Journal of Computational Algebra 18 (2026) 100045

Contents lists available at ScienceDirect

Journal of Computational Algebra

journal homepage: www.elsevier.com/locate/jaca

The fractal symmetry in multiplicative structures of $\mathfrak{su}(2^n)$ with applications to dynamical Lie algebra computation ☆

Moody T. Chu

Department of Mathematics, North Carolina State University, Raleigh, NC, 27695, USA

ARTICLE INFO

MSC:
17B45
15B30
81R50
65F60

Keywords:
Pauli string coding
Dynamical Lie algebra
Generators
Fractal-like structure
Recursive algorithm

ABSTRACT

This work pursues two intertwined objectives. First, it elucidates a nested, cyclic pattern inherent in the multiplication of basis elements within $\mathfrak{su}(2^n)$, revealing a delicate, fractal-like symmetry at the heart of the algebra. Second, building upon this structural insight, it introduces a recursive algorithm that efficiently determines the dynamical Lie algebra generated by any collection of elements in $\mathfrak{su}(2^n)$.

1. Introduction

The collection of all possible operations that can be performed on a quantum system is mathematically described by a Lie group, specifically the special unitary group $SU(2^n)$, where n is the number of qubits [10,12,19,24]. Each quantum gate can naturally be represented by an element of this group, thereby embodying a permissible unitary transformation. Associated with this group is its Lie algebra $\mathfrak{su}(2^n)$, whose elements are precisely all trace-free, skew-Hermitian matrices in $\mathbb{C}^{2^n\times 2^n}$. These matrices serve as the infinitesimal generators of unitary evolution, providing the fundamental structure for continuous transformations in quantum mechanics [21,29].

Given an arbitrary subset $V \subset \mathfrak{su}(2^n)$, referred to henceforth as a generator, the minimal subalgebra $\mathfrak{g}(V)$ spanned by all possible nested commutators of the elements in V is known as the dynamical Lie algebra (DLA) [30,33]. In recent years, the concept of DLA has garnered considerable attention across a range of practical applications. This is because it describes the mathematical framework characterizing all possible unitary evolutions accessible to a quantum system, given a specified set of generating Hamiltonians or quantum gates. For instance, in the fundamentally important Hamiltonian simulation problem where an efficient approximation of the time evolution operator $e^{-\iota\mathcal{H}t}$ is desired [3,8,16,17,26,27], we are interested in using $\mathfrak{g}(V)$ to parameterize the system when V contains all terms in the underlying Hamiltonian $\mathcal{H}$ multiplied by the imaginary unit ι [4,7,11,23,25,31].

One of the central tasks in deriving the DLA is to precisely describe the structure of this subalgebra and to ascertain its dimension. Despite its importance, there is presently no general, effective method for explicit construction of the subalgebra $\mathfrak{g}(V)$ for arbitrary $V \subset \mathfrak{su}(2^n)$, nor is there a known approach to provide an *a priori* bound for the dimension of $\mathfrak{g}(V)$ in the general case. For certain V, the dimension of $\mathfrak{g}(V)$ can grow exponentially with n, a manifestation of the curse of dimensionality that makes these problems especially challenging. Consequently, the explicit determination and systematic classification of dynamical Lie algebras in this context remain unresolved and open for further investigation.

It is known that any subalgebra of $\mathfrak{su}(N)$ is either Abelian or a direct sum of compact simple Lie algebras and a center [13,24]. A significant contribution in the paper [34] is its 95-page supplementary material, which uses trees to describe all possible irreducible simple subalgebras, providing a complete lattice of irreducible simple subalgebras of $\mathfrak{su}(N)$ for $2 \le N \le 2^{15}$. Given $V \in \mathfrak{su}(2^n)$ with $1 \le n \le 15$, we can theoretically break down the subalgebra $\mathfrak{g}(V)$ to identify its isomorphic irreducible

☆ This research was supported in part by the National Science Foundation under grants DMS-1912816 and DMS-2309376.
E-mail address: chu@math.ncsu.edu.

https://doi.org/10.1016/j.jaca.2026.100045
Received 1 October 2025; Received in revised form 8 March 2026; Accepted 16 March 2026

components within the lattice. However, this process is challenging in practice.

In certain quantum models, the Hamiltonian $\mathcal{H}$ exhibits a distinctive structure [2,9,15,18,22]. It is possible to exploit the structure to get an analytic description of $\mathfrak{g}(\imath\mathcal{H})$. For instance, the Heisenberg model assumes that the n spin-½ particles interact with only the nearest neighbors. The corresponding Hamiltonian appears in the form

$$\mathcal{H} = -\frac{1}{2}\sum_{j=1}^{n-1}(J_X\sigma_j^X * \sigma_{j+1}^X + J_Y\sigma_j^Y * \sigma_{j+1}^Y + J_Z\sigma_j^Z * \sigma_{j+1}^Z) + \gamma\sum_{j=1}^{n}\sigma_j^Z, \quad (1)$$

where $\sigma_j^A := I^{\otimes(j-1)} \otimes A \otimes I^{\otimes(n-j)}$ and J_A is the coupling constant with A standing for any of the Pauli matrices

$$X := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad Y := \begin{bmatrix} 0 & -\imath \\ \imath & 0 \end{bmatrix}; \quad Z := \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad (2)$$

γ denotes a transverse interfering magnetic field, and $*$ denotes the conventional matrix multiplication. In this case, it can be argued that $\dim(\mathfrak{g}(\imath\mathcal{H})) = 4^n - 4$ [25]. For the Ising model

$$\mathcal{H} = -\frac{1}{2}\sum_{j=1}^{n-1} J_Z\sigma_j^Z * \sigma_{j+1}^Z + \gamma\sum_{j=1}^{n-1}\sigma_j^X, \quad (3)$$

we find that $\dim(\mathfrak{g}(\imath\mathcal{H})) = n(2n-1)$. Indeed, both models (1) and (3) are special cases of the general translation-invariant 2-local spin chain Hamiltonians,

$$\mathcal{H} = \sum_{j=1}^{n-1}\sum_{\mathbf{a}\otimes\mathbf{b}\in\mathcal{A}} c_{j,\mathbf{a},\mathbf{b}} I^{\otimes(j-1)} \otimes \mathbf{a} \otimes \mathbf{b} \otimes I^{\otimes(n-j-1)}, \quad (4)$$

where $\mathcal{A} \subset \{I, X, Y, Z\}^{\otimes 2}$ characterizes a special 2-local lattice configuration of the system. It has been discovered that, for the system (4), there are 17 unique dynamical Lie subalgebras, all classified in [33]. While the closed-form analysis for special cases of V is intriguing, such classifications yield insight into $\mathfrak{g}(\imath\mathcal{H})$ only within the confines of specific quantum models, and thus fall short of offering a unified perspective that extends across more general settings.

A recent study introduces the concept of an anti-commutation graph for arbitrary collections of Pauli strings, demonstrating that, through graph contraction techniques, any such graph can be systematically reduced to a concise set of canonical forms [1]. This reduction offers a principled approach to classifying the Lie subalgebras generated by minimal subsets of Pauli strings, as the structure of the anti-commutation graph and its canonical representative uniquely characterize the associated algebra. Within this framework, it is theoretically possible to enumerate all dynamical Lie subalgebras that arise from subsets of Pauli strings. Nevertheless, while the theoretical foundations are being established, a concrete computational procedure for carrying out these reductions has not yet been described. In contrast, our method directly encodes $2^n \times 2^n$ Pauli strings in single integers and enlists all elements in $\mathfrak{g}(V)$ for any general subset V.

The paper is organized as follows. In Section 2, we present an integer-based encoding scheme for the basis elements of $\mathfrak{su}(2^n)$, which enables efficient computation of commutator brackets through straightforward index manipulation and permutation. This representation, which associates integers with $2^n \times 2^n$ matrices, serves as the foundation for our exploration in Section 3, where we reveal a nested, cyclic, and fractal-like pattern underlying the multiplication table of the basis elements of $\mathfrak{su}(2^n)$. This structure appears to be novel in the literature. This recurring pattern motivates a divide-and-conquer approach, reminiscent of the fast Fourier transform, for computing the algebra $\mathfrak{g}(V)$. Two updating schemes, one following the Jacobi method and the other inspired by Gauss-Seidel, are proposed in Section 4. Finally, Section 5 offers computer experiments that validate our theoretical findings.

Table 1
Matrix-to-matrix multiplication table of $\{I, X, Y, Z\}$ identified as $\{0, 1, 2, 3\}$.

$\downarrow * \rightarrow$	0	1	2	3
0	0	1	2	3
1	1	0	$\imath 3$	$-\imath 2$
2	2	$-\imath 3$	0	$\imath 1$
3	3	$\imath 2$	$-\imath 1$	0

2. Encoding Pauli strings and Lie brackets

Together with the identity matrix I, an element in the set $\{I, X, Y, Z\}^{\otimes n}$, i.e., a tensor product of n matrices selected from the set $\{I, X, Y, Z\}$, is referred to as a Pauli string. Elements in $\{I, X, Y, Z\}^{\otimes n}$ are mutually orthogonal with respect to the Frobenius inner product over $\mathbb{C}^{2^n\times 2^n}$. There are 4^n Pauli strings, so they span all possible $2^n \times 2^n$ Hermitian matrices. Each Pauli string is of size $2^n \times 2^n$. Constructing such large matrices explicitly is impractical, and performing the Lie bracket operation directly is even more so. The following procedure, previously discussed in [5,6], offers an efficient scheme for encoding each Pauli string as a unique integer, and for decoding integers back to Pauli strings, thereby enabling effective computation of the Lie bracket operation. We briefly summarize the essential concepts here to ensure clarity and self-containment.

Let the Fraktur numerics $\mathfrak{0}, \mathfrak{1}, \mathfrak{2}, \mathfrak{3}$ represent both the Pauli operators I, X, Y, Z, respectively, and the digits in the quaternary numeral system. Each element in $\{I, X, Y, Z\}^{\otimes n}$ can thus be interpreted naturally as a unique n-digit identifier:

$$\mathfrak{d}_n \otimes \mathfrak{d}_{n-1} \otimes \ldots \otimes \mathfrak{d}_1 \Rrightarrow \mathfrak{d}_n\mathfrak{d}_{n-1}\ldots\mathfrak{d}_1, \quad \mathfrak{d}_j \in \{\mathfrak{0}, \mathfrak{1}, \mathfrak{2}, \mathfrak{3}\}, \quad (5)$$

which can be mapped bijectively to an ordinal number ℓ via

$$\ell := \sum_{j=1}^{n} 4^{j-1}\mathfrak{d}_j + 1, \quad (6)$$

and vice versa. Thus, regardless of the value n which could be large, we represent a $2^n \times 2^n$ Pauli string compactly by a single integer. Let B_ℓ, $\ell = 1, \ldots, 4^n$, denote the Pauli strings multiplied by the imaginary number $\imath$, we may use the same conversion to represent B_ℓ by the single integer ℓ and decode the integer ℓ to see the content of B_ℓ. By removing the element $B_1 = \imath I^{\otimes n}$, the remaining B_ℓ's, $\ell = 2, \ldots 4^n$, form a mutually orthogonal basis for the Lie algebra $\mathfrak{su}(2^n)$ which is of real dimension $4^n - 1$. Elements within a specified subset $V \subset \mathfrak{su}(2^n)$ are characterized by a finite collection $\{B_{\ell_j}\}_{j=1}^k$. To construct $\mathfrak{g}(V)$, it suffices to focus on the minimal subalgebra that is generated by the associated collection $\{B_{\ell_j}\}_{j=1}^k$.

It is obvious that if $m < n$, then $\mathfrak{su}(2^m) \hookrightarrow \mathfrak{su}(2^n)$, i.e., $\mathfrak{su}(2^m)$ is naturally embedded in $\mathfrak{su}(2^n)$ by regarding $\mathfrak{d}_k = \mathfrak{0}$ for $k = m+1, \ldots, n$. Thus, the ordinal assigned to a Pauli string in $\mathfrak{su}(2^m)$ remains unchanged when considered as an element in $\mathfrak{su}(2^n)$. If the maximal ordinal number of elements in a given $V \subset \mathfrak{su}(2^n)$ is less than 4^{m-1}, then it suffices to work with $\mathfrak{su}(2^m)$ to obtain $\mathfrak{g}(V)$.

Pauli matrices enjoy many interesting properties and are readily implementable on a quantum machine [14,32]. They satisfy the relationships

$$X * X = Y * Y = Z * Z = -\imath X * Y * Z = I, \quad (7)$$

by which the multiplications among X, Y, Z are established in Table 1. Notably they exhibit cyclic relationships which resemble, but are distinct from, the quaternion multiplications. From this table, is also easy to the Lie bracket operation which we present below.

Suppose that B_i is expressed in terms of its n-digit ID as $B_i = \imath\mathfrak{d}_{i_n}\ldots\mathfrak{d}_{i_1}$, where $\mathfrak{d}_{i_k} \in \{\mathfrak{0}, \mathfrak{1}, \mathfrak{2}, \mathfrak{3}\}$ for $k = 1, \ldots, n$. Then the commutator $[B_i, B_j]$ is given by

$$[B_i, B_j] = -((\mathfrak{d}_{i_n} \dots \mathfrak{d}_{i_1}) * (\mathfrak{d}_{j_n} \dots \mathfrak{d}_{j_1}) - (\mathfrak{d}_{j_n} \dots \mathfrak{d}_{j_1}) * (\mathfrak{d}_{i_n} \dots \mathfrak{d}_{i_1}))$$
$$= -((\mathfrak{d}_{i_n} * \mathfrak{d}_{j_n}) \dots (\mathfrak{d}_{i_1} * \mathfrak{d}_{j_1}) - (\mathfrak{d}_{j_n} * \mathfrak{d}_{i_n}) \dots (\mathfrak{d}_{j_1} * \mathfrak{d}_{i_1})). \quad (8)$$

According to (7), each of the matrix-to-matrix multiplications of the form $\mathfrak{d}_{i_k} * \mathfrak{d}_{j_k}$, $k = 1, \dots, n$, can be efficiently computed using Table 1. Note further that, since the products $\mathfrak{d}_{i_k} * \mathfrak{d}_{j_k}$ and $\mathfrak{d}_{j_k} * \mathfrak{d}_{i_k}$ differ only by a possible negative sign, the two tensor products on the right side of equation (7) are essentially the same. We have either $[B_i, B_j] = 0$ or $[B_i, B_j] = -2(\mathfrak{d}_{i_n} * \mathfrak{d}_{j_n}) \dots (\mathfrak{d}_{i_1} * \mathfrak{d}_{j_1})$. We have thus proved the following lemma whose cyclic property is useful for avoiding repeated calculation.

Lemma 2.1. *If the commutator $[B_i, B_j]$ of two distinct B_i, B_j is not zero, then $[B_i, B_j] = cB_k$ for some $k \neq i$ or j, and c is either 0 or -2. In this case, it is also true that $[B_j, B_k] = cB_i$ and $[B_k, B_i] = cB_j$. The proportional constants c depend on the makeup of the Pauli strings.*

More specifically, the matrix product $\mathfrak{d}_{i_k} * \mathfrak{d}_{j_k}$ must assume one of the three possible types, $\imath\mathfrak{d}^{(\alpha)}$, $-\imath\mathfrak{d}^{(\beta)}$, or $\mathfrak{d}^{(\gamma)}$, with $\mathfrak{d}^{(\alpha)}, \mathfrak{d}^{(\beta)}, \mathfrak{d}^{(\gamma)} \in \{0, 1, 2, 3\}$. The reverse product $\mathfrak{d}_{j_k} * \mathfrak{d}_{i_k}$ yields accordingly $-\imath\mathfrak{d}^{(\alpha)}$, $\imath\mathfrak{d}^{(\beta)}$, or $\mathfrak{d}^{(\gamma)}$. Since $\mathfrak{d}^{(\gamma)}$ does not change sign, let α and β denote the total numbers of factors in types $\imath\mathfrak{d}^{(\alpha)}$ and $-\imath\mathfrak{d}^{(\beta)}$, respectively, in the tensor product $(\mathfrak{d}_{i_n} * \mathfrak{d}_{j_n}) \dots (\mathfrak{d}_{i_1} * \mathfrak{d}_{j_1})$. Then $[B_i, B_j]$ is of the form

$$[B_i, B_j] = -(\imath^{\alpha+\beta}((-1)^\beta - (-1)^\alpha))(\mathfrak{d}_{\ell_n} \dots \mathfrak{d}_{\ell_1}), \quad (9)$$

for some $\mathfrak{d}_{\ell_k} \in \{0, 1, 2, 3\}$. The n-digit ID of the bracket $[B_i, B_j]$ is thus completely determined, without executing any matrix or tensor multiplications at all.

Lemma 2.2. *When α and β are simultaneously even or odd in (9), $[B_i, B_j] = 0$. Equivalently, if the matrix product B_iB_j is real-valued, then B_i and B_j commute.*

The process described above computes the Lie bracket with minimal index retrievals or swaps. A fundamental aspect of constructing the dynamical Lie subalgebra $\mathfrak{g}(V)$ is the iterative verification of element membership, which demands careful implementation to ensure computational efficiency. Toward this end, Algorithm 1 presents a streamlined prototype that captures the essential steps of the method. To demonstrate the core concepts, we omit the additional refinements that would eliminate redundant calculations. Arithmetic operations are expressed using select Matlab commands to maintain conciseness and transparency.

A particularly noteworthy feature of the implementation is the handling of matrix-to-matrix multiplications at Lines 16 and 24. Rather than executing each multiplication $\mathfrak{d}_{i_t} * \mathfrak{d}_{j_t}$ for $t = 1, \dots n$ individually by referencing Table 1 for every possible digit pair (i_t, j_t), we use the variable col to efficiently aggregate the relevant column indices for all digits across all elements in V. This is achieved by treating Table 1 as a single, contiguous one-dimensional array in memory, as is standard in computer architecture, and then systematically mapping all pairs of indices (i_t, j_t) to positions within this linear array. This enables vectorized memory access, allowing the simultaneous multiplication of a single element in V with all others. The result is a significant improvement in computational efficiency.

To illustrate the procedure with greater clarity, consider the scenario that $n = 3$ and $V = \{28, 38, 53\}$. The function f4b maps these ordinals to Pauli strings, whose digital representations, when read from bottom to top, form the columns of the matrix $\begin{bmatrix} 3 & 1 & 0 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}$, respectively. The variable $\mathsf{col} = \begin{bmatrix} 4 & 2 & 1 \\ 3 & 2 & 2 \\ 2 & 3 & 4 \end{bmatrix}$, produced by Line 12, designates the column indices for each digit in Table 1. Consequently, the entries in the column $\begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix} + (\begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}) * 4 = \begin{bmatrix} 8 \\ 7 \\ 10 \end{bmatrix}$ correspond to the positions in the linear array of Table 1 for the product $B_{28} * B_{38}$. Specifically, without performing explicit calculations, we retrieve $B_{28} * B_{38} = \imath(1 \otimes 2 \otimes 3) * \imath(2 \otimes 1 \otimes 1) = -(1 * 2) \otimes (2 * 1) \otimes (3 * 1)$ directly from the 8th, 7th and 10th entries of the linear array in Table 1, which are $\imath 2$, $-\imath 3$ and $\imath 3$, respectively. At the end, it can be verified that $[B_{28}, B_{38}] = -2B_{63}$ and that the algorithm produces $\mathfrak{g}(V) = \text{span}\{11, 18, 28, 38, 53, 63\}$.

The two for-loops at Lines 15 and 23, though structurally identical, fulfill distinct roles: the first iterates over rectangular blocks for checking newly incorporated elements, while the second is restricted to triangular blocks to take advantage of anti-symmetry. Importantly, the iterations within each loop are mutually independent and can be executed in parallel, further enhancing computational efficiency. Additional improvements are possible by exploiting the cyclic structure inherent in the problem, a key aspect that we explore in the subsequent discussion.

To elucidate the computational demands, we consider the function $T(n)$, which quantifies the time required to compute either the product or the resulting indices of two Pauli strings within the algebra $\mathfrak{su}(2^n)$. By the fact that

$$B_i * B_j = -(\mathfrak{d}_{i_n} \dots \mathfrak{d}_{i_1}) * (\mathfrak{d}_{j_n} \dots \mathfrak{d}_{j_1})$$
$$= -(\mathfrak{d}_{i_n} * \mathfrak{d}_{j_n})((\mathfrak{d}_{i_{n-1}} \dots \mathfrak{d}_{i_1}) * (\mathfrak{d}_{j_{n-1}} \dots \mathfrak{d}_{j_1})),$$

we see that $T(n)$ is related to $T(n-1)$ by one additional product $\mathfrak{d}_{i_n} * \mathfrak{d}_{j_n}$ which is an operation in $\mathfrak{su}(2)$. Indeed, this product is determined via a constant-time lookup in the 4×4 table $\mathfrak{T}^{(1)}$, which also yields the associated phase factor $\{1, \imath, -\imath\}$. Denoting the cost for this constant-time lookup by $O(1)$, we obtain the recurrence relationship

$$T(n) = T(n-1) + O(1) = T(n-2) + 2O(1) = \dots = O(n). \quad (10)$$

Instead of performing high-dimensional matrix multiplications in the conventional way which costs $O((2^n)^3)$, the fact that the complexity grows linearly with the number n of qubits is remarkable.

In Algorithm 1, our strategy is to multiply a single element by all others in the current set V simultaneously using the col aggregation. Let $|V|$ denote the number of elements in V. The cost to update the current V in one iteration is dominated by the vectorized lookup which implies the complexity $O(|V|n)$. The algorithm terminates when the subspace V reaches closure under the bracket product. Under the assumption that each iteration adds at least one new basis element, the algorithm takes at most $\dim(\mathfrak{g}(V))$ iterations. Therefore, the total theoretical complexity is bounded by $O((\dim \mathfrak{g}(V))^2 n)$. In the case when $\mathfrak{g}(V)$ grows to the entire space $\mathfrak{su}(2^n)$, the worst-case total cost is $O(16^n n)$.

3. Nested cyclic structure

There is an additional structure inherent in the multiplications. The pattern occurs repeatedly to form a nested, cyclic, fractal-like structure. In this section, we explain how the nature of this structure arises, which is not only mathematically intriguing but also holds promise for streamlining arithmetic operations involving ordinal numbers. Drawing on the relationship in (9), it suffices to consider the matrix multiplications only. Throughout, we operate directly on the ordinal indices associated with Pauli strings.

Let $\mathfrak{T}^{(1)}$ denote the 4×4 matrix in Table 1 which represents the basic matrix-to-matrix multiplications over $\mathfrak{su}(2^1)$. We can extend $\mathfrak{T}^{(1)}$ to the multiplication table $\mathfrak{T}^{(n)}$ over $\mathfrak{su}(2^n)$ without involving the bit-to-bit multiplication as we have described in the preceding section. Indeed, we can carry out the multiplication by blocks.

To demonstrate our point of this extension, we consider the transition from $n = 1$ to $n = 2$. Partition the ordinal numbers of the Pauli strings in $\mathfrak{su}(2^2)$ into four subsets, denoted by

$$\mathfrak{G}^{(2)}_\ell := \{4\ell, 4\ell + 1, 4\ell + 2, 4\ell + 3\}, \quad \ell = 0, 1, 2, 3. \quad (11)$$

Algorithm 1 Dynamical Lie Algebra Generator.

procedure DLA(n, V) ▷ Given the dimension n and the generator V
  $M = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3\imath & -2\imath \\ 2 & -3\imath & 0 & 1\imath \\ 3 & 2\imath & -1\imath & 0 \end{bmatrix}$; ▷ Basic multiplication table
  % Define anonymous functions for digit conversion.
  pow4 = 4.^(0 : $n-1$)
  f4b = @(j) floor(rem(j./pow4, 4)); ▷ Converting ordinal j to quaternary b
  b4f = @(b) b * pow4′; ▷ Converting quaternary b to ordinal j
  $loop = 1$;
  $start = 1$;
  $\Delta = V$;
  % Check blocks in loops until no new element is found. Integers only. No $\imath$.
  while $loop = 1$ **do**
    col = f4b($\Delta' - 1$)′ + 1; ▷ Prepare the column numbers
    $gnew = []$;
    %% Can replace **for** by **parfor** for parallel computation
    for $t = 1 : start$ **do**
      $P0 = M(col(:, t) + (col(:, start + 1 : end) - 1) * 4)$; ▷ Perform products
      α = sum(imag($P0$) > 0);
      β = sum(imag($P0$) < 0);
      nz = find((−1)^(β) ≠ (−1)^(α)); ▷ Finding nonzeros based on (9)
      new=setdiff(b4f((abs(real($P0(:, nz)$)) + abs(imag($P0(:, nz)$)))′)′ + 1, Δ));
      $gnew = [gnew, new]$;
    end for
    for $t = start + 1$: length(Δ) − 1 **do**
      $P0 = M(col(:, t) + (col(:, t + 1 : end) - 1) * 4)$;
      α = sum(imag($P0$) > 0);
      β = sum(imag($P0$) < 0);
      nz = find((−1)^(β) ≠ (−1)^(α));
      new=setdiff(b4f((abs(real($P0(:, nz)$)) + abs(imag($P0(:, nz)$)))′)′ + 1, Δ));
      $gnew = [gnew, new]$;
    end for
    %% Check to see if any new element is found
    δ = setdiff($gnew, \Delta$);
    if isempty(δ) == 0 **then**
      $\Delta = [\Delta, \delta]$;
      $start$ = size(Δ, 2) − length(δ);
    else
      $loop = 0$;
    end if
  end while
  return Δ ▷ This is the dynamical Lie algebra $\mathfrak{g}(V)$
end procedure

For $s, t \in \{0, 1, 2, 3\}$, let $\mathfrak{T}^{(2)}_{st} \in \mathbb{C}^{4\times 4}$ denote the multiplication table for matrices in $\mathfrak{G}^{(2)}_s$ with those in $\mathfrak{G}^{(2)}_t$. Collectively, these form the array $\mathfrak{T}^{(2)} := [\mathfrak{T}^{(2)}_{st}]$. Observe that $\mathfrak{T}^{(1)}$ is naturally embedded in $\mathfrak{T}^{(2)}$ as the block $\mathfrak{T}^{(2)}_{00}$. We assert that each $\mathfrak{T}^{(2)}_{st}$ can be obtained from $\mathfrak{T}^{(1)}$ via a suitable shift and, if necessary, multiplying by a factor of $\pm\imath$. To be more specific about this construction, write $\mathfrak{T}^{(1)} := [\tau^{(1)}_{st}]$ in terms of its entries. Let each entry be factorized as the product

$$\tau^{(1)}_{st} = \lambda^{(1)}_{st}\omega^{(1)}_{st}, \tag{12}$$

where $\lambda^{(1)}_{st}$ and $\omega^{(1)}_{st}$ taking values from the sets $\{1, \imath, -\imath\}$ and $\{0, 1, 2, 3\}$, respectively. For future reference, we refer to $\lambda^{(1)}_{st}$ as the sign and $\omega^{(1)}_{st}$ as the magnitude of $\tau^{(1)}_{st}$. The full set of multiplication tables in $\mathfrak{T}^{(2)}$ is generated as follows.

Lemma 3.1. *For a given pair (s, t) with $s, t \in \{0, 1, 2, 3\}$, the multiplication table $\mathfrak{T}^{(2)}_{st}$ is constructed by first adding the integer $4\omega^{(1)}_{st}$ to the magnitude integer of every entry in $\mathfrak{T}^{(1)}$ without changing its original type and sign, and then multiplying all by $\lambda^{(1)}_{st}$. In other words, $\mathfrak{T}^{(2)}_{st}$ is of the form in Table 2.*

Table 2
Multiplication Table $\mathfrak{T}^{(2)}_{st}$.

↓∗→	$4t$	$4t+1$	$4t+2$	$4t+3$
$4s$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st})$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+1)$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+2)$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+3)$
$4s+1$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+1)$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st})$	$\imath\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+3)$	$-\imath\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+2)$
$4s+2$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+2)$	$-\imath\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+3)$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st})$	$\imath\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+1)$
$4s+3$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+3)$	$\imath\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+2)$	$-\imath\lambda^{(1)}_{st}(4\omega^{(1)}_{st}+1)$	$\lambda^{(1)}_{st}(4\omega^{(1)}_{st})$

Proof. In the algebra $\mathfrak{su}(2^2)$, each Pauli strings is labeled by a two-digit identifier whose second digit $\mathfrak{d}_2$ ranges over $\{0, 1, 2, 3\}$. This extends the labeling in $\mathfrak{su}(2^1)$, which uses only a single digit. According to (6), all elements in $\mathfrak{G}^{(2)}_s$ share the same second digit $\mathfrak{d}_2 = s$. The special case $\mathfrak{d}_2 = 0$ corresponds to precisely $\mathfrak{su}(2^1)$ whose multiplication is already given in Table 1. Therefore, the multiplication of elements in $\mathfrak{G}^{(2)}_s$ with those in $\mathfrak{G}^{(2)}_t$ is the tensor product of the corresponding elements in $\mathfrak{T}^{(1)}$ with 4 times of the matrix product $s * t$ whose value is precisely $\tau^{(1)}_{st}$ by definition. □

By applying Lemma 3.1, we can fully determine the multiplication table $\mathfrak{T}^{(2)}$ over $\mathfrak{su}(2^2)$ block by block, as is seen in Table 3.

An interesting structural regularity is manifested in Table 3. Observe that the diagonal blocks satisfy

$$\mathfrak{T}^{(2)}_{ss} = \mathfrak{T}^{(2)}_{00} = \mathfrak{T}^{(1)},$$

and that

$$\mathfrak{T}^{(2)}_{0t} = \mathfrak{T}^{(2)}_{t0},$$

for all $s, t \in \{0, 1, 2, 3\}$. Note also that for indices $s \neq t$ with neither equal to 0, the relationship

$$\mathfrak{T}^{(2)}_{st} = -\mathfrak{T}^{(2)}_{ts}$$

holds. Furthermore, the entire block structure in $\mathfrak{T}^{(2)}$ is determined by the first row of blocks. A close examination reveals the intriguing fact that the same pattern of structure, block symmetry and anti-symmetry, and the rule of "first-row-for-all," is a recurring theme. This motif first emerges in $\mathfrak{T}^{(1)}$ and is consistently present within each block $\mathfrak{T}^{(2)}_{st}$. Ultimately, it extends to $\mathfrak{T}^{(2)}$ as a whole.

More specifically, this recurring theme is depicted in Table 4, where the curly brackets indicate subsets of specific ordinal numbers and the unbracketed letters denote matrices or scalars of the corresponding dimensions. For example, in the block $\mathfrak{T}^{(2)}$, we select the subsets $\{\tilde{A}\} = \{\hat{A}\} = \{0, 1, 2, 3\}$, $\{\tilde{B}\} = \{\hat{B}\} = \{4, 5, 6, 7\}$, and so forth; whereas for the block $\mathfrak{T}^{(2)}_{12}$, we take $\{\tilde{A}\} = \{4\}$, $\{\tilde{B}\} = \{5\}$, $\{\hat{A}\} = \{8\}$, $\{\hat{B}\} = \{9\}$, and so on. We observe in Table 3 that the multiplication rules align precisely with the structure depicted in Table 4. Notably, observe that within each block the values in the first row, labeled A, B, C, and D, determine all remaining values in the block.

Building upon our foundational understanding of the structure of $\mathfrak{T}^{(2)}$, we now generalize this insight to higher-dimensional cases. This reveals a recursive pattern that persists across all dimensions, providing a unified framework that governs the behavior of these multiplications. As will become clear from the argument below, it is sufficient to focus on the case where the sets of ordinal numbers grouped within the curly brackets in Table 4 are given by

$$\mathfrak{G}^{(n)}_\ell := \{4^{n-1}\ell, 4^{n-1}\ell + 1, \ldots, 4^{n-1}\ell + (4^{n-1} - 1)\}, \quad \ell = 0, 1, 2, 3. \tag{13}$$

When expressed in their standard n-digit representations as in (5), every element of $\mathfrak{G}^{(n)}_\ell$ shares a common last digit $\mathfrak{d}_n = \ell$. Partition $\mathfrak{T}^{(n)} := [\mathfrak{T}^{(n)}_{st}]$ into 4×4 blocks, where each block $\mathfrak{T}^{(n)}_{st}$ is the $4^{n-1} \times 4^{n-1}$ multiplication table corresponding to the product of elements from $\mathfrak{G}^{(n)}_s$ with those in $\mathfrak{G}^{(n)}_t$. Assume that the table $\mathfrak{T}^{(n-1)} = [\tau^{(n-1)}_{ij}] \in \mathbb{C}^{4^{n-1}\times 4^{n-1}}$ has already been constructed with each entry decomposed as

$$\tau^{(n-1)}_{ij} = \lambda^{(n-1)}_{ij}\omega^{(n-1)}_{ij}, \tag{14}$$

Table 3
Multiplication Table $\mathfrak{T}^{(2)}$.

↓∗→	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	3ι	-2ι	5	4	7ι	-6ι	9	8	11ι	-10ι	13	12	15ι	-14ι
2	2	-3ι	0	1ι	6	-7ι	4	5ι	10	-11ι	8	9ι	14	-15ι	12	13ι
3	3	2ι	-1ι	0	7	6ι	-5ι	4	11	10ι	-9ι	8	15	14ι	-13ι	12
4	4	5	6	7	0	1	2	3	12ι	13ι	14ι	15ι	-8ι	-9ι	-10ι	-11ι
5	5	4	7ι	-6ι	1	0	3ι	-2ι	13ι	12ι	-15	14	-9ι	-8ι	11	-10
6	6	-7ι	4	5ι	2	-3ι	0	1ι	14ι	15	12ι	-13	-10ι	-11	-8ι	9
7	7	6ι	-5ι	4	3	2ι	-1ι	0	15ι	-14	13	12ι	-11ι	10	-9	-8ι
8	8	9	10	11	-12ι	-13ι	-14ι	-15ι	0	1	2	3	4ι	5ι	6ι	7ι
9	9	8	11ι	-10ι	-13ι	-12ι	15	-14	1	0	3ι	-2ι	5ι	4ι	-7	6
10	10	-11ι	8	9ι	-14ι	-15	-12ι	13	2	-3ι	0	1ι	6ι	7	4ι	-5
11	11	10ι	-9ι	8	-15ι	14	-13	-12ι	3	2ι	-1ι	0	7ι	-6	5	4ι
12	12	13	14	15	8ι	9ι	10ι	11ι	-4ι	-5ι	-6ι	-7ι	0	1	2	3
13	13	12	15ι	-14ι	9ι	8ι	-11	10	-5ι	-4ι	7	-6	1	0	3ι	-2ι
14	14	-15ι	12	13ι	10ι	11	8ι	-9	-6ι	-7	-4ι	5	2	-3ι	0	1ι
15	15	14ι	-13ι	12	11ι	-10	9	8ι	-7ι	6	-5	-4ι	3	2ι	-1ι	0

Table 4
Recurring Structure in Matrix Multiplications.

↓∗→	$\{\hat{A}\}$	$\{\hat{B}\}$	$\{\hat{C}\}$	$\{\hat{D}\}$
$\{\tilde{A}\}$	A	B	C	D
$\{\tilde{B}\}$	B	A	$\imath D$	$-\imath C$
$\{\tilde{C}\}$	C	$-\imath D$	A	$\imath B$
$\{\tilde{D}\}$	D	$\imath C$	$-\imath B$	A

where $\lambda_{ij}^{(n-1)} \in \{1, \imath, -\imath\}$ denotes the sign and $\omega_{ij}^{(n-1)} \in \{0, 1, \dots 4^{n-1} - 1\}$ the magnitude. Define

$$\Lambda^{(n-1)} := [\lambda_{ij}^{(n-1)}].$$

The following theorem establishes a systematic procedure for constructing of $\mathfrak{T}^{(n)}$ from its predecessor $\mathfrak{T}^{(n-1)}$, thereby extending the multiplication table to any dimension. Its elegance lies in the concise recursive formula, which enables the generation of the entire structure $\mathfrak{T}^{(n)}$ with remarkable simplicity.

Theorem 3.2. *For $n \geq 2$, the multiplication tables satisfy the relationship*

$$\mathfrak{T}^{(n)} = \Lambda^{(1)} \otimes \mathfrak{T}^{(n-1)} + 4^{n-1}\mathfrak{T}^{(1)} \otimes \Lambda^{(n-1)}. \tag{15}$$

Proof. If suffices to show that the blocks of $\mathfrak{T}^{(n)}$ can be obtained from the formula

$$\mathfrak{T}_{st}^{(n)} = (\mathfrak{T}^{(n-1)} + 4^{n-1}\omega_{st}^{(1)}\Lambda^{(n-1)})\lambda_{st}^{(1)}, \quad s, t = 0, 1, 2, 3. \tag{16}$$

The relationship (16) holds by virtue of the fact that $\mathfrak{T}^{(n-1)}$ is already embedded in $\mathfrak{T}^{(n)}$ as the block $\mathfrak{T}_{00}^{(n)}$. Elements in $\mathfrak{G}_s^{(n)}$ differ from those in $\mathfrak{G}_t^{(n)}$ only at the nth digit $\mathfrak{d}_n$. If an element in $\mathfrak{G}_s^{(n)}$ is to be multiplied by an element in $\mathfrak{G}_t^{(n)}$, we only need to identify the corresponding entry in $\mathfrak{T}^{(n-1)}$, omitting the final digit, which is then combined via tensor product with 4^{n-1} times of the matrix product $s * t$ whose value is precisely captured by $\tau_{st}^{(1)}$. □

The relationship (16) implies that the block $\mathfrak{T}_{st}^{(n)}$ is a shift of the block $\mathfrak{T}^{(n-1)}$ with at most a change of sign by $\lambda_{st}^{(1)}$. Note that all entries in $\mathfrak{T}_{st}^{(n-1)}$ are uniformly shifted by the amount $4^{n-1}\omega_{st}^{(1)}$ in the same direction as $\mathfrak{T}^{(n-1)}$ because the signs in $\Lambda^{(n-1)}$ are precisely those in $\mathfrak{T}^{(n-1)}$.

Corollary 3.3. *The block structure characterized by the "first-row-for-all" property, as detailed in Table 4, is preserved in $\mathfrak{T}^{(n)}$.*

Proof. With the formula (16) in hand, we compare $\mathfrak{T}_{23}^{(n)}$ against $\mathfrak{T}_{01}^{(n)}$. Clearly, $\omega_{01}^{(1)} = \omega_{23}^{(1)} = 1$, $\lambda_{01}^{(1)} = 1$, and $\lambda_{23}^{(1)} = \imath$. Therefore, $\mathfrak{T}_{23}^{(n)} = \imath\mathfrak{T}_{01}^{(n)}$. Analogous relations also hold for the other cases. □

Moreover, since $\tau_{0t}^{(1)} = t$ for any $t \in \{0, 1, 2, 3\}$, we can further reduce the calculation to considering only the first block as the following observation reveals.

Corollary 3.4. *Suppose that $\mathfrak{T}^{(n)}$ is partitioned according to the choice of subsets (13). Then the blocks B, C, and D in Table 4 can be obtained by shifting $A = \mathfrak{T}^{(n-1)}$ in the same direction $\Lambda^{(n-1)}$ by 4^{n-1}, $2 \times 4^{n-1}$, and $3 \times 4^{n-1}$, respectively.*

Since other blocks $\mathfrak{T}_{st}^{(n)}$ are obtainable according to the structure specified in Table 4, the product $(\mathfrak{d}_{i_n} \dots \mathfrak{d}_{i_1}) * (\mathfrak{d}_{j_n} \dots \mathfrak{d}_{j_1})$ in $\mathfrak{T}^{(n)}$ can be identified as the product $(\mathfrak{d}_{i_n-1} \dots \mathfrak{d}_{i_1}) * (\mathfrak{d}_{j_n-1} \dots \mathfrak{d}_{j_1})$ in $\mathfrak{T}^{(n-1)}$. This process can be applied recursively as needed.

The shifting strategy outlined above is, in essence, mathematically identical to the previously discussed bitwise multiplication approach. What distinguishes it, however, is the ability of performing the multiplication block by block rather than bit by bit.

We have mentioned earlier that the nested, cyclic behavior, characterized by Theorem 3.2, repeats itself to form fractal-like structure. This phenomenon is vividly illustrated in Fig. 1, which displays the magnitudes of the entries in $\mathfrak{T}^{(4)}$. The left panel already hints at an emerging fractal pattern. When the surface is tilted, as shown on the right, one can discern that each visible "triangle" harbors further layers of intricate, nested fractal detail within its boundaries. This fractal-like symmetry, prevalent in the multiplication tables $\mathfrak{T}^{(n)}$ of any dimension n, is formally grounded in the recursive tensor structure established in Theorem 3.2. Specifically, the expression (15) characterizes a discrete self-similarity, where the global structure of the $\mathfrak{su}(2^n)$ algebra is hierarchically constructed from scaled and shifted replicas of its lower-dimensional counterparts. This nested mapping ensures that the fundamental cyclic properties of $\mathfrak{su}(2)$ are preserved across all scales of n, satisfying the conceptual definition of a fractal as an object that is exactly or approximately similar to a part of itself.

Finally, as established in (9), the bracket product table can be directly derived from the matrix multiplication table by taking the imaginary part of each entry and multiplying it by -2. This method provides a clear and direct pathway from the structure of the matrix multiplication table to the corresponding bracket product table. For instance, Table 3 is transformed into Table 5 by this procedure. It is important to note, however, that the convenient "first-row-for-all" pattern observed in Table 4 no longer holds under these new circumstances.

4. Recursive algorithm

Our objective is to construct $\mathfrak{g}(V)$ in a manner that is both efficient and conceptually clear. Instead of laboriously assembling the entire multiplication table in advance and consulting it for each bracket computation, we seek a more direct approach to determining membership in

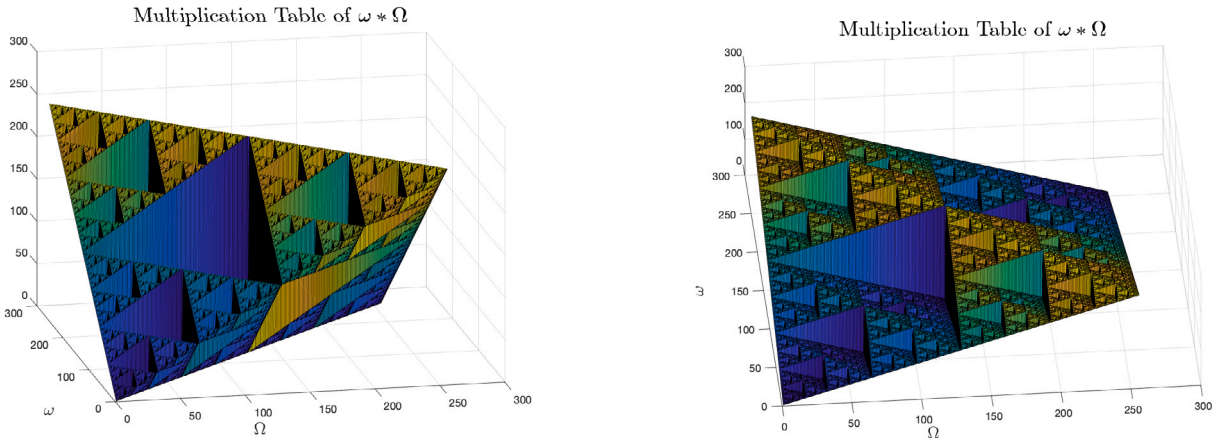

Fig. 1. Fractal-like structure within the multiplication table $\mathfrak{T}^{(4)}$.

Table 5
Bracket Product Table for $n = 2$.

$\frac{[\downarrow,\rightarrow]}{-?}$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	$3\imath$	$-2\imath$	0	0	$7\imath$	$-6\imath$	0	0	$11\imath$	$-10\imath$	0	0	$15\imath$	$-14\imath$
2	0	$-3\imath$	0	$1\imath$	0	$-7\imath$	0	$5\imath$	0	$-11\imath$	0	$9\imath$	0	$-15\imath$	0	$13\imath$
3	0	$2\imath$	$-1\imath$	0	0	$6\imath$	$-5\imath$	0	0	$10\imath$	$-9\imath$	0	0	$14\imath$	$-13\imath$	0
4	0	0	0	0	0	0	0	0	$12\imath$	$13\imath$	$14\imath$	$15\imath$	$-8\imath$	$-9\imath$	$-10\imath$	$-11\imath$
5	0	0	$7\imath$	$-6\imath$	0	0	$3\imath$	$-2\imath$	$13\imath$	$12\imath$	0	0	$-9\imath$	$-8\imath$	0	0
6	0	$-7\imath$	0	$5\imath$	0	$-3\imath$	0	$1\imath$	$14\imath$	0	$12\imath$	0	$-10\imath$	0	$-8\imath$	0
7	0	$6\imath$	$-5\imath$	0	0	$2\imath$	$-1\imath$	0	$15\imath$	0	0	$12\imath$	$-11\imath$	0	0	$-8\imath$
8	0	0	0	0	$-12\imath$	$-13\imath$	$-14\imath$	$-15\imath$	0	0	0	0	$4\imath$	$5\imath$	$6\imath$	$7\imath$
9	0	0	$11\imath$	$-10\imath$	$-13\imath$	$-12\imath$	0	0	0	0	$3\imath$	$-2\imath$	$5\imath$	$4\imath$	0	0
10	0	$-11\imath$	0	$9\imath$	$-14\imath$	0	$-12\imath$	0	0	$-3\imath$	0	$1\imath$	$6\imath$	0	$4\imath$	0
11	0	$10\imath$	$-9\imath$	0	$-15\imath$	0	0	$-12\imath$	0	$2\imath$	$-1\imath$	0	$7\imath$	0	0	$4\imath$
12	0	0	0	0	$8\imath$	$9\imath$	$10\imath$	$11\imath$	$-4\imath$	$-5\imath$	$-6\imath$	$-7\imath$	0	0	0	0
13	0	0	$15\imath$	$-14\imath$	$9\imath$	$8\imath$	0	0	$-5\imath$	$-4\imath$	0	0	0	0	$3\imath$	$-2\imath$
14	0	$-15\imath$	0	$13\imath$	$10\imath$	0	$8\imath$	0	$-6\imath$	0	$-4\imath$	0	0	$-3\imath$	0	$1\imath$
15	0	$14\imath$	$-13\imath$	0	$11\imath$	0	0	$8\imath$	$-7\imath$	0	0	$-4\imath$	0	$2\imath$	$-1\imath$	0

$\mathfrak{g}(V)$. To this end, we propose to exploit the recurring patterns present in Table 4, enabling us to implement either a Jacobi or Gauss-Seidel style update scheme, proceeding block by block. This approach is inspired by the divide-and-conquer strategy underlying the fast Fourier transform, allowing us to decompose the problem into manageable subproblems and thereby achieve both efficiency and clarity in the construction.

At the heart of the method lies a fundamental procedure, hereafter referred to as Do_It, which computes the product $[p_i, q_j]$ directly for all elements $p_i \in \{P\}$ and $q_j \in \{Q\}$, where $\{P\}$ and $\{Q\}$ are two given sets of ordinal numbers. When it becomes necessary to emphasize the particular subsets involved, we write this operation as $[\{P\}, \{Q\}]$. Since this procedure is expensive, we invoke the routine Do_It only when the cardinalities of $\{P\}$ and $\{Q\}$ fall below a specified threshold. When the threshold is not met, we further break down $\{P\}$ and $\{Q\}$ systematically according to the specific rules described below. By partitioning the problem into blocks that reflect this structure in Table 4, we may employ a divide-and-conquer strategy reminiscent of the Fast Fourier Transform (FFT), which has the potential to significantly improve computational complexity.

We first discuss a Jacobi type updating mechanism block by block. Given V, we begin by partitioning it into four mutually disjoint groups $\{A\}, \{B\}, \{C\}, \{D\}$ according to (13) with $\{A\} \subset \mathfrak{G}_0^{(n)}$, $\{B\} \subset \mathfrak{G}_1^{(n)}$, and so on. Some of the groups could be empty. For clarity and convenience in the exposition of the algorithm, we introduce the labels $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ to denote the parent groups. As previously noted, each group may equivalently be identified by the last digit $\mathfrak{d}_n$, which is the convention adopted in the implementation. These identifiers remain fixed throughout the calculation, and we will use these labels interchangeably with the corresponding subsets $\mathfrak{G}_0^{(n)}$, $\mathfrak{G}_1^{(n)}$, and so forth. To construct $\mathfrak{g}(V)$, it is sufficient to examine the bracket products among the ten possible pairings of these groups which, by the recursive structure of the construction, naturally fall into four distinct categories, each corresponding precisely to one of the parent identifiers:

$$[\{A\}, \{A\}], [\{B\}, \{B\}], [\{C\}, \{C\}], [\{D\}, \{D\}] \subset \mathfrak{G}_0^{(n)}, \tag{17}$$

$$[\{A\}, \{B\}], [\{C\}, \{D\}] \subset \mathfrak{G}_1^{(n)}, \tag{18}$$

$$[\{A\}, \{C\}], [\{B\}, \{D\}] \subset \mathfrak{G}_2^{(n)}, \tag{19}$$

$$[\{A\}, \{D\}], [\{B\}, \{C\}] \subset \mathfrak{G}_3^{(n)}. \tag{20}$$

The partitioning strategy is to ensure that computations within each of the four categories proceed entirely independently, enabling full concurrency. Results remain confined to their originating category, and even within a single category, bracket products can be evaluated in parallel. Should the cardinality of a particular pair $[\{\omega\}, \{\Omega\}]$ fail to meet the threshold, we enter the recursive mode together with its inherited parent identifier $\boldsymbol{\Gamma}$. The procedure is illustrated in the flowchart shown in Fig. 2. Each time a new element is added to the updated set V in the outer loop, the algorithm performs an additional iteration to ensure completeness. When the Do_It routine is called, it is possible to track and omit previously computed results, thereby eliminating redundant bracket product calculations at the cost of increased memory usage.

The process in the recursive mode mirrors the previous approach, with the distinction that each set $\mathfrak{G}_\ell^{(n)}$ is further subdivided according to the parent identifier $\boldsymbol{\Gamma}$. Specifically, we partition each $\mathfrak{G}_\ell^{(n)}$, $\ell = 0, 1, 2, 3$, into four contiguous segments of equal size, each comprising consecutive ordinal numbers, to systematically check for subcategories:

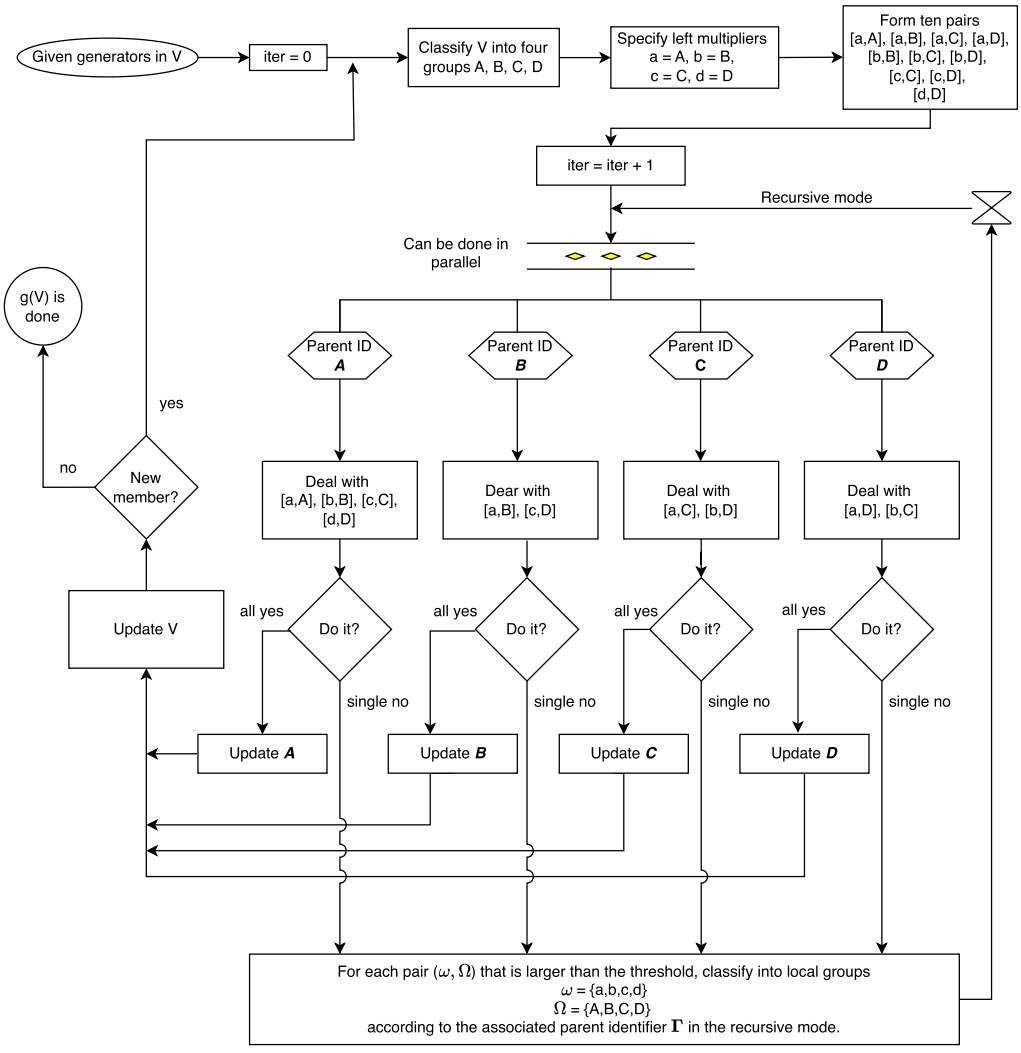

Fig. 2. Scheme of the recursive Jacobi-type iteration.

$$\mathfrak{G}_{\ell}^{(n)} = \bigcup_{s=0}^{3} \underbrace{\{4^{n-1}\ell + 4^{n-2}s, \ldots 4^{n-1}\ell + 4^{n-2}(s+1) - 1\}}_{\mathfrak{G}_{\ell,s}^{(n)}}. \tag{21}$$

Just as the set $\mathfrak{G}_{\ell}^{(n)}$ is uniquely identifiable by the last digit $\mathfrak{d}_n = \ell$, the subset $\mathfrak{G}_{\ell,s}^{(n)}$ is uniquely specified by the pair of final digits $(\mathfrak{d}_n, \mathfrak{d}_{n-1}) = (\ell, s)$. Suppose $\{\omega\} \subset \mathfrak{G}_i^{(n)}$ and $\{\Omega\} \in \mathfrak{G}_j^{(n)}$. We then subdivide $\{\omega\}$ into four mutually disjoint groups, denoted $\{A_\omega\}$, $\{B_\omega\}$, $\{C_\omega\}$, and $\{D_\omega\}$, such that $\{A_\omega\} \in \mathfrak{G}_{i,0}^{(n)}$, $\{B_\omega\} \in \mathfrak{G}_{i,1}^{(n)}$, and so forth. In the same manner, $\{\Omega\}$ is partitioned into $\{A_\Omega\}$, $\{B_\Omega\}$, $\{C_\Omega\}$, and $\{D_\Omega\}$ with $\{A_\Omega\} \in \mathfrak{G}_{j,0}^{(n)}$, and analogously for the remaining subsets. Some of these subsets could be empty. To calculate $[\{\omega\}, \{\Omega\}]$, we examine the newly formed ten bracket products, of which each resulting new element, if any, is collected into the same parent structure $\mathbf{\Gamma}$.

Again, each bracketed product described above can be computed independently, making the approach naturally amenable to parallelization. Should any resulting pair, after subdivision, still exceed the designated size threshold, the process recurses: the pair is further partitioned, and the bracket operation is applied to these smaller fragments. This recursive partitioning continues until all pairs are sufficiently small to be handled directly by the routine Do_It, which is the base case that terminates the recursion. In this way, the algorithm achieves both efficiency and scalability. Nevertheless, care must be taken in selecting the threshold for partitioning: if it is set too low, the recursion may become excessively deep, potentially resulting in errors such as "stack too deep" or "maximum recursion limit reached," indicating that the number of recursive calls exceeds the available stack space. The recursive procedure is embedded in the flowchart in Fig. 2.

Table 6
Initial setup for Jacobi-type Recursion.

$\frac{[\downarrow,\rightarrow]}{-2}$	$\{6,10,12\}$	$\{16,18,28\}$	$\{34\}$,	$\{50\}$
$\{6,10,12\}$	A	B	C	D
$\{16,18,28\}$		A	D	C
$\{34\}$			A	B
$\{50\}$				A

It is important to note that this diagram presents a simplified view: at each decision point, a single negative response prompts the procedure to re-enter the recursive mode. Conversely, when a positive decision is reached, any newly identified elements should be incorporated into $\mathbf{\Gamma}$ or retained in memory until the process concludes. This final action is denoted as "all yes" in the diagram. The return of this recursive mode serves to update V which will be checked in the main routine for any new members and, if necessary, the process continues iteratively.

To illustrate the operations of this recursive method, we consider a simple example with a low-dimensional setting. Take $n = 4$ and $V = \{7, 11, 13, 17, 19, 29, 35, 51\}$. These values represent the ordinal numbers of Pauli strings, mapped according to the rule described in equation (6). Following the algorithm, the set V is initially partitioned into four

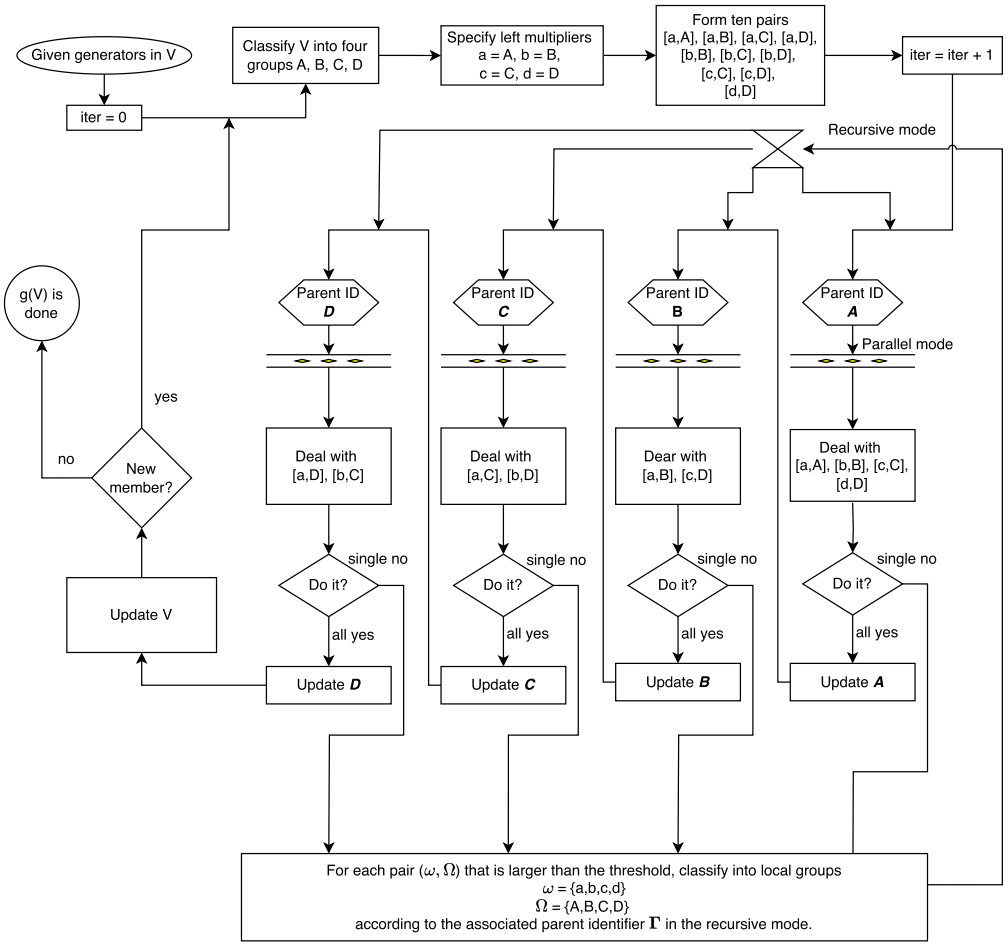

Fig. 3. Scheme of the recursive Gauss-Seidel-type iteration.

Table 7
Initial setup for Jacobi-type Recursion.

$\frac{\{1,\to\}}{\to 2}$	{6}	{10}	{12}	{16, 18}	{28}	{34},	{50}
{6}	0	$14\imath$	$-8\imath$	0	0	0	0
{10}		0	$4\imath$	0	0	0	0
{12}			0	0	0	0	0
{16, 18}				0	0	$\{50\imath, 48\imath\}$	$\{-34\imath, -32\imath\}$
{28}					0	$62\imath$	$-46\imath$
{34}						0	$16\imath$
{50}							0

groups: $A = \{6, 10, 12\}$, $B = \{16, 18, 28\}$, $C = \{34\}$, and $D = \{50\}$. Subsequently, ten pairs are formed, corresponding to the upper triangular blocks with parent ID identified in Table 6. Suppose, for illustrative purposes, that an artificial threshold of 2 is imposed. Under this constraint, the sets A and B must be subdivided further according to (21). We thus enter the recursive mode. Table 7 keeps the respective parent IDs in the corresponding blocks, but notice that the routine Do_It is invoked for "all" subblocks. As a result, the groups are updated to $A = \{4, 6, 8, 10, 12, 14\}$, $B = \{16, 18, 28\}$, $C = \{32, 34, 46\}$, and $D = \{48, 50, 62\}$, respectively. This completes the first iteration (iter $=$ 1) of the divide-and-conquer process. Since new members are added to V, we are ready to perform the second iteration with the updated groups. Since these groups are larger, the divide-and-conquer will go deeper. When doing so, however, previously computed results can be avoided to reduce redundant calculations. Ultimately, for this particular example, $\mathfrak{g}(V)$ consists of 30 basis elements.

Fig. 3 depicts the Gauss-Seidel-type updating mechanism, which differs from the Jacobi scheme in a fundamental way. In the Gauss-Seidel-type approach, as soon as the computation for the brackets with parent identifier ***A*** is complete, the updated subset $\{a\}$ within $\mathfrak{G}_0^{(n)}$ is immediately utilized in all subsequent calculations for brackets with parent identifiers ***B***, ***C***, and ***D***. This sequential updating continues: once the brackets associated with ***B*** are refreshed, their new values are incorporated into the computations for ***C*** and ***D***, and so on. In this way, each update involves the most recently available information. Note that while parallel computation is possible within each category, the progression from one category to the next remains inherently serial.

5. Computer experiments

The core Algorithm 1, together with the recursive Gauss-Seidel-type iterative procedure, has been meticulously implemented in Matlab. These codes are available to interested readers upon request. Below, we present a selection of numerical experiments that employ the recursive Gauss-Seidel-type approach.

We begin by considering one-dimensional 2-local spin systems, where the Hamiltonian takes the form described in (4). Within this framework, we present two representative choices for $\mathcal{A}$ from our broader set of experimental investigations. The comprehensive classification of the 17 distinct dynamical Lie subalgebras associated with such Hamiltonians, thoroughly catalogued in [33], serves as a robust and trusted benchmark against which we can assess the precision and dependability of our algorithm. Then, we extend our experiment to other generators V, whose corresponding dynamical Lie algebras have not yet been explored in the existing literature.

It is essential to acknowledge from the outset that the curse of dimensionality remains a formidable and unavoidable challenge. In particular,

Table 8
Cardinalities of $\mathfrak{g}(V)$ for the TFXY model.

n	3	4	5	6	7	8	9	10
V	7	10	13	16	19	22	25	28
$\mathfrak{g}(V)$	15	28	45	66	91	120	153	190
$\mathfrak{g}(V+1)$	30	126	510	2,046	8,190	32,766	131,070	–
$\mathfrak{g}(V-1)$	36	255	1,023	4,095	16,383	65,535	–	–

when dealing with physical models, the parameter n must often be restricted, as the sheer size of the set $\mathfrak{g}(V)$ can become prohibitively large.

Example 1. The Hamiltonian of the 1-dimensional Heisenberg chain with open boundary conditions is of the form

$$H = \sum_{i=1}^{n-1} (X_i X_{i+1} + Y_i Y_{i+1} + Z_i Z_{i+1}),$$

which is a special case of (4) with $\mathcal{A} = \{XX, YY, ZZ\}$. By our encoding system (6), the generator V for the case $n = 9$ has 24 elements represented by

$$V = \left\{ \begin{matrix} 6 & 11 & 16 & 21 & 41 & 61 & 81 & 161 & 241 & 321 & 641 & 961 \\ 1281 & 2561 & 3841 & 5121 & 10241 & 15361 & 20481 & 40961 & 61441 & 81921 & 163841 & 245761 \end{matrix} \right\}.$$

Our algorithm constructs the Lie algebra $\mathfrak{g}(V)$, comprising exactly 65,535 elements. This exhaustive enumeration is achieved by systematically performing 3,757,981,697 multiplications, as dictated by our theoretical framework, and subsequently translating these results into bracket products via the mechanism described in (9). Note that $\mathfrak{g}(V)$ is a subalgebra embedded in $\mathfrak{su}(2^9)$ which is of dimension 262,143. According to the work [33], however, this dynamical Lie subalgebra is isomorphic to $\mathfrak{su}(2^8)$, whose cardinality matches that of our $\mathfrak{g}(V)$. This correspondence provides strong affirmation of the accuracy and validity of our results. Furthermore, our computations provide a detailed account of the genuine elements comprising $\mathfrak{g}(V)$.

Example 2. The Hamiltonian for an XY spin chain with open boundary conditions in the presence of a random, normally distributed transverse field Z, also known as the TFXY model in the literature, is of the form [25]

$$H = \sum_{i=1}^{n-1} (X_i X_{i+1} + Y_i Y_{i+1}) + \sum_{i=1}^{n} b_i Z_i.$$

For the case $n = 10$, the generator V comprises 28 distinct elements represented by

$$V = \left\{ \begin{matrix} 4 & 6 & 11 & 13 & 21 & 41 & 49 & 81 & 161 & 193 \\ 321 & 641 & 769 & 1281 & 2561 & 3073 & 5121 & 10241 & 12289 & 20481 \\ 40961 & 49153 & 81921 & 163841 & 196609 & 327681 & 655361 & 786433 & & \end{matrix} \right\}$$

Remarkably, despite the wide spread of this generating set, the resulting algebra $\mathfrak{g}(V)$ is composed of only 190 elements. Table 8 presents the cardinalities of both the generator set V and the corresponding $\mathfrak{g}(V)$ for a range of values of n.

When we adjust the generators by increasing each ordinal in V by one, the total number of generators remains constant, yet the underlying structure is profoundly altered. The cardinality of the algebra associated with $\mathfrak{g}(V+1)$ changes dramatically. Remarkably, the cardinality of $\mathfrak{g}(V-1)$ is more than twice that of $\mathfrak{g}(V+1)$, underscoring the sensitivity of these algebras to the configuration of V. These observations reveal a delicate interplay between the configuration of V and the inherent algebraic constraints that shape $\mathfrak{g}(V)$.

Example 3. The study in [33] focuses exclusively on generators derived from translation-invariant, 2-local spin chain Hamiltonians. Within this well-defined setting, the authors identify meticulously seventeen distinct dynamical Lie subalgebras, providing a complete map of all possible scenarios. This achievement represents a significant theoretical advance, illuminating the structure of these systems with remarkable clarity. While the isomorphisms between the relevant sets are explicitly described, practical applications often require the explicit identification of elements generated by a specific choice of V. Our algorithm is crafted to meet this demand, offering a direct and flexible approach that applies even when V is arbitrary.

Table 9
Cardinalities of $\mathfrak{g}(V)$ for the 3-local model.

n	3	4	5	6	7	8	9
V	3	6	9	12	15	18	21
$\mathfrak{g}(V)$	6	36	272	2,016	8,256	32,640	131,328
$\mathfrak{g}(V-1)$	3	13	42	992	8,128	32,640	130,816
$\mathfrak{g}(V-2)$	6	30	255	2,046	16,383	65,535	–

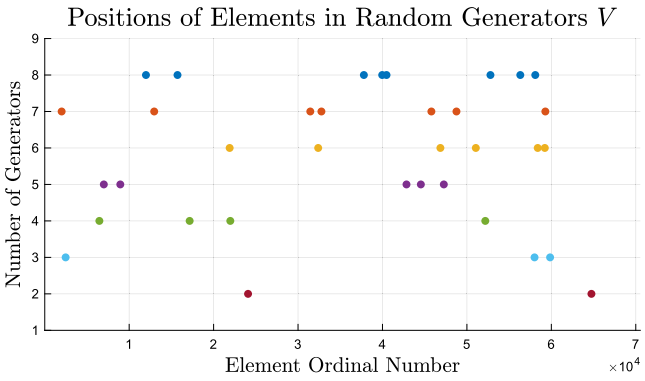

Fig. 4. Random generators V and the resulting $|\mathfrak{g}(V)|$ in $\mathfrak{su}(2^8)$.

Table 10
Cardinalities of $\mathfrak{g}(V)$ with k random elements in V.

k	2	3	4	5	6	7	8
$\mathfrak{g}(V)$	3	20	272	4,092	65,535	65,535	65,535
$\mathfrak{g}(V+1)$	3	11	137	4,092	16,512	65,535	65,535
$\mathfrak{g}(V-1)$	3	20	255	8,184	65,535	65,535	65,535

Consider a 3-local system characterized by $\mathcal{A} = \{XXY, YYZ, XYZ\}$. While this particular configuration may not directly correspond to any established physical model, and the analytic form of $\mathfrak{g}(V)$ remains to be determined, it nonetheless presents a rich mathematical landscape worthy of thoughtful exploration. The generator V for the case n = 9 has 21 elements:

$$\left\{ \begin{matrix} 38 & 58 & 59 & 149 & 229 & 233 & 593 & 913 & 929 & 2369 & 3649 \\ 3713 & 9473 & 14593 & 14849 & 37889 & 58369 & 59393 & 151553 & 233473 & 237569 & \end{matrix} \right\},$$

from which we may sequentially extract generators corresponding to all smaller values of n. The cardinalities of both the generator set V and the corresponding $\mathfrak{g}(V)$, $\mathfrak{g}(V-1)$ and $\mathfrak{g}(V-2)$ for various values of n are shown in Table 9.

Example 4. The Lie algebra $\mathfrak{su}(2^n)$ encompasses $2^{4^n-1} - 4^n$ subsets of size greater than or equal to two. Any such subset may serve as a generator V. In the previous examples, the generators V are chosen with particular structure, allowing sometimes for a more precise analysis of the resulting subalgebra $\mathfrak{g}(V)$. In this example, we consider the case $n = 8$, and select k random integers, $k = 2, \ldots, 8$, from the interval $[1, 65535]$ to act as generators.

The choices of generators in this experiment are marked in Fig. 4, where each dot along a horizontal line represents the ordinal position of a generator for a given k. It is important to note that these selections are entirely unstructured. The cardinalities of the resulting $\mathfrak{g}(V)$, as well as those of the slightly perturbed $\mathfrak{g}(V-1)$ and $\mathfrak{g}(V+1)$ are reported in Table 10.

It is remarkable that, in this particular setting, selecting as few as 6, 7, or 8 random elements for the generator V is sufficient to generate the entire Lie algebra $\mathfrak{su}(2^8)$ which comprises 65,535 elements. When this happens, the gate set is universal, meaning any unitary operation can be

approximated arbitrarily well by a finite sequence of these gates. This is a core concept in the Solovay-Kitaev theorem [20,28]. The underlying reasons for this phenomenon, as well as the specific properties that such a small set of generators must possess to span the full algebra, present intriguing mathematical questions worthy of further exploration.

6. Conclusion

We have demonstrated that the multiplicative structure of $\mathfrak{su}(2^n)$ possesses an inherent fractal symmetry that can be exploited for computational efficiency. By adopting a recursive integer-encoding scheme, we transform the problem of generating a dynamical Lie algebra from a memory-intensive matrix operation into a streamlined combinatorial task. Our analysis confirms that this approach scales linearly with the number of qubits for the fundamental string product.

For certain structured models, such as translation-invariant 2-local spin chains, the dynamical Lie algebra admits a closed-form analytic characterization. In more general settings, where a complete classification remains elusive, our framework provides a robust and scalable means to compute $\mathfrak{g}(V)$ for any prescribed generator set V.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jaca.2026.100045.

Data availability

Data will be made available on request.

References

[1] G. Aguilar, S. Cichy, J. Eisert, L. Bittel, Full classification of Pauli Lie algebras, arXiv:2408.00081, 2024.
[2] D.W. Berry, A.M. Childs, R. Cleve, R. Kothari, R.D. Somma, Simulating Hamiltonian dynamics with a truncated Taylor series, Phys. Rev. Lett. 114 (2015) 090502.
[3] A.M. Childs, N. Wiebe, Hamiltonian simulation using linear combinations of unitary operations, Quantum Inf. Comput. 12 (2012) 901.
[4] F. Casas, A. Murua, M. Nadinic, Efficient computation of the Zassenhaus formula, Comput. Phys. Commun. 183 (11) (2012) 2386–2391.
[5] M.T. Chu, Lax dynamics for Cartan decomposition with applications to Hamiltonian simulation, IMA J. Numer. Anal. 44 (3) (2024) 1406–1434.
[6] M.T. Chu, Preparing Hamiltonians for quantum simulation: a computational framework for Cartan decomposition via Lax dynamics, Math. Comput. 94 (356) (2025) 2867–2893.
[7] M.T. Chu, On the commutative substructure within Cartan pairs of subalgebras in $\mathfrak{su}(2^n)$, Result. Math. 80 (5) (2025) 164, 2025.
[8] J.I. Cirac, P. Zoller, Goals and opportunities in quantum simulation, Nat. Phys. 8 (4) (2012) 264–266.
[9] L. Clinton, J. Bausch, T. Cubitt, Hamiltonian simulation algorithms for near-term quantum hardware, Nat. Commun. 12 (1) (2021) 4989.
[10] G. Cybenko, Reducing quantum computations to elementary unitary operations, Comput. Sci. Eng. 3 (2) (2001) 27–32.
[11] M. Dağlı, D. D'Alessandro, J.D.H. Smith, A general framework for recursive decompositions of unitary quantum evolutions, J. Phys. A, Math. Theor. 41 (15) (2008) 155302.
[12] M. Dağli, Lie algebra decompositions with applications to quantum dynamics, PhD dissertation, Iowa State University, 2008, Retrospective Theses and Dissertations, 15721.
[13] D. D'Alessandro, Introduction to Quantum Control and Dynamics, 2nd ed., Chapman and Hall/CRC, 2021.
[14] T.G. de Brugière, M. Baboulin, B. Valiron, C. Allouche, Synthesizing quantum circuits via numerical optimization, in: Lecture Notes in Computer Science, Springer International, 2019, pp. 3–16.
[15] D.P. DiVincenzo, Quantum computation, Science 270 (5234) (1995) 255–261.
[16] B. Drury, P. Love, Constructive quantum Shannon decomposition from Cartan involutions, J. Phys. A, Math. Theor. 41 (39) (Sep. 2008) 395305.
[17] R.P. Feynman, Simulating physics with computers, in: Physics of Computation, Part II, Dedham, Mass., 1981, Int. J. Theor. Phys. 21 (6–7) (1981/1982) 467–488.
[18] J. Haah, M.B. Hastings, R. Kothari, G.H. Low, Quantum algorithm for simulating real time evolution of lattice Hamiltonians, SIAM J. Comput. (2018), FOCS18–250–284.
[19] T. Hawkins, Emergence of the Theory of Lie Groups: An Essay in the History of Mathematics 1869–1926, Sources and Studies in the History of Mathematics and Physical Sciences, Springer-Verlag, New York, 2000.
[20] M. Hayashi, Quantum Information Theory: Mathematical Foundation, second edition, Graduate Texts in Physics, Springer-Verlag, Berlin, 2017.
[21] J.E. Humphreys, Introduction to Lie Algebras and Representation Theory, second edition, Graduate Texts in Mathematics, vol. 9, Springer-Verlag, New York-Berlin, 1978.
[22] S. Jin, X. Li, N. Liu, Quantum simulation in the semi-classical regime, Quantum 6 (2022) 739.
[23] N. Khaneja, S.J. Glaser, Cartan decomposition of $SU(2^n)$ and control of spin systems, Chem. Phys. 267 (1) (2001) 11–23.
[24] A.W. Knapp, Lie Groups Beyond an Introduction, digital 2nd edition, Springer Science & Business Media, vol. 140, 2023.
[25] E. Kökcü, T. Steckmann, J.K. Freericks, E.F. Dumitrescu, A.F. Kemper, Fixed depth Hamiltonian simulation via Cartan decomposition, Phys. Rev. Lett. 129 (2022) 070501.
[26] S. Lloyd, Universal quantum simulators, Science 273 (5278) (1996) 1073–1078.
[27] G.H. Low, I.L. Chuang, Optimal Hamiltonian simulation by quantum signal processing, Phys. Rev. Lett. 118 (2017) 010501.
[28] M.A. Nielsen, I.L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, Cambridge University Press, 2010.
[29] W. Walter Pfeifer, The Lie Algebras $\mathrm{su}(N)$: An Introduction, Birkhäuser Verlag, Basel, 2003.
[30] M. Ragone, B.N. Bakalov, F. Sauvage, A.F. Kemper, C. Ortiz Marrero, M. Larocca, M. Cerezo, A Lie algebraic theory of barren plateaus for deep parameterized quantum circuits, Nat. Commun. 15 (1) (2024) 7172.
[31] H.N. Sá Earp, J.K. Pachos, A constructive algorithm for the Cartan decomposition of $\mathrm{SU}(2^N)$, J. Math. Phys. 46 (8) (2005) 082108.
[32] V. Shende, S. Bullock, I. Markov, Synthesis of quantum-logic circuits, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 25 (6) (2006) 1000–1010.
[33] R. Wiersema, E. Kökcü, A.F. Kemper, B.N. Bakalov, Classification of dynamical Lie algebras for translation-invariant 2-local spin systems in one dimension, arXiv:2309.05690, 2023.
[34] R. Zeier, T. Schulte-Herbrüggen, Symmetry principles in quantum systems theory, J. Math. Phys. 52 (11) (2011) 113510.