ChatGPT

Designing a Generalized Quantum Simulation Engine for Atoms, Molecules, and Subatomic Particles

Introduction and Scope

Building a unified computational engine for **atomic, molecular, and subatomic modeling** is an ambitious goal. Such an engine must span quantum chemistry and nuclear/particle physics, accurately simulating electrons in molecules as well as quarks in hadrons. It would use fundamental **quantum mechanical principles** to treat a wide range of systems, from chemical reactions to subatomic particle interactions. The engine should be **modular and scalable**, serving as the core of an open-source tool that researchers (or even AI reasoning systems) can easily use and extend. In this report, we outline the design of this engine, covering its theoretical foundations, internal representations of quantum states, computational methods, acceleration techniques, and integration with reference benchmarks.

Key design goals include: **(1)** faithfully implementing quantum-mechanical models (wavefunctions, fields, etc.) for different regimes, **(2)** incorporating multiple **levels of theory** (from basic approximations to high-accuracy methods) for validation and comparison, **(3)** employing **acceleration strategies** (symmetry, effective theories, machine learning, etc.) to boost performance, and **(4)** making the system accessible and extensible via existing libraries and documentation. The ultimate vision is a **research-grade** simulation platform that can be invoked in automated workflows or AI assistants, enabling on-demand reasoning about molecular or subatomic phenomena.

Quantum-Mechanical Foundations

At its core, the engine rests on the **laws of quantum mechanics**. For atomic and molecular systems, this means solving the **time-independent Schrödinger equation** (or relativistic variants) for electrons and nuclei. In subatomic contexts, it means solving equations of **quantum field theory** (e.g. Quantum Chromodynamics for quarks and gluons) using lattice or continuum methods. Despite differences in scale, these share common principles: the system's **state** is described by a **wavefunction or quantum state vector** (or a **field** in QFT), and measurable quantities are expectation values of operators in that state.

Wavefunctions and Orbitals: In non-relativistic molecular quantum mechanics, the fundamental object is the wavefunction ψ. According to Born's interpretation, ψ itself is a complex probability **amplitude**, and its squared magnitude $|\psi|^2$ gives the probability density of finding particles in a given configuration [1] [2]. For example, a one-electron atomic wavefunction (an *orbital*) describes the probability distribution of an electron around a nucleus [2]. Wavefunctions of many-electron systems reside in a high-dimensional configuration space. The engine will likely represent many-electron states in a **basis** of one-electron orbitals – e.g. using linear combinations of atomic orbitals or plane-waves – building up many-body wavefunctions from these orbital building blocks. Atomic orbitals are simply one-particle wavefunctions (solutions of the hydrogenic Schrödinger equation); their $|\psi|^2$ yield familiar shapes (s, p, d,... orbitals) that indicate where an

electron is most likely to be found [2]. In general, the wavefunction encodes all quantum correlations and must obey required symmetries (such as antisymmetry for fermions like electrons, enforced via determinants as discussed below).

Born-Oppenheimer Approximation and Vibrations: For molecules, the engine can employ the Born-Oppenheimer separation of electronic and nuclear motion. Electronic wavefunctions are solved at fixed nuclei positions to yield potential energy surfaces, and then nuclei vibrational states are obtained by solving nuclear motion (typically as quantum harmonic oscillators in normal mode coordinates). A molecule's **normal modes** are independent vibrational patterns of the atoms [3] [4]. Each normal mode acts like a quantized harmonic oscillator with a vibrational wavefunction and quantized energy levels [4] [5]. The engine should include tools to compute normal mode frequencies and vibrational wavefunctions after an equilibrium structure is found. This involves constructing the mass-weighted Hessian matrix of the molecule and diagonalizing it to get mode frequencies and eigenvectors [6] [4]. By treating each normal coordinate Q_j as an independent 1D oscillator, the total vibrational wavefunction factorizes into a product $\psi_{vib}(Q_1, Q_2,\dots) = \prod_j \psi_{vib,j}(Q_j)$ [4], and the total vibrational energy is the sum of each mode's energy levels [5]. These vibrational quantizations let the engine simulate IR/Raman spectra and thermodynamic zero-point energies.

Field-Theoretical Description: For subatomic physics, especially Quantum Chromodynamics (QCD), the state of a system (like a proton or nucleus) is described by quantum fields (quark and gluon fields) rather than simple wavefunctions. The engine must switch to a **field-theoretical representation**, such as a lattice discretization of QCD fields. In **Lattice QCD**, the continuum space-time is replaced by a finite grid; quark fields live on lattice sites and gluon fields on links between sites [7]. The engine would then work with objects like gauge field configurations $U_{\mu}(x)$ (matrices on each link encoding the gluon field) and quark field amplitudes at each site. The dynamics follow from a Lagrangian or action – simulations use Monte Carlo sampling of field configurations weighted by e^{-S} (the Euclidean action) to compute observables [8] [9]. In essence, the engine's subatomic module would perform **path integral** evaluations via Monte Carlo, which is computationally demanding but ab initio. Lattice QCD is known to reproduce experimental results with high fidelity; for instance, it can calculate the proton's mass with <2% error from first principles [10]. This makes it the "gold standard" for QCD calculations, providing a benchmark for any approximate model of hadrons. The engine should treat Lattice QCD results as reference data to validate any faster approximate schemes for nuclear or particle interactions.

Second Quantization Framework: A unifying mathematical framework for the engine is **second quantization**, where states of multiple indistinguishable particles are described by occupation numbers in a set of modes (orbitals or momentum states). The engine can define creation and annihilation operators for electrons, allowing it to represent an N-electron wavefunction in a Fock space. This is natural for implementing methods like Configuration Interaction or Coupled Cluster (which are formulated in second quantization). Likewise, for bosonic fields or phonon modes, second quantization provides ladder operators. Adopting a second-quantized algebra of operators allows flexible handling of different particle species, and it aligns with how many quantum libraries (e.g. OpenFermion) operate [11] [12]. It also simplifies mapping to quantum computing representations if needed (by mapping fermionic operators to qubit operators, which OpenFermion facilitates [12]).

In summary, the engine's foundation is the rigorous quantum description of matter: whether it's electrons bound in atoms, atoms bound in molecules, or quarks bound in nucleons, the states are represented either

by wavefunctions (with appropriate symmetry and dimensionality) or by field configurations, and these states evolve or are solved for according to the relevant Hamiltonians.

Internal Representations of States and Observables

To simulate such varied systems, the engine must support multiple **representations** of quantum states:

- **Real-space and Basis Representations:** The wavefunction can be represented on a real-space grid or in a chosen **basis set**. For molecular electrons, Gaussian orbital bases or plane-wave bases are common. The engine might allow flexible choice of basis (e.g. Gaussian-type orbitals for quantum chemistry or plane-waves for periodic solids). In contrast, for fields (like lattice QCD) the natural "basis" is the value of the field on each lattice site/link, and Monte Carlo samples those field values.

- **Single-Particle vs Many-Body Basis:** The engine will often construct many-body states from single-particle orbitals. For example, a Hartree–Fock state is a single Slater determinant built from one-particle orbitals. More exact states can be expanded as superpositions of Slater determinants (configuration interaction expansion). The engine should internally represent a multi-electron state either by storing the coefficients of such an expansion or by an object that can generate amplitudes on the fly (as done in Coupled Cluster, which uses an exponential ansatz described below).

- **Density Matrices and Densities:** For certain tasks (like DFT or effective field theories), it is useful to represent the state via **densities** rather than explicit wavefunctions. In DFT, the key quantity is the electron density $\rho(\mathbf{r})$ – a function of three spatial coordinates instead of the $3N$ coordinates of a wavefunction [13] [14]. The engine's DFT module would focus on evolving $\rho(\mathbf{r})$ until self-consistent, rather than an explicit many-electron ψ. Similarly, in nuclear physics, one may use **density functional** approaches or mean-field (Hartree-Fock-Bogoliubov) with densities of nucleons. The engine should include data structures for densities and Green's functions as needed for these formalisms.

- **Hamiltonians and Operators:** A core part of the engine is assembling the **Hamiltonian operator** for the system of interest. This could be in second-quantized form (sums of creation-annihilation operators with integrals as coefficients), or in matrix form in a finite basis. For molecular systems, integrals like $\langle \chi_i \chi_j | \chi_k \chi_l \rangle$ (two-electron integrals in a basis ${\chi}$) are needed. The engine can leverage quantum chemistry libraries (like **Libint** or PySCF's integral routines) to compute these. For lattice QCD, the Hamiltonian is implicit in the path integral Monte Carlo – but for certain approximate subatomic models (like quark models or nuclear shell model), explicit Hamiltonian matrices can be constructed.

- **Vibrational and Reaction Coordinates:** In addition to electronic degrees, the engine should represent nuclear coordinates and allow moving them (for geometry optimization or reaction pathways). Representations like internal coordinates or normal mode coordinates help here. Also, the concept of **potential energy surfaces** (PES) is central – the engine may represent a PES (from electronic structure) and allow nuclear quantum motion on it (via vibrational wavefunctions or path integrals for nuclei in advanced cases).

By supporting these representations, the engine ensures it can switch between different levels of theory or regimes. For example, one might optimize a molecular geometry with DFT (density representation), then perform a high-accuracy CCSD(T) single-point energy (wavefunction expansion representation) as a benchmark. Or use a lattice QCD computation to extract an effective potential between two nucleons (field configuration representation distilled into a simpler operator form).

Ab Initio Methods and Algorithms

A truly general simulation engine must incorporate a **spectrum of methods**. On one end are highly accurate but expensive *ab initio* methods that serve as benchmarks; on the other end are faster, approximate methods and even data-driven models for scaling up to larger systems. Here we survey the key methods to be included:

Full Configuration Interaction (FCI): FCI is the brute-force exact diagonalization of the electronic Hamiltonian in a finite basis set. It considers **all possible Slater determinants** (all configurations of N electrons among the chosen orbitals), thereby capturing 100% of the electron correlation within that basis [15]. In essence, FCI directly solves the Schrödinger equation by finding the exact eigenvalues of the Hamiltonian matrix (dimension skyrockets combinatorially with system size). FCI is the ultimate reference for small systems – our engine can use FCI results to validate approximate methods on small molecules. We note that FCI is **formally exact** (in the limit of a complete one-electron basis) for the non-relativistic electronic structure problem [16] [17]. Because the FCI Hilbert space grows combinatorially (factorially) with the number of electrons and orbitals, FCI calculations are only feasible for very small molecules (often fewer than 12 electrons in ~12 orbitals). Indeed, "few full CI energies using flexible basis sets have been obtained" due to the extreme cost [16]. Nonetheless, including an FCI solver (e.g. via integration with a library like *PySCF*, which has an FCI module) is crucial for benchmarking. Our engine can use FCI to compute "exact" energies for small models, against which cheaper methods (HF, DFT, CC, ML models) are compared. For example, we could run an FCI on a water molecule in a minimal basis to have a ground truth to compare a DFT energy against.

Hartree–Fock (HF) and Mean-Field Methods: Hartree–Fock is the starting point for most electronic structure methods [18]. It uses a **single Slater determinant** as a variational ansatz for the N-electron wavefunction, optimizing the orbitals to minimize the energy. HF accounts for the Pauli principle (via the determinant) but treats electron–electron interactions only in an average way (missing dynamic correlation). The engine will include an HF module (Roothaan equations solution) to provide initial orbitals and reference states. HF is relatively fast (N^4 scaling with system size for standard algorithms) and provides orbital energies and molecular orbitals that can be used in post-HF methods. Importantly, HF provides the **reference Slater determinant** for more advanced correlation methods like Coupled Cluster and CI. In many cases, a single Slater determinant already captures a significant portion of the total bonding energy (and the rest – the correlation energy – is added by post-HF methods). The HF method also establishes the *Fermi level* and canonical MOs which are useful for interpreting chemical reactivity. Our engine might use existing implementations from libraries (e.g. PySCF or Psi4's SCF modules) to get robust and optimized HF calculations.

Coupled Cluster (CC) Methods: Coupled Cluster is a **high-accuracy correlation method** that has become the gold standard for molecular energetics. In particular, CCSD(T) – Coupled Cluster with singles, doubles, and perturbative triples – is often called *"the gold standard of quantum chemistry"* due to its excellent accuracy across a broad range of molecules [19] [20]. CC uses an *exponential ansatz* for the wavefunction: $|

\Psi_{\rm CC}\rangle = e^T |\Phi_{\rm HF}\rangle$, where $|\Phi_{\rm HF}\rangle$ is the HF determinant and T is a cluster operator (e.g. $T_1 + T_2$ for CCSD) that generates excited determinants when expanded. This compact exponential form effectively sums an infinite series of many-body diagrams, yielding results that are often close to FCI but at a much reduced cost compared to FCI. CCSD(T) typically scales as $O(N^7)$ (where N is system size) [19] – very expensive, but still feasible for small-medium molecules. Our engine will incorporate CCSD(T) as the primary **high-accuracy energy** method for molecules. We will use it as a benchmark to calibrate faster methods (e.g., comparing a new ML surrogate's prediction to a CCSD(T) result on a test set). Because of the high cost, one cannot use CCSD(T) for very large systems directly, but there are linear-scaling approximations (local CC) that could be included later [21]. The engine might rely on libraries like *Psi4*, *PySCF*, or *NWChem* for the actual CCSD(T) implementation, as writing a CC solver from scratch is complex. The **accuracy of CCSD(T)** is well documented – it often yields reaction energies and geometries in agreement with experiment to within a few kJ/mol or better [22]. Indeed, many in the field treat CCSD(T)/complete-basis-set as a "truth" for chemistry problems (hence the term *gold standard*). We will ensure the engine can output CCSD(T) energies and perhaps density matrices, so that one can compare them to, say, a DFT energy for the same geometry.

Density Functional Theory (DFT): DFT is indispensable for treating larger systems because of its favorable cost-to-accuracy ratio. Instead of an explicit many-electron wavefunction, DFT uses the electron **density** $\rho(\mathbf{r})$ as the primary variable [13]. By the Hohenberg-Kohn theorems, the ground-state energy can be obtained by minimizing a universal energy functional $E[\rho]$ [13]. In practice, DFT requires approximating the exchange-correlation functional (the part of $E[\rho]$ not known exactly). The engine will include **well-established DFT functionals** such as *B3LYP* and *PBE0* (hybrid functionals), and perhaps GGA functionals like PBE or BP86. These serve as "workhorse" methods: for example, B3LYP has historically been one of the most widely used functionals in chemistry, often achieving decent accuracy (errors ~2–3 kcal/mol on small molecule benchmarks) [23]. PBE0 (also known as PBE1PBE, a hybrid with 25% HF exchange) is another highly regarded functional that often outperforms B3LYP, especially for inorganic chemistry problems [24]. By including multiple functionals, the engine allows users to compare results (e.g., see how a property changes with B3LYP vs PBE0 vs a coupled-cluster benchmark). DFT scales roughly $O(N^3)$ for typical algorithms (due to integration grids and formation of the Fock matrix), making it far more scalable than CCSD(T). With linear-scaling techniques, DFT can handle hundreds to thousands of atoms. Our engine will likely leverage existing DFT code – for instance, linking to *Quantum ESPRESSO* for plane-wave DFT on periodic systems, or using PySCF/Psi4 for molecular DFT. **Quantum ESPRESSO** is an open-source suite specifically for electronic-structure and materials modeling using plane-wave DFT and pseudopotentials [25]. It would enable our engine to simulate periodic crystals or surfaces with DFT, complementing the molecular Gaussian-basis DFT of Psi4/PySCF. In summary, DFT provides the **main practical simulation capability** for medium-to-large systems, with functionals like B3LYP and PBE0 giving a balance of accuracy and efficiency [23] [24].

Multireference and Advanced Quantum Chemistry: Some systems (e.g. transition metal complexes, bond breaking, etc.) require beyond single-reference methods. The engine can include *Complete Active Space Self-Consistent Field (CASSCF)* for capturing static correlation, and then use perturbation (CASPT2) or configuration interaction (MRCI) for dynamic correlation on top of that. These are niche but important for certain strongly correlated molecules. Additionally, **quantum Monte Carlo (QMC)** methods like Diffusion Monte Carlo could be included for an alternative high-accuracy approach (QMC treats many electrons with stochastic projection, often reaching near-CC accuracy and good scaling on parallel machines). QMC would utilize the same trial wavefunctions (Slater-Jastrow) which our engine can produce from HF orbitals plus a Jastrow factor.

Lattice QCD and Nuclear Physics Methods: On the subatomic side, the engine must incorporate methods to simulate hadrons and nuclei. **Lattice QCD** has been mentioned – practically, this means incorporating or interfacing with existing Lattice QCD codes (e.g., the engine might call an external lattice simulator written in C++/Fortran with high-performance optimizations). Lattice QCD uses importance sampling to compute correlation functions from which particle masses and matrix elements are extracted [26] [10]. This is extremely computationally intensive, typically requiring supercomputers; it won't be something a user runs on a laptop via our engine. Instead, our engine could make use of *precomputed lattice QCD results* or simple lattice routines for small lattice sizes for demonstration. Alternatively, **Effective Field Theories (EFTs)** can be used as a lighter-weight approach: for example, in low-energy nuclear physics, *Chiral Effective Field Theory* provides nucleon-nucleon interaction potentials that embody QCD's low-energy symmetry constraints. The engine could include parameterizations from chiral EFT for nuclear forces, allowing it to simulate light nuclei quantum-mechanically (by solving Schrödinger equation for nucleons with those potentials). Another method is the *Nuclear Shell Model*, which is basically configuration interaction for nucleons in an mean-field potential of a nucleus – conceptually similar to electronic CI but for protons/neutrons in nuclear orbitals. A unified engine might allow solving a small shell-model problem as well.

Quantum Algorithms (Optional): Though not strictly required, the engine can prepare for future integration with quantum computing algorithms. For example, it might use **OpenFermion** to output the fermionic Hamiltonian of a molecule in a form ready for quantum algorithms (like the qubit Hamiltonian via Jordan-Wigner transformation) [11] [12]. This would allow users to test quantum algorithms (like VQE or phase estimation) on the same Hamiltonians. OpenFermion serves exactly this purpose as "the electronic structure package for quantum computers" [11], providing tools to transform chemistry problems into quantum circuit form. While our immediate focus is on classical simulation, this forward-compatibility with quantum computing could future-proof the engine.

In summary, our engine will house a **suite of solvers**: from mean-field (HF/DFT), through high-accuracy post-HF (CCSD(T), FCI for small systems), to specialized methods for special cases (multireference, QMC), and including quantum field theory solvers for subatomic problems (lattice QCD, EFTs). To keep development manageable, we will heavily leverage **existing open-source libraries** for many of these methods, integrating them via a common Python interface. For instance, the engine might call **PySCF** for an FCI or CCSD(T) energy [27] [28], call **Psi4** for a high-spin open-shell DFT calculation (since Psi4 supports many quantum chemical methods) [29], or call **Quantum ESPRESSO** for plane-wave DFT of a crystal [25]. Each of these libraries has API access (Python bindings or command-line interfaces) that we can wrap. PySCF, for example, is designed as a Python framework with accessible modules for SCF, MP2, CC, etc., and is **lightweight and easily extensible** [27] [28]. Psi4 provides a powerful engine with Python scripting as well, covering a wide variety of methods and **efficient C++ backend for integrals and coupled-cluster**. OpenFermion, as mentioned, bridges to quantum computing algorithms. By uniting these under a single interface, users can seamlessly switch between methods and cross-validate results.

Acceleration Strategies and Approximations

Covering large systems or long time scales requires more than just choosing an efficient method – we often must invoke smart approximations or accelerations. Our engine will incorporate multiple **strategies to accelerate computations** without severely sacrificing accuracy:

- **Exploiting Symmetry:** Recognizing and utilizing symmetries of the physical system can reduce computational cost drastically. For molecular systems, identifying the **point group symmetry**

(spatial symmetry of the molecule) allows the engine to block-diagonalize Hamiltonian matrices and avoid repeating equivalent calculations. For example, integrals that are symmetry-equivalent are computed only once [30]. In point group C_{2v}, if the molecule has a mirror plane, integrals related by that reflection are equal and can be reused. Quantum chemistry programs routinely take advantage of this to speed up Hartree-Fock, CI, etc. [30]. Our engine will automatically detect the symmetry of a given molecular geometry and classify orbitals by symmetry labels (irreducible representations) [31] [30]. The benefit is both in reduced computation and easier interpretation of results (you can label states by symmetry, like "A1" or "T2g" representations). In solids or periodic systems, **translational symmetry** is exploited via Bloch's theorem: instead of N-atom unit cell calculations being $O(N^3)$, one works with **reciprocal space (k-points)** to only simulate a single cell with various phase factors. The engine's materials module (via Quantum ESPRESSO) will use k-point sampling to leverage periodicity, effectively reducing an infinite lattice to a representative set of points in the Brillouin zone.

Symmetry is also crucial in subatomic simulations: for example, imposing rotational symmetry (angular momentum conservation) in nuclear shell models, or isospin symmetry between protons and neutrons to reduce independent parameters. Our engine's nuclear module can use group-theoretical techniques to classify nuclear states by total angular momentum and parity, which simplifies configuration spaces by excluding combinations that don't yield the correct symmetry.

- **Effective Field Theories (EFTs):** An **effective field theory** approach simplifies a problem by focusing only on relevant degrees of freedom at the energy scale of interest, while systematically accounting for (or parameterizing) the effects of higher-energy processes [32] [33]. For instance, in nuclear physics, instead of explicitly simulating quarks and gluons for low-energy nuclear structure, one uses hadrons (protons, neutrons, pions) as degrees of freedom with interactions constrained by symmetry (chiral symmetry, etc.) and fit to data. This is a form of coarse-graining that can drastically reduce complexity. Our engine can incorporate **EFT-based potentials** (like a chiral nucleon-nucleon potential) to allow quantum mechanical simulation of nuclei without full QCD. Another example is in electronic structure: **pseudopotentials** (discussed next) can be viewed as an EFT concept – they eliminate core electrons (high-energy degrees) and replace them with an effective potential acting on valence electrons [34] [35]. The engine's architecture will allow swapping in an effective Hamiltonian in place of a fundamental one when appropriate, greatly accelerating calculations while retaining accuracy for low-energy observables. The guiding philosophy from EFT is *"make everything as simple as possible, but no simpler"* [36] – we include only necessary degrees of freedom and symmetries to reach the desired accuracy, and we have **expansion parameters** that quantify the neglected physics (e.g. expansion in (E/Λ) where Λ is a high-energy cutoff) [33]. This provides a path to improving the approximation systematically if needed (by adding higher-order terms). Effective Hamiltonians also appear in quantum chemistry (for example, the *CASPT2* method effectively builds an effective Hamiltonian for the active space by second-order perturbation). Our engine might automate constructing effective Hamiltonians for a subset of degrees of freedom given a full treatment of a smaller system – akin to integrating out high-frequency modes.

- **Pseudopotentials and Frozen Core Approximations:** In atomic and solid-state simulations, *pseudopotentials* (also known as **Effective Core Potentials, ECPs**) are a standard acceleration technique. The idea is to eliminate the need to explicitly treat tightly bound core electrons by replacing their effect with an effective potential acting on the valence electrons [34] [35]. Core electrons do not significantly participate in chemical bonding, yet they dramatically increase basis

set size and computational cost, and relativistic effects for core electrons can be important for heavy elements. By using an ECP, one can **reduce electron count** and also incorporate scalar relativistic effects implicitly. Our engine will have a library of pseudopotentials for the periodic table (e.g., norm-conserving pseudopotentials or projector augmented-wave datasets for planewave DFT, and effective core potentials for Gaussian basis calculations). When a user simulates a molecule containing, say, iodine, the engine can by default apply a pseudopotential to replace inner-shell electrons beyond the valence. The pseudopotential is constructed to reproduce the scattering properties (and valence orbital energies) of the eliminated core electrons [35]. Notably, a good pseudopotential ensures valence orbitals remain orthogonal to the (excluded) core orbitals and include the correct nodal behavior [35]. This approximation can cut down computational effort enormously – for example, a platinum atom with 78 electrons can be treated as if it had effectively 18 electrons if a pseudopotential represents the inner 60 electrons. The engine will allow toggling pseudopotentials on/off for testing. In cases where ultimate accuracy is needed, one might do an all-electron calculation (especially for lighter elements or when core polarization is relevant). For most routine simulations, however, pseudopotentials will be used by default as they "model [core electrons'] effects in some simpler way" without significantly affecting valence chemistry [34]. Quantum ESPRESSO's infrastructure for plane-wave DFT, for instance, inherently uses pseudopotentials to represent nuclei + core, because plane-wave basis is impractical for core orbitals. We'll integrate that capability. Similarly, for Gaussian-based calculations, we can utilize effective core potentials from standard libraries (the engine could interface with QChem's ECP data or Basis Set Exchange to fetch ECP definitions).

- **Imposing Physical Constraints and Symmetry in Ansätze:** When using variational approaches, carefully choosing the form of the **wavefunction ansatz** can accelerate convergence and improve accuracy. For example, using a **Slater determinant** automatically imposes the antisymmetry required by fermions (electrons) [37], whereas a simple Hartree product would violate the Pauli principle. The engine's default for multi-fermion systems will always use antisymmetrized ansätze (Slater determinants or combinations thereof). Additional known constraints can be built in: for instance, if we know the molecule's total spin and spatial symmetry, we can choose a wavefunction that is an eigenfunction of $\hat{S}^2$ and of the point group symmetry operations. This may involve symmetry-adapted linear combinations of determinants. By working in a symmetry-adapted basis, we reduce the number of free parameters and ensure the variational algorithm searches in the physically relevant subspace (no need to later project out symmetry contaminants). A concrete implementation: in Hartree-Fock, one can enforce e.g. a restricted closed-shell form (all orbitals doubly occupied with proper spin pairing) or use *symmetry-broken* solutions when appropriate. Our engine will provide options for **Restricted, Unrestricted, and Generalized Hartree-Fock** depending on spin-state requirements, etc.

- **Variational Ansätze & Tensor Networks:** Beyond single determinants, we consider **parametric ansätze** to capture correlation. The **Coupled Cluster ansatz** ($|\Psi\rangle = e^T | \Phi_{\rm HF} \rangle$) is one example that we have for high accuracy. Another class of ansätze gaining attention are **tensor network states**, like Matrix Product States (MPS) or more generally Projected Entangled Pair States (PEPS), especially for strongly correlated systems and in condensed matter contexts. An MPS (underlying the DMRG algorithm) can efficiently represent 1D/quasi-1D system wavefunctions. The engine could incorporate a DMRG solver (there are libraries like Block2 or ITensor) for molecules where an MPS is appropriate (often when there is a linear arrangement or a limited entanglement

structure). This again is an acceleration because it uses a more compact representation than CI in many cases.

- **Neural Network Quantum States (Machine-Learned Ansätze):** A cutting-edge approach is to use neural networks to represent wavefunctions. For instance, **FermiNet** and **PauliNet** are neural-network-based variational ansätze for many-electron wavefunctions that have shown promise in reaching near CC quality for atoms and small molecules [38]. They essentially use deep networks to encode the complicated dependency of ψ on all particle coordinates, while maintaining the required antisymmetry by construction (e.g., FermiNet outputs a matrix that is antisymmetrized via a determinant) [37] [38]. These **neural wavefunctions** can be extremely flexible, potentially generalizing across molecule configurations or learning compact representations of correlation. However, training them (via Variational Monte Carlo) is computationally heavy, albeit embarrassingly parallel. Our engine will include an interface to define and train such neural network wavefunctions. It might leverage frameworks like **PyTorch or JAX** to implement the networks and compute gradients (which are needed for the variational optimization of parameters) [39] [40]. For instance, we could integrate DeepMind's open-source code for FermiNet or a simpler network like a Restricted Boltzmann Machine for lattice models. The advantage of using PyTorch or JAX is that they provide **automatic differentiation** (autograd) and GPU acceleration, so one can efficiently compute the gradient of the energy (expected value of Hamiltonian) with respect to potentially thousands of network parameters [39] [40]. This is crucial for training neural ansätze. Moreover, JAX in particular can **just-in-time compile** computations and run NumPy-like code on accelerators (GPU/TPU) with ease [40], which is beneficial for scaling up VMC optimization. By including neural quantum states, our engine stays at the frontier of research – these methods may allow tackling systems that are beyond reach of traditional methods by learning from data or capturing correlation in a more compact functional form than polynomial expansion of determinants. For example, recent studies have shown a *single FermiNet* (with one determinant) can outperform a huge CI expansion of 64 determinants for bond-breaking in N_2 [41] [42], indicating the power of a good ansatz. Our engine's modular design (and use of Python ML libraries) will make it easy for researchers to plug in custom neural network ansätze as needed.

- **Machine Learning Surrogates and Potentials:** Apart from representing wavefunctions, machine learning can accelerate simulations by acting as *surrogate models*. One successful application is **machine-learned interatomic potentials**. Instead of computing *ab initio* energies and forces for every new molecular geometry (which could require a full DFT or CCSD calculation each time), one can train a neural network (or kernel model) on a set of reference calculations and then use it to predict energies for new geometries nearly instantly. Techniques such as **Behler-Parrinello Neural Network potentials**, **Gaussian Approximation Potentials (GAP)**, or graph neural network models like **SchNet** and **ANI** have demonstrated that one can achieve near-DFT accuracy at orders-of-magnitude lower cost [43] [44]. Our engine can incorporate an ML potential module: for example, one could train a potential on a particular molecule's conformational energy surface using CCSD(T) data and then use that potential to do molecular dynamics or optimize structures quickly, with the confidence that it's calibrated to high-level data. The engine would facilitate generating the training data (running many single-point calculations with say CCSD(T) via the methods above), feeding that to an ML training pipeline (likely using PyTorch or JAX for flexibility), and then deploying the resulting model for fast predictions. In essence, this is **transfer of computational effort**: do heavy computations offline to train an ML model, then use cheap model evaluations online during simulation. Another place ML surrogates help is in speeding up **DFT functional evaluations**.

Recently, there's interest in using ML to develop new exchange-correlation functionals or to **learn the DFT mapping** from density to energy by training on high-level data (so-called Δ-learning, where an ML model corrects a cheaper method to match a more expensive method's results) [45]. Our engine can offer a "DFT+ML" option where, say, a neural network functional (like DeePHF or DM21) is used instead of or alongside a traditional functional. Additionally, ML can assist in **optimizing basis sets**, extrapolating results to the basis set limit, or predicting good initial guesses for geometry optimization, all of which could be modules in the engine.

In the subatomic domain, ML surrogates could help emulate results of lattice QCD. For example, one could imagine training a neural network to predict nucleon-nucleon scattering phase shifts (or even the nucleon mass) as a function of some input parameters, effectively capturing the results of many lattice QCD runs. This could then serve as a quick evaluator to compare with experimental data or to include in a larger simulation (embedding the ML model as a component in a multi-physics simulation).

- **Effective Sampling and Reduced Models:** For dynamical simulations (e.g., molecular dynamics or path integral Monte Carlo for finite-temperature properties), acceleration techniques like **importance sampling, rare-event sampling (e.g. umbrella sampling)**, or using **coarse-grained models** can be integrated. While the question primarily focuses on static or ground-state calculations, a comprehensive engine could support time evolution or thermodynamic ensembles. For instance, one might run Born-Oppenheimer molecular dynamics where forces come from the quantum engine (DFT or ML potential), using larger timesteps via a multiple-timestep algorithm or constraining high-frequency modes (like using SHAKE algorithm for bonds). These ideas ensure that if the engine is extended to dynamics, it remains efficient.

Each of these acceleration strategies will be available as **options** that users can turn on as needed. A researcher might start with a brute-force approach for a small test system (say, no pseudopotentials, full configuration interaction) to gather baseline data, and then for production runs on larger systems, enable pseudopotentials, switch to DFT or ML potentials, and rely on symmetry and effective models to get results in a feasible time. The engine's design will emphasize **easy interchange of modules** – e.g., one can run a single SCF calculation with and without symmetry to see the speed difference, or easily swap a Hamiltonian with its effective version by a single parameter change. Extensive **documentation and references** will be provided for each approximation (for instance, citing the source of a pseudopotential dataset, or the paper introducing a particular ML model or ansatz), so users understand the provenance and validity range [34] [43].

Comparison with Trusted Reference Methods

An essential feature of our engine is the ability to **validate and benchmark** approximations against high-fidelity methods. We will ensure the engine can seamlessly compare its results with trusted references in each domain:

- **Validation Against FCI and Exact Diagonalization:** For small molecules or model systems (like the Hubbard model), the engine can perform full configuration interaction or exact diagonalization and use those results as a "ground truth" to gauge other methods. For example, we might compare the energy of a H_4 molecule in a minimal basis computed by a neural network ansatz versus the exact FCI energy – the difference tells us the ansatz's accuracy. Because FCI is exact (within a given one-particle basis) [17], any discrepancy quantifies the correlation missing in an approximate

method. We will also compare properties beyond energy, if possible: e.g., compare dipole moments, or orbital occupations, etc., from an approximate method to those from FCI. This helps in fine-tuning approximations or deciding if a cheaper method is reliable for the property of interest. The engine will likely output such comparisons in reports or automatically compute error metrics if asked (like "the CCSD energy is within 1 kcal/mol of FCI" or "DFT orbital energies deviate from HF by X").

- **Coupled Cluster (CCSD(T)) Benchmarks:** As discussed, CCSD(T) is the de facto standard for chemical accuracy [20]. Our engine will use CCSD(T) results as a baseline for medium-sized molecules (where FCI is impossible but CCSD(T) is still doable). For instance, when developing a new machine learning potential, one might train it on CCSD(T) energies of thousands of molecular geometries – here CCSD(T) provides the supervisory signal. Or if using a DFT functional, one might evaluate its performance by comparing to CCSD(T) reaction energies or barrier heights. We will include example benchmark sets (like the popular G2 or S22 sets of molecular energies) and make it easy to run CCSD(T) on them for reference [23]. This allows users to say, "method X achieves a mean unsigned error of Y relative to CCSD(T) on this benchmark." Because CCSD(T) itself has some residual error (it's not exact), for small systems we can cross-check CCSD(T) with FCI or experimental data to ensure quality. The engine's documentation will note that CCSD(T) is very accurate for single-reference systems (and give references to its known failures, e.g., multi-reference cases where it might break down).

- **Standard DFT Functionals as Baseline:** While DFT is an approximation, certain well-known functionals like B3LYP and PBE0 are so extensively used that they serve as a point of comparison. If our engine introduces a new approximate scheme (say a low-cost HF-D3 or an ML functional), we should compare its results to a reputable DFT functional on various properties. For instance, one might want to confirm that the new method at least matches B3LYP for bond lengths and vibrational frequencies of a test set. B3LYP is known to typically give bond lengths with small errors and energies within a few kcal/mol for many organic molecules [23]. If a new approach is worse than B3LYP across the board, it may not be attractive unless it has other advantages (like much lower cost). Thus, including these standard DFT references in our engine means a user can easily compute, say, the PBE0 energy and compare it to their new method's energy, ensuring they can contextualize the performance.

- **Lattice QCD and Experimental Data for Subatomic Physics:** In the nuclear/particle domain, **experimental measurements** and lattice QCD calculations act as the benchmark. Our engine will incorporate known results, such as the experimental deuteron binding energy, nucleon-nucleon scattering phase shifts, or hadron mass spectra from experiments and high-precision lattice computations [10]. For any approximate nuclear model integrated (for example, a simple potential model or a mean-field model), the engine can compare predicted observables to these reference values. Lattice QCD, being a first-principles nonperturbative method, provides numbers like the pion mass, proton mass, and so on, which our engine can use to calibrate effective models. Indeed, lattice QCD has **"successfully agreed with many experiments"**, e.g., reproducing the proton's mass with <2% error [10]. It also predicts phenomena like the quark-gluon plasma transition at certain temperatures [10]. So if our engine includes a simpler QCD-inspired model, we would verify it reproduces such known values within reasonable error.

- **Cross-Checks Between Methods:** The engine enables internal cross-checks. For example, we can compare a high-level **Moller-Plesset perturbation (MP2)** or CCSD result to a DFT result for the same

system to see how large correlation effects are. Or compare a **multi-reference** result to a single-reference CC to identify multi-reference character. Or check that an **EFT potential** calibrated to lattice or experimental data yields similar phase shifts in a two-body scattering calculation. By providing multiple methods under one roof, the engine encourages this practice of verification. Often, discrepancies between methods can highlight interesting physics: e.g., if DFT differs greatly from CCSD(T) for a certain type of system, it might indicate strong static correlation or a limitation of the functional.

- **Regression Tests with Library Documentation Examples:** We will incorporate standard examples from library documentation (e.g., Psi4 and PySCF examples of certain calculations) and ensure our engine reproduces those. For instance, Psi4 might list an expected energy for CCSD(T)/cc-pVTZ on water; our engine's output should match that to the precision given. This not only validates our integration of those libraries but also provides users assurance that the engine yields results consistent with known sources. Each method's documentation often provides reference values [20] which we can cite or include in our test suite.

Ultimately, by baking in these comparisons, the engine helps researchers trust new approaches. A user should be able to ask: *"How does my novel variational ansatz or ML potential perform compared to CCSD(T) or experiment?"* and get an automated report. Since we plan integration into AI reasoning tools, the engine could even **explain** differences – e.g., an AI agent using the engine might note "the DFT result is 5 kcal/mol off from the CCSD(T) benchmark, which is a typical error for DFT on this kind of system, likely due to insufficient treatment of dispersion." We envision an AI like Anthropic's Claude or others could query the engine for such comparisons and get back both numerical data and contextual analysis (with our curated references as a knowledge base). This level of insight will make the engine not just a black box number-cruncher, but a teaching and discovery aid.

Software Design and Integration

To achieve all the above, careful software design is needed. The engine will be written in a high-level language (Python) for ease of integration and modularity, with performance-critical parts in lower-level languages (C/C++/Fortran, or leveraging vectorized libraries and GPU support via PyTorch/JAX). Here we outline the design:

Modular Architecture: The engine will consist of loosely coupled modules, each handling a specific domain or task, for example:

- `QuantumChemistry` module – handling molecules and electrons. This module interfaces with PySCF, Psi4, and other quantum chemistry libraries. It will have sub-components like `SCFDriver` (for HF/DFT), `PostHF` (for MP2, CC, CI), `Properties` (for dipole moments, frequencies via numerical differentiation or analytic Hessians if available), etc. Using PySCF is advantageous because it is **Python-based and easily integrated**, and offers a **comprehensive set of electronic structure methods** (HF, DFT, MP2, CC, MCSCF, etc.) with the ability to customize or extend them [28]. Psi4 is similarly accessible via its Python API and could be used for methods PySCF lacks (or to double-check). We will hide the complexity behind a uniform interface: e.g., a function `compute_energy(method="CCSD(T)", basis="cc-pVTZ", molecule=water)` will internally call the appropriate library and return the energy. The user doesn't need to know if it was PySCF or

Psi4 or Q-Chem under the hood, but our documentation will cite the underlying library's manual for details (for instance, pointing to Psi4's documentation for what "CCSD(T)" entails) [20].

- `Materials` module – handling periodic boundary conditions and crystalline materials. This will likely wrap **Quantum ESPRESSO** (or potentially other plane-wave codes like ABINIT or SIESTA for localized basis). Quantum ESPRESSO can be called via input files and executed, but recently Python interfaces or direct libraries (via its API or through Atomic Simulation Environment, ASE) exist. We will include the ability to do a DFT calculation for a crystal using a selected pseudopotential and k-point mesh, then extract band structures, density of states, etc. All the heavy lifting is done by QE's well-optimized Fortran code; our engine just prepares input and parses output. We will cite QE's user manual for how to interpret results and what approximations are made (e.g., "using plane-wave cutoff of X Ry, k-point sampling YZ" etc.) [25]. This module makes it possible to treat, say, a catalyst surface or a bulk material and then perhaps use those results in a larger workflow (like computing reaction energies using a surface model).

- `SubatomicPhysics` module – covering lattice QCD and nuclear physics. For lattice QCD, we are likely to **interface with existing codes** from the Lattice community (such as the **Chroma library** for lattice QCD). Chroma, for example, provides data-parallel operations and can handle Monte Carlo generation of gauge field configurations [46]. Instead of reinventing that, we can call Chroma as a library (it has C++ interfaces we could wrap via pybind11 or similar). The engine might allow specifying a lattice size, coupling, quark mass, and then generating configurations and measuring observables (like a Wilson loop or hadron correlator). However, given the complexity, an alternative is to use stored configurations from literature or simplified models. For nuclear structure, we could code simpler routines (like solving the Schrödinger equation for a few-body system with a given potential, which is achievable with, say, variational methods or a partial-wave expansion). There are also libraries for nuclear EDFs (energy-density functionals) and the like, but those might be beyond initial scope. Regardless, this module will clearly separate the high-energy physics aspects from the chemistry aspects, as they have different requirements.

- `MachineLearning` module – providing tools for training and using ML surrogates. This will leverage **PyTorch** and **JAX** heavily. For example, if a user wants to train a neural network potential, this module will provide a class `NeuralPotential` with methods `train(training_data)` and `predict(coordinates)` which under the hood set up a PyTorch model, use autograd for training gradients, etc. PyTorch's strengths are ease of use and a large ecosystem (and it provides the necessary GPU acceleration and autodiff) [39]. JAX is great for composable function transformations (like JIT and vectorization) and can be used for certain things like automatically differentiating through an entire simulation (if needed for doing ML-driven adaptive simulations). We anticipate using **PyTorch for user-facing ML models** (since more scientists know its API) and possibly JAX internally for some performance-critical or research prototypes (noting that JAX effectively combines autograd + XLA for speed [40]). The ML module will also interface with data management – reading in reference data from disk, splitting into training/test, etc., to make the training process smooth for scientists who may not be ML experts. We will provide documented examples like training an ANI-style neural network on a small molecule dataset, referencing literature on QML (quantum machine learning) that demonstrates the efficiency gains [43].

- `Infrastructure` modules – things like an input parser, unit conversion, parallel job scheduling (e.g., if running many independent calculations, spread across cores or cluster), and result logging.

Since one aim is integration into AI tools, we will ensure the engine can be called programmatically (as a Python library or via an API) and returns data in structured formats (like JSON or Python dictionaries). This makes it easier for an AI system to parse results. For example, an AI agent might call a function to optimize a geometry and get back an object containing the optimized coordinates, the final energy, the method used, and references to sources (like "optimized with B3LYP functional [23]").

Use of Community Libraries and Documentation: As emphasized, we stand on the shoulders of giants by using community-validated libraries. Each integration will be accompanied by references to the library's own documentation or validation. For instance, when we use PySCF, we'll cite the PySCF paper and manual to note its capabilities and trustworthiness [27] [28]. PySCF is known for its lightweight, easily hackable nature and has been used in many research projects, which gives confidence. Psi4's documentation states it is designed for high-throughput quantum chemistry with robust implementations [29] – we'll lean on that for tasks like scanning a potential energy surface with many calculations. Quantum ESPRESSO's manifesto highlights its use of plane-waves and pseudopotentials for materials [25]; we will follow its recommended practices for convergence (e.g., increase plane-wave cutoffs or k-point density until energies converge, etc.). The **Chroma lattice QCD library** we mentioned comes from the USQCD collaboration and supports large-scale parallel LQCD calculations [46]. We will cite its manual for any parameters we expose (like the gauge action used, algorithms for Monte Carlo updates, etc.). By keeping these connections visible, advanced users can delve deeper into each component's theory and implementation via the cited sources, and our engine remains transparent.

Code-Level Implementation Example: To illustrate the integration, consider how a **geometry optimization** might be implemented in the engine's code. We might have:

```
molecule = Molecule.from_xyz("h2o.xyz")
method = QuantumChemistryMethod(name="B3LYP", basis="6-31G*")
optimizer = GeometryOptimizer(method=method, convergence={'grad_tol': 1e-4})
result = optimizer.optimize(molecule)
```

Under the hood, `QuantumChemistryMethod` could be a wrapper that calls Psi4 or PySCF. For B3LYP, it would set up a DFT calculation with the specified basis and functional [23]. The `GeometryOptimizer` might use a standard algorithm like Berny optimization (RFO method), calling `method.compute_energy_and_gradient(molecule)` each step. If we choose Psi4 as the backend for DFT, that function would call Psi4's Python API: e.g., `psi4.geometry(molecule.to_string())` then `psi4.energy("B3LYP/6-31G*")` and `psi4.gradient("B3LYP/6-31G*")`. The result object would contain the optimized geometry and final energy. It would also carry provenance information, e.g., result.method_provenance = "Psi4 1.4, B3LYP functional (Becke 1988 exchange [24] + Lee-Yang-Parr 1988 correlation)". This way, if an AI is reading the result, it could say: "Optimization completed with B3LYP/6-31G*. Sources indicate B3LYP is a widely-used hybrid functional known to give ~0.1 Å accuracy in bond lengths [23]."

Another example, for **neural network wavefunction optimization**, pseudo-code might be:

```
hamiltonian = Hamiltonian.from_pyscf(molecule, basis="STO-3G")  # get 1e and 2e
integrals
ansatz = NeuralWavefunctionAnsatz(n_electrons=hamiltonian.n_electrons,
n_orbitals=hamiltonian.n_orbitals)
vmc = VariationalMonteCarlo(hamiltonian, ansatz, optimizer=Adam(lr=1e-3))
vmc.run(n_steps=10000)
energy_est = vmc.energy_mean
```

This would construct the Hamiltonian in a second-quantized form (maybe using OpenFermion to get a FermionOperator) [11], then define a neural network ansatz (the internal of `NeuralWavefunctionAnsatz` uses PyTorch to set up, say, a FermiNet architecture with appropriate antisymmetry). The VMC class then samples electron configurations, evaluates the ansatz probability amplitude, and uses autograd to adjust parameters minimizing the energy. We'd rely on PyTorch's **autograd engine** for the gradient of the network parameters with respect to a loss (energy), which is exactly what torch's `autograd` provides [39]. The heavy parts (evaluating many samples) run on GPU thanks to PyTorch. This is a complex workflow but each part is manageable by reusing known tools (OpenFermion for Hamiltonian mapping [12], existing neural net code for FermiNet, etc.). Upon completion, we could compare `energy_est` to, say, the PySCF FCI energy for STO-3G and output the difference.

Integration into AI Systems: Because the user specifically mentioned integration into an AI like Claude or automated reasoning tool, a final aspect is an interface for such systems. This likely means the engine should have a **programmatic API** (which we are anyway designing) and produce outputs that are easy for an NLP or reasoning system to interpret. Possibly, one would build a layer on top of the engine that formats results in human-readable terms with explanations. For instance, after a calculation, the engine could output:

"The binding energy of H_2 computed with CCSD(T)/aug-cc-pVDZ is 109.5 kcal/mol, which is within 1 kcal/mol of the experimental value (110.5 kcal/mol) [47]. The high-level method CCSD(T) is known as the 'gold standard' due to its accuracy [19], so this result is highly trustworthy."

This kind of output combines numeric results with contextual knowledge (some drawn from our citations). To facilitate this, the engine might tag results with references or short notes. Our extensive citation of literature in this report is in line with that goal: each technique and result can be linked to a source. If an AI (with access to those sources) were using the engine, it could cite those to justify answers.

In conclusion, by following solid software engineering (modularity, leveraging well-tested libraries) and thoroughly documenting and citing the approaches, the engine will serve both as a powerful computational tool and an educational platform. It brings under one roof the traditionally separate domains of quantum chemistry and quantum field theory, enabled by advances in algorithms and computing (and aware of emerging help from machine learning). With such an engine, researchers can tackle questions spanning from **How do electrons behave in a new material?** to **What is the mass of a hypothetical particle?**, comparing answers from various angles, and always cross-checking against the best knowledge available. [10] [19]

---

1 2 5.5: Quantum Theory and Atomic Orbitals - Chemistry LibreTexts
https://chem.libretexts.org/Courses/Williams_School/Chemistry_I/05%3A_Electronic_Structure_and_Periodic_Properties/5.05%3A_Quantum_Theory_and_Atomic_Orbitals

3 4 5 6 Normal Modes - Chemistry LibreTexts
https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Spectroscopy/Vibrational_Spectroscopy/Vibrational_Modes/Normal_Modes

7 8 9 10 26 Lattice QCD - Wikipedia
https://en.wikipedia.org/wiki/Lattice_QCD

11 12 OpenFermion | Google Quantum AI
https://quantumai.google/openfermion

13 14 23 24 Density functional theory - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC2777204/

15 16 17 vergil.chemistry.gatech.edu
https://vergil.chemistry.gatech.edu/notes/ci.pdf

18 Hartree–Fock method - Wikipedia
https://en.wikipedia.org/wiki/Hartree%E2%80%93Fock_method

19 22 LSDALTON_CAAR_TALK_BYKOV2
https://www.olcf.ornl.gov/wp-content/uploads/2017/11/2018UM-Day2-Bykov.pdf

20 21 ORCA Input Library - Coupled cluster
https://sites.google.com/site/orcainputlibrary/coupled-cluster

25 Quantum Espresso - Advancing quantum simulations of materials
for everyone
https://www.quantum-espresso.org/

27 28 Quantum chemistry with Python — PySCF
https://pyscf.org/

29 GitHub - psi4/psi4: Open-Source Quantum Chemistry
https://github.com/psi4/psi4

30 31 4.3.5 Symmetry‣ 4.3 Basic SCF Job Control ‣ Chapter 4 Self-Consistent Field Ground-State Methods ‣ Q-Chem 6.4 User’s Manual
https://manual.q-chem.com/latest/sec_symm.html

32 33 36 ictp-saifr.org
https://www.ictp-saifr.org/wp-content/uploads/2020/06/ICTP-EFT-notes-v4.pdf

34 35 8.10.1 Overview‣ 8.10 Introduction to Effective Core Potentials (ECPs) ‣ Chapter 8 Basis Sets and Effective Core Potentials ‣ Q-Chem 6.1 User’s Manual
https://manual.q-chem.com/6.1/Ch8.S10.SS1.html

37 38 41 42 Explicitly antisymmetrized neural network layers for variational Monte Carlo simulation - ScienceDirect
https://www.sciencedirect.com/science/article/abs/pii/S0021999122008282

39 GitHub - pytorch/pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration
https://github.com/pytorch/pytorch

40 hpc.nih.gov
https://hpc.nih.gov/apps/JAX.html

43 44 Ab Initio Machine Learning in Chemical Compound Space - PMC
https://pmc.ncbi.nlm.nih.gov/articles/PMC8391942/

45 Fast Near Ab Initio Potential Energy Surfaces Using Machine Learning
https://authors.library.caltech.edu/records/ypgg5-h7k64

46 The Chroma Library for Lattice Field Theory
https://jeffersonlab.github.io/chroma/

47 Averting the Infrared Catastrophe in the Gold Standard of Quantum ...
https://link.aps.org/doi/10.1103/PhysRevLett.131.186401