTree algorithm.

are to the left and to the right of the parent node correspondingly. If a node does not have a child, we store NaN instead of a pointer. For example, the node 0 at level 2 does not have a child with the value 0, and thus $\mathcal{T}_{i,\downarrow}^{\mathcal{U}}[2,0] = ?$. At the same time, the same node *does* have a child with the bit value 1, which is node 0 at level 3. Thus, $\mathcal{T}_{i,\downarrow}^{\mathcal{U}}[2,0] = \mathit{0}$. At the last prefix tree level nodes do not have children and thus $\mathcal{T}_{N,\downarrow}^{\mathcal{U}}$ is a redundant array filled with NaNs.

For the sake of completeness, in Fig. 7.11 we provide the prefix tree data structures for the set $\mathcal{XY}$.

7.4.2 ConstructPrefixTree implementation

The idea of ConstructPrefixTree is to obtain the relevant prefix tree structures level by level starting from $\left(\mathcal{U} \to \mathcal{T}_0^{\mathcal{U}}\right)$ and $\mathcal{T}_{0,\uparrow}^{\mathcal{U}}$. We illustrate how the arrays of level 1 can be obtained from the arrays of level 0 in Fig. 7.12. More generally, the same procedure is applicable to obtain the arrays of level i from the arrays of level $i-1$.

1. First, one stacks the array containing i-th bits of all vectors in $\mathcal{U}$ on top of the array $\left(\mathcal{U} \to \mathcal{T}^{\mathcal{U}}_{i-1}\right)$ to form an array of shape $[N_{\text{unq}}, 2]$. This array essentially contains all *non-unique* prefixes at i-th level: the first row stores node values, while the second row contains pointers to the parents.

2. Two non-unique prefixes that have the same bit value and parent are considered identical and should be merged into a single prefix. Thus, one feeds non-unique prefixes into a UniqueColumns function which produces two output arrays. The first output is a 2D array of shape $\left[\left|\mathcal{T}^{\mathcal{U}}_i\right|, 2\right]$ containing only unique prefixes; by inspection it can be seen that this array is equivalent $\mathcal{T}^{\mathcal{U}}_{i,\uparrow}$. The second output is an array containing pointers from non-unique prefixes to the unique ones. Since each non-unique prefix is associated with a basis vector in $\mathcal{U}$, this array represents $\left(\mathcal{U} \to \mathcal{T}^{\mathcal{U}}_i\right)$. The UniqueColumns is our custom lower-level function and we cover it in Section 7.5.

3. Finally, one uses the obtained arrays to fill the array $\mathcal{T}^{\mathcal{U}}_{i-1,\downarrow}$ at the *previous* level. First, one creates an array filled with NaNs of the same shape as $\mathcal{T}^{\mathcal{U}}_{i-1,\uparrow}$. Then, one scatters into it the values from array $\mathcal{T}^{\mathcal{U}}_{i,\downarrow}$ according to the following rule:

$$\mathcal{T}^{\mathcal{U}}_{i-1,\downarrow}[\mathcal{T}^{\mathcal{U}}_{i,\uparrow}[l,1], \mathcal{T}^{\mathcal{U}}_{i,\uparrow}[l,0]] = l \;\forall l \in 1..\left|\mathcal{T}^{\mathcal{U}}_i\right|. \tag{7.7}$$

   In other words, for each node in $\mathcal{T}^{\mathcal{U}}_i$ we find its parent node $\mathcal{T}^{\mathcal{U}}_{i,\uparrow}[l,1]$ and indicate that this node has a child l having the bit value $\mathcal{T}^{\mathcal{U}}_{i,\uparrow}[l,0]$

7.4.3 LoopOverTrie implementation

The LoopOverTrie algorithm described in Chapter 6 builds coupled pairs $(\mathbf{x}, \mathbf{x}')$ bit by bit. For a given $\mathbf{x}$ and prefix tree level i it calculates all acceptable prefixes $\mathbf{x}'_{<i}$ and the corresponding coupling $\underline{\mathbf{XY}}_{<i}$. The idea of our GPU implementation is to store the *pointers* to valid $\mathbf{x}'_{<i}$ and $\underline{\mathbf{XY}}_{<i}$ in an array for each $\mathbf{x}$ and update them level by level. In addition, we concatenate all such pointer arrays to vectorise the algorithm across all $\mathbf{x}$ in $\mathcal{U}$.

Specifically, for each prefix tree level we obtain three arrays shown in Fig. 7.13:

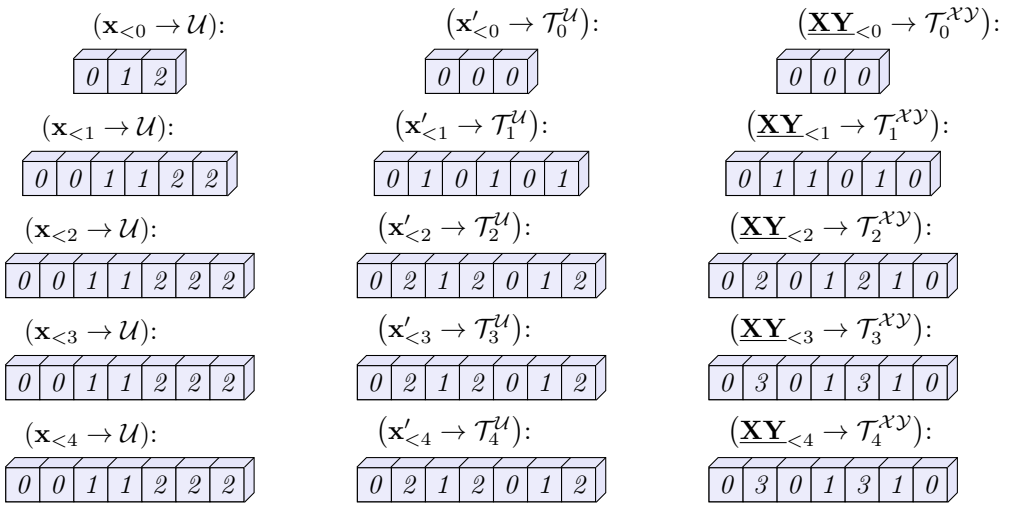


Figure 7.13: Arrays obtained during one full run of LoopOverTrie.

1. The array $(\mathbf{x}_{<i} \to \mathcal{U})$ serves the vectorisation purpose and indicates chunks of other arrays which correspond to different $\mathbf{x}$ being input to ConstructPrefixTree. For example, at the level 0 we start with empty prefixes $\varnothing$ corresponding to every vector in $\mathcal{U}$ and thus $(\mathbf{x}_{<0} \to \mathcal{U}) = [0, 1, 2]$.

2. The array $\left(\mathbf{x}'_{<i} \to \mathcal{T}_i^{\mathcal{U}}\right)$ stores pointers to valid $\mathbf{x}'_{<i}$ prefixes for every $\mathbf{x}$ in $\mathcal{U}$. For example, at level 0 any vector in $\mathcal{U}$ can be coupled only to a single prefix $\varnothing$, and thus $\left(\mathbf{x}'_{<0} \to \mathcal{T}_0^{\mathcal{U}}\right) = [0, 0, 0]$.

3. The array $\left(\underline{\mathbf{XY}}_{<i} \to \mathcal{T}_i^{\mathcal{XY}}\right)$ stores pointers to valid $\underline{\mathbf{XY}}_{<i}$ prefixes in a way similar to $\left(\mathbf{x}'_{<i} \to \mathcal{T}_i^{\mathcal{U}}\right)$.

A single iteration of the algorithm which produces level 2 arrays given the arrays at level 1 is shown in Fig. 7.14. More generally, obtaining arrays for level i from the arrays of level $i-1$ involves the following steps.

1. First, we substitute every pointer to a valid prefix $\mathbf{x}'_{<i-1}$ ($\underline{\mathbf{XY}}_{<i-1}$) with the pointers to its possible children (including NaNs for non-existing children). This produces the arrays of candidate pointers $\left(\mathbf{x}'_{<i}? \to \mathcal{T}_i^{\mathcal{U}}\right)$ and $\left(\underline{\mathbf{XY}}_{<i}? \to \mathcal{T}_i^{\mathcal{XY}}\right)$ twice the length of the initial arrays. Specifically, entries of

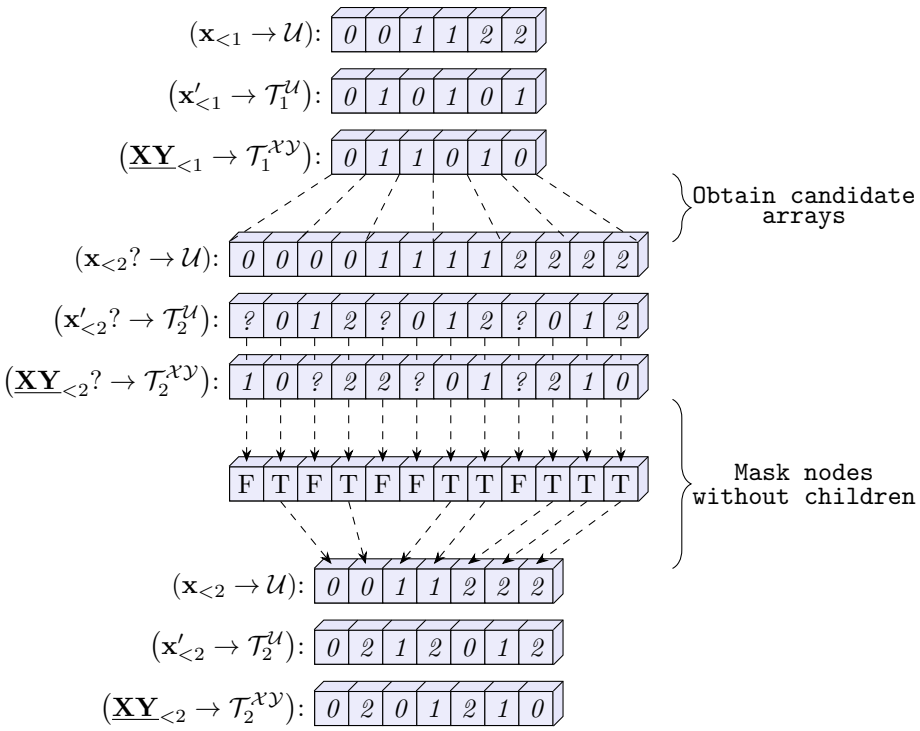


Figure 7.14: An iteration of LoopOverTrie GPU implementation.

the array $\left(\mathbf{x}'_{<i}? \to \mathcal{T}_i^{\mathcal{U}}\right)$ are as follows:

$$\begin{aligned} \left(\mathbf{x}'_{<i} \to \mathcal{T}_i^{\mathcal{U}}\right)[2k] &= \mathcal{T}_{i-1,\downarrow}^{\mathcal{U}}[k, 0]; \\ \left(\mathbf{x}'_{<i} \to \mathcal{T}_i^{\mathcal{U}}\right)[2k+1] &= \mathcal{T}_{i-1,\downarrow}^{\mathcal{U}}[k, 1]; \end{aligned} \tag{7.8}$$

In other words, the k-th valid prefix pointer in the array $\left(\mathbf{x}'_{<i-1} \to \mathcal{T}_{i-1}^{\mathcal{U}}\right)$ is substituted with the pointers to its left and right children at the positions $2k$ and $2k+1$ respectively. The array $\left(\underline{\mathbf{XY}}_{<i}? \to \mathcal{T}_i^{\mathcal{XY}}\right)$ is obtained in a similar way, however, with a subtlety: for a given $\underline{\mathbf{XY}}_{<i-1}$ the order of its children in the array $\left(\underline{\mathbf{XY}}_{<i}? \to \mathcal{T}_i^{\mathcal{XY}}\right)$ depends on the i-th bits of *both* $\mathbf{x}$ and $\mathbf{x}'$. For example, $\left(\mathbf{x}'_{<i} \to \mathcal{T}_i^{\mathcal{U}}\right)[2k]$ corresponds to moving to the *left* child of $\mathbf{x}'_{<i-1}$, i.e. appending 0 to it. However, if $x_i = 1$, then $XY_i = x_i \otimes 0 = 1$, and thus $\left(\underline{\mathbf{XY}}_{<i}? \to \mathcal{T}_i^{\mathcal{XY}}\right)[2k]$ should contain the pointer to the *right* child of $\left(\underline{\mathbf{XY}}_{<i-1} \to \mathcal{T}_{i-1}^{\mathcal{XY}}\right)[k]$. This is expressed with the following equations:

$$\begin{aligned} \left(\underline{\mathbf{XY}}_{<i} \to \mathcal{T}_i^{\mathcal{XY}}\right)[2k] &= \mathcal{T}_{i-1,\downarrow}^{\mathcal{XY}}[k, 0 \otimes \mathbf{x}_k[i]]; \\ \left(\underline{\mathbf{XY}}_{<i} \to \mathcal{T}_i^{\mathcal{XY}}\right)[2k+1] &= \mathcal{T}_{i-1,\downarrow}^{\mathcal{XY}}[k, 1 \otimes \mathbf{x}_k[i]]; \end{aligned} \tag{7.9}$$

Finally, to keep track of the vectorised $\mathbf{x}$, we double each element in $(\mathbf{x}_{<i-1} \to \mathcal{U})$:

$$(\mathbf{x}_{<i} \to \mathcal{U})\,[2k] = (\mathbf{x}_{<i} \to \mathcal{U})\,[2k+1] = (\mathbf{x}_{<i-1} \to \mathcal{U})\,[k] \tag{7.10}$$

The new arrays can be obtained using a combination of Tile, Gather and Reshape PyTorch primitives and we refer the reader to our code for more details [92].

2. Once the arrays are produced, we calculate a boolean mask which is equal to True only if child pointers are not NaN for *both* $\left(\mathbf{x}'_{<i} \to \mathcal{T}_i^{\mathcal{U}}\right)$ and $\left(\underline{\mathbf{XY}}_{<i} \to \mathcal{T}_i^{\mathcal{XY}}\right)$.

3. Finally, we use this boolean mask to obtain arrays of valid prefixes for the current level i.

The array $(\mathbf{x}_{<N} \to \mathcal{U})$ obtained at the last level of LoopOverTrie is clearly equivalent to the array $(\mathbf{x}' \to \mathcal{U})$ produced by FindCoupledPairs procedure. The arrays $(\mathbf{x}' \to \mathcal{U})$ and $(\underline{\mathbf{XY}} \to \mathcal{XY})$ can be obtained from $\left(\mathbf{x}'_{<N} \to \mathcal{T}_N^{\mathcal{U}}\right)$ and $\left(\underline{\mathbf{XY}}_{<N} \to \mathcal{T}_N^{\mathcal{XY}}\right)$ by using the correspondence between the indices of the nodes at the last level of a prefix tree, and their indices in the initial bit vector set. As described above, this correspondence is expressed with the arrays $\left(\mathcal{U} \to \mathcal{T}_N^{\mathcal{U}}\right)$ and $\left(\mathcal{XY} \to \mathcal{T}_N^{\mathcal{XY}}\right)$ obtained during ConstructPrefixTree.

7.5 Auxiliary functions

7.5.1 FindAInB implementation

As mentioned in Section 7.3.1, FindAInB is a function which takes two arrays `A` and `B` as input and produces two arrays $\texttt{A} \in \texttt{B}$ and $((\texttt{A} \cap \texttt{B}) \to \texttt{B})$ as output. Crucially, the array `B` should not contain any repeated values. We split its implementation into three stages.

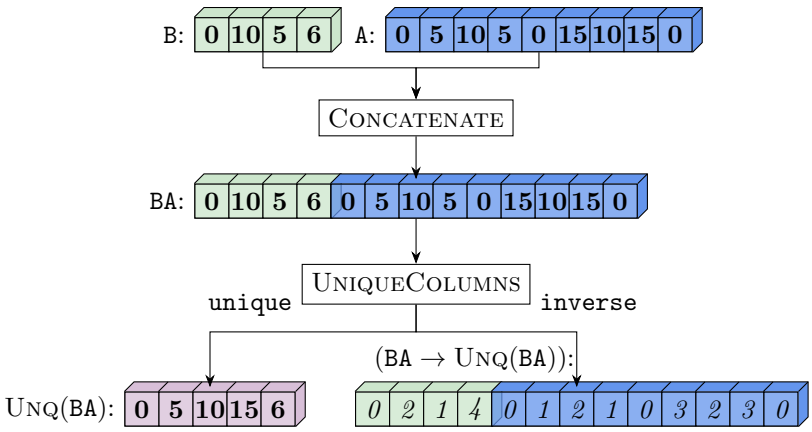


Figure 7.15: The first stage of FindAInB. One concatenates arrays `A` and `B` to find unique elements in their union. As well, one obtains the positions of those unique elements in the initial arrays.

Stage 1: Find unique elements in `A` and `B` union

We concatenate the arrays `A` and `B` into a single array and apply our custom function UniqueColumns to the concatenated array. We use UniqueColumns instead of default PyTorch Unique function because, in general, when several integers are used per $\mathbf{x}$, the arrays `A` and `B` will be two-dimensional, with integer values for each $\mathbf{x}$ stored in columns. Since the default PyTorch Unique function works only with 1D arrays, we have to employ our custom UniqueColumns function explained further in this section.

As depicted in Fig. 7.15, UniqueColumns returns two arrays. The first array Unq(`BA`) contains unique elements found in the union of `A` and `B` arrays. The second array (`BA` → Unq(`BA`)) can be considered as a concatenation of two arrays (`B` → Unq(`BA`)) and (`A` → Unq(`BA`)), each containing pointers from elements in `B` and `A` to the found unique values Unq(`BA`).

Stage 2: Finding the reverse pointers to `B`

The purpose of this stage is to find the reverse pointers from Unq(`BA`) to `B`. To that end, we create an array (Unq(`BA`) → `B`) of the same shape as Unq(`BA`) filled with NaNs. Then, we Scatter into this array *indices* of elements in `B`. For example, the second element in the (`B` → Unq(`BA`)) array is *2*, which means that `B`[1] is the

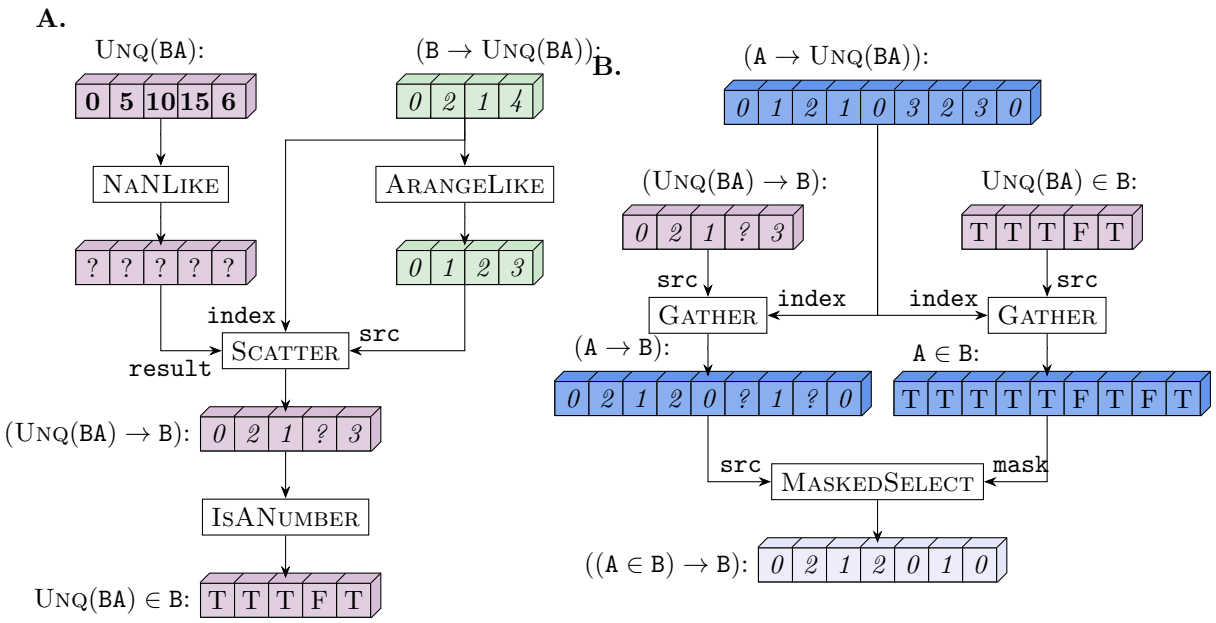


Figure 7.16: The two last stages of FindAInB. **A.** Having the pointers from B to Unq(BA), one employs Scatter to find the *reverse* pointers from Unq(BA) to B. As well, one obtains a boolean mask indicating whether an element in Unq(BA) can be found in B. B. One uses Gather to fetch the pointers from Unq(BA) to B for every element in A and thus obtains the pointers (A → B). In addition, one obtains a boolean mask indicating whether an element in A can be found in B.

third element of Unq(BA). Thus, we scatter *2* to the third position in the array (Unq(BA) → B) as depicted in Fig. 7.16. If, after this operation, some values are still equal to NaN, it means that there are values in Unq(BA) that can be found in A but not in B. Hence, one can apply a standard IsANumber function to (Unq(BA) → B) and obtain a boolean mask indicating which elements of Unq(BA) belong to B.

Stage 3: Finding pointers from A to B

At the final stage we Gather pointers (Unq(BA) → B) using the array (A → Unq(BA)) as `index`. As shown in Fig. 7.16B, we thus obtain an array (A → B) with pointers from A to B (if some element in (A → Unq(BA)) is NaN, it means that the corresponding element in A does not belong to B). In a similar way, we can Gather boolean mask Unq(BA) ∈ B and obtain A ∈ B. Finally, we apply the boolean mask A ∈ B to (A → B) to keep only the valid pointers to B, as required by the use of FindAInB in FindCoupledPairs.

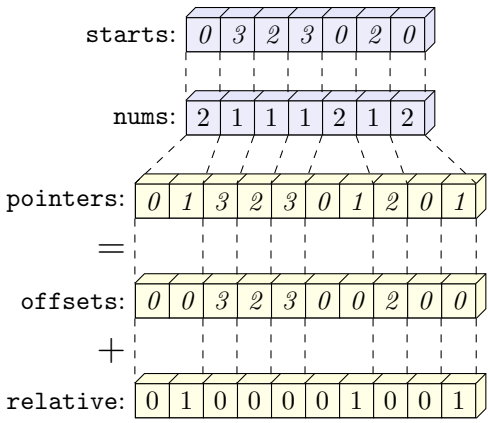


Figure 7.17: An example of the ExpandPointers routine. A `starts`$[i]$ pointer is substituted with a set of `nums`$[i]$ consecutive (i.e `starts`$[i], \texttt{starts}[i]+1, \ldots, \texttt{starts}[i]+\texttt{nums}[i]-1$). For the sake of simplicity, the final array of pointers is represented as a sum of two arrays `offsets` and `saw`.

Computational complexity

The dominating cost of FindAInB is finding unique columns with UniqueColumns. As discussed further in Section 7.5.3 it is of complexity $\mathcal{O}\left((|\texttt{A}| + |\texttt{B}|)\log(|\texttt{A}| + |\texttt{B}|)\right)$, which matches the figures given in Section 6.2.2.

7.5.2 ExpandPointers implementation

The ExpandPointers procedure serves the following purpose. Suppose one has an array `memory` whose elements are grouped into contiguous chunks of various length. This is akin to such Hamiltonian data arrays as h_l, $\mathbf{Y}^{(l)}$ and $\underline{\mathbf{YZ}}^{(l)}$ which can be grouped according to their $\underline{\mathbf{XY}}$. ExpandPointers is a function which allows one to retrieve those contiguous chunks of `memory` by specifying their starting positions `starts` and their lengths `nums`. As output ExpandPointers produces an array `pointers` which replaces each starting pointer `starts`$[i]$ with a *sequence* of `nums`$[i]$ consecutive pointers to each element in the corresponding contiguous chunk. An example of the ExpandPointers action is given in Fig. 7.17. We split its implementation into the following steps:

1. We represent `pointers` as a sum of two arrays: `offsets` and `relative`. For each *expanded* pointer in `pointers`, the `offsets` array contains the starting

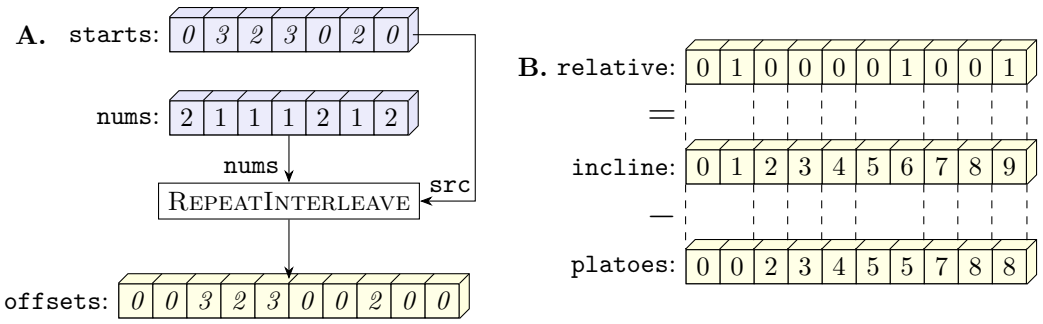


Figure 7.18: A. Calculation of `starts` array. **B.** Decomposition of `saw` array into a sum of `incline` and `platoes` arrays.

position of the chunk to which the expanded pointer belongs to. As illustrated in Fig. 7.18A it is trivially obtained with the standard PyTorch routine RepeatInterleave: each element of `starts`$[i]$ array is repeated `nums`$[i]$ times. The `relative` array contains the relative position of each extended pointer with respect to the chunk start. Obtaining `relative` is slightly more involved and is performed in the next steps.

2. To calculate `relative` we represent it as a difference of two arrays: `incline` and `platoes`. The `incline` array is an arithmetic progression of integers with the step 1 starting from 0. This progression represents the fact that the neighbouring expanded pointers belonging to the same chunk should differ by one. This array can be easily obtained using the standard Cumsum PyTorch function as shown in Fig. 7.19A.

3. However, `incline` does not account for the fact that this progression breaks down at the chunk borders, where increments should start from scratch. The array `platoes` corrects for it by counting the number of increments that are accumulated by the start of each chunk and belong to the *previous* chunks. The calculation of `platoes` is shown in Fig. 7.19B.

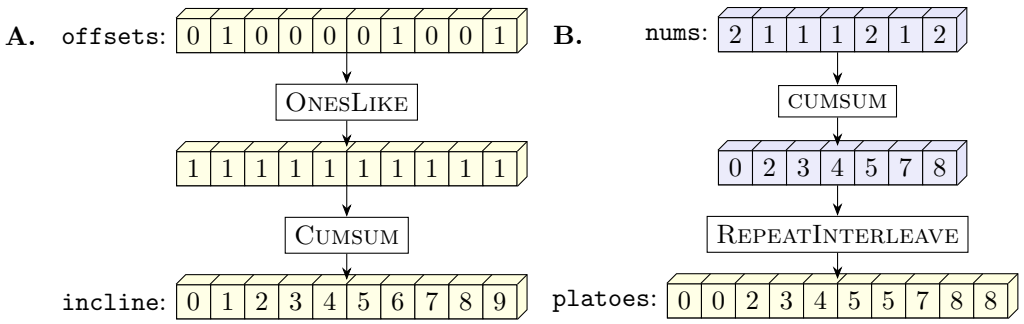


Figure 7.19: A. Calculation of `incline` array. **B.** Calculation of `platoes` array.

7.5.3 UniqueColumns implementation

As discussed, both ConstructPrefixTree and FindAInB require finding unique elements in an array, where each element is represented with more than one integer value. Since the standard PyTorch Unique routine operates only with 1D arrays, we employ a custom function UniqueColumns to find unique elements of a 2D array, where the integers representing each value are stored along the second dimension (i.e. in columns). Our implementation is based on the following idea: we devise an algorithm that assigns an integer label to each column, so that identical columns have identical labels. Then, we apply the conventional PyTorch Unique function to these labels and find unique among them. Finally, we map the unique labels back to columns. More specifically, let us consider an example of columns of size 2; its generalisation to larger column sizes is straightforward and can be found in our code.

1. Suppose we are given an array `array` of shape $[L, 2]$. Let `row1` $\coloneqq$ `array`$[:, 0]$ and `row2` $\coloneqq$ `array`$[:, 1]$ be its first and second row correspondingly.

2. We apply the PyTorch Unique function to `row1` and obtain an array of its unique values `unq_row1` $\coloneqq$ Unique(`row1`). We also obtain an array of pointers from `row1` to `unq_row1`, which we refer to as `unq_row1_inv` $\coloneqq$ (`row1` $\to$ `unq_row1`). We apply the PyTorch Unique function to `row2` to obtain the analogously defined `unq_row2` and `unq_row2_inv`.

3. We note that there are |unq_row1|·|unq_row2| possible pairs of unique elements in the first and second rows, where |·| denotes the array length. We assign to a pair (unq_row1[i], unq_row2[j]) an integer label |unq_row2| · $i + j$.

4. The arrays unq_row1_inv and unq_row2_inv store indices i and j of the corresponding unique elements. Thus, for non-unique columns of array we calculate the array of labels as follows: labels ≔ |unq_row2|·unq_row1_inv+ unq_row2_inv.

5. We employ the PyTorch Unique function to obtain unique labels unq_labels, as well as the array unq_labels_inv of pointers from labels to unq_labels. Since there is a one-to-one correspondence between labels and array columns, unq_labels_inv is actually an array of pointers from array to its unique columns: unq_labels_inv ≡ (array → UniqueColumns(array)).

6. Finally, we obtain the unique columns themselves. To that end, for each unique label we calculate the indices i and j introduced above. This is achieved as follows: i ≔ unq_labels // |unq_row2|; j ≔ unq_labels mod |unq_row2|, where // denotes integer division. Having i and j for each unique column, it is straightforward to obtain the columns by applying the Gather operation to the arrays unq_row1 and unq_row2 and stacking the results.

Computational complexity

The dominating cost of UniqueColumns corresponds to the repeated calls of the Unique PyTorch routine, which finds the unique elements of an array by sorting it with $\mathcal{O}(L \log L)$ complexity.

8 Numerical results

Contents

In this section we present the results of our numerical experiments. We start by sketching out some of their important features, while the full self-contained code used to run the experiments can be found in the supplementary GitHub repository [92]. We apply the $E_{\text{loc}}^{\text{var}}$-based optimisation to study molecular systems of previously inaccessible sizes and reveal that for certain systems ANQS are capable to outperform traditional quantum chemistry methods and achieve state-of-the-art accuracies. To ensure a balanced and comprehensive analysis, we also consider scenarios where ANQS struggle to produce high-quality energies. Finally, we provide results of various ablation studies which informed our choice of hyperpameters in

the main set of experiments.

8.1 Experiment particulars

General setup

All experiments were performed on a single NVIDIA RTX A5000 GPU with 24 GB of RAM. For a given molecule and its geometry, we obtain the qubit Hamiltonian with the OpenFermion software library [93], which uses PySCF [94] and Psi4 [95] quantum chemistry packages as underlying backends to calculate one- and two-body integrals. We study all molecules in the minimal basis set STO-3G, unless specified otherwise. For each molecule we obtain the maximum set of symmetries and ensure symmetry-aware sampling as described in Chapter 5. For each geometry we run calculations using three different seeds of the underlying pseudorandom number generator. When discussing achieved energies we report the *minimum* value obtained across seeds unless specified otherwise. For other metrics we report the *mean* value. We invite the reader interested in specific values of the hyperparameters and/or reproducing the experimental results to explore the supplementary GitHub repository [92].

Ansatz architecture

We employ an ANQS architecture that represents each conditional wave function with *two* real-valued subnetworks, separately encoding the absolute value $\log|\psi|$ and the phase φ of an amplitude $\psi = \mathrm{e}^{\log|\psi|+\mathrm{i}\varphi}$. This allows us to perform SR gradient postprocessing without worrying about numerical instability issues arising with complex-valued subnetworks.

Each ANQS subnetwork is a multi-layer perceptron with two hidden layers of width 64. There is a residual connection adding the output of the first hidden layer to the output of the second *before* the second layer is activated. The hidden layers are activated with the hyperbolic tangent function, while no elementwise activation is applied to the last layer. Instead, we apply a *global* activation to the last layer which we describe below.

Local pruning strategy

Contrary to the findings of Chapter 5, in our experiments we employ the Mask-Unphysical-0 local pruning strategy, even though we expect it to restrict the ansatz expressivity and hinder the training. Our choice roots in the lack of understanding on how to reconcile discarding the samples required by Mask-Unphysical-2 and Discard-Unphysical with autoregressive sampling without replacement. In particular, it is not clear how dropping the partially formed unique samples during the Gumbel top-K sampling changes the empiric probability distribution of the samples in the batch $\mathcal{U}$. However, we alleviate the expressivity restriction brought about by our masking strategy by grouping qubits into qudits as explained below.

Grouping qubits into qudits

We group every $N_{\frac{\text{qubit}}{\text{qudit}}}$ qubits into a qudit and sample the latter. If N is not a multiple of $N_{\frac{\text{qubit}}{\text{qudit}}}$, the last subnetwork groups the remaining $\left(N \mod N_{\frac{\text{qubit}}{\text{qudit}}}\right)$ qubits into the last qudit.

To explain the main motivation behind such approach, let us show that if *all* N qubits are grouped into a single qudit, there is *no* expressivity difference between MU-0 and DU strategies. In this case the ANQS consists of a single subnetwork producing a probability distribution over 2^N elements. The MU-0 strategy modifies this probability distribution so that the unphysical elements are manually assigned zero probabilities. However, when the whole probability distribution is produced by the network at once, it is equivalent to simply renormalising the probabilities of all physical samples. As a result, these are produced from the same probability distribution as in the case of the DU strategy, which renormalises the empiric frequencies of the physical samples after discarding the unphysical ones. Therefore, when $N_{\frac{\text{qubit}}{\text{qudit}}} = N$, the MU-0 strategy is as expressive as DU.

Grouping all N qubits into a single qudit is clearly unfeasible since it requires a subnetwork with an exponential number of outputs. Hence, we aim to strike the balance between $N_{\frac{\text{qubit}}{\text{qudit}}} = 1$, which is known to result in poor convergence, and $N_{\frac{\text{qubit}}{\text{qudit}}} = N$. Specifically, we chose $N_{\frac{\text{qubit}}{\text{qudit}}}$ equal to 6 as a result of an ablation

study described further. Another advantage of grouping qubits into qudits is faster sampling and amplitude evaluation, which we also cover in the same ablation study.

Global activation

Once the log-amplitude subnetwork produces the unnormalised values for $\log|\psi(x_i|\mathbf{x}_{<i})|$, we apply a global activation function to them. Specifically, we shift them by their average, i.e. $\log|\psi(x_i|\mathbf{x}_{<i})| \to \log|\psi(x_i|\mathbf{x}_{<i})| - \sum_{x_i}\log|\psi(x_i|\mathbf{x}_{<i})|$, where the summation over x_i includes all possible $2^{\frac{N_{\text{qubit}}}{\text{qudit}}}$ values of i-th qudit. In our preliminary experiments we observed empirically that such global activation improves the optimisation convergence.

8.2 Main results

The main implicit assumption behind our approach is that a large enough N_{unq} allows sampling the most important amplitudes as long as the true ground state wave function is peaked enough. In this section we empirically demonstrate the validity of this assumption. We obtain energies that are better than those provided by state-of-the art quantum chemistry methods and/or below chemical accuracy. Equally as importantly, our approach provides an order of magnitude computational speedup compared to existing works.

8.2.1 Li_2O and BeF_2 dissociation curves

In the first set of experiments we study Li_2O and BeF_2 molecules both requiring $N = 30$ qubits in the minimum basis set STO-3G. These are amongst the largest molecules still amenable to exact diagonalisation. Thus, it is possible to check whether ANQS can achieve chemical accuracy with respect to the true ground state energy E_{FCI}. In addition, the unfavourable scaling of local energy calculations is already apparent for these molecules, and in the previous works an optimisation based on the full E_{loc} required an excessive compute time ranging from hours [48] to days [45]. As a result, little progress on extending the ANQS calculations beyond these molecules has been reported so far [46].

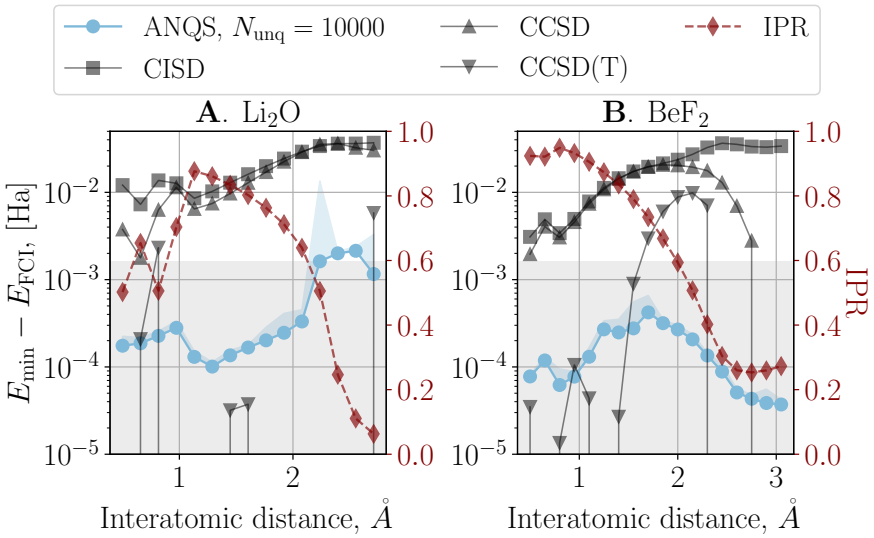


Figure 8.1: Dissociation curves for Li_2O and BeF_2 molecules. The grey shaded areas correspond to energy differences within chemical accuracy. The blue shaded areas depict the spread of values from minimum to maximum across seeds. We plot energy differences to the exact diagonalisation (FCI) result ((left y-axis) and ground state IPR (right y-axis) as a function of interatomic distance.

We quantify the peakedness of the ground states using the *inverse participation ratio (IPR)* defined as follows:

$$\text{IPR} \coloneqq \mathbb{E}\left[p(\mathbf{x})\right] = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot p(\mathbf{x}). \tag{8.1}$$

The maximum value of IPR is 1 and it is achieved when only one basis vector contributes to a wave function, and its minimum is $\frac{1}{2^N}$, reached when $\forall \mathbf{x}\ |\psi(\mathbf{x})| = \frac{1}{\sqrt{2^N}}$.

For each molecule we explore a range of interatomic distances and for each distance we optimise an ANQS with $N_{\text{unq}} = 10^4$. The minimum achieved variational energies depicted in Fig 8.1 consistently surpass those obtained by the conventional methods like CISD and CCSD. While ANQS does not always outperform CCSD(T), it is important to remember that the latter can yield unphysical energies below E_{FCI}, thus warranting cautious interpretation of such comparison. Crucially, ANQS reliably achieves energies within chemical accuracy for nearly all points on dissociation curves, with a notable exception around 2.4–2.6 Å for Li_2O, a region where traditional methods deliver poorer energies too.

We also plot the IPR corresponding to each interatomic distance. For Li_2O, a higher peakedness generally correlates with the improved optimisation accuracy.

Molecule	Li_2S	Li_2Se	Li_2Te	$BeCl_2$	$BeBr_2$	BeI_2	C_2
N	38	56	74	46	82	118	56
$\lvert\mathcal{H}\rvert$	$5.7 \cdot 10^9$	$9.7 \cdot 10^{12}$	$1.5 \cdot 10^{15}$	$7.8 \cdot 10^7$	$1 \cdot 10^{10}$	$2 \cdot 10^{11}$	$1.4 \cdot 10^{11}$

Table 8.1: Numbers of qubits and Hilbert space sizes for the molecules studied in Section 8.2.2.

However, the accuracy for BeF_2 shows a more complicated pattern: it drops with an initial decrease in IPR from approximately 0.9 to 0.75, before improving again as the IPR decreases further to around 0.25. Thus, while the data are compatible with the hypothesis that high peakedness is sufficient for successful optimisation, the overall success may also be influenced by other factors, such as the phase structure of the ground state. Identifying these factors is an important direction for future research.

Finally, let us note that for Li_2O molecule with the geometry taken from PubChem database [91] we achieve chemical accuracy on average in 750 seconds. This is 25 times faster compared to Ref. [48], which to the best of our knowledge reported the fastest optimisation for the same molecule so far. The per-iteration time also improves to 0.18 seconds as compared to 8.3 seconds reported in the same work.

8.2.2 Further Li and Be compounds

In the next set of experiments we expand the ANQS quantum chemistry calculations to previously challenging system sizes. We obtain dissociation curves for two groups of molecules of increasing size; the first group includes Li_2S, Li_2Se and Li_2Te molecules, while the second consists of $BeCl_2$, $BeBr_2$ and BeI_2. We base our selection on their chemical similarity with the already studied Li_2O and BeF_2: the anion is replaced by a heavier element in the same group 6 (7) of the periodic table. Thus we can systematically scale the number of qubits N, the Hilbert space size and the number of terms in Hamiltonian N_T. We note, however, that since the Hilbert space dimension is $|\mathcal{H}| := \binom{N}{n_e}$, the largest values of N do not necessarily amount to the largest values of $|\mathcal{H}|$. For example, the largest studied Hilbert space is that of Li_2Te molecule with 74 qubits and $\sim 10^{15}$ Slater determinants (see Table 8.1), which is three orders of magnitude more than the largest FCI study performed

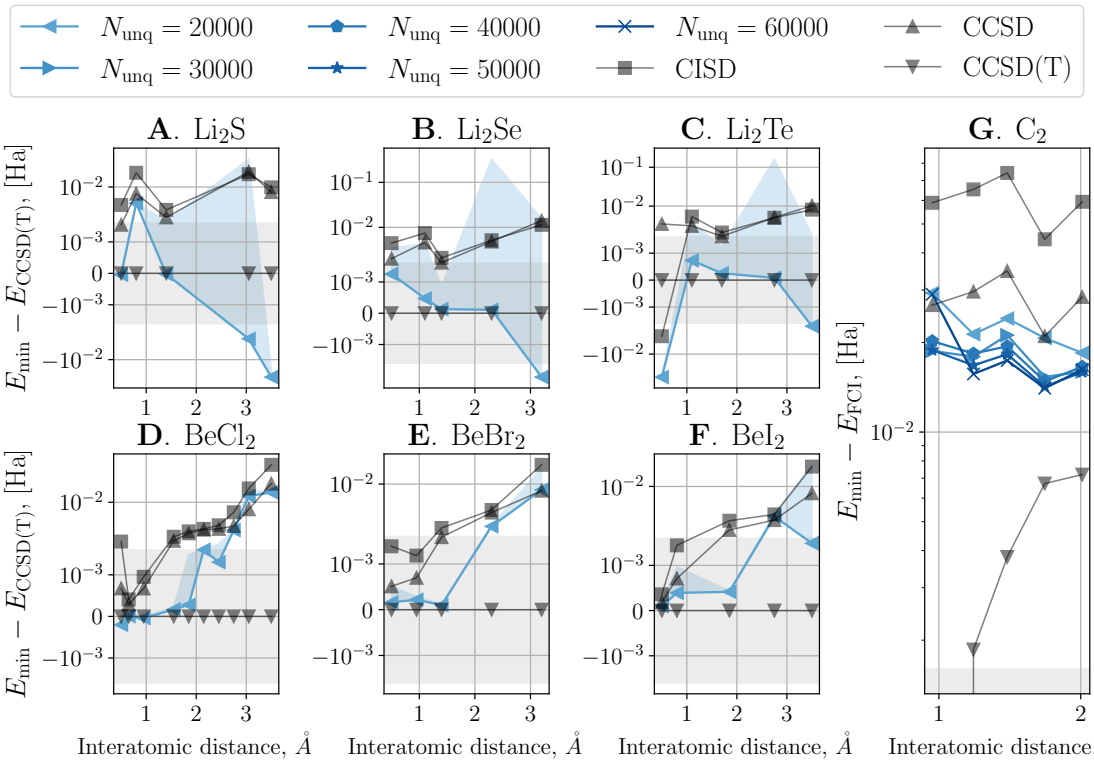


Figure 8.2: Dissociation curves for bigger molecules as described in the main text. The C_2 molecule is considered in cc-pVDZ basis set. We plot energy differences to a reference energy as a function of interatomic distance. **A**–**F.** The reference energy is obtained with CCSD(T). **G.** The reference energy is obtained with FCI as reported in [96]. The grey shaded areas correspond to the energy differences within chemical accuracy. The blue shaded areas depict the spread of values from minimum to maximum across seeds.

so far [97]. At the same time, the Hilbert space size of BeI_2 molecule with 118 qubits is on the order of "mere" 10^{11} Slater determinants.

We plot the results of this set of experiments in Fig. 8.2A–F. We do not have access to FCI energies for these molecules, and thus we plot the difference between the minimum variational energies achieved by ANQS and those obtained with CCSD(T). It can be seen that in the dominant majority of experiments ANQS outperforms traditional quantum chemistry methods such as CISD and CCSD. When compared to CCSD(T), ANQS results mostly come close to within chemical accuracy, and in some cases surpass them. We consider this as an evidence in favour of the remarkable prospect of ANQS.

8.2.3 C_2 molecule in cc-pVDZ basis set

We also apply ANQS to a paradigmatic strongly correlated system: C_2 molecule in the cc-pVDZ basis set [96]. The results are presented in Fig 8.2G, where we plot the difference between the lowest energy achieved by ANQS and E_{FCI} as given in Ref. [96]. While we find that increasing N_{unq} systematically improves the result, none of N_{unq} values allowed reaching chemical accuracy or surpassing the CCSD(T) energies, even though C_2 has a moderate number of qubits (56) and Hilbert space size ($\sim 10^{11}$) as compared to Li and Be compounds studied in the previous section. Nevertheless, an increase in N_{unq} does result in better accuracy, and, starting from $N_{\mathrm{unq}} = 40000$, ANQS energies surpass those of CISD and CCSD methods. We leave it for future work to explore whether increasing N_{unq} further would continue to improve the convergence, or whether the limiting factor is rather the expressivity of the chosen ansatz.

8.3 Ablation studies

In this section we present the results of the ablation studies. First, we study how replacing the local energy with its variational proxy affects the optimisation accuracy and computational efficiency. Second, we benchmark autoregressive Gumbel sampling against the *adaptive* approaches to autoregressive statistics sampling. These approaches aim to obtain the desired number of samples N_{unq} at every optimisation iteration by means of resamplings with adjusted batch size. Third, we study how grouping qubits into qudits affects the performance of variational optimisation. Fourth, we numerically investigate the impact of $N_{\mathrm{unq}}^{\mathrm{SR}}$ on the accuracy and speed of ANQS quantum chemistry calculations. Finally, we compare three implementations of the FindCoupledPairs procedure described in Chapter 6.

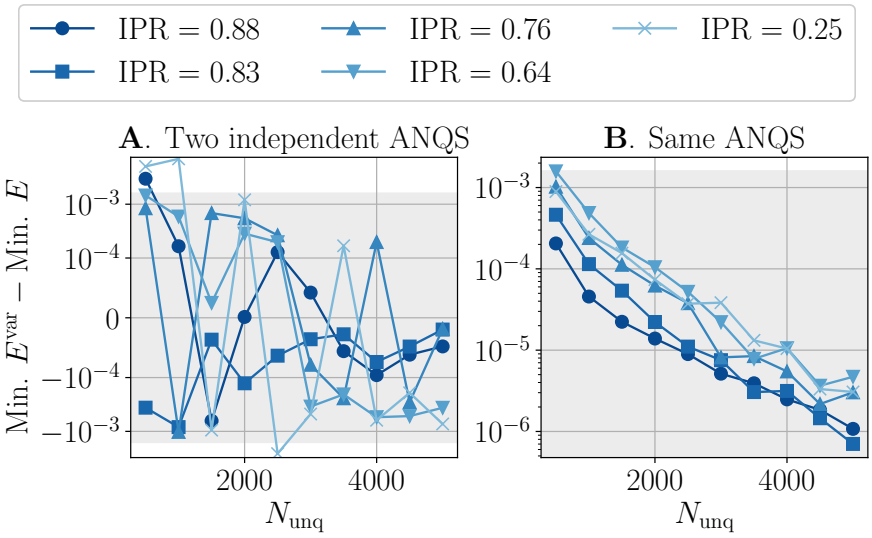


Figure 8.3: Impact of $E_{\text{loc}}^{\text{var}}$ on the accuracy of variational optimisation. The grey shaded areas correspond to the energies within chemical accuracy of 1.6 mHa. We depict difference between the minimum achieved E_{loc} and $E_{\text{loc}}^{\text{var}}$ for two cases. **A.** Two independent ANQS are optimised via E_{loc} and $E_{\text{loc}}^{\text{var}}$ evaluation respectively. **B.** A *single* ANQS is optimised via the full E_{loc} evaluation.

8.3.1 $E_{\text{loc}}^{\text{var}}$ vs E_{loc}

Accuracy comparison

In the first set of ablations we investigate whether substituting E_{loc} with $E_{\text{loc}}^{\text{var}}$ affects the minimal energies achieved during the optimisation. Li_2O is a linear molecule, and by varying the distance between the nuclei we can obtain ground states with different values of IPR, which allows us to track the impact of the peakedness as well.

We consider five molecular geometries with IPRs ranging from 0.25 to 0.88. For each geometry and for each flavour of ANQS optimisation (i.e. employing either E_{loc} or $E_{\text{loc}}^{\text{var}}$) we run optimisation for ten different values of N_{unq} ranging from 500 to 5000. Thus, we can study whether capturing a larger fraction of the Hilbert space at each iteration makes any difference for the proxy-based optimisation. We do not employ stochastic reconfiguration and run each optimisation for 10^4 iterations.

For each optimisation instance we extract the minimum achieved energy, which we denote as E and E^{var} in the case of conventional and proxy-based optimisation respectively. We plot the difference $E^{\text{var}} - E$ for each N_{unq} and different values of IPR in Fig. 8.3A. Three trends are pronounced: first, both energies lie within

chemical accuracy to each other and no method has a decisive advantage — in some cases $E^{\text{var}} - E < 0$, which means that proxy-based optimisation delivered a better energy, while in some cases the opposite holds. Second, the difference between two methods seems to become less substantial as N_{unq} grows. Third, the absolute value of energy difference $E^{\text{var}} - E$ is bigger for geometries with smaller IPR.

We also study the influence of IPR in a complementary way: for each instance of optimisation employing the full local energy we additionally calculate the variational energy of the state and compare it to the value estimated via the full local energy; the results are presented in Fig. 8.3B. Similarly to Fig. 8.3A, the difference between energies becomes smaller as N_{unq} grows. However, the absolute value of the gap between two energies is different and depends on IPR: for $N_{\text{unq}} = 2000$ the gap is $\approx 10^{-5}$ Ha for IPR $= 0.88$, while for a smaller value of IPR $= 0.64$ the gap is approximately ten times larger. Let us note, however, that for certain N_{unq} proxy-based optimisation has a smaller gap for IPR $= 0.25$ than for IPR $= 0.68$.

We consider this set of experiments to be a numerical evidence supporting the notion that for certain systems proxy-based optimisation can achieve energies comparable to or better than the conventional one.

Computational performance comparison

In the second set of experiments we ablate the *practical advantage* of proxy-based optimisation. To that end we turn our attention to the set of already considered $\{\text{Li}_2\text{O},\ \text{Li}_2\text{S},\ \text{Li}_2\text{Se},\ \text{Li}_2\text{Te}\}$ molecules. For each molecule we pick a single geometry and run 10^2 optimisation iterations for values of N_{unq} ranging from 500 to 5000. We choose such a modest number of iterations due to the prohibitive cost of the full local energy calculations. Similarly to the previous set of experiments, we do not employ stochastic reconfiguration. During each run we measure the average iteration time as well as the time required to calculate the values of $E_{\text{loc}}(\mathbf{x})$ ($E^{\text{var}}_{\text{loc}}(\mathbf{x})$ in the case of proxy-based optimisation) after having sampled the set of unique basis vectors $\mathcal{U}$.

In Fig. 8.4 we show the speedup provided by the proxy-based optimisation, which we define as the ratio $\frac{T_{\text{iter}}[E_{\text{loc}}]}{T_{\text{iter}}\left[E^{\text{var}}_{\text{loc}}\right]}$, where $T_{\text{iter}}\left[E_{\text{loc}}\right]$ and $T_{\text{iter}}\left[E^{\text{var}}_{\text{loc}}\right]$ are the average

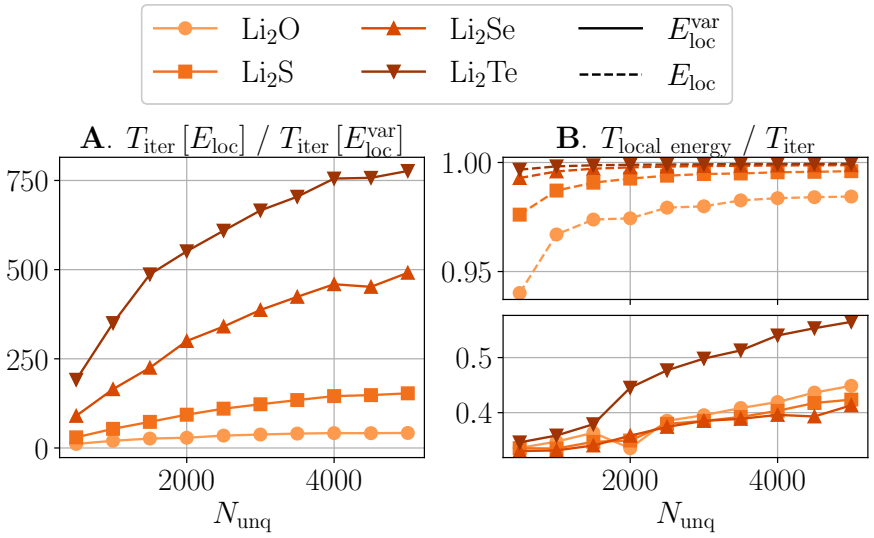


Figure 8.4: Impact of $E_{\text{loc}}^{\text{var}}$ on the computational performance of variational optimisation. **A.** A speedup provided by $E_{\text{loc}}^{\text{var}}$ with respect to E_{loc} minimisation. **B.** A fraction of the iteration time taken by the energy evaluation for both types of optimisation.

iteration times for the conventional and proxy-based optimisations respectively. For the smaller Li_2O molecule the proxy-based advantage is rather modest, with the largest tenfold speedup achieved for $N_{\text{unq}} = 5000$. However, the larger the molecule is, the larger the proxy-based advantage grows, and for Li_2Te the starting value of $\frac{T_{\text{iter}}[E_{\text{loc}}]}{T_{\text{iter}}[E_{\text{loc}}^{\text{var}}]}$ is 200 for $N_{\text{unq}} = 500$, which further grows to 750 at $N_{\text{unq}} = 5000$. In wall-clock time it corresponds to $T_{\text{iter}}[E_{\text{loc}}]$ of approximately 2.5 minutes, while $T_{\text{iter}}[E_{\text{loc}}^{\text{var}}]$ was only 0.2 seconds. As a result, an attempt to run 10^4 iterations of the conventional optimisation would require 17 days, while the proxy-based optimisation will be completed in just above half an hour.

Apart from reducing the per iteration computational cost, the proxy-based optimisation also changes the relative cost of different iteration parts. In Fig. 8.4B we plot which fraction of the iteration is taken by the calculation of E_{loc} ($E_{\text{loc}}^{\text{var}}$), including the time spent on sampling and amplitude evaluation. For the conventional optimisation evaluation of E_{loc} has the dominant cost and, for example, for Li_2Te more than 99% of iteration is spent on it. At the same time, in the case of the proxy-based optimisation, the calculation of $E_{\text{loc}}^{\text{var}}$ still constitutes a substantial part of the total iteration time, but it amounts for only $\sim 50\%$ of it. Thus, the

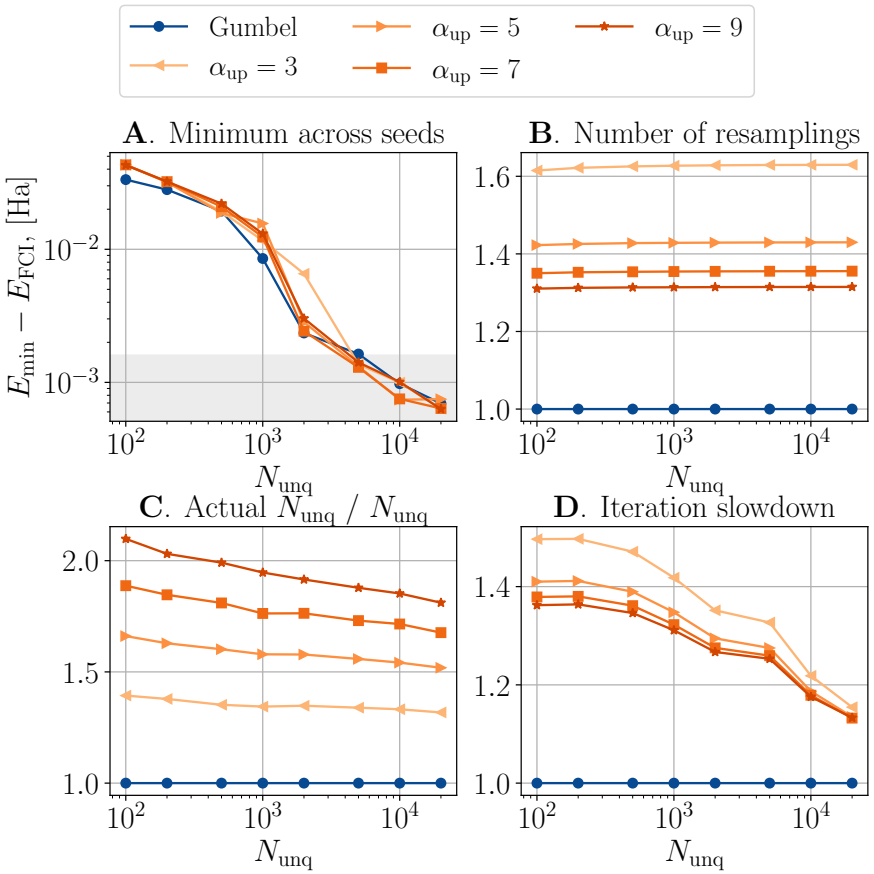


Figure 8.5: An ablation study comparing Gumbel sampling to adaptive statistics sampling strategies for several values of α_{up}. **A.** Achieved energy differences with respect to E_{FCI}; the grey shaded area corresponds to the energy differences within chemical accuracy of 1.6 mHa. **B.** Number of resamplings required to obtain the desired number of unique samples. **C.** Ratio of the actual number of unique samples produced during the last resampling to the desired N_{unq}. **D.** Slowdown in the total iteration time brought by adaptive sampling as compared to the Gumbel sampling iteration time.

contributions from other iteration parts are now discernible and further attention can be paid to optimising their performance too.

8.3.2 Autoregressive Gumbel sampling

In this ablation study we compare autoregressive Gumbel sampling to adaptive autoregressive statistics sampling. As discussed in Chapters 3 and 6, autoregressive Gumbel sampling is advantageous due to its ability to produce the number of unique samples N_{unq} *exactly* as specified by the user. However, it is possible to obtain the desired number of samples at every iteration with the conventional autoregressive

statistics sampling [45, 48] too. To that end one has to resort to adaptive sampling: if the number of produced unique samples is less than N_{unq}, then one increases N_{s} according to some batch size schedule and samples again.

Specifically, we consider the following adaptive scheme: if the number of produced unique samples is less than N_{unq}, we increase N_{s} by a factor of α_{up} and sample again. As soon as the number of obtained unique samples exceeds N_{unq}, we stop sampling and select the first N_{unq} unique samples with the highest probabilities $p(\mathbf{x})$ produced by the ansatz. In addition, to avoid soaring values of N_{s}, we *decrease* the batch size by a factor of $\alpha_{\text{down}} = 2$ before *the first* sampling attempt of each iteration. One might consider various other adaptive sampling approaches, however this remains outside the scope of this thesis.

For our benchmark we optimise an ANQS corresponding to the Li_2O molecule for $2 \cdot 10^4$ iterations using plain Adam optimiser across a range of N_{unq}. We choose five possible values of $\alpha_{\text{up}} = \{3, 5, 7, 9\}$. In addition, we run calculations using only autoregressive Gumbel sampling.

The results of optimisation are presented in Fig. 8.5. One can see in Fig. 8.5A that the energy errors achieved by all sampling strategies are similar to a good degree of accuracy, with no sampling scheme having a competitive edge over others. At the same time all sampling approaches apart from Gumbel require on average more than one sampling per iteration, as displayed in Fig. 8.5B. Evidently, the number of resamplings decreases as α_{up} grows, since it takes fewer attempts to reach higher values of N_{s}. However, this comes at a cost of increased computational burden as shown in Fig. 8.5C: the actual number of unique samples produced during each successful sampling attempt is larger than N_{unq} for all adaptive sampling strategies. As a result, every adaptive sampling scheme increases iteration time with respect to Gumbel sampling as presented in Fig. 8.5D. Notably, lower values of α_{up} slow down the computation more, since restarting sampling multiple times is more expensive than producing more unique samples as long as the batch of unique samples fits in the GPU RAM.

Finally, let us note that while the slowdowns displayed in Fig. 8.5D are rather moderate, this is mainly due to sampling taking less than 15% of the total iteration time as shown in Fig. 8.10. However, we expect the advantage of Gumbel sampling to become more pronounced for larger molecules, with sampling taking a larger part of iteration time.

8.3.3 Grouping qubits into qudits

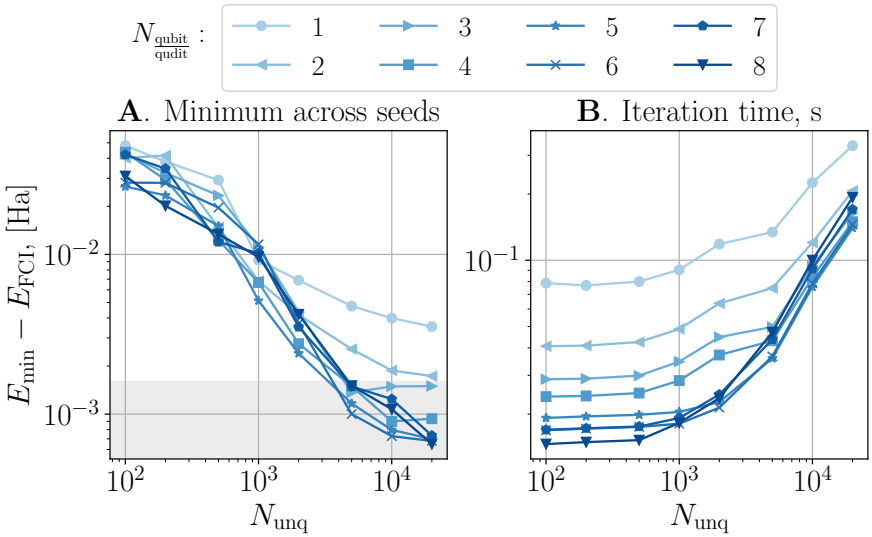


Figure 8.6: An ablation studying the impact of grouping qubits into qudits. **A.** Achieved energy differences with respect to E_{FCI}; the grey shaded area corresponds to the energy differences within chemical accuracy of 1.6 mHa. **B.** Iteration time averaged across seeds.

In this ablation study we investigate the impact of grouping qubits into qudits. We optimise the ANQS representing the Li_2O molecule for $2 \cdot 10^4$ iterations without resorting to SR. We consider every possible value of $N_{\frac{\text{qubit}}{\text{qudit}}}$ from 1 to 8 and show the results in Fig. 8.6. One can see in Fig. 8.6A that low values of $N_{\frac{\text{qubit}}{\text{qudit}}}$ such as 1 and 2 consistently perform the worst in terms of the achieved energy error, even though they have the largest number of parameters as shown in the table in Fig. 8.6. This is in agreement with the results of Chapter 5 indicating that masking conditional wave functions required for correct symmetry-aware sampling reduces the expressivity of an ANQS [48, 72, 98]. At the same time, increasing $N_{\frac{\text{qubit}}{\text{qudit}}}$ results in steady improvement of the achieved accuracy, with $N_{\frac{\text{qubit}}{\text{qudit}}} = 5, 6$

N_{p}	1	2	3	4	5	6	7	8
$N_{\frac{\mathrm{qubit}}{\mathrm{qudit}}}$	317,048	161,528	112,288	97,128	85,376	91,648	118,408	148,224

Table 8.2: Number of parameters for ansatzes constructed with different $N_{\frac{\mathrm{qubit}}{\mathrm{qudit}}}$ and used in the ablation study of Section 8.3.3.

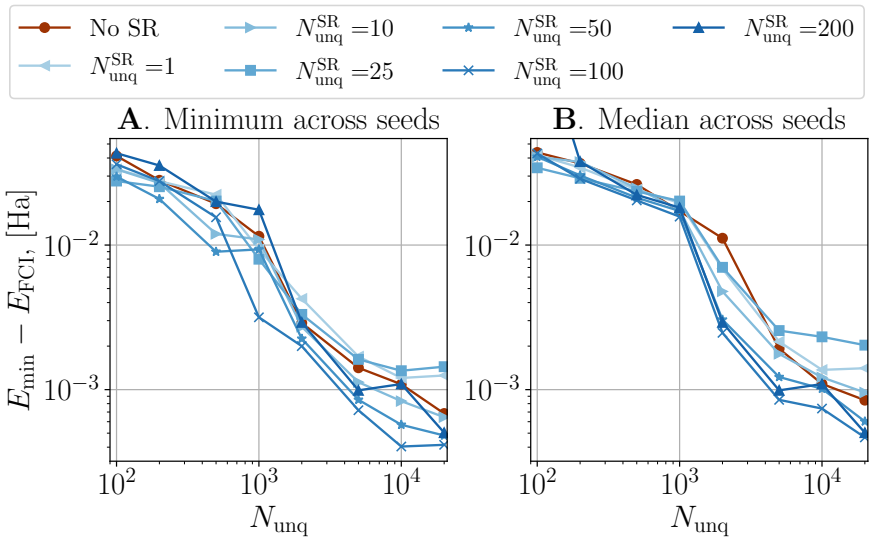


Figure 8.7: Impact of the hyperparameter $N_{\mathrm{unq}}^{\mathrm{SR}}$ in stochastic reconfiguration. We plot the achieved energy differences with respect to E_{FCI}; the grey shaded area corresponds to energy differences within the chemical accuracy of 1.6 mHa. Figures **A** and **B** show minimum and median statistics calculated across five seeds respectively.

providing the best energies at $N_{\mathrm{unq}} = 20000$. Importantly, these values of $N_{\frac{\mathrm{qubit}}{\mathrm{qudit}}}$ also result in the least time spent on sampling and amplitude evaluation as depicted in Fig. 8.6. This is due to the fact that larger $N_{\frac{\mathrm{qubit}}{\mathrm{qudit}}}$ amount to fewer conditional wave functions $\psi(x_i|\mathbf{x}_{<i})$ constituting the ansatz (the number of conditional wave functions is given by $\left\lceil \frac{N}{N_{\frac{\mathrm{qubit}}{\mathrm{qudit}}}} \right\rceil$).

Similar regularities were obtained in our preliminary experiments on other molecules. As a result, we chose $N_{\frac{\mathrm{qubit}}{\mathrm{qudit}}} = 6$ for all numerical experiments described in Section 8.2.

8.3.4 Stochastic reconfiguration

In this ablation study we investigate how SR affects the accuracy and computational efficiency of ANQS quantum chemistry calculations. Prior to comparing the optimisation with and without SR, we select the optimal hyperparameter $N_{\mathrm{unq}}^{\mathrm{SR}}$.

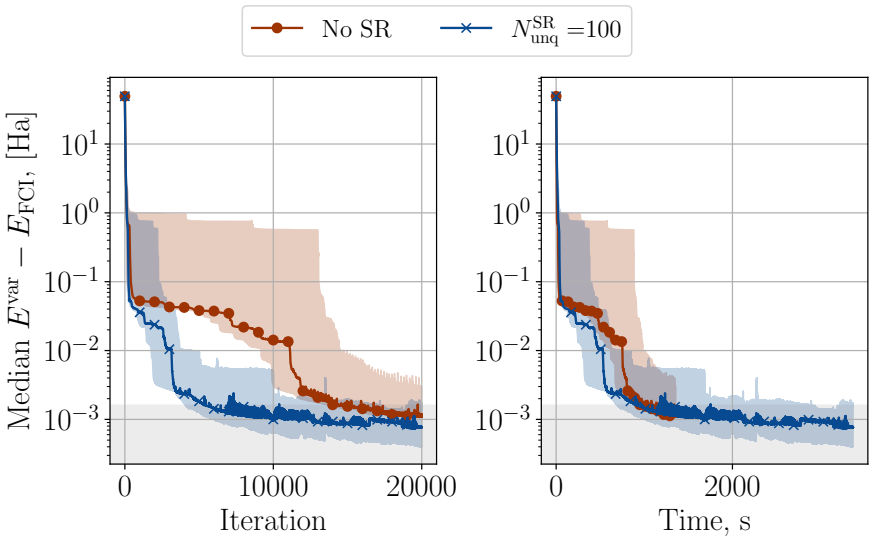


Figure 8.8: Example training curves for Li_2O molecule at $N_{\text{unq}} = 10000$. We show median energy differences to the exact diagonalisation (FCI) result as a function of (**A**) iteration number; (**B**) wall clock time. The coloured shaded areas depict the spread of values from minimum to maximum across seeds. The grey shaded areas correspond to the energy differences within chemical accuracy of 1.6 mHa.

Similarly to the previous ablation studies, we consider the Li_2O molecule and run ANQS optimisation with different values of $N_{\text{unq}}^{\text{SR}} \in \{1, 10, 25, 50, 100, 200\}$; as well we run optimisation instances which do not employ stochastic reconfiguration. We run each optimisation configuration with 5 different seeds of the underlying pseudorandom number generator and in each run we measure the minimum achieved energy.

In Fig. 8.7A and Fig. 8.7B we show, respectively, the minimum and median achieved energies across the five seeds. Generally, larger values of $N_{\text{unq}}^{\text{SR}}$ result in lower achieved energies, with $N_{\text{unq}}^{\text{SR}} = 100$ providing the best energy starting from $N_{\text{unq}} = 10^3$. Rather surprisingly, stochastic reconfiguration does not always translate into the improved convergence: for example, $N_{\text{unq}}^{\text{SR}} = 1$ and $N_{\text{unq}}^{\text{SR}} = 25$ provide worse energies than purely Adam-based optimisation. The larger value of $N_{\text{unq}}^{\text{SR}}$, namely 200, does not result in better energies either, even though one might naïvely expect it to provide a more accurate estimate of the metric tensor S. We attribute such behaviour to sharp singular spectrum of S typical for NQS. A larger $N_{\text{unq}}^{\text{SR}}$ gives rise to multiple, very small singular values, introducing numerical

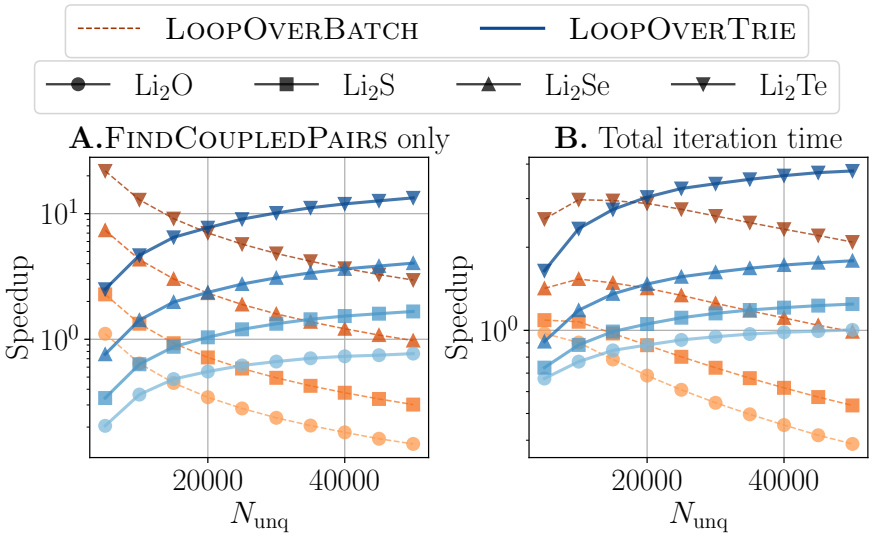


Figure 8.9: The speedup of LoopOverBatch and LoopOverTrie routines with respect to baseline LoopOverTerms. The speedups are calculated for: **A.** Only FindCoupledPairs procedure; **B.** Total iteration time.

noise which affects the accuracy of the inversion of S.

In Fig. 8.8 we show exemplary training curves for optimisation of the Li_2O molecule with $N_{\text{unq}} = 10^4$. Specifically, we show how E^{var} changes with the iteration number (Fig. 8.8A) and elapsed time (Fig. 8.8B) for two optimisations run (i) with $N_{\text{unq}}^{\text{SR}} = 100$ and (ii) without SR. It can be seen that the SR-driven optimisation achieves lower energies substantially earlier. However, once the required iteration time is taken into account, the optimisation performed without SR catches up and might eventually achieve similar energies in a similar *time*. The optimal strategy could therefore be to start the training with SR and switch it off as soon as the iterations "break through" to energies below a certain threshold.

8.3.5 Comparison of FindCoupledPairs implementations

In the final set of experiments we benchmark three implementations of the FindCoupledPairs procedure. In Fig. 8.9A we plot the speedups achieved by the LoopOverBatch and LoopOverTrie algorithms compared to the baseline LoopOverTerms implementation. For small values of N_{unq} the LoopOverBatch implementation performs the best, offering up to 10x reduction in time. However, as

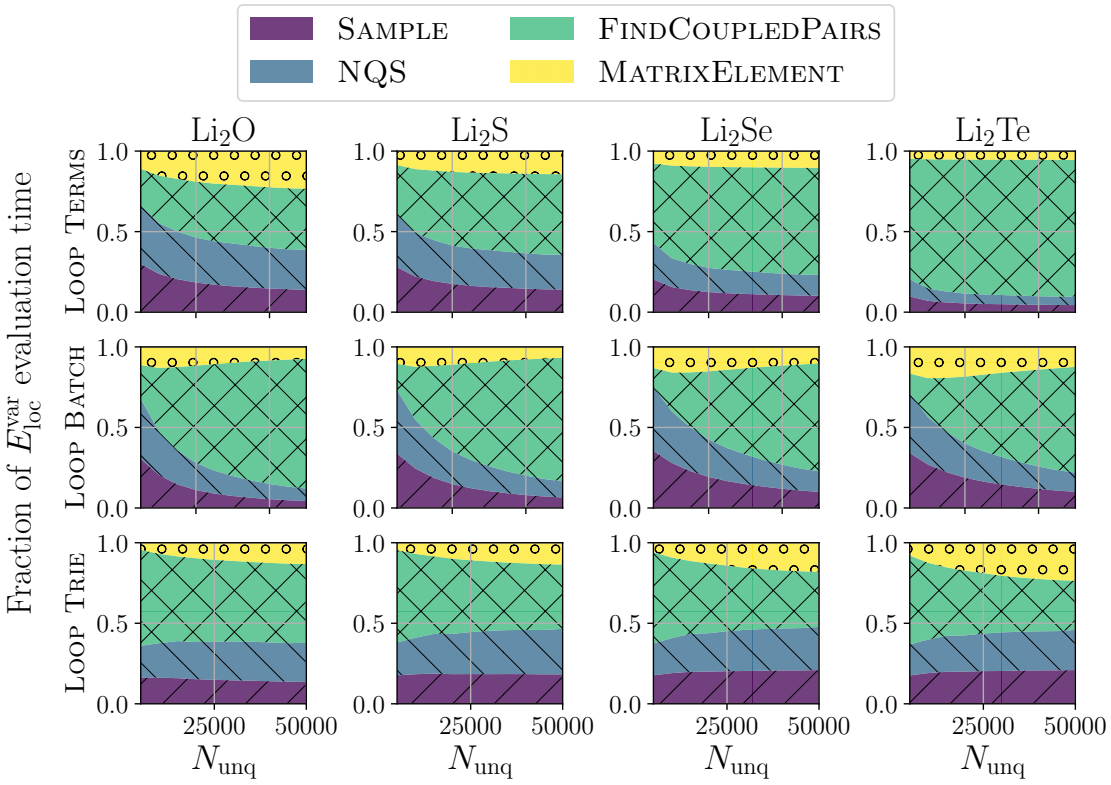


Figure 8.10: The fraction of the energy evaluation time spent on each of four crucial iteration parts: (i) sampling; (ii) evaluation of amplitudes for basis vectors in $\mathcal{U}$; (iii) finding the coupled pairs $(\mathbf{x}, \mathbf{x}')$; (iv) calculating the matrix elements between the coupled pairs. Top to bottom rows correspond to different implementations of FindCoupledPairs procedure: LoopOverTerms, LoopOverBatch, LoopOverTrie.

N_{unq} grows, the advantage of LoopOverBatch becomes less and less pronounced, until it vanishes. For example, for the Li_2O molecule and $N_{\mathrm{unq}} = 5 \cdot 10^4$ it is an order of magnitude slower than the other approaches.

The opposite holds for LoopOverTrie: at low N_{unq} it struggles to compete with LoopOverBatch. As discussed in Chapters 6 and 7, this is due to overhead related to explicit looping over qubits inherent to LoopOverTrie. However, as *both* N_{unq} and N_{T} grow, LoopOverTrie becomes more and more advantageous: for example, for Li_2Se molecule and $N_{\mathrm{unq}} = 5 \cdot 10^4$ it is five times faster than either LoopOverTerms or LoopOverBatch. For Li_2Te molecule and the same N_{unq} it demonstrates a tenfold speedup compared to LoopOverTerms.

To accurately estimate the speedup associated with these new techniques, it is important to keep in mind that FindCoupledPairs takes only a part of the

compute time in each iteration. The local energy calculation also involves three other crucial subroutines: sampling of $\mathbf{x}$, amplitude evaluation and computing the matrix elements $H_{\mathbf{x}\mathbf{x}'}$. In Fig. 8.9B, we show the gains associated with the new FindCoupledPairs algorithms with respect to the total iteration time. In the case of Li_2Te molecule and $N_{\text{unq}} = 5 \cdot 10^4$, the tenfold improvement of LoopOverTrie shrinks down to a factor of four.

In Fig. 8.10 we plot the fraction of the time spent on each of the four subroutines during the energy evaluation. Both for LoopOverTerms and LoopOverBatch the fraction of FindCoupledPairs grows as N_{unq} increases. For the largest molecule (Li_2Te) at $N_{\text{unq}} = 5 \cdot 10^4$, it amounts for the majority of iteration time, while the remaining subroutines take at most 10% of the calculations each. At the same time, FindCoupledPairs implemented as LoopOverTrie constitutes only a third of iteration time, and, more generally, decreases as both N_{unq} and N_{T} grow. Thus, computations related directly to the local energy calculation (as opposed to sampling and amplitude evaluation) are no longer a bottleneck of NQS quantum chemistry.

Conclusion

Carleo and Troyer introduced neural quantum states to leverage the expressive power of neural networks and tackle challenging quantum many-body problems within the framework of variational Monte Carlo. In this thesis we studied the application of *autoregressive* neural quantum states to the paradigmatic problem of quantum chemistry in second quantisation. However, when dealing with quantum many-body problems one inevitably faces the second half of the Anna Karenina principle: "Each quantum many-body problem is hard in its own way". Quantum chemistry was no exception: we had to adjust neural quantum states to a multitude of challenges not present in interacting spin systems, the initial domain of neural quantum states application.

We showed that $\mathbb{Z}_2$ symmetries initially conceived in the context of variational quantum algorithms are also beneficial in the context of NQS. They allow one to account for the point group symmetries of a molecule. This reduces the computational space of the problem to the level operated with by the conventional quantum chemistry methods. As a result, we reached the level of accuracy reported in previous works with more than an order of magnitude speedup for a range of molecules with multiple $\mathbb{Z}_2$ symmetries.

We also suggested that the peaked ground state wave functions typical for molecules should be not only mitigated in ANQS quantum chemistry calculations, but rather celebrated. In particular, it allowed us to leverage an ANQS-specific sampling algorithm which produces the desired number of unique samples. Thus, we were able to focus on a small physically relevant portion of the Hilbert space and greatly accelerate the stochastic estimation of the variational energy by ignoring the contributions from the unsampled part of the Hilbert space. Consequently, we were able to achieve energies that compete with those obtained with the traditional

quantum chemistry methods, for computational spaces several orders of magnitude larger than previously attempted with ANQS. Equally as important, our innovations translated previously challenging calculations into routine. For example, the first application of ANQS to quantum chemistry in Ref. [45] required roughly two days to obtain chemical accuracy for the Li_2O molecule. In our experiments the same level of accuracy can be achieved within just above ten minutes.

Remarkably, we even came across certain geometries where the peakedness was less pronounced, and yet focusing only on the sampled subspace still yielded energies within chemical accuracy to the ground state — this was the case, for example, with BeF_2. At the same time, we identified cases when our method struggled to reach competitive energies, such as C_2. However, it remains to be explored, whether this indicates a general limitation of our method or whether it is due to unrelated issues like bottlenecks in the optimisation [99].

More generally, while solid intuition was developed for many NQS use cases and the field matures more with every published work, every resolved issue is still followed with more open questions on whether a better solution exists.[1] Let us offer a few further directions for algorithmic improvements and theoretical analysis that would scale up our method.

The first observation is that our method is readily amenable to parallelisation across several GPUs [46, 47]. We expect this to enable addressing even larger problem sizes, potentially reaching complex organic compounds.

Additionally, the impact of the underlying neural network architecture on the accuracy of optimisation remains a largely unexplored topic. Recently, neural network architectures that are specifically tailored to fermionic systems showed significant promise in application to quantum many-body problems [78, 100], including quantum chemistry [36, 50]. Combining the enhanced expressivity

[1] Deep learning, the parent field of neural quantum states, is often compared to art: the art of choosing the right architecture, right set of hyperparameters, right data preprocessing pipeline. Sometimes even a stronger statement seems appropriate for neural quantum states: not only it is an art, it is also *sorcery*.

exhibited by these models with the sampling efficiency of ANQS can potentially give rise to a powerful new approach [101].

We also lack understanding on how the choice of N_{unq} influences the learning ability of ANQS. However daunting this question might seem, first steps in a similar direction were already made by Astrakhantsev *et al.* [102]. This work analysed an algorithmic phase transition in the learning dynamics of a variational quantum circuit related to the number of samples produced at each gradient descent step. It is natural to extend these techniques to ANQS optimisation.

Finally, as discussed, the variational energy (6.1) is an upper bound to the true energy of the NQS, but this upper bound may not be sufficiently tight. The so-called *projected* energies provide a more accurate estimate [103]. It is to be explored whether ANQS optimisation can be synthesised with this approach to energy calculation for improved optimisation.

Regardless of the actual research path taken, we believe that our work showcases the high promise of ANQS *ab initio* quantum chemistry calculations beyond the FCI limit. We are optimistic that the ideas presented in this thesis will democratise and facilitate further experiments, thus serving as a fruitful basis for the rapid future progress.

Acknowledgements

So, this is it. Multiple times during my DPhil I was reading theses of other people — and each time I was drawn to reading their acknowledgements, eager to find a trace of what was happening behind the scenes, to see not only the scientific content, but how did the person actually feel in the process. And, of course, every time I was imagining what *my* acknowledgements section would look like. So, here I am — what do I have to say?

Well, it was quite a ride! Don't get me wrong, being a physicist was my dream since childhood, and still is, and doing a DPhil went a long way towards it; the topic I was working on had a perfect mix of physics and computation, and I immensely enjoyed it; being an Oxford student amounted to extremely entertaining life outside work too!

However, there was a good amount of grief and desperation too. Many times, much more than I would like it, I doubted myself and was not sure whether there was any way forward in my research. I know it happens to everyone, and, in retrospect, there always was a way out — however that does not make the memory of pain disappear, right?

On top of that, I wouldn't call optimal experiencing global pandemics at the start of a DPhil, when one is supposed to make connections. However, in that regard I don't have much to complain about, since I had amazing housemates and my college managed to somehow keep the library open 24/7 for a large part of COVID.

What I couldn't shake off that easily, and still can't, is the Russian invasion of Ukraine. It is outright evil, causing enormous suffering in millions of innocent people. I couldn't help but think that the world is at some dangerous watershed, prone to collapse into grim times. It is a black cloud living in my mind, blocking the view of any happy future.

However, there is a very powerful remedy I found against the feeling of being powerless. This remedy is remembering and being grateful to all the people I've met on the way, who made this "quite a ride" very special. Thinking of them gives me hope.[2]

First of all, I would like to glance back at my school time in Sevastopol and thank Viktor Fedorovich Sannikov, Ivan Mistryanu and Ivan Ivanovich Kazachek. Viktor Fedorovich was teaching Olympiad math with rigour, Ivan was teaching Olympiad programming with kindness and Ivan Ivanovich was teaching Olympiad physics with passion. I owe them both my interest in science and an ability to actually do something there, which I am still living off. I would also like to thank my school director, Viktor Albertovich Oganesyan for being our Professor Dumbledore and kindly letting us prioritise Olympiad preparation above everything else. As well, I want to commemorate Ludmila Sergeevna Starkova, our late class teacher, who was our Professor McGonagall, making sure me and my friends don't get lazy (thank you Rebekka for pointing out my spelling error, which has been with me for the better part of my life) or complacent.

More generally, I would like to pay tribute to teachers making Ukrainian Physics Olympiad run for the last thirty years, regardless of what happens in the outer world, especially Oleh Yuriyovich Orlyanskii (who has served in the Armed Forces of Ukraine since the first days of invasion), Yurii Yakovich Pasikhov (who always came across to me as some sort of Cossack hetman), Pavlo Andriyovich Viktor (who recently received well-deserved recognition with a million followers on his YouTube channel) and Vadim Leonidovich Manakin (who was the first to tell me that a photon is *neither* a wave, *nor* a particle, but rather what we make of it ourselves by measuring).

In a less distant past I would like to thank people who gave me a first introduction into being a scientist and made my studies at Oxford possible. This includes Evgenii

[2]There is a little cloud shadowing my sky, though. Rumour has it some people I would like to thank are nearly supporting the Russian invasion of Ukraine. However, this is but a rumour, and I would like to give them the benefit of a doubt. Should the rumour be true, I would feel very sorry for them, as they must be living a rather miserable life, devoid of basic humanity.

Mikhaylovich Khorov and Artem Nikolayevich Krasilov of Institute for Information Transmission Problems (IITP), who taught me scientific discipline and helped me to publish my first papers. As well, I am grateful to Evgenii Kiktenko and Aleksey Fedorov for inviting me to the Russian Quantum Centre when I felt that studying wireless networks wasn't quite my calling. They also actively helped me to get a PhD position and planted the idea of applying to Oxford in my head, which, as you might guess, worked out quite nicely.

I would like to express my gratitude to Andrei Nikolayevich Sobolevskii, the director of IITP, who kindly took the time to write my recommendation letters despite being busy running a whole research institute. Recently, I watched interviews with Vladimir Kara-Murza following his release from a Russian prison, and I was reminded of the integrity, honesty, and scholarship I have witnessed in Andrei Nikolayevich. Unfortunately, he was lately demoted from his position by Russian authorities who seem to despise independent and decent people. I wish him all the best in this dark era.

Moving to Oxford meant leaving many friends behind, and, to be honest, I wasn't as diligent in replying to their messages as they deserved. Nevertheless, they welcomed me warmly each time I returned for a vacation, which I see as yet another testament to their big hearts. So, I am happy to use this opportunity and reach out and thank them: the "Го фильм" team of Maksik Bystrov, Tanya Selezneva, Ksyusha Yagafarova and Roma Makarov, keen to listen to my rowing stories until the early morning. To the Phystech.Radio people, especially Dasha Kublikova, Dima Proshutinskii, Dasha Sokol and Polina (я на колінах) Kitaytseva, whom I want to reassure that, as always, дождь на окнах рисует. Thanks to Kostya Chukreev and Leva Oganesyan for a night unexpectedly full of "дорогие полуночники" as well as "basically indeed." To Anya Artamonova, thank you for hearing the music of waves and wind. To Oksana Kostiuk, thank you for the great Spotify playlists and our shared passion for Viktor Pavlik. And to Anya Starovoytova, thank you for providing me with a chance to find unexplored parts of Yalta.

I was also very lucky to maintain connections with my school friends and form a strong support circle online. This kept me from going crazy during the first months of COVID when offline it was basically me, myself, and I. So, I thank Valya Myagkova, Nastya Krasnova, Danya Belosheikin, Ivanna Apostolova and *Dr* Masha Zeziuliia for the months of my favorite Friday sitcom (i.e., our Zoom meetings), which seem to have receded, but I am sure the producers will come up with one or two Christmas specials. And yes, the calls might have ended, but the Telegram stickers will surely remain with us forever.

To Sanya Snorkin(g), thank you for your vitality and for teaching me how to speak to the code, as well as when to press Ctrl-Z several times at the right time and place. To Nikita Koritskii, thank you for our countless "Where where", "Вот объективно", and for pointing me toward good quantum field theory textbooks. They are on my bookshelf now, and it seems like just after submission is a good time to finally do some reading. To both of you, thank you for our morning check-up emails interleaved with philosophical musings, which will undoubtedly be published one day alongside Marcus Aurelius's *Meditations*.

Apart from thanking the people I knew before Oxford, I also want to of course acknowledge those I met during my time there. I would like to thank my supervisor, Prof. Alex Lvovsky, for accepting me as a DPhil student and inadvertently providing me with a topic outside of his area of expertise, which turned out to perfectly match my preferences. I am grateful for him teaching me how to explain and think about things by stripping away all the mathematics until only the bare physics remains. Additionally, I really appreciate him asking how I was doing on the first day of the Russian invasion and confessing that he couldn't stop doomscrolling either.

Of course, I would like to thank my groupmates for going through the ups and downs of COVID, group meetings, and jointly exploring how to manage our supervisor: Tom Barrett, Parth Patel, Giorgio Maltese, Nastia Pushkina, Jose da Costa Filho, Xianxin Guo, Jernej Frank, and Kaden Bearne. I would especially like to thank Mert Esencan for his wondrous Turkish coffee machine and Matty Filipovich for being a great teammate both when writing a paper and rowing, as

well as for meddling precisely when needed. Thanks to Sasha Duplinskii for being the official Her Royal Majesty purveyor of top-notch memes. Special thanks go to my Bachelor student, Lennart Bürger, for being a joy to work with, but also, for teaching me how to be a better and kinder supervisor.

And, of course, to my fellow traveler *Dr* James Spall, who was kind enough to pick me up from library confinement and provide a sympathetic ear during COVID lunches when I struggled to find research ideas. He introduced me to the wonders of British culture and politics, including those damn pub games. In acknowledging James, I also express my gratitude to the Linacre (aka Hufflepuff) team of kind and funny people I had the pleasure of meeting through him: Charli Cox, Laurence Scheunemann, Valentin Darre, Alice Turner, Laura Martin, Kayman Siva, Charlie Newton, Simon Merchant, Chloe Miossec, and Emil Grove Dyrvik.

Through the Oxford Physics Department, I also met a few other remarkable people beyond my groupmates. I would like to thank Jan Ole Ernst — when I sometimes lost confidence in my self-worth as a scientist, our accidental chats in the corridor about quantum computing helped to restore my hope. To Mark IJspeert, thank you for being an enjoyable and thoughtful partner in lab demonstrations. To Henrik Dreyer, thank you for sharing your story of academic struggle and your passion for changing the world, as well as teaching me about strongly correlated electrons. Finally, I would like to thank Prof. Steve Simon for his bright, lucid, engaging, and equally important, humorous teaching style. I wish all courses could blend actual knowledge with important (and often funny) context to the extent he does. I learned a lot from his Condensed Matter and Topological Quantum classes.

One of the brightest spots on my academic path was my summer residency at Xanadu, where I experienced an incredibly stimulating, welcoming, and enjoyable working environment. Before arriving there, I struggled to produce any decent results, but during my two-month stay in Toronto, I laid the groundwork for a large part of this thesis. For that, I am immensely grateful to Juan Miguel Arrazola, who offered me the position and supported me throughout the process, especially

when I faced challenges translating the results into a paper. He is a man who truly knows what he is doing and I wish him and his family an exciting future.

I would also like to take this opportunity to thank all the summer residents who made this journey joyful. A special thanks goes to Korbinian Kottmann, thank you for helping me discover that I actually *love* karaoke bars, as well as for being the first person to tell me that he also doesn't understand all the buzz around classical shadows. To Matija Medvidović, thank you for your gift as a comedian, teaching me more about neural quantum states and Hamiltonian sampling, and showing me how to play ping-pong at a higher level. Speaking of which, credit for my ping-pong skills also goes to Davide Castaldo, who additionally sent me an important reference, without which this thesis wouldn't have been possible. It was also funny to discover that we wholeheartedly dislike the same paper.

Apart from fellow residents, I would like to thank Ilan Tzitrin for his incredible, if not exquisite, sense of humor, and our chat on creative writing. And to Anna Pressler, thank you for being a great housemate in otherwise somewhat lonely Toronto, as well as for opening up the world of asparagus to me.

For another aspect of my academic journey, I would like to thank Markus Schmitt, who invited me for a postdoc interview but, funnily enough, did *not* offer me a position. Despite this, he kindly agreed to become my mentor when I struggled to organise my thoughts, eventually becoming a true scientific collaborator. I really appreciate him sharing his expertise in neural quantum states, as well as his calm presence and positive outlook. I doubt I would have been able to compose this thesis without him, and I look forward to future discussions with him.

To conclude the academic acknowledgements, I would like to express my deep gratitude to Prof. Eugene Demler. He offered me a postdoc position and, when it became clear I wouldn't be able to submit the thesis on time, allowed me to write it up while working on his projects. This means a lot to me, and I can't wait to fully dedicate myself to the research at ETH once I finish writing this acknowledgement section and submit. I would also like to thank each member of the Demler group for being welcoming and entertaining top-class physicists; every conversation with any

of you is a joy: Alex Gómez Salvador, Jonathan Curtis, Luka Školč, Radu Andrei, Yi-Fan Qu, Filip Marijanović, Aaron Müller, Arthur Christiansen, Duilio De Santis, Ivan Morera Navarro, Sambuddha Chattopadhyay, Luuk van der Berg, and Ola Carlsson. To Utso Bhattacharya, thank you for being an excellent collaborator and padel partner, as well as for simply being a good friend to me.

I would also like to acknowledge some seemingly unrelated but important parts of my Oxford stay. First, I want to thank the people behind the scenes of everyday Wikipedia life. Writing is a labour for me, and I am truly impressed by those who spend their spare time creating encyclopedic content free for everyone. I don't know what kind of person I would have been without access to so much knowledge at my fingertips. I would also like to thank the Oxford University Ceilidh Band providing an opportunity for me to uncover my deep love for ceilidhs, which I believe are one of the most powerful kinds of spells, capable of cheering up anyone who is feeling severely down.[3] Another thanks goes to the man behind the counter at Itsu on Cornmarket Street. He always greeted me warmly and occasionally gave me free items when I would rush in just minutes before closing after a long evening at work, hoping to grab some half-price sushi. Once, when I mentioned that Rebekka was vegetarian and no vegetarian options were left, he even offered me half of his shelved dinner. To my deep shame, I don't even know his name, but to me, he embodies the kind spirit of Oxford.

Another important part of my Oxford experience was its collegiate system, which allowed me to meet an overwhelming number of talented and bright people. I must be honest: I chose St Edmund Hall (affectionately known as Teddy Hall) as my college of preference simply because I was tempted by its claim of being "the oldest surviving academic society to house and educate undergraduates in any university". However, in hindsight, I am inclined to believe it was the Sorting Hat directing me to Gryffindor (Teddy's colors are gold and maroon!) where "dwell the brave at heart". And, of course, being in Teddy meant a lot of rowing, which I think is truly the sport dear to my heart.

[3]Unfortunately, no amount of ceilidhs would help against depression. So, if one is feeling down for an extended period of time, reaching out to a specialist is a perfectly normal way to go.

I would like to thank George Heywood for kindly letting me occupy his room, which had a much better WiFi signal, during the first lockdown. To Jenna Elliott, thank you for teaching me when to say "not quite" and for bringing me and Yusuf to an 11 pm ice hockey taster session. I don't think I enjoyed ice hockey much, but the cycle back through the empty night city was definitely a prologue to many solo rides I took during the first coronavirus spring. To James Morley, thank you for being a diligent and kindhearted student. It was great to refresh my knowledge of undergraduate physics courses with you. To Václav Janeček, thank you for the incredibly entertaining lunches at Teddy, which taught me that instead of quietly reading my book, I can learn much more by engaging with the people around me.

To Fernando Jimenez-Gallardo, for adding a little bit of mayhem in any endeavour, as well as for teaching me that I can always do a law conversion degree and allegedly enjoy my life much more. To Thijs van der Plas and Julia Carver, thank you for showing me how to cook ridiculously tasty food out of thin air. To Yana Lishkova, thank you for being my first guide in Oxford. I guess your Cambridge undergrad should have come in handy then ;) To Zubin Deyal, thank you for conquering the hearts of my relatives with your wide smile, which is simply a reflection of your cheerful nature. To Yusuf İkbal Oldaç and James O'Donovan, thank you for your hilarious stories and your talent to take those neat "we are fashion bloggers, but funny"-style photos.

To Raggy Ravichandran, thank you for an unforgettable drive to Henley T&V. Being under the shower rain on an actual Henley course in the middle of the British summer (whose mild +20°C temperatures I grew to love) is one of my fondest Oxford memories. To Giorgos Kalantzis, thank you for joining many morning ergs, even though they often meant sacrificing proper sleep. I'm sorry I never joined you for a Teddy breakfast. I grew to like them in my last year and wish we could have enjoyed some of them together. To Olly Chen, thank you for your determination in work and rowing, and for your readiness to help others. To Mark Ewanz Rocher, thank you for being a role model of an athlete and a PhD student, who simply *gets*

things done. To Anya Glazer, thank you for your kindness and for always finding the right words, as well as for adding a creative splash to the life of SEHBC.

To Rohit Sahasrabuddhe, thank you for your cheerful presence and our discussions on the statistical physics of society (I still need to learn what your research is about!). To Nicco Fontana, thank you for our discussions on emergence and for being our embodied battle cry during the City Bumps regatta. To Marek Grzesiuk, thank you for being an absolute legend, defying common sense and the laws of physics with your jaw-dropping erg scores. To Giovanni Rolandino, thank you for being almost prohibitively Italian in all photos and for having a generous and sincere heart, always ready to help.

To Raghav Khaund, thank you for being truly the kindest person I've ever met. Sometimes I think you are *the* keeper, the Atlas holding the world, a third rescue ranger coming to help Chip and Dale (or at least to save a morning outing). To Ambre and Iris Bertrand, thank you for being two rays of sunshine, not only teaching me the word "peppy", but also embodying it with your personal example, and, of course, for sharing the love for a good boogie! To Olly Beaven, thank you for our cycles back from Abingdon, revealing our shared interest in meditation, The Smiths, cooking, and, of course, clotted cream. These belong to my fondest memories too. To Cyril Schroeder, thank you for the most beautiful rendition of "Auld Lang Syne" the world has ever witnessed. My friend, auld acquaintance should *not* be forgot.

I would also like to acknowledge the Oxford University Ukrainian Society. They made the start of the Russian invasion much less scary for me, as I found people to be busy with. And oh boy, we were busy! The Ukrainian flags flying high everywhere in Oxford, a rally of a thousand people on the Radcliffe Camera square, and a retail spot full of goods brought by local people for our donation drive speak for themselves. To Roma Tokaryk, thank you for being the best auctioneer in the world, gathering donations out of thin air. To Emma and Charly Mateo, thank you for carrying our events on your shoulders (sometimes quite literally, from a car). I owe you all an "Honourable Banderivets" medal. To Olya Homonchuk and Mats van Es, thank you for being such a focused and proactive duo who not only runs

events from start to finish but also convinces her employer to double the donations we gathered. To Vika Khalanchuk, thank you for being such an avid fighter on the cultural front. To Zhora Pervushin, thank you for halting your bike ride to check up on me when I needed it the most, and, of course, for your ability to put a fine joke in the middle of any WhatsApp conversation. To Artur Doschchyn, thank you for all the incredible banderivets jokes you told me as a distraction while we walked from a protest near the Russian Embassy in London to a protest near 10 Downing Street. To Nastia Vasylchenkova, thank you for being in London and reviving all the good memories from the good old times of school Olympiads. And, of course, to Stesha Badovska for completely shattering my idea of being the only Crimean person in Oxford, but also for giving me a chance to reconnect with my homeland, as well as for being an inexhaustible source of energy, ready to move mountains.

I would also like to thank my Russian friends. To Vasily Sorokin, thank you for our sometimes desperate library sessions. To Eugenia Vorobeva and Dzhirgala Badmaeva, thank you for our impromptu Christmas trip to Edinburgh, a city where you can almost see fairies gently lighting the streets. To Lena Racheva, thank you for stepping up to ask a question to a director of one of Serhii Zhadan's books when I was worried that I, as the host, would be the only one left. To Sergei Zotkin, thank you for your eagerness to help others and our kitchen discussions on how to survive a PhD in an unknown field.

I also have four friends, with whom we were trying to stay close together in the first days of invasion, so that we had somebody to live through the pain and grief with. To Anya Vlasenkova, thank you for being our makeup guru, but also for increasing my chances in quiz games, where now for a British composer I have *three* options: Handel, Elgar and *Britten* (the topic of Anya's Master's thesis). To Anya Kogan, thank you for coming to help when I needed it the most and for checking up on me whenever you saw me through the Physics Department window, ensuring there was a head attached to my legs. To Liza Belkina (кукусики!!!), thank you for our salsa classes and our walks back after, as well as for "ничё-ничё", also known as the most powerful therapeutic technique. And of course to Borya

Shteynas, thank you for your clear explanations of physical problems, for your all-encompassing humanitarian knowledge, and the language of literary quotes so familiar to me. Thank you also for the bicycle rides "with a theme" and long summer evenings in the Physics Department, when nothing worked, but at least there was a pint to grab afterward with a friend. It is hard to imagine what Oxford would have been without you guys.

A special credit goes to a group of people notoriously known as "Berry mugfins in the Cotswolds", or, in some dossiers, "Cardi-gang". These people are my wonderful housemates, who went with me through trials and tribulations of being PhD students. It is hard to believe there was a period in my life when I could come home in the evening and get a hearty dose of friendly banter. Or, when things took a darker turn, find advice and a sympathetic ear. And, of course, a big fat birthday (cheese)cake!

So, to Marie-Claire Jalaguier and Lukáš Melichar, thank you for being an absolutely iconic rowing couple, Her Royal Majesty suppliers of Scottish accent to our household, and valiant explorers of meal plan companies.

To *Dr* Paul Hintermayer, thank you for teaching me that if something looks like a frog, swims like a frog, and croaks like a frog, then it *still* doesn't have to be a frog. Thank you as well for enlightening me and changing my worldview twice: first, by introducing me to scones just before the start of the coronavirus, and second, three years later, by showing me how much better they taste when toasted.

To *Dr* Lizzy Jones, thank you for making me believe in magic each time you touched the dough, for the alchemy happening in our kitchen (stingy nettle beer! What sort of philosopher's stone is that?!). And, of course, thank you for all your cheeky sayings that are destined for eternity. In other words, I solemnly confirm that potatoes are, and will always be, delicious in any form.

To *Dr* Ioannis (Giannhs) Spanos, thank you for being that cool uncle I never had. When we first met I wasn't sure whether you were slightly crazy or just joking. To an extent, I am still not sure, but these four years were worth it anyway. I could always count on you to release the pressure with an inappropriate joke when needed, but more importantly, I could always count on you in general.

To *Dr* Lucy Joy Cornell, who is a joy indeed. Hey Luc, thank you for agreeing to join me in my first ceilidh, for exposing me to "Hamilton" and all our subsequent sing-alongs, as well as making all our Dr Z trips happen. But also, thank you for taking such good care of me, whether by teaching me how to cook a proper carbonara (it's good, but the M&S one is also dear to my heart) or gently checking up on me when I would come home and silently stare at my dinner. And, of course, I would like to express my gratitude to Denise and David Cornell who were second in line after Lucy to take care of all our household!

Now, I want to acknowledge my closest people: my family. My mama Alyona, my papa Oleg, my grandma Anya and grandpa Misha, my second grandma and grandpa Valya and Kolya, and my late grandma Galya and grandpa Bronya. Without you, without your relentless and unconditional support *none* of this would have happened. I have only recently started to realise how much you have to deal with, and how lucky I am to have you as a family. Thank you for making miracles part of my life, be it unfathomable Ded Moroz presents (he always seemed to narrowly escape me!), or completing a PhD in Oxford. Малышевы, Андрахановы, и Косты, у меня всё хорошо! Я по вам скучаю, и я надеюсь скоро настане день, закінчиться війна, и я вас увижу. И да, бабушка Аня, я покушал!

Finally, I would like to thank my Rebekka for seeing through it all with me, but also, of course, for so much more than that. Thank you for saying things I was about to say myself, for pointing out to me the enchanted lanes I was about to point out to you, for your laughter during our little dances, for all the silly photos on my phone and overflowing warmth in my heart. I am very happy to have met you. And yes, this "quite a ride" is coming to an end, yet this is but a stop to refuel the car and ourselves (cheeky Maccies?). We depart for the next adventure in three... Two... One...

Go!

References

[1] Paul Adrien Maurice Dirac and Ralph Howard Fowler. "Quantum mechanics of many-electron systems". In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 123.792 (1929), pp. 714–733. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.1929.0094. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1929.0094.

[2] Giuseppe Carleo and Matthias Troyer. "Solving the quantum many-body problem with artificial neural networks". In: *Science* 355.6325 (2017), pp. 602–606. URL: https://science.sciencemag.org/content/355/6325/602.

[3] Di Luo et al. *Autoregressive Neural Network for Simulating Open Quantum Systems via a Probabilistic Formulation*. 2020. arXiv: 2009.05580.

[4] Markus Schmitt and Markus Heyl. "Quantum many-body dynamics in two dimensions with artificial neural networks". In: *Physical Review Letters* 125.10 (2020), p. 100503.

[5] Kenny Choo et al. "Symmetries and Many-Body Excitations with Neural-Network Quantum States". In: *Phys. Rev. Lett.* 121 (16 Oct. 2018), p. 167204. URL: https://link.aps.org/doi/10.1103/PhysRevLett.121.167204.

[6] Giacomo Torlai et al. "Neural-network quantum state tomography". In: *Nature Physics* 14.5 (2018), pp. 447–450. URL: https://doi.org/10.1038/s41567-018-0048-5.

[7] Juan Carrasquilla et al. "Reconstructing quantum states with generative models". In: *Nature Machine Intelligence* 1.3 (2019), pp. 155–161. URL: https://doi.org/10.1038/s42256-019-0028-1.

[8] R. B. Laughlin. "Anomalous Quantum Hall Effect: An Incompressible Quantum Fluid with Fractionally Charged Excitations". In: *Phys. Rev. Lett.* 50 (18 May 1983), pp. 1395–1398. URL: https://link.aps.org/doi/10.1103/PhysRevLett.50.1395.

[9] Ulrich Schollwöck. "The density-matrix renormalization group in the age of matrix product states". In: *Annals of Physics* 326.1 (2011). January 2011 Special Issue, pp. 96–192. URL: https://www.sciencedirect.com/science/article/pii/S0003491610001752.

[10] Federico Becca and Sandro Sorella. *Quantum Monte Carlo Approaches for Correlated Systems*. Cambridge University Press, 2017.

[11] Filippo Vicentini et al. "NetKet 3: Machine Learning Toolbox for Many-Body Quantum Systems". In: *SciPost Phys. Codebases* (2022), p. 7. URL: https://scipost.org/10.21468/SciPostPhysCodeb.7.

[12] Román Orús. "Tensor networks for complex quantum systems". In: *Nature Reviews Physics* 1.9 (2019), pp. 538–550.

[13] J. Eisert, M. Cramer, and M. B. Plenio. "Colloquium: Area laws for the entanglement entropy". In: *Rev. Mod. Phys.* 82 (1 Feb. 2010), pp. 277–306.

[14] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[16] Federico Becca and Sandro Sorella. *Quantum Monte Carlo Approaches for Correlated Systems*. Cambridge University Press, 2017.

[17] James Stokes et al. "Quantum Natural Gradient". In: *Quantum* 4 (May 2020), p. 269. URL: https://doi.org/10.22331/q-2020-05-25-269.

[18] Johannes Jakob Meyer. "Fisher Information in Noisy Intermediate-Scale Quantum Applications". In: *Quantum* 5 (Sept. 2021), p. 539. URL: https://doi.org/10.22331/q-2021-09-09-539.

[19] James Stokes et al. "Quantum Natural Gradient". In: *Quantum* 4 (May 2020), p. 269. URL: https://doi.org/10.22331/q-2020-05-25-269.

[20] Ao Chen and Markus Heyl. *Efficient optimization of deep neural quantum states toward machine precision*. 2023. arXiv: 2302.01941 [cond-mat.dis-nn].

[21] Riccardo Rende et al. *A simple linear algebra identity to optimize Large-Scale Neural Network Quantum States*. 2023. arXiv: 2310.05715 [cond-mat.str-el].

[22] Jason Ansel et al. "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. URL: https://pytorch.org/assets/pytorch2-2.pdf.

[23] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[24] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[25] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[26] Jiuxiang Gu et al. "Recent advances in convolutional neural networks". In: *Pattern recognition* 77 (2018), pp. 354–377.

[27] Zachary C Lipton, John Berkowitz, and Charles Elkan. "A critical review of recurrent neural networks for sequence learning". In: *arXiv preprint arXiv:1506.00019* (2015).

[28] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[29] Mohamed Hibat-Allah et al. "Recurrent neural network wave functions". In: *Physical Review Research* 2.2 (2020), p. 023358.

[30] Kenny Choo, Titus Neupert, and Giuseppe Carleo. "Two-dimensional frustrated J_1-J_2 model studied with neural network quantum states". In: *Phys. Rev. B* 100 (12 Sept. 2019), p. 125124. URL: https://link.aps.org/doi/10.1103/PhysRevB.100.125124.

[31] Yusuke Nomura. "Helping restricted Boltzmann machines with quantum-state representation by restoring symmetry". In: *Journal of Physics: Condensed Matter* 33.17 (2021), p. 174003.

[32] Christopher Roth and Allan H MacDonald. "Group convolutional neural networks improve quantum state accuracy". In: *arXiv preprint arXiv:2104.05085* (2021).

[33] Luciano Loris Viteritti, Riccardo Rende, and Federico Becca. "Transformer variational wave functions for frustrated quantum spin systems". In: *arXiv preprint arXiv:2211.05504* (2022).

[34] Hannah Lange et al. *Neural network approach to quasiparticle dispersions in doped antiferromagnets*. 2023. arXiv: 2310.08578 [cond-mat.str-el].

[35] Michael Y. Pei and Stephen R. Clark. *Specialising Neural-network Quantum States for the Bose Hubbard Model*. 2024. arXiv: 2402.15424 [cond-mat.quant-gas].

[36] Jannes Nys, Gabriel Pescia, and Giuseppe Carleo. *Ab-initio variational wave functions for the time-dependent many-electron Schrödinger equation*. 2024. arXiv: 2403.07447 [cond-mat.str-el].

[37] Corey Adams et al. "Variational Monte Carlo Calculations of $A \leq 4$ Nuclei with an Artificial Neural-Network Correlator Ansatz". In: *Phys. Rev. Lett.* 127 (2 July 2021), p. 022502. URL: https://link.aps.org/doi/10.1103/PhysRevLett.127.022502.

[38] Alessandro Lovato et al. "Hidden-nucleons neural-network quantum states for the nuclear many-body problem". In: *Phys. Rev. Res.* 4 (4 Dec. 2022), p. 043178. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.4.043178.

[39] Y. L. Yang and P. W. Zhao. "Deep-neural-network approach to solving the ab initio nuclear structure problem". In: *Phys. Rev. C* 107 (3 Mar. 2023), p. 034320. URL: https://link.aps.org/doi/10.1103/PhysRevC.107.034320.

[40] David Pfau et al. "Ab initio solution of the many-electron Schrödinger equation with deep neural networks". In: *Phys. Rev. Res.* 2 (3 Sept. 2020), p. 033429. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.2.033429.

[41] Jan Hermann, Zeno Schätzle, and Frank Noé. "Deep-neural-network solution of the electronic Schrödinger equation". In: *Nature Chemistry* 12.10 (2020), pp. 891–897. URL: https://www.nature.com/articles/s41557-020-0544-y.

[42] Kirill Neklyudov et al. "Wasserstein Quantum Monte Carlo: A Novel Approach for Solving the Quantum Many-Body Schrödinger Equation". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 63461–63482. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/c8450235f227f136242f774b2799581f-Paper-Conference.pdf.

[43] Kenny Choo, Titus Neupert, and Giuseppe Carleo. "Two-dimensional frustrated J 1- J 2 model studied with neural network quantum states". In: *Physical Review B* 100.12 (2019), p. 125124.

[44] Kenny Choo, Antonio Mezzacapo, and Giuseppe Carleo. "Fermionic neural-network states for ab-initio electronic structure". In: *Nature Communications* 11.1 (2020), p. 2368.

[45] Thomas D. Barrett, Aleksei Malyshev, and A. I. Lvovsky. "Autoregressive neural-network wavefunctions for ab initio quantum chemistry". In: *Nature Machine Intelligence* 4.4 (Apr. 2022), pp. 351–358. URL: https://doi.org/10.1038/s42256-022-00461-z.

[46] Tianchen Zhao, James Stokes, and Shravan Veerapaneni. "Scalable neural quantum states architecture for quantum chemistry". In: *Machine Learning: Science and Technology* 4.2 (June 2023), p. 025034. URL: https://dx.doi.org/10.1088/2632-2153/acdb2f.

[47] Yangjun Wu et al. "NNQS-Transformer: an Efficient and Scalable Neural Network Quantum States Approach for Ab initio Quantum Chemistry". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '23. , Denver, CO, USA, Association for Computing Machinery, 2023. URL: https://doi.org/10.1145/3581784.3607061.

[48] Aleksei Malyshev, Juan Miguel Arrazola, and A. I. Lvovsky. *Autoregressive Neural Quantum States with Quantum Number Symmetries*. 2023. arXiv: 2310.04166 [quant-ph].

[49] Nobuyuki Yoshioka, Wataru Mizukami, and Franco Nori. "Solving quasiparticle band spectra of real solids using neural-network quantum states". In: *Communications Physics* 4.1 (2021), pp. 1–8.

[50] An-Jun Liu and Bryan K. Clark. *Neural network backflow for ab-initio quantum chemistry*. 2024. arXiv: 2403.03286 [physics.chem-ph].

[51] Tiago Mendes-Santos, Markus Schmitt, and Markus Heyl. "Highly Resolved Spectral Functions of Two-Dimensional Systems with Neural Quantum States". In: *Phys. Rev. Lett.* 131 (4 July 2023), p. 046501. URL: https://link.aps.org/doi/10.1103/PhysRevLett.131.046501.

[52] Markus Schmitt et al. "Quantum phase transition dynamics in the two-dimensional transverse-field Ising model". In: *Science Advances* 8.37 (2022), eabl6850. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.abl6850. URL: https://www.science.org/doi/abs/10.1126/sciadv.abl6850.

[53] Kaelan Donatella et al. "Dynamics with autoregressive neural quantum states: Application to critical quench dynamics". In: *Phys. Rev. A* 108 (2 Aug. 2023), p. 022210. URL: https://link.aps.org/doi/10.1103/PhysRevA.108.022210.

[54] Alessandro Sinibaldi et al. "Unbiasing time-dependent Variational Monte Carlo by projected quantum evolution". In: *Quantum* 7 (Oct. 2023), p. 1131. URL: https://doi.org/10.22331/q-2023-10-10-1131.

[55] Giacomo Torlai and Roger G. Melko. "Latent Space Purification via Neural Density Operators". In: *Phys. Rev. Lett.* 120 (24 June 2018), p. 240503. URL: https://link.aps.org/doi/10.1103/PhysRevLett.120.240503.

[56] Michael J. Hartmann and Giuseppe Carleo. "Neural-Network Approach to Dissipative Quantum Many-Body Dynamics". In: *Phys. Rev. Lett.* 122 (25 June 2019), p. 250502. URL: https://link.aps.org/doi/10.1103/PhysRevLett.122.250502.

[57] Filippo Vicentini et al. "Variational Neural-Network Ansatz for Steady States in Open Quantum Systems". In: *Phys. Rev. Lett.* 122 (25 June 2019), p. 250503. URL: https://link.aps.org/doi/10.1103/PhysRevLett.122.250503.

[58] E. S. Tiunov et al. "Experimental quantum homodyne tomography via machine learning". In: *Optica* 7.5 (May 2020), pp. 448–454. URL: https://opg.optica.org/optica/abstract.cfm?URI=optica-7-5-448.

[59] Murali K. Kurmapu et al. "Reconstructing Complex States of a 20-Qubit Quantum Simulator". In: *PRX Quantum* 4 (4 Dec. 2023), p. 040345. URL: https://link.aps.org/doi/10.1103/PRXQuantum.4.040345.

[60] Ekaterina Fedotova et al. "Continuous-variable quantum tomography of high-amplitude states". In: *Phys. Rev. A* 108 (4 Oct. 2023), p. 042430. URL: https://link.aps.org/doi/10.1103/PhysRevA.108.042430.

[61] Bjarni Jónsson, Bela Bauer, and Giuseppe Carleo. *Neural-network states for the classical simulation of quantum computing*. 2018. arXiv: 1808.05232 [quant-ph].

[62] Matija Medvidović and Giuseppe Carleo. "Classical variational simulation of the quantum approximate optimization algorithm". In: *npj Quantum Information* 7.1 (2021), pp. 1–7.

[63] Matija Medvidović and Javier Robledo Moreno. *Neural-network quantum states for many-body physics*. 2024. arXiv: 2402.11014 [cond-mat.dis-nn].

[64] Hannah Lange et al. *From Architectures to Applications: A Review of Neural Quantum States*. 2024. arXiv: 2402.09402 [cond-mat.dis-nn].

[65] Nicolas Le Roux and Yoshua Bengio. "Representational power of restricted Boltzmann machines and deep belief networks". In: *Neural computation* 20.6 (2008), pp. 1631–1649.

[66] Dong-Ling Deng, Xiaopeng Li, and S. Das Sarma. "Quantum Entanglement in Neural Network States". In: *Phys. Rev. X* 7 (2 May 2017), p. 021021. URL: https://link.aps.org/doi/10.1103/PhysRevX.7.021021.

[67] Xun Gao and Lu-Ming Duan. "Efficient representation of quantum many-body states with deep neural networks". In: *Nature Communications* 8.1 (2017), p. 662. URL: https://doi.org/10.1038/s41467-017-00705-2.

[68] Giuseppe Carleo, Yusuke Nomura, and Masatoshi Imada. "Constructing exact representations of quantum many-body systems with deep neural networks". In: *Nature Communications* 9.1 (2018), p. 5322. URL: https://doi.org/10.1038/s41467-018-07520-3.

[69] Yoav Levine et al. "Quantum Entanglement in Deep Learning Architectures". In: *Phys. Rev. Lett.* 122 (6 Feb. 2019), p. 065301. URL: https://link.aps.org/doi/10.1103/PhysRevLett.122.065301.

[70] Or Sharir, Amnon Shashua, and Giuseppe Carleo. "Neural tensor contractions and the expressive power of deep neural quantum states". In: *Phys. Rev. B* 106 (20 Nov. 2022), p. 205136. URL: https://link.aps.org/doi/10.1103/PhysRevB.106.205136.

[71] Kyle Sprague and Stefanie Czischek. "Variational Monte Carlo with Large Patched Transformers". In: *arXiv preprint arXiv:2306.03921* (2023).

[72] Moritz Reh, Markus Schmitt, and Martin Gärttner. "Optimizing Design Choices for Neural Quantum States". In: *arXiv preprint arXiv:2301.06788* (2023).

[73] Or Sharir et al. "Deep Autoregressive Models for the Efficient Variational Simulation of Many-Body Quantum Systems". In: *Phys. Rev. Lett.* 124 (2 Jan. 2020), p. 020503. URL: https://link.aps.org/doi/10.1103/PhysRevLett.124.020503.

[74] Attila Szabo and Neil S Ostlund. *Modern quantum chemistry: introduction to advanced electronic structure theory*. Courier Corporation, 2012.

[75] Sam McArdle et al. "Quantum computational chemistry". In: *Rev. Mod. Phys.* 92 (1 Mar. 2020), p. 015003.

[76] Xiang Li et al. "A Nonstochastic Optimization Algorithm for Neural-Network Quantum States". In: *Journal of Chemical Theory and Computation* 19.22 (2023). PMID: 37962975, pp. 8156–8165. eprint: https://doi.org/10.1021/acs.jctc.3c00831. URL: https://doi.org/10.1021/acs.jctc.3c00831.

[77] Xiang Li et al. *Improved Optimization for the Neural-network Quantum States and Tests on the Chromium Dimer*. 2024. arXiv: 2404.09280 [physics.chem-ph].

[78] Di Luo and Bryan K. Clark. "Backflow Transformations via Neural Networks for Quantum Many-Body Wave Functions". In: *Phys. Rev. Lett.* 122 (22 June 2019), p. 226401. URL: https://link.aps.org/doi/10.1103/PhysRevLett.122.226401.

[79] Zakari Denis, Alessandro Sinibaldi, and Giuseppe Carleo. "Comment on" Can Neural Quantum States Learn Volume-Law Ground States?"" In: *arXiv preprint arXiv:2309.11534* (2023).

[80] Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *International conference on machine learning*. PMLR. 2016, pp. 1747–1756.

[81] Aaron van den Oord et al. "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499* (2016).

[82] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[83] Wouter Kool, Herke Van Hoof, and Max Welling. "Ancestral gumbel-top-k sampling for sampling without replacement". In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 1726–1761.

[84] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*. Vol. 33. US Government Printing Office, 1954.

[85] Chris J Maddison, Daniel Tarlow, and Tom Minka. "A^* Sampling". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/309fee4e541e51de2e41f21bebb342aa-Paper.pdf.

[86] Di Luo et al. "Gauge-invariant and anyonic-symmetric autoregressive neural network for quantum lattice models". In: *Phys. Rev. Res.* 5 (1 Mar. 2023), p. 013216. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.5.013216.

[87] Kanav Setia et al. "Reducing Qubit Requirements for Quantum Simulations Using Molecular Point Group Symmetries". In: *Journal of Chemical Theory and Computation* 16.10 (2020). PMID: 32833450, pp. 6091–6097. eprint: https://doi.org/10.1021/acs.jctc.0c00113. URL: https://doi.org/10.1021/acs.jctc.0c00113.

[88] Sergey Bravyi et al. "Tapering off qubits to simulate fermionic Hamiltonians". In: *arXiv preprint arXiv:1701.08213* (2017).

[89] Ville Bergholm et al. "Pennylane: Automatic differentiation of hybrid quantum-classical computations". In: *arXiv preprint arXiv:1811.04968* (2018).

[90] Juan Miguel Arrazola et al. "Differentiable quantum computational chemistry with PennyLane". In: *arXiv preprint arXiv:2111.09967* (2021).

[91] Sunghwan Kim et al. "PubChem substance and compound databases". In: *Nucleic Acids Res.* 44.D1 (2016), pp. D1202–D1213.

[92] Aleksei Malyshev. *Autoregressive neural quantum states for quantum chemistry*. https://github.com/Exferro/anqs_quantum_chemistry. 2024.

[93] Jarrod R McClean et al. "OpenFermion: the electronic structure package for quantum computers". In: *Quantum Science and Technology* 5.3 (June 2020), p. 034014. URL: https://dx.doi.org/10.1088/2058-9565/ab8ebc.

[94] Qiming Sun et al. "Recent developments in the PySCF program package". In: *The Journal of Chemical Physics* 153.2 (July 2020), p. 024109. URL: https://doi.org/10.1063/5.0006074.

[95] Daniel G. A. Smith et al. "PSI4 1.4: Open-source software for high-throughput quantum chemistry". In: *The Journal of Chemical Physics* 152.18 (May 2020), p. 184108. URL: https://doi.org/10.1063/5.0006002.

[96] Sandeep Sharma and Ali Alavi. "Multireference linearized coupled cluster theory for strongly correlated systems using matrix product states". In: *The Journal of Chemical Physics* 143.10 (Aug. 2015), p. 102815. eprint: https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/1.4928643/15502606/102815_1_online.pdf. URL: https://doi.org/10.1063/1.4928643.

[97] Hong Gao et al. "Distributed Implementation of Full Configuration Interaction for One Trillion Determinants". In: *Journal of Chemical Theory and Computation* 20.3 (2024). PMID: 38314701, pp. 1185–1192. eprint: https://doi.org/10.1021/acs.jctc.3c01190. URL: https://doi.org/10.1021/acs.jctc.3c01190.

[98] Massimo Bortone, Yannic Rath, and George H. Booth. "Impact of conditional modelling for a universal autoregressive quantum state". In: *Quantum* 8 (Feb. 2024), p. 1245. URL: https://doi.org/10.22331/q-2024-02-08-1245.

[99] Marin Bukov, Markus Schmitt, and Maxime Dupont. "Learning the ground state of a non-stoquastic quantum Hamiltonian in a rugged neural network landscape". In: *SciPost Phys.* 10 (2021), p. 147. URL: https://scipost.org/10.21468/SciPostPhys.10.6.147.

[100] Javier Robledo Moreno et al. "Fermionic wave functions from neural-network constrained hidden states". In: *Proceedings of the National Academy of Sciences* 119.32 (2022), e2122059119.

[101] Stephan Humeniuk, Yuan Wan, and Lei Wang. "Autoregressive neural Slater-Jastrow ansatz for variational Monte Carlo simulation". In: *SciPost Phys.* 14 (2023), p. 171. URL: https://scipost.org/10.21468/SciPostPhys.14.6.171.

[102] Nikita Astrakhantsev et al. "Phenomenological theory of variational quantum ground-state preparation". In: *Phys. Rev. Res.* 5 (3 Sept. 2023), p. 033225. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.5.033225.

[103] Deidre Cleland et al. "Taming the First-Row Diatomics: A Full Configuration Interaction Quantum Monte Carlo Study". In: *Journal of Chemical Theory and Computation* 8.11 (2012). PMID: 26605580, pp. 4138–4152. eprint: https://doi.org/10.1021/ct300504f. URL: https://doi.org/10.1021/ct300504f.