Automated product-formula error bounds for translation-invariant Hamiltonians

Quantum simulation remains in the frontier of quantum information research due to the ability to simulate systems that classical computers cannot efficiently. Although tremendous efforts have been made to present theoretical errors bounds for product formulae, these theoretical bounds, especially for higher-order formulae, are complicated and unclear. In this paper, we introduce a python package and general classical algorithms that would allow for efficient computation of these theoretical bounds for any Hamiltonian for Pauli Strings and Fermionic Operators. In particular, we take advantage of the symmetry of translationally invariant systems to remove system-size dependence of commutator calculations. In addition to arbitrary Hamiltonians, we design a more sophisticated pipeline to compute higher-order product formula bounds. We consider the application of this package for Pionless-EFTs, where we implement sophisticated graph coloring algorithms for tighter norm calculations. We present time-step and circuit depth calculations for Pionless-EFTs which achieve comparable performance to previous methods.

1 Introduction

Despite their simplicity, product formulae methods for quantum simulation are among the best performing, easiest to implement, and generally most popular algorithms to simulate the evolution of quantum systems. Although their asymptotic performance is worse than other more modern methods, such as quantum signal processing (QSP) [1], empirically we see that their performance for relevant timescales is comparable to or even better than QSP errors. On the other hand, product-formula error bounds often depend on complicated expressions of nested commutators which are difficult to compute, and several of the authors have been personally victimised by these computations. Childs et al. [2] provided an explicit expression for product formulae, but the commutator-norm error scaling dependence on Hamiltonians make the expressions difficult to understand. Watson et. al [3] explicitly wrote out expressions for order $p = 1, 2$ error bounds of pionless-EFTs and did asymptotic analysis for error bounds of nuclear Hamiltonians. Such approaches are unfeasible when the complexity of the problem is scaled up. For example, more complex hamiltonians such as pionful hamiltonians or calculations for higher-order approximations of pionless-EFTs would be much harder to compute and be time consuming. Therefore, we introduce generalizable computer algorithms for computation of error bounds. In our algorithms, we choose to work with the fermionic operators directly to take advantage the symmetries involved in translation invariance. No coding package to the best of our knowledge is able to handle applications error bounds calculations presented in [2] for translationally invariant Hamiltonians.

In this paper, we provide a general algorithm for computing product-formulae error bounds. These algorithms are applicable to k-local spin Hamiltonians and fermion Hamiltonians. We also provide special support for fermion Hamiltonians consisting of hopping and number operators. We

present a framework and python code to handle translation invariance of k-local Hamiltonians for arbitrary lattice size. We implement code which represent Pauli Operators and Fermion Operators symbolically, as well as their respective commutator relations. We provide time-step comparisons between previous methods and present implementation details compared to [3] for the Pionless-EFT, as well as show plots for hyperparameters that are specific to our implementation.

We organize the rest of our paper as follows. In section 2, we discuss crucial definitions and background for definitions and variables we will reference in the later parts of the paper. In section 3, we write out details of our framework for handling commutator-norm expressions, as well as proofs that our methods are sound. In Section 4, we discuss implementation-specific details of our algorithms and the short-comings of explicitly computing error bounds involving commutator-norms. In Section 5, we show experimental results and plots of our methods. Finally, in Section 6, we summarize our contributions and provide future directions for work.

2 Preliminaries

In this section, we define general notation and background relevant to our paper.

2.1 Notation and Terminology

Let us first define some notation. Given two positive integers, x, y, we denote their least common multiple (LCM) as $\mathrm{lcm}(x, y)$. We define LCM between vectors $\vec{x}$ and $\vec{y}$ similarly by a vector of the LCM of each component. We denote the set of natural numbers $1, 2, \ldots, D$ as $[D]$. $\|\cdot\|$ will denote the operator norm, $\|\cdot\|_1$ will denote the 1-norm, and $\|\cdot\|_\eta$ will denote the semi-norm of an operator for a system with η particles.

Consider a Hamiltonian

$$H = \sum_{i=1}^{N} h_i \tag{1}$$

where N is the number of terms and each h_i is a Hermitian operator. A Hamiltonian H is k-local if each h_i acts on at most k qubits. We consider two types of k-local Hamiltonians: Pauli Hamiltonians and fermion Hamiltonians. Pauli Hamiltonians are defined by sums of Pauli strings, where we can define each term by $h_i = \prod_{j\in[k]} h_{i,(j,s_j)}$, where j is the operator index and s_j represents the qubit index the operator acts on. Each $h_{i,(j,s_j)} \in \{I, X, Y, Z\}$, where X, Y, Z denote the Pauli Operators and I represents the 2×2 identity matrix. Fermion Hamiltonians can be defined similarly by $h_i := \prod_{j\in[k]} u_{i,(j,s_j)}$ where each $u_{i,(j,s_j)} \in \{a^\dagger_{s_j}, a_{s_j}, I_{s_j}\}$, where $a^\dagger_{s_j}$ and a_{s_j} are the creation and annihilation operators on qubit s_j, respectively. We will denote a term of a Fermion Hamiltonian as a fermion string. As before, I_{s_j} denotes the identity on qubit s_j, or equivalently in this case, the identity on the entire system.

Because of the k-locality for both fermion strings and Pauli strings, we can define the translation operator $\mathcal{T}_v$ for both, where v denotes a vector. For Pauli strings, given the previous definition of $h_i = \prod_{j\in[k]} h_{i,(j,s_j)}$, we can define the translation operator similarly:

$$\mathcal{T}_v(h_i) = \prod_{j\in[k]} h_{i,(j,s_j+v)} \tag{2}$$

For fermions, suppose that $h_i := \prod_{j\in[k]} u_{i,(j,s_j)}$ where each $u_{i_k} \in \{a^\dagger_{i_k}, a_{i_k}, I_{i_k}\}$. Then,

$$\mathcal{T}_v(h_i) = \prod_{j\in[k]} u_{i,(j,s_j+v)} \tag{3}$$

Putting these all together, a Hamiltonian H is translationally invariant if we can express H in the following form:

$$H = \sum_{i=1}^{M} \sum_{v \in \vec{x_i}\mathbb{Z}^N} \mathcal{T}_v(h_i) \tag{4}$$

Definition 1. *If H is a translationally invariant Hamiltonian, the* ***canonical decomposition*** *is the unique selection of $\{x_i\}_{i=1}^M$ and $\{h_i\}_{i=1}^M$ in Equation 4 such that*

1. *For each $H_i \coloneqq \sum_{v \in \vec{x_i}\mathbb{Z}^N} \mathcal{T}_v(h_i)$, $[H_i, H_i] = 0$.*
2. *$\max \|\vec{x_i}\|_{fro}$ is minimized.*

The canonical decomposition should be assumed from now on, unless otherwise stated. We define h_i a *base term* of H. Each inner summation of Equation 4 is denoted as a translationally invariant term of H. We define this to emphasize the distinction between a single term of H and a single translationally invariant term of H. We also alternatively represent each translation-invariant term of the inner-summation in Equation 4 by the tuple $(h_i, \vec{x_i})$, where h_i is a base term and $\vec{x_i}$ is the translation of the base term across the lattice. For each tuple $(h_i, \vec{x_i})$, we can separately define m_i as the smallest qubit index the base term h_i acts non-trivially on i.e. $\min \text{supp}(h_i)$ and l_i represents the difference $\max(\text{supp}(h_i)) - \min(\text{supp}(h_i))$. For multi-dimensional qubit indices, the definition is the same, except we compute these values component-wise. We define each inner summation of H as a translationally invariant term of H, we will reference each translationally invariant term of H by their tuple representations later in the algorithm.

We formally define the semi-norm of an operator as follows:

Definition 2 (Fermionic Semi-Norm (Section 2.3 of Ref. [4]))**.** *Let X be a number-preserving operator and let $|\psi_\eta\rangle$ and $|\phi_\eta\rangle$ be normalized states with exactly η fermions. Then, the fermionic semi-norm of X is*

$$\|X\|_\eta = \max_{|\psi_\eta\rangle, |\phi_\eta\rangle} |\langle \psi_\eta | X | \phi_\eta \rangle|. \tag{5}$$

Furthermore, if X is Hermitian, then

$$\|X\|_\eta = \max_{|\psi_\eta\rangle} |\langle \psi_\eta | X | \psi_\eta \rangle|. \tag{6}$$

For nuclear systems with a fixed amount of particles, it is important that we consider the fermionic semi-norm instead. We will discuss applications of our methods towards these nuclear systems later.

2.2 Product Formula Bounds

Product-formula simulation seeks to construct an estimate the evolution of a Hamiltonian through a decomposition comprised of products of smaller, efficient evolutions for small time steps. More specifically, for the unitary e^{-itH}, where $H = \sum_{\gamma=1}^{\Gamma} H_\gamma$, suppose that each e^{-itH_γ} can be implemented exactly and efficiently for any desired time t by a simple quantum circuit. Suzuki [5] presented approximations for p-order formulas defined by the following:

$$\mathcal{P}_1(t) \coloneqq \prod_{\gamma=1}^{\Gamma} e^{-itH_\gamma}, \tag{7}$$

$$\mathcal{P}_2(t) \coloneqq \prod_{\gamma=1}^{\Gamma} e^{-itH_\gamma/2} \prod_{\gamma=\Gamma}^{1} e^{-itH_\gamma/2}, \tag{8}$$

$$\mathcal{P}_{p+2}(t) \coloneqq \mathcal{P}_p^2(s_p t)\mathcal{P}_p((1-4s_p)t)\mathcal{P}_p^2(s_p t), \qquad p \in 2\mathbb{N}, p \geq 2 \tag{9}$$

where

$$s_p \coloneqq (4 - 4^{1/(p+1)})^{-1}, \Upsilon \coloneqq 2 \times 5^{p/2-1}$$

A general upper bound to these product-formulae estimates can be described by the following [2, Theorem 6 and Appendix E]:

$$\left\| e^{-itH} - \mathcal{P}_p(t) \right\| \leq 2\Upsilon^{p+1} \frac{t^{p+1}}{p+1} \tilde{\alpha}_{\text{comm}}^{(p)}, \tag{10}$$

where

$$\tilde{\alpha}_{\text{comm}}^{(p)} \coloneqq \sum_{\gamma_{p+1},\gamma_p,\ldots,\gamma_1=1}^{\Gamma} \left\| [H_{\gamma_{p+1}}[H_{\gamma_p}, \ldots [H_{\gamma_2}, H_{\gamma_1}]]] \right\| \tag{11}$$

for any $p \in 2\mathbb{N} \cup \{1\}$

Tighter upper bound formulations for product formulae approximations can be constructed as follows:

$$\left\| e^{-itH} - \mathcal{P}_1(t) \right\| \leq \frac{t^2}{2} \sum_{\gamma_1=1}^{\Gamma} \left\| \left[H_{\gamma_1}, \sum_{\gamma_2=\gamma_1+1}^{\Gamma} H_{\gamma_2} \right] \right\|. \tag{12}$$

$$\begin{aligned} \left\| e^{-itH} - \mathcal{P}_2(t) \right\| \leq \frac{t^3}{12} \sum_{\gamma_1=1}^{\Gamma} \left\| \left[\sum_{\gamma_3=\gamma_1+1}^{\Gamma} H_{\gamma_3}, \left[\sum_{\gamma_2=\gamma_1+1}^{\Gamma} H_{\gamma_2}, H_{\gamma_1} \right] \right] \right\| \\ + \frac{t^3}{24} \sum_{\gamma_1=1}^{\Gamma} \left\| \left[H_{\gamma_1} \left[H_{\gamma_1}, \sum_{\gamma_2=\gamma_1+1}^{\Gamma} H_{\gamma_2} \right] \right] \right\| \end{aligned} \tag{13}$$

We denote the $p = 1$ and $p = 2$ error bounds from equations 12 and equations 13 as the tighter bound of $p = 1$ and $p = 2$ product formulae approximations. For any product-formulae formulation, we can see that we get an arbitrarily close approximation to e^{-itH} if we select small enough t. For any arbitrary error bound, we can represent it as such:

$$\left\| e^{-itH} - \mathcal{P}_p(t) \right\| \leq t^{p+1} F(p), \tag{14}$$

for some function F. To determine the number of time steps required to simulate e^{-itH} up to error ϵ_{prod}, consider the use of triangle inequality to achieve r components:

$$\left\| e^{-itH} - \mathcal{P}_p(t) \right\| \leq r \left\| e^{-i(t/r)H} - \mathcal{P}_p(t/r) \right\|. \tag{15}$$

Substituting equation 14 into equation 15 and setting the right hand side equal to ϵ_{prod}, we get:

$$r = [\frac{t^{p+1}}{\epsilon_{\text{prod}}} F(p)]^{\frac{1}{p}} \tag{16}$$

Regardless of the form of F, this process of computing r allows us to generalize time-step calculations for any error bound formulation. We will reference variables defined in these formulas to describe algorithms for computing these expressions later in the paper.

2.3 Pionless-EFT

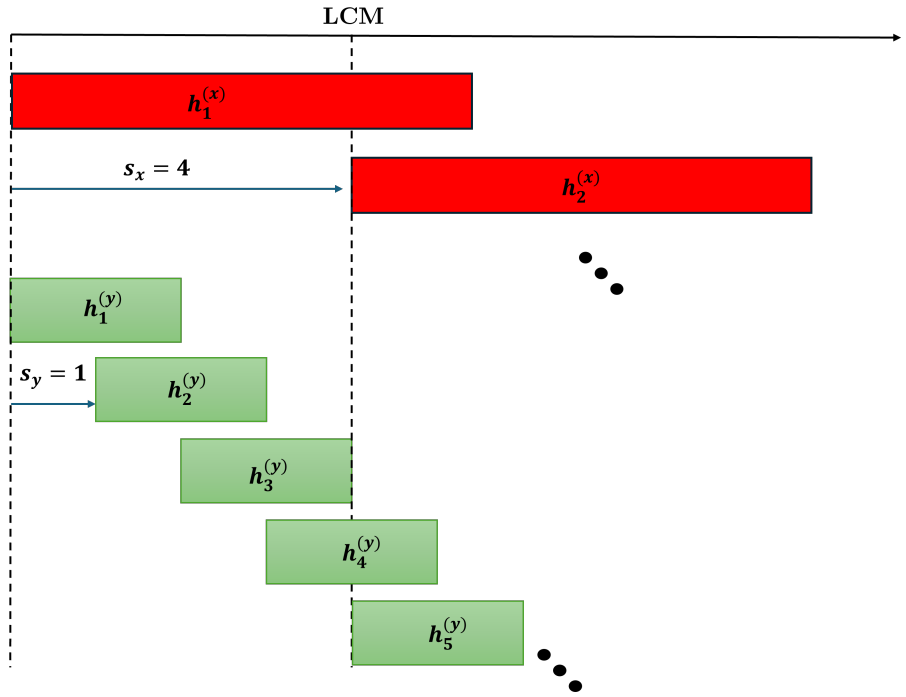


Figure 1: Two sets of translationally invariant terms acting on a $1D$ chain, highlighted in red and green respectively. We see that the commutator $[h_1^{(x)}, h_5^{(y)}]$ is not a translation of any terms which begin within the first $\text{lcm}(s_x, s_y)$ qubits, and hence needs to be considered separately. We also see that $[h_2^{(x)}, h_5^{(y)}]$ is a translation of $[h_1^{(x)}, h_1^{(y)}]$ by with a shift of $\text{lcm}(s_x, s_y)$, demonstrating that we do not need to consider it separately.

3 General Algorithm

One major component of the product formula error bound is the computation of the norms of the nested commutators, which depends on p. To compute these nested commutators, we first show how to compute a single commutator between two translationally invariant terms. Then, we use this as a subroutine to compute nested commutators. Then, we apply these algorithms towards the Pionless-EFT Hamiltonian, where we take advantage of additional properties of nuclear Hamiltonians to introduce more clever algorithms for norm calculations.

We strictly assume that the Hamiltonians are k-local and are translationally invariant. In addition, we assume translation invariance on an infinite lattice. We make this choice for the following reasons: 1) If the lattice is too small, one cannot take advantage of translation invariance

properly and would be better off computing trotter error bounds directly, 2) the algorithms for considering translation invariance on an infinite lattice is almost exactly the same as considering translation invariance and 3) It is possible to generate finite trotter error bounds even while considering an infinite lattice.

3.1 Commutators of Translationally Invariant Terms

We emphasize that we can compute the commutator quantity $[H_i, H_j]$ without losing this property of translation invariance, where H_i, H_j represent summations of translationally invariant terms. In other words, there exists $\{z_k\}_{k=1}^{M'} \in \mathbb{Z}^N$, $M' \in \mathbb{Z}$, and $\{h'_k\}_{k=1}^{M'}$ such that

$$[H_i, H_j] = \sum_{k=1}^{M'} \sum_{v \in \vec{z_k}\mathbb{Z}^N} \mathcal{T}_v(h'_k) \tag{17}$$

One can see this by examining a commutator between two arbitrary translationally invariant terms:

$$\left[\sum_{v \in \vec{x}\mathbb{Z}^N} \mathcal{T}_v(h), \sum_{v' \in \vec{y}\mathbb{Z}^N} \mathcal{T}_{v'}(h') \right] \tag{18}$$

Every commutator interaction between the first and second translationally invariant terms repeats every $\mathrm{lcm}(x, y)$, defined component-wise. Figure 1 represents a one-dimensional visualization of this idea. We can group together each unique commutator interaction to form new translationally invariant terms. We provide an explicit algorithm now to capture this idea.

Algorithm 3 (Commutator Calculations, $p = 1$)**.**
Input: *Tuple representations of translationally invariant terms* $(h_1, \vec{x_1}), (h_2, \vec{x_2})$
Output: *A collection of tuples representing* $[\sum_{v_1 \in \vec{x_1}\mathbb{Z}^N} \mathcal{T}_{v_1}(h_1), \sum_{v_2 \in \vec{x_2}\mathbb{Z}^N} \mathcal{T}_{v_2}(h_2)]$

1. *Initialize resulting collection of tuples* $T = \{\}$
2. *Calculate* $\mathrm{lcm}(\vec{x_1}, \vec{x_2})$ *and find all* $(\vec{v_1}, \vec{v_2}) \in (\vec{x_1}\mathbb{Z}^N, \vec{x_2}\mathbb{Z}^N)$ *where either* $\vec{v_1} \leq \mathrm{lcm}(\vec{v_1}, \vec{v_2})$ *or* $\vec{v_2} \leq \mathrm{lcm}(\vec{v_1}, \vec{v_2})$ *in any component. Let* C *define this collection of* $(\vec{v_1}, \vec{v_2})$.
3. *For each* $(\vec{v_1}, \vec{v_2}) \in C$*, Compute the commutator* $h_{(\vec{v_1}, \vec{v_2})} = [\mathcal{T}_{v_1}(\vec{h_1}), \mathcal{T}_{v_2}(\vec{h_2})]$*. Add the tuple* $(h_{(v_1, v_2)}, \mathrm{lcm}(\vec{x_1}, \vec{x_2}), m_{(\vec{v_1}, \vec{v_2})}, l_{(\vec{v_1}, \vec{v_2})})$ *to* T.
4. *return* T

We will now provide supplemental proofs to show the validity of Algorithm 3. More specifically, we will show that the search we provide in step 2 of 3 is sufficient.

Lemma 4. *Suppose we have two fermion strings* h_1, h_2 *which are number preserving and satisfy number ordering. Then* $\mathrm{supp}([h_1, h_2]) \subseteq \mathrm{supp}(h_1) \cup \mathrm{supp}(h_2)$ *where* $\mathrm{supp}(A)$ *is defined on the lattice sites where the operator* A *acts non-trivially on.*

Proof. Suppose $\vec{x} \in supp([h_1, h_2])$. Then, $\vec{x} \in \mathrm{supp}(h_1)$ or $\vec{x} \in \mathrm{supp}(h_2) \implies \vec{x} \in \mathrm{supp}(h_1) \cup \mathrm{supp}(h_2)$. □

Now, we will show that base terms can only occur inside the within $\max(m_1, m_2) + \mathrm{lcm}(x_1, x_2)$ of two translationally invariant terms $(h_1, \vec{x_1}), (h_2, \vec{x_2})$. This proposition shows that Algorithm 3 removes any dependence of the time-complexity on the size of the system.

Proposition 5. *Suppose we have two tuple representations of translationally invariant terms* $(h_1, \vec{x_1}, \vec{m_1}, \vec{l_1}), (h_2, \vec{x_2}, \vec{m_2}, \vec{l_2})$. *Then, a base term that arises as a by-product of the commutator cannot have an offset* m' *that is larger than* $\max(m_1, m_2) + \mathrm{lcm}(s_1, s_2)$ *in any dimension i.e.* $m' \leq \max(m_1, m_2) + \mathrm{lcm}(x_1, x_2)$ *in all dimensions.*

Proof. Suppose not. Then, there is a translationally invariant term $T^* = (h_*, \vec{x_*}, \vec{m_*}, \vec{l_*})$ that has an initial offset m^* greater than $\max(m_1, m_2) + \mathrm{lcm}(s_1, s_2)$. Suppose that the commutator $[\mathcal{T}_{\vec{v_1}}(h_1), \mathcal{T}_{\vec{v_2}}(h_2)]$ where $(\vec{v_1}, \vec{v_2}) \in (x_1 Z^N, x_2 Z^N)$ produces this T^*. By lemma 4, we obtain that $v_1, v_2 \geq \max(m_1, m_2) + \mathrm{lcm}(s_1, s_2)$. Then, consider the commutator $T' := [\mathcal{T}_{\vec{v_1} - \mathrm{lcm}(v_1, v_2)}(h_1), \mathcal{T}_{\vec{v_2} - \mathrm{lcm}(v_1, v_2)}(h_2)]$. Certainly, this commutator is computed in Algorithm 3. By construction, we see that T' is a base term by definition for T^*. This contradicts the assumption that h^* is a base term ⨳. □

For nested commutators, by linearity, we can extend an iterative process taking advantage of the calculations provided in Algorithm 1.

Algorithm 6 (Arbitrary Nested Commutators)**.**
Input: *An integer* $p \in 2\mathbb{Z} \cap 1$ *and a list* J *of tuple representations, where* $J[i] := (h_i, \vec{x_i}, \vec{m_i}, \vec{l_i})$ *and* $|J| = p + 1$.
Output: *A collection of tuples representing*
$[\sum_{v_{p+1} \in x_{p+1}^{\vec{}} \mathbb{Z}^N} \mathcal{T}_{v_{p+1}}(h_{p+1}), \ldots, [\sum_{v_2 \in \vec{x_2} \mathbb{Z}^N} \mathcal{T}_{v_2}(h_2), \sum_{v_1 \in \vec{x_1} \mathbb{Z}^N} \mathcal{T}_{v_1}(h_1)]]$

- *Initialize resulting collection of tuples* $T = \{\}$ *and current term* $curr = \{J[1]\}$. *Initialize rolling set* $temp = \{\}$

1. *For i=1..p (inclusive)*
   a. *For all tuples x in curr, Input J[i+1] and x into Algorithm 3 and add the output to temp.*
   b. *Set* $curr = temp$. *If curr is empty as a result, return* 0. *Set* $temp = \{\}$.
2. *Set* $T = curr$. *Return* T.

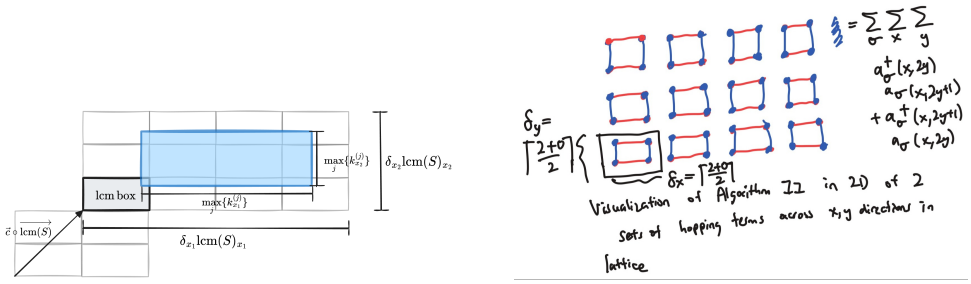
The ability to keep translation invariance through commutators allows us to recursively call Algorithm 3. In addition, the linearity of the commutator operator makes it possible to only consider a simplified version of the whole problem to nested commutators of single translationally invariant terms.

3.2 Semi-Norm Calculations

In fermionic Hamiltonian simulation, we assume that the systems are **number-preserving**, i.e. the number fermions that exist in the system must be constant. As mentioned before, for product formulae error bounds of fermionic Hamiltonians, we consider the semi-norm instead of the usual norm in Equation 10. To compute the semi-norm, Watson et al. [3] proved the following theorem:

Theorem 7. *Consider a set of fermionic modes,* M. *Let* $\vec{i} = (i_1, i_2, \ldots, i_{k_{\vec{i}}})$ *denote a tuple of* $k_{\vec{i}}$ *indices for some constant* $k_{\vec{i}}$, *and let* $\Omega = \{\vec{i_1}, \vec{i_2} \ldots\}$ *be a set of such tuples such that no tuple shares indices with any other tuple:* $\forall \vec{i_a}, \vec{i_b} \in \Omega$ *with* $a \neq b$, $\vec{i_a} \cap \vec{i_b} = \emptyset$. *Define the fermionic operator*

$$X_\Omega = \sum_{\vec{i} \in \Omega} J_{\vec{i}} h_{\vec{i}}, \tag{19}$$



(a) (b)

Figure 2: In figure (a), the lcm box represents the lcm of all translations of operators across the lattice. The light-box represents the restriction $\widetilde{G}$ of operators presented in Algorithm 10. We emphasize here that the light-blue box is a multiple of the gray lcm box. This explicit construction ensures that the direct tiling of this box across the lattice maintains a valid coloring of graph G. Figure (b) shows an example of the calculation of $\widetilde{G}$ using a simplified example in 2D involving hopping operators.

such that each $h_{\vec{i}}$ is a Number Preserving Fermionic Operator (NPFO) acting on the fermionic modes in $\vec{i} \subset M$. Then, the fermionic semi-norm can be bounded as

$$\|X_\Omega\|_\eta \leq J_{\max} \min\left\{ \left\lceil \frac{\eta}{\lceil k_{\min}/2 \rceil} \right\rceil, |\Omega| \right\}, \tag{20}$$

where $k_{\min}$ is the minimum locality of $h_{\vec{i}}$ and $J_{\max} = \max_{\vec{i}\in\Omega}\{|J_{\vec{i}}|\}$.

The result of this theorem is the basis for the semi-norm calculations. We seek to take the triangle inequality over operators with disjoint support to then use Theorem 7 to upper bound all calculations. To accomplish this, we construct a graph where the nodes represent individual fermionic operators, and an edge exists between two operators if and only if they share an index. Then, we utilize the a greedy graph coloring algorithm to construct subsets of operators with completely different support.

A direct consequence of the previous theorem is the following:

Corollary 8. *Consider a fermionic Hamiltonian $H = \sum_{j=1}^n \alpha_j h_j$. Construct a graph G as follows: for each local term h_j introduce a vertex, and draw an edge between two vertices if they act on at least one of the same fermionic modes. Then the fermionic semi-norm of H can be bound as:*

$$\|H\|_\eta \leq \chi(G) \frac{\eta}{\lceil k_{\min}/2 \rceil} \max\{\alpha_i\}_{i=1}^n$$

where $\chi(G)$ is the chromatic number of G and $k_{\min}$ is the minimum locality of $h_{\vec{i}}$.

Proof. Let G be the graph defined above, suppose graph G has a minimal colour set S. We can triangle inequality along these disjoint sets to apply Theorem 7. We have:

$$\|H\|_\eta = \sum_{i\in[|S|]} \left\| \sum_{j\in S_i} \alpha_j h_j \right\|_\eta \leq \sum_{i\in[|S|]} \max\{\alpha_j\}_{j\in S_i} \frac{\eta}{\lceil k_{\min}/2 \rceil} \leq \chi(G) \frac{\eta}{\lceil k_{\min}/2 \rceil} \max\{\alpha_j\}_{j\in[n]}$$

□

Corollary 8 gives us an upper bound of the semi-norm of a fermionic operator by the chromatic number of its graph. We choose to use this result to estimate semi-norms of fermionic operators.

Our choice of this upper bound can be attributed to the idea that the chromatic number of $\chi(G)$ is finite. To compute $\chi(G)$, The symmetry of translationally invariant operators can be used again to instead compute a coloring of a smaller graph $\widetilde{G}$. This $\widetilde{G}$ can be tiled across the lattice to generate a complete coloring for G. The idea is that there is are closed interval size such that, when this closed interval is tiled across the lattice, the graph induced by each region is isomorphic to each other. We define the region closest to the origin as $\widetilde{G}$. We explicitly define the nodes of $\widetilde{G}$ as follows:

Definition 9. *Define* $S \coloneqq \{(h_i, \vec{x_i})\}_{i=1}^n$ *represent a collection of translationally invariant terms. Define* $\mathrm{lcm}(S) \coloneqq LCM\{\vec{x_i}\}_{i=1}^n$. *Let* $\vec{\delta} \in (\mathbb{Z}^+)^D$ *be defined component-wise as*

$$\vec{\delta} = \left\lceil \frac{\max\{l_i\}_{i=1}^n}{\mathrm{lcm}(S)} \right\rceil.$$

and let $\vec{c} \in (\mathbb{Z}^+)^D$ *be defined as*

$$\vec{c} = \left\lceil \frac{\max\{m_i\}_{i=1}^n}{\mathrm{lcm}(S)} \right\rceil.$$

Then, the nodes of $\widetilde{G}$ *can be defined by*

$$N(\widetilde{G}) = \{\min \mathrm{supp}(h) \in [\vec{c} \circ lcm(S), (\vec{c} + \vec{z}) \circ \mathrm{lcm}(S)] \mid h \in N(G)\}.$$

where $N(G)$ *defines the nodes of* G.

By construction, we can rewrite any $S = \{(h_i, \vec{x_i})\}_{i=1}^n$ as $S = \{(h_j, \mathrm{lcm}(S))\}_{j=1}^{|\widetilde{G}|}$ where $\{h_j\}_{j=1}^{|\widetilde{G}|}$ represents the set of all operators in $\widetilde{G}$. We denote each $h \in \widetilde{G}$ as a **representative** of G. The representative of any $h \in G$ shall be denoted by $\tilde{h}$. To align the construction of $\widetilde{G}$ with the separation of G into sets where each operator acts on a different of qubits, we seek to draw edges between any two operators that act on a similar qubit. It is crucial we also consider when representative operators and non-representative operators act on the same qubit. We can formalize the creation of edges of $\widetilde{G}$ in the following algorithm:

Algorithm 10 (Finding Independent Sets of Translation Invariant Fermionic Operators)**.**
Input: A set of tuple representations $\{(h_i, \vec{x_i})\}_{i=1}^n$
Output: A norm

1. *Construct a graph* $\widetilde{G}$.
2. *Add edges to* $\widetilde{G}$ *according to the following rules: For* $u, v \in G$
   - *If* $u, v \in \widetilde{G}$ *and* $\mathrm{supp}(u) \cap \mathrm{supp}(v) \neq \emptyset$ *add* (u, v) *to* $E(\widetilde{G})$
   - *If* $u \in \widetilde{G}$ *and* $v \notin \widetilde{G}$ *and* $\mathrm{supp}(u) \cap \mathrm{supp}(v) \neq \emptyset$ *add* $(u, \tilde{v})$ *to* $E(\widetilde{G})$
3. *Vertex color the graph* $\widetilde{G}$.
4. *Assign all* $u \in G$ *to the set associated with the color of its representative* $\tilde{u} \in \widetilde{G}$. *These sets will consist of disjoint operators.*
5. *For each set of disjoint operators, use theorem 7 to compute norms. Sum all these norms together and return.*

A visualization for 10 is illustrated in Fig. 2 for $D = 2$. The key step in the algorithm is Step 2. By construction, these rules ensure that if $(u, v) \in E(G)$ (i.e. if two terms $u, v \in T[O]$ intersect) then their representative will share an edge in $E(\widetilde{G})$. That is, $(u, v) \in E(G) \implies (\tilde{u}, \tilde{v}) \in E(\widetilde{G})$. So coloring $\widetilde{G}$ to divide terms into independent sets and then assigning sets based on the coloring of the representatives is guaranteed to provide a valid (if not necessarily minimal) coloring of $T[O]$. The construction of the additional edges in step 2 of the algorithm such that the in the construction of $\widetilde{G}$, no self-edges between any node can occur. We now show that this is the case.

Theorem 11. *The graph construction of G described in Algorithm 10 has no self-edges.*

Proof. Each node $g \in G$, by construction, has a corresponding 1-1 edge incidents to its representative node $\tilde{g} \in \widetilde{G}$. It follows that since $\tilde{g}$ is a self edge by construction of $\widetilde{G}$ that $g \in G$ must not have any self edges either. □

For larger hamiltonians, the size of $\widetilde{G}$ can get very large, and the direct construction is intractable. One way to circumvent the explicit construction of a graph is simply to color each translationally invariant term independently. By considering each translationally invariant operator individually, unknown interactions of multiple terms translating across the lattice do not need to be considered, drastically simplifying calculations. Algorithm 10 requires the explicit construction of a graph to compute norms, where for higher-order commutators of the Pionless-EFT, are extremely time-consuming to calculate. To avoid the quadratic scaling of creating $\widetilde{G}$, We take advantage of the fact that we can use any coloring for G to describe a weaker coloring of $\widetilde{G}$ which does not require an explicit construction of a graph. To illustrate this, by the triangle inequality, we have

$$\|H\|_\eta = \sum_{i=1}^{M} \sum_{v \in \vec{x_i}\mathbb{Z}^N} \|\mathcal{T}_v(h_i)\|_\eta \tag{21}$$

Coloring individual terms is much easier and can be accomplished through basic arithmetic of h_i, l_i, x_i. We denote this alternative coloring as the **brute-force coloring algorithm**. The brute-force coloring algorithm can be described as follows:

Algorithm 12 (Brute Force Coloring)**.**

Inputs: The set $\{(h_i, \vec{x_i})\}_{i=1}^n$ Outputs: A norm

1. *Initialize total sum S.*
2. *For all translationally invariant operators in H,*
   a. *Component-wise, compute the vector $\vec{C} = \lfloor (l_i - 1)/x_i \rfloor + 1$*
   b. *$S = S + \prod C$, where $\prod C$ is the product of all components of vector C.*
3. *Return C*

We now put everything together to describe an explicit algorithm for computing α_{comm} in equation 10. We introduce the idea of a threshold M to avoid the explicit creation of a graph that would be inefficient to compute.

Algorithm 13 (General algorithm for computing α_{comm})**.**
Inputs: Hamiltonian H, order of approximation p, size threshold M

1. *Initialize Symbolic representations of $H := \sum_i H_i$, where $[H_i, H_i] = 0 \ \forall i$.*

2. *Use Theorem 6 to calculate the* $[H_{\gamma+1}, [H_\gamma, ...[H_{\gamma_2}, H_{\gamma_1}]]]$ *component of* α_{comm} $\forall \gamma_{p+1}, \gamma_p, ..., \gamma_2, \gamma_1$, *given order of approximation* p. *Define*

$$G_{(\gamma_1-1)\cdot(\gamma_2-1)\cdot...\cdot(\gamma_p-1)\cdot(\gamma_{p+1}-1)} \coloneqq [H_{\gamma+1}, [H_\gamma, ...[H_{\gamma_2}, H_{\gamma_1}]]] \ \forall \gamma_{p+1}, \gamma_p, ..., \gamma_2, \gamma_1$$

Save these calculations separately.

3. *For each* G_i, *calculate the norm via Theorem 10 or Theorem 12.*

4. *Return* $a_{comm} \coloneqq \sum_i G_i$

4 Implementation

Here, we discuss choices we made in-practice for the implementation of the algorithms above. We used the python programming language because it is the easiest and most widely known. For the operators in our package, We implement all the operators symbolically to avoid the exponential scaling of the matrix size of operators. We provide general support Pauli strings and fermionic operators, while implementing separate operators to handle error bounds with the Pionless-EFT. To represent translationally invariant operators, we store them in a python class, representing each by their tuple representation. As much as possible, for symmetric Hamiltonians like the Pionless-EFT, translationally invariant terms with the same initial offset and translation vector are kept together into one python class. This way, simplifications are much easier to implement. To color the graphs created by our graph-coloring algorithm, we choose to use networkx's [6] greedy_color() function, and we chose to use the "largest_first" strategy to make our algorithm deterministic. We calculate the constants of $F(p)$ along with the commutators since they depend on p and it's easiest to generate them as further nesting is applied. To avoid duplicate calculations with commutators that return 0, entire orders of commutators are computed one at a time. As a result, all nested commutators are computed before any norm calculations are being done. One notable drawback of this approach is the memory that is taken up to store entire orders of commutators instead of processing them immediately, which becomes relevant for higher-order commutators.

The independence of commutator-norm calculations without each other allows us to parallelize computations. The direct parallelization of these commutator norms would be ineffective since the larger commutators would take more time to compute graph coloring, leaving all the compute to a few processes at the end. In practice, most commutators take very short amounts of time, so instead only the graph coloring computations are parellelized. Due to memory constraints, some commutators are too large to produce an explicit graph representation $\widetilde{G}$ effectively. This led us to implement the looser brute-force coloring which does not require an explicit graph representation for larger graphs. We choose to define a threshold value T to denote the graph size of when we choose to use this brute-force coloring. In section 5, we provide plots to how the choice of T affects error bounds.

To conduct these experiments, we used a computer with 128 CPUs with 1 TB memory for parallel graph-coloring calculations. Here, we discuss why such intense resources are necessary.

4.1 Memory Limitations

We can see that the time complexity scaling of Theorem 13 is extremely poor, with a time complexity scaling of $O(N^{2p})$, where N is the number of base terms in the hamiltonian. In practice, this upperbound in time complexity is extremely loose because it is expected that many commutators return 0. Even so, it is reasonable that for higher-order calculations, excessive amounts of memory

are required. For example, for the 3D Pionless-EFT, there are 64 base terms, where each base term requires at least 3000 bytes of memory to represent. For $p = 4$ commutators, empirical analysis suggests that 500 gigabytes of RAM are required to store all the resulting base terms. Using the asymptotic bounds, we get upper bound of around $8 \cdot 10^{28}$, though in practice we end up with $5 \cdot 10^7$ base terms. Such analysis suggests that $p = 6$ commutator calculations are not feasible at all.

Another part of the algorithm with memory bottlenecks occurs in the graph coloring algorithm, where the creation of the graph described in Theorem 10 has quadratic memory scaling. Because of the already large amounts of memory the nested commutators take up, the added overhead of constructing a graph for each nested commutator where the graphs are too large puts the job requires more than 1 TB if we were just to construct a graph for each nested commutator. Therefore, in Theorem 13, we implement a threshold idea where graphs that would be too large to generate are calculated using the brute-force coloring.

4.2 Time Complexity Issues

This algorithm suffers greatly from a time complexity standpoint as well. The time complexity of the commutator algorithm scales $O(N^{2^p})$, though in practice, many commutators fully commute, leading to the algorithm being more efficient than theory suggests. The entire algorithm for $p = 2$ took 20 seconds, while for $p = 4$ commutators of the Pionless-EFT, Step 2 of Theorem 13 took 16 hours. This suggests that $p = 6$ commutators are out of reach from a time-complexity standpoint.

5 Application

Here, we present comparisons to previous estimates of α_{comm}. We provide plots to show the effects of changing the threshold value M in 13 for circuit depths in the case of the Pionless-EFT Hamiltonian.

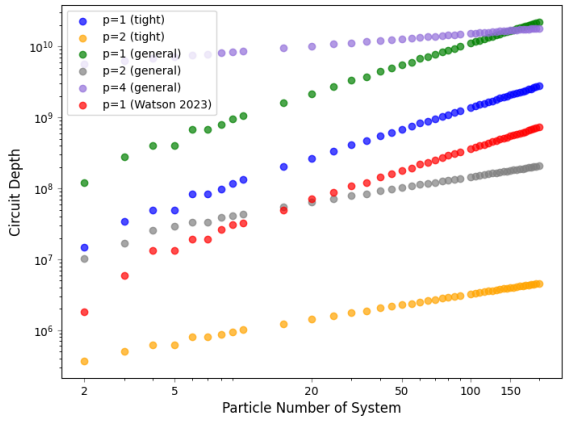


Figure 3: Results from Theorem 13 using order $p = 1, 2, 4$ for systems with $\eta = 2, 10, 30, 50, 70, 90$ fermions. The red plots represent the scaling of theoretical upper bound calculations done in [3].

As expected, the general methods for $p = 1$ and $p = 2$ performed worse compared to the their tighter formulations. Compared to previous methods, our computer-calculated bounds perform worse compared to $p = 1$ bounds presented in Watson et. al [3], but this is expected due to our choice of our algorithms representing more general bounds.

Our calculations suggest that computing product-formulae bounds past $p = 2$ offers limited returns for the amount of computations it requires due to only having better performance than $p = 2$

product-formulae for systems with extremely large particle number. From figure 3, the crossover from $p = 4$ over $p = 2$ product-formulae error bounds only happens at around 10000 particles We only expect the crossover to get worse for more complicated systems such as the Pionful-EFT where the commutator-norms in equation 10 are more non-zero, thus scale more exponentially.

6 Discussion

We have provided and described algorithms to help compute product formulae error bounds for any arbitrary translationally invariant k-local Hamiltonian that does not depend on the lattice size. We also provide a python package that implements these algorithms. This python package provide support for the Pionless-EFT, while also providing support for computing commutators of arbitrary Pauli and fermion strings. Finally. We provided experimental to show comparable results with same asymptotic bound and error values to previous calculations.

Several results of our work can be further investigated. For example, our calculations suggest that it wouldn't be appropriate to simulate past order 2 product formulae. More theory to prove or disprove our methods would help understand product formulae error scaling much better. In addition, one could examine the symmetries provided in the Pionless-EFT and derive tighter and easier to implement error bounds for this specific use case. More specifically, analyze how each nested commutator scales as order p increases. These theories would help us better analyze much higher-order product formulae $p = 6$ and generalize to other well-known nuclear Hamiltonians.

References

[1] Guang Hao Low and Isaac L. Chuang. "Optimal hamiltonian simulation by quantum signal processing". Physical Review Letters**118** (2017).

[2] Andrew M. Childs, Yuan Su, Minh C. Tran, Nathan Wiebe, and Shuchen Zhu. "Theory of trotter error with commutator scaling". Phys. Rev. X **11**, 011020 (2021).

[3] James D Watson, Jacob Bringewatt, Alexander F Shaw, Andrew M Childs, Alexey V Gorshkov, and Zohreh Davoudi. "Quantum algorithms for simulating nuclear effective field theories" (2023). arXiv preprint arXiv:2312.05344.

[4] Yuan Su, Hsin-Yuan Huang, and Earl T. Campbell. "Nearly tight trotterization of interacting electrons". Quantum **5**, 495 (2021).

[5] Masuo Suzuki. "General theory of fractal path integrals with applications to many-body theories and statistical physics". Journal of Mathematical Physics **32**, 400–407 (1991).

[6] Aric Hagberg, Pieter J. Swart, and Daniel A. Schult. "Exploring network structure, dynamics, and function using networkx". In Proceedings of the 7th Python in Science Conference (SciPy2008). Los Alamos National Laboratory (LANL) (2007). url: https://www.osti.gov/biblio/960616.