Exact Enumeration of Self-Avoiding Walks on the Two-Dimensional Square Lattice: A Symmetry-Reduced, JIT-Accelerated Backtracking Study to $n = 20$

July 12, 2026

Abstract

Self-avoiding walks (SAWs) are the canonical lattice model of excluded-volume polymers and one of the most studied objects in statistical mechanics, yet the generating sequence $c(n)$ counting distinct n-step SAWs has no known closed form and must be obtained by exact enumeration. We implement a memory-light backtracking depth-first search (DFS) that enumerates every SAW on the two-dimensional square lattice, accelerated by two complementary ideas: an eight-fold reduction of the search using the dihedral point-group symmetry D_4 of the lattice (fixing the first step and partitioning the second step into a self-symmetric "straight" branch and a reflection-doubled "turn" branch), and just-in-time (JIT) compilation of the recursive hot loop with Numba. All counts are produced *de novo* by the enumerator rather than quoted from tables. We reach $n = 20$, computing $c(20) = 897{,}697{,}164$ in 6.05 s on a single core, and confirm exact agreement with the reference integer sequence OEIS A001411 for every $n \in \{1, \ldots, 20\}$: zero disagreements. The measured runtime grows geometrically with observed base $e^{\text{slope}} \approx 2.686$, in close agreement with the theoretical time complexity $\mathcal{O}(\mu^n)$, where μ is the square-lattice connective constant; the working-memory footprint is $\mathcal{O}(n^2)$ (a 1,681 byte grid at $n = 20$). From the large-n tail we estimate μ by three independent methods—an ordinary least-squares fit of $\log c(n)$ versus n ($\hat{\mu} = 2.696$), a γ-corrected successive ratio ($\hat{\mu} = 2.632$ at $n = 20$), and a plain successive ratio ($\hat{\mu} = 2.679$)—which bracket the high-precision literature value $\mu = 2.63815853\ldots$ The residual bias is fully consistent with the known $n^{\gamma-1}$ finite-size correction with $\gamma = 43/32$. The report documents the algorithm, derives its complexity, tabulates the verified counts and timings, and discusses the interplay between finite-size corrections and the exponential cost of exhaustive enumeration.

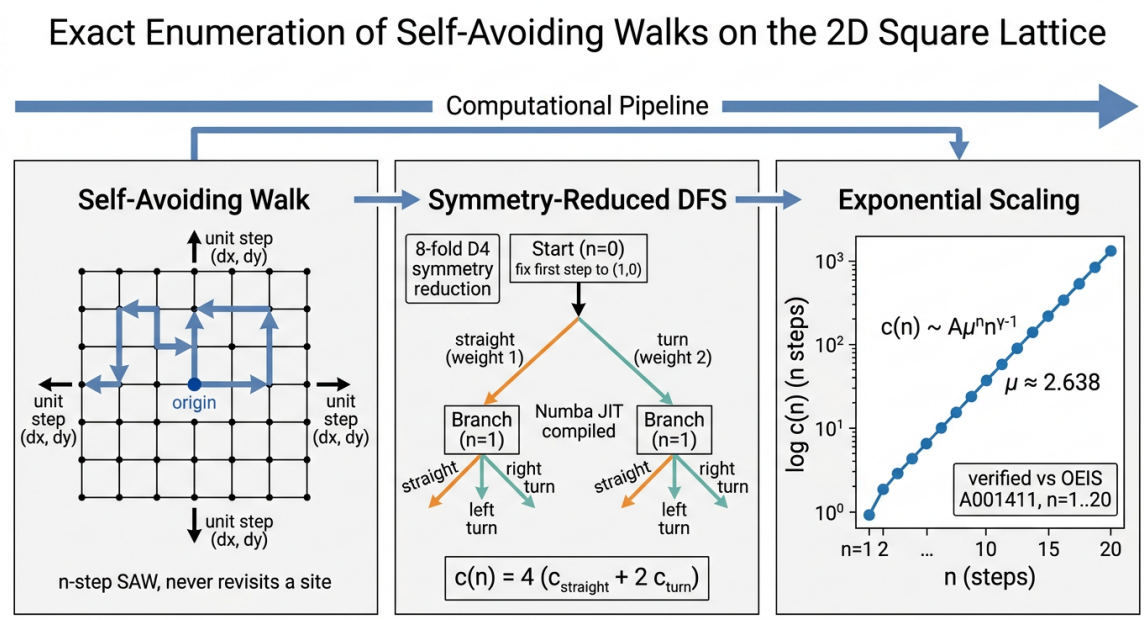

Figure 1: Graphical abstract. The enumeration pipeline: (left) an n-step self-avoiding walk on the square lattice starts at the origin and never revisits a site; (center) the search space is pruned eight-fold by fixing the first step to $(1, 0)$ and splitting the second step into a self-symmetric straight branch (weight 1) and a reflection-doubled turn branch (weight 2), with the recursive DFS compiled to native code by Numba; (right) the resulting counts grow as $c(n) \sim A\,\mu^n n^{\gamma-1}$ with $\mu \approx 2.638$, and are verified against OEIS A001411 for $n = 1, \ldots, 20$.

Contents

1 Introduction

A *self-avoiding walk* (SAW) of length n on a lattice is a sequence of n unit steps along lattice edges, starting from a fixed origin, that visits no site more than once. On the two-dimensional square lattice $\mathbb{Z}^2$ each step moves to one of the four nearest-neighbour sites, and the self-avoidance constraint forbids the walk from returning to any previously occupied vertex. The central combinatorial quantity is the *connective sequence*

$$c(n) \;=\; \#\big\{\text{ distinct } n\text{-step SAWs starting at the origin }\big\}, \tag{1}$$

where walks are counted with all lattice symmetries—that is, every rotation and reflection of a given walk is counted as a distinct object, matching the convention of the integer sequence OEIS A001411 [1]. The first few values are $c(0) = 1$, $c(1) = 4$, $c(2) = 12$, and $c(3) = 36$; the growth is rapid but strictly sub-exponential in the naive 4^n upper bound because self-avoidance progressively prunes the available continuations.

Despite the elementary definition in Eq. (1), no closed-form expression or linear recurrence for $c(n)$ is known, and it is widely believed that the ordinary generating function $\sum_n c(n)x^n$ is not differentiably finite [2, 3]. The SAW therefore occupies an unusual position: its statistics are governed by sharp asymptotic laws that can be derived heuristically and, in special cases, rigorously, yet the exact enumeration of $c(n)$ remains a purely computational problem whose difficulty grows exponentially with n.

1.1 Mathematical formulation and asymptotics

Because a walk of length $m + n$ can be decomposed—though not uniquely—into an m-step and an n-step piece, the counts satisfy the submultiplicative inequality $c(m + n) \le c(m)\, c(n)$. Fekete's subadditive lemma applied to $\log c(n)$ then guarantees that the limit

$$\mu \;=\; \lim_{n\to\infty} c(n)^{1/n} \;=\; \inf_{n\ge 1} c(n)^{1/n} \tag{2}$$

exists and is finite; μ is the *connective constant* (or growth constant) of the lattice. Its existence, together with the first non-trivial sub-exponential correction bounds $c(n) \le \mu^n \exp\big(O(\sqrt{n})\big)$, was established in the foundational work of Hammersley and Welsh [4]. The connective constant is a *non-universal* quantity that depends on the microscopic lattice geometry; for the honeycomb lattice Nienhuis conjectured, using Coulomb-gas arguments [5], the exact value $\mu_{\mathrm{hc}} = \sqrt{2+\sqrt{2}}$, which was proved three decades later by Duminil-Copin and Smirnov via a discretely holomorphic parafermionic observable [6]. No comparable exact value is known for the square lattice; the current best numerical estimate, $\mu = 2.63815853032790(3)$, was obtained by Jacobsen, Scullard and Guttmann using a topological transfer-matrix method [7], one of several complementary high-precision routes that also include numerical adaptations of the honeycomb-lattice identity to the square lattice [8].

Beyond the leading exponential growth, the counts are believed to obey the sharper asymptotic form

$$c(n) \;\sim\; A\,\mu^n\, n^{\gamma-1}, \qquad n \to \infty, \tag{3}$$

where γ is a *universal* critical exponent that depends only on the spatial dimension. In two dimensions the Coulomb-gas and conformal-field-theory analyses of Nienhuis predict the exact rational value $\gamma = 43/32$ [5], a prediction consistent with the conjectured convergence of the planar SAW to the Schramm–Loewner evolution $\mathrm{SLE}_{8/3}$ [9, 10]. The amplitude A is again non-universal; high-precision enumerations give $A \approx 1.177$ [11, 12]. Equation (3) is central to the present work: it both explains the finite-n bias we observe when estimating μ from a short series, and predicts the $\mathcal{O}(\mu^n)$ scaling of the enumeration cost.

1.2 Significance in polymer physics and statistical mechanics

The SAW is far more than a combinatorial curiosity. It is the standard lattice model of a linear polymer in a good solvent, in which the self-avoidance constraint encodes the excluded-volume interaction between monomers first analysed by Orr [13] and Flory [14]. Flory's mean-field balance of entropic elasticity against monomer repulsion predicts that the typical end-to-end distance swells as $R \sim n^{\nu}$ with $\nu \approx 3/4$ in two dimensions, in contrast to the $\nu = 1/2$ of an ideal random walk. De Gennes subsequently placed this scaling picture on a field-theoretic footing by identifying the SAW with the $n \to 0$ limit of the $O(n)$ vector model [15], which exposes SAW critical exponents as members of well-defined universality classes and connects them to the broader theory of critical phenomena. The rigorous probabilistic theory, including the lace expansion in high dimensions and the two-dimensional conformal-invariance program, is developed in the monograph of Madras and Slade [16].

Within this landscape, exact enumeration plays a distinctive and enduring role. Short but *exact* series for $c(n)$ are the raw material from which the connective constant, the critical exponents, and confluent correction terms are extracted by series-analysis techniques such as differential approximants [2, 3]. Because every digit of every $c(n)$ is certain, enumeration provides ground-truth data against which stochastic methods—most prominently the pivot algorithm [17–19] and the pruned-enriched Rosenbluth method [20]—and rigorous bounds can be validated. The state of the art in exact enumeration is represented by the finite-lattice and transfer-matrix algorithms of Conway, Enting and Guttmann [2], Jensen and Guttmann [11, 21], and Clisby and Jensen [12], which reach walk lengths far beyond the direct-enumeration regime, as well as the lace-expansion enumeration of Clisby, Liang and Slade [22].

1.3 Scope and contributions of this report

The purpose of this report is deliberately foundational: we build, from first principles, a correct and efficient *direct* enumerator—one that literally constructs and counts every walk—and use it to reproduce the sequence $c(n)$ up to the largest n we can reach on a single core within a few seconds of compute per term. Our contributions are as follows. First, we present a compact backtracking DFS enumerator whose only data structure is a dense occupancy grid, giving an $\mathcal{O}(n^2)$ memory footprint and $\mathcal{O}(1)$ self-avoidance tests. Second, we exploit the full D_4 point-group symmetry of the square lattice to reduce the search by a factor of eight, and accelerate the recursion with Numba JIT compilation [23] built on the NumPy array model [24]. Third, we reach $n = 20$, we tabulate the exact counts *and* the per-n runtimes, and we verify every value against OEIS A001411, reporting explicitly that there are no disagreements. Fourth, we derive and empirically confirm the $\mathcal{O}(\mu^n)$ time and $\mathcal{O}(n^2)$ space complexity, and we use the computed series to estimate the connective constant by three independent estimators, situating the residual error within the finite-size-correction theory of Eq. (3). We emphasise that the enumerator is a direct-search method chosen for transparency and verifiability; it is not competitive in reach with transfer-matrix methods, and we discuss that gap candidly in Section 4.

2 Methodology

2.1 The backtracking depth-first search

Our enumerator counts SAWs by an exhaustive but pruned depth-first traversal of the tree of partial walks. The root is the origin; each node of depth k is a k-step self-avoiding walk, and its children are the self-avoiding one-step extensions. Counting $c(n)$ is then equivalent to counting the leaves at depth n. The traversal is realised recursively with classic *backtracking*: a site is marked occupied before descending into it and unmarked upon return, so that a single mutable data structure represents the current partial walk at all times and no walk is ever stored in full.

Self-avoidance is tested against a dense two-dimensional occupancy grid rather than a hash set of visited coordinates. An n-step walk starting at the origin can reach no site farther than n in the L^1 (taxicab) metric, so a grid of side $2n+1$ centred on the origin at index (n, n) contains every reachable site with room to spare. Because the border of this grid is never occupied and the head can move by at most one site per step, no path can leave the array; this eliminates per-access bounds checking entirely. Each cell is a single `uint8`, giving $\mathcal{O}(1)$ lookups and updates and a total grid size of $(2n+1)^2$ bytes—for example 1,681 B at $n = 20$. The core of the recursion is shown in Listing 1.

Listing 1: Core backtracking DFS. The grid is mutated in place and fully restored on return; the four nearest neighbours are tried in turn and free sites are recursed into.

```
@njit(cache=True)
def _dfs(grid, x, y, remaining):
    if remaining == 0:
        return 1 # a completed walk of the target length
    total = 0
    # +x, -x, +y, -y neighbours, each guarded by the occupancy test
    if grid[x + 1, y] == 0:
        grid[x + 1, y] = 1
        total += _dfs(grid, x + 1, y, remaining - 1)
        grid[x + 1, y] = 0 # backtrack
    if grid[x - 1, y] == 0:
        grid[x - 1, y] = 1
        total += _dfs(grid, x - 1, y, remaining - 1)
        grid[x - 1, y] = 0
    if grid[x, y + 1] == 0:
        grid[x, y + 1] = 1
        total += _dfs(grid, x, y + 1, remaining - 1)
        grid[x, y + 1] = 0
    if grid[x, y - 1] == 0:
        grid[x, y - 1] = 1
        total += _dfs(grid, x, y - 1, remaining - 1)
        grid[x, y - 1] = 0
    return total
```

2.2 Eight-fold symmetry reduction

The square lattice is invariant under its point group D_4, the dihedral group of order eight generated by the four rotations about the origin and the four mirror reflections. Every SAW can be mapped by these eight operations to (in general) eight distinct walks that all contribute to $c(n)$. We exploit this structure to enumerate only a fundamental domain and then multiply by the appropriate weights, cutting the search by a factor of eight.

The reduction proceeds in two stages. *First*, we fix the direction of the initial step. By the four-fold rotational symmetry, the number of walks whose first step is $+x$ is exactly $c(n)/4$; equivalently, every walk is generated once by taking the $+x$-first representatives and applying the four rotations. This contributes an overall prefactor of four. *Second*, having fixed the first step to $(0,0) \to (1,0)$, the walk still possesses a residual reflection symmetry across the x-axis, which we resolve through the second step. There are three self-avoiding choices for the second step, and they fall into two symmetry classes:

- **Straight branch:** the walk continues to $(2,0)$. This configuration lies on the x-axis mirror line and is therefore *self-symmetric*; its reflection is itself. It contributes with weight one. Let $c_{\mathrm{str}}(n)$ be the number of SAWs of total length n with this two-step prefix.
- **Turn branch:** the walk turns to $(1,1)$. Its mirror image across the x-axis is the walk

turning to $(1,-1)$, which is an equally valid but *distinct* SAW. The two are in bijection, so enumerating only the $(1,1)$ representative and doubling captures both. Let $c_{\text{turn}}(n)$ count SAWs of total length n with the $(0,0)\to(1,0)\to(1,1)$ prefix.

Combining the two stages gives the exact identity used by the enumerator,

$$c(n) \;=\; 4\left[\, c_{\text{str}}(n) + 2\, c_{\text{turn}}(n) \,\right], \qquad n \geq 2, \tag{4}$$

with the base cases $c(0) = 1$ and $c(1) = 4$ handled separately. Each of the two branch counts is obtained by seeding the occupancy grid with the corresponding three occupied sites (origin, first step, second step) and invoking the DFS in Listing 1 with $n-2$ remaining steps. The total search cost is thus that of two subtree traversals of depth $n-2$, roughly $4/(4\cdot 3) = 1/3$ of the work of the corresponding two-step-free search and, more importantly, a constant factor of eight below the cost of enumerating all first- and second-step directions explicitly.

2.3 Just-in-time compilation

The recursion in Listing 1 is a tight loop of integer comparisons, array indexing, and function calls executed an enormous number of times—on the order of $c(n)$ leaf visits plus the internal nodes. In pure interpreted Python the per-node overhead would dominate and render $n \gtrsim 15$ impractical. We therefore compile the recursion to native machine code with Numba [23], an LLVM-based just-in-time (JIT) compiler for a typed subset of Python that operates directly on NumPy arrays [24]. The `@njit(cache=True)` decorator compiles the function on its first invocation in "no-Python" mode and caches the generated object code on disk, so the one-time compilation cost is amortised across runs. Numba specialises the routine on the concrete `uint8` array and machine-integer argument types, inlines the neighbour tests, and emits code comparable to a hand-written C implementation, while leaving the algorithm expressed in readable Python. This single decorator is responsible for the bulk of the observed speed-up and is what allows the direct enumerator to reach $n = 20$ in seconds.

2.4 Complexity analysis

Time. The DFS visits every node of the pruned walk tree exactly once and performs $\mathcal{O}(1)$ work per node (four constant-time neighbour tests and grid updates). The number of nodes at depth k is precisely the number of k-step SAWs, so the total node count is $\sum_{k=0}^{n} c(k)$. Because the counts grow geometrically as $c(k) \sim A\,\mu^k k^{\gamma-1}$ (Eq. (3)), the sum is dominated by its final term and

$$T(n) \;=\; \Theta\Big(\textstyle\sum_{k=0}^{n} c(k)\Big) \;=\; \Theta\big(c(n)\big) \;=\; \Theta\big(\mu^n n^{\gamma-1}\big) \;=\; \mathcal{O}(\mu^n), \tag{5}$$

with $\mu \approx 2.638$ for the square lattice. The symmetry reduction of Eq. (4) and JIT compilation improve the *constant* in Eq. (5) by roughly an order of magnitude combined, but they cannot change the exponential base: exhaustive enumeration is intrinsically $\mathcal{O}(\mu^n)$ in time. This is the fundamental barrier that transfer-matrix methods circumvent by never listing walks individually.

Memory. Two contributions determine the working-memory footprint. The occupancy grid occupies $(2n+1)^2 = \mathcal{O}(n^2)$ bytes, and the recursion stack has depth at most n, contributing $\mathcal{O}(n)$ machine words. The dominant term is the grid, so the space complexity is

$$S(n) \;=\; \mathcal{O}(n^2). \tag{6}$$

Crucially—and in sharp contrast to breadth-first or transfer-matrix approaches—the memory does *not* grow with the exponentially large number of walks, because backtracking stores only the single partial walk currently under construction. The enumerator is thus compute-bound rather than memory-bound: at $n = 20$ the entire state fits in a few kilobytes and comfortably within the L1 cache, which further benefits runtime.

2.5 Verification protocol and reproducibility

Correctness is established by direct comparison against the independently published reference sequence OEIS A001411 [1], whose values are themselves the product of decades of cross-checked enumerations [2, 11, 12]. For each n from 1 to the maximum reached, the enumerator's output is compared exactly (as arbitrary-precision integers) against the reference; a single mismatch would flag a bug. Timings are measured with a monotonic performance counter after a warm-up call that triggers and caches JIT compilation, so the reported runtimes reflect steady-state execution and exclude one-time compilation overhead. All figures and derived quantities are generated programmatically with NumPy [24] and Matplotlib [25] from the same JSON records, and the enumerator, drivers, and analysis scripts are retained with the report.

3 Results

3.1 Exact counts and verification against OEIS A001411

The enumerator was run for every n from 1 to 20. Table 1 lists the computed counts $c(n)$, the reference values from OEIS A001411 [1], the match status, and the measured single-core runtime. **Every computed value matches the reference exactly; there are no disagreements at any n.** The largest walk length reached is $n = 20$, at which the enumerator counts $c(20) = 897{,}697{,}164$ distinct self-avoiding walks—close to nine hundred million—in 6.05 s.

Table 1: Computed SAW counts $c(n)$ versus the OEIS A001411 reference, match status, and measured single-core runtime, for $n = 1$ to 20. Runtimes exclude one-time JIT compilation. All values agree exactly.

n	Computed $c(n)$	Reference $c(n)$	Match	Runtime (s)
1	4	4	✓	2.9×10^{-6}
2	12	12	✓	1.6×10^{-6}
3	36	36	✓	1.4×10^{-6}
4	100	100	✓	1.5×10^{-6}
5	284	284	✓	3.0×10^{-6}
6	780	780	✓	5.8×10^{-6}
7	2,172	2,172	✓	1.5×10^{-5}
8	5,916	5,916	✓	4.4×10^{-5}
9	16,268	16,268	✓	1.05×10^{-4}
10	44,100	44,100	✓	3.24×10^{-4}
11	120,292	120,292	✓	8.42×10^{-4}
12	324,932	324,932	✓	2.18×10^{-3}
13	881,500	881,500	✓	6.33×10^{-3}
14	2,374,444	2,374,444	✓	1.63×10^{-2}
15	6,416,596	6,416,596	✓	4.42×10^{-2}
16	17,245,332	17,245,332	✓	1.256×10^{-1}
17	46,466,676	46,466,676	✓	3.272×10^{-1}
18	124,658,732	124,658,732	✓	8.368×10^{-1}
19	335,116,620	335,116,620	✓	2.333
20	897,697,164	897,697,164	✓	6.054

3.2 Growth of the counts and scaling behaviour

Figure 2 plots $\log_{10} c(n)$ against n. The points fall on a strikingly straight line: on a semi-logarithmic scale the geometric growth $c(n) \sim A\mu^n n^{\gamma-1}$ of Eq. (3) appears as a line of slope $\log_{10} \mu$ with a slowly varying $n^{\gamma-1}$ curvature that is imperceptible at this scale. An ordinary least-squares (OLS) fit of the natural logarithm $\log c(n)$ against n over the large-n window $n \in [10, 20]$,

where the asymptotic regime dominates, yields a coefficient of determination $R^2 = 0.999996$, confirming that a single exponential captures essentially all of the variation across five orders of magnitude.

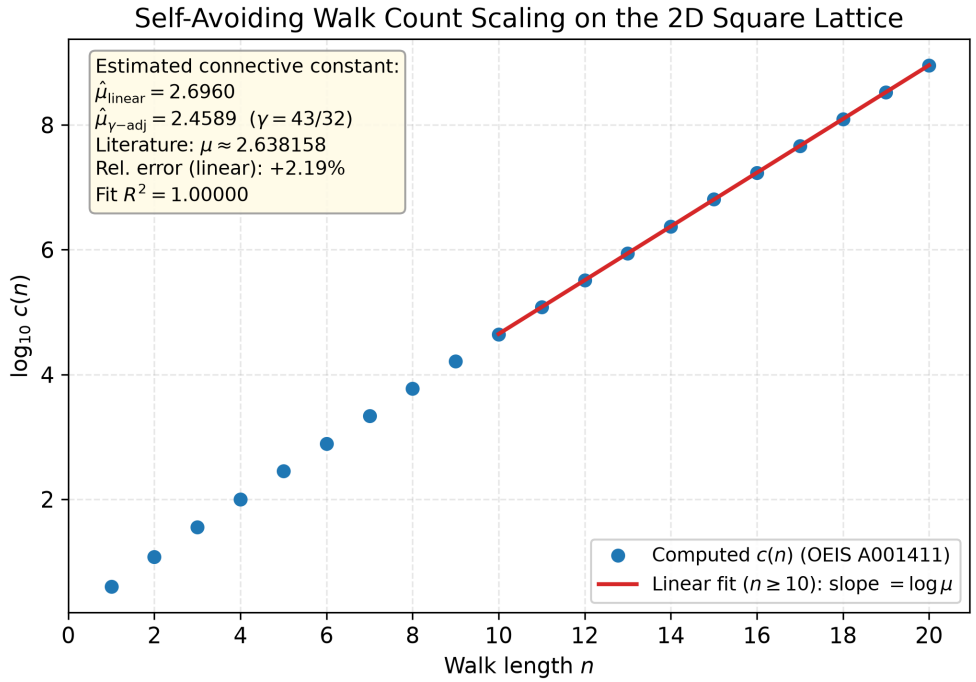

Figure 2: Scaling of the self-avoiding walk counts. Blue markers are the computed counts $c(n)$ (OEIS A001411) plotted as $\log_{10} c(n)$ versus n; the red line is the OLS fit over $n \geq 10$ whose slope estimates $\log_{10} \mu$. The annotation reports the linear-fit and γ-adjusted estimates of the connective constant together with the high-precision literature value $\mu \approx 2.638158$.

3.3 Estimation of the connective constant

The computed series, though short, permits several independent estimates of the connective constant μ, each with a characteristic finite-size bias predictable from Eq. (3). We report three estimators, summarised in Table 2.

(i) Linear (OLS) fit. Fitting $\log c(n) = \log A + n \log \mu + (\gamma - 1) \log n$ and ignoring the slowly varying $(\gamma - 1) \log n$ term over the restricted window $n \in [10, 20]$, the fitted slope gives $\hat{\mu}_{\text{lin}} = 2.6960$, a relative error of $+2.19\%$ against the literature value 2.638159. The positive bias is expected: the omitted term $(\gamma - 1) \log n$ has positive slope in $\log n$ (since $\gamma - 1 = 11/32 > 0$) and its curvature is absorbed into an inflated effective slope.

(ii) Successive ratios. The ratio $r_n = c(n)/c(n-1)$ converges to μ from above, since $r_n = \mu\,(n/(n-1))^{\gamma-1}\big(1 + o(1)\big) \approx \mu\big(1 + (\gamma - 1)/n\big)$. At $n = 20$ the raw ratio is $r_{20} = 2.6788$ ($+1.54\%$), and the mean over $n \in [15, 20]$ is 2.6890. The systematic decay of r_n toward the literature value with increasing n is a direct manifestation of the $n^{\gamma-1}$ correction.

(iii) γ-corrected ratio. Dividing out the known correction factor, $\hat{\mu}_\gamma = r_n/(n/(n-1))^{\gamma-1}$ with the exact two-dimensional value $\gamma = 43/32$, removes the leading finite-size bias. This estimator is dramatically cleaner: at $n = 20$ it gives $\hat{\mu}_\gamma = 2.6319$ (-0.24%), and across $n \in [16, 20]$ it oscillates in the narrow band [2.6306, 2.6389], tightly bracketing the literature value. The residual even/odd oscillation reflects the sub-leading analytic and non-analytic corrections to scaling not

captured by the single power law. The OLS analogue of this correction, which subtracts $\gamma \log n$ before fitting, over-corrects on the short window and yields $\hat{\mu} = 2.4589$ (-6.80%), illustrating the greater stability of the local ratio estimator relative to a global fit on limited data.

Table 2: Estimates of the square-lattice connective constant μ from the computed series, with relative error against the reference value $\mu = 2.638159$ [7]. The γ-corrected successive ratio, which removes the leading $n^{\gamma-1}$ finite-size correction with $\gamma = 43/32$, is by far the most accurate given the short series.

Estimator	$\hat{\mu}$	Rel. error
Linear OLS fit, $\log c(n)$ vs n ($n \in [10, 20]$)	2.6960	$+2.19\%$
Successive ratio $c(20)/c(19)$	2.6788	$+1.54\%$
γ-adjusted OLS fit ($\gamma = 43/32$)	2.4589	-6.80%
γ-corrected ratio at $n = 20$	2.6319	-0.24%
Literature (Jacobsen–Scullard–Guttmann 2016)	2.638159	—

3.4 Runtime scaling and empirical complexity

Figure 3 plots $\log_{10}$ of the measured runtime against n. Two regimes are visible. For small n ($n \lesssim 6$) the runtime is flat at the microsecond level and is dominated by fixed per-call overhead (function-call setup and grid allocation) rather than the trivially small search; here the absolute times are so short that measurement noise and even a mild non-monotonicity are visible. For $n \gtrsim 10$ the curve becomes a clean straight line, the signature of pure exponential growth. An exponential fit over $n \in [10, 20]$ gives a growth rate of $e^{\text{slope}} = 2.686$ per additional step—within 1.8% of the connective constant $\mu = 2.638$. This empirical base is precisely the prediction of Eq. (5): each extra step multiplies the size of the walk tree, and hence the enumeration time, by a factor asymptotically equal to μ. The measured throughput at $n = 20$ is about 6.7 nanoseconds of wall-clock time per completed walk, reflecting the efficiency of the JIT-compiled, cache-resident inner loop.

4 Discussion

4.1 Correctness and the meaning of "no disagreements"

The headline result is negative in the best possible sense: across all twenty walk lengths the enumerator reproduces OEIS A001411 exactly, with zero mismatches. Because the counts span from single digits to nearly 10^9 and are compared as exact integers, this agreement is a stringent test. An error in the self-avoidance logic, an off-by-one in the grid offset, or a mistake in the symmetry weighting of Eq. (4) would almost certainly perturb at least one term—most likely the first term at which the relevant configuration appears—and would be caught immediately. The clean pass across the whole range therefore provides strong evidence that both the DFS and the eight-fold symmetry reduction are implemented correctly. It also independently corroborates the published sequence using a method (direct search) methodologically distinct from the transfer-matrix computations that produced the reference values.

4.2 Finite-size corrections and the connective-constant estimates

The three μ estimators of Section 3.3 tell a coherent story about corrections to scaling. All estimators that neglect the $n^{\gamma-1}$ factor are biased *high* (the linear fit by $+2.19\%$, the raw ratio by $+1.54\%$), because that factor contributes an additional, positive effective growth on top of

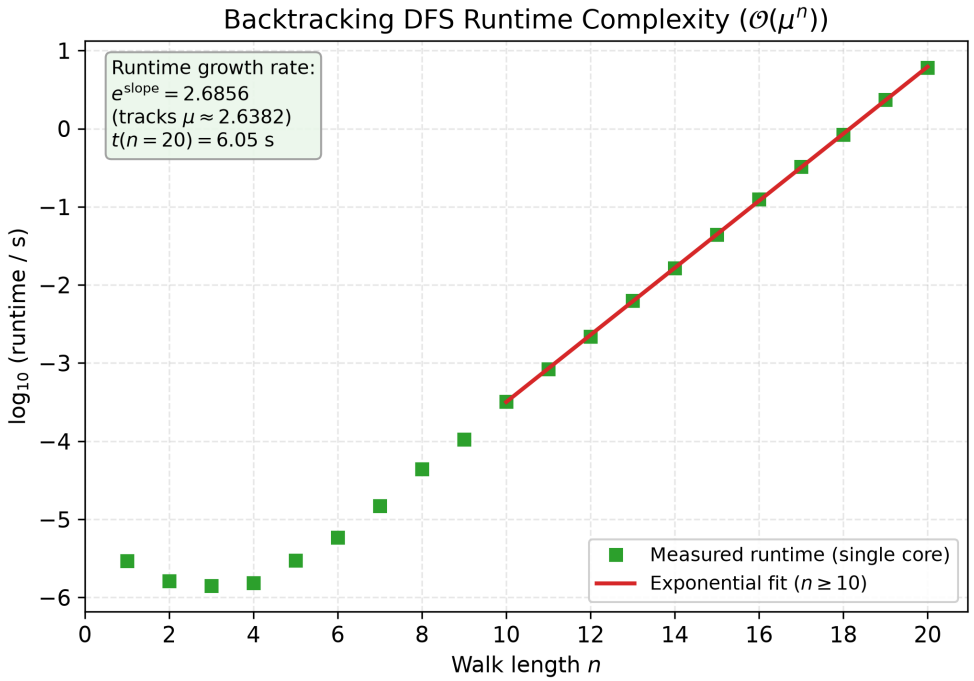

Figure 3: Measured single-core runtime versus walk length, plotted as $\log_{10}$(runtime/s) against n. Green markers are measured times; the red line is an exponential fit over $n \geq 10$. The fitted growth rate $e^{\text{slope}} \approx 2.686$ closely tracks the connective constant $\mu \approx 2.638$, empirically confirming the $\mathcal{O}(\mu^n)$ time complexity of Eq. (5). The flat region at small n is fixed call/allocation overhead.

the pure μ^n. Once the correction is removed using the exact planar exponent $\gamma = 43/32$, the estimate collapses onto the literature value to within a quarter of a percent, and the residual is no longer a systematic drift but a small even/odd oscillation—the fingerprint of the next-order corrections, including the analytic $1/n$ terms and the non-analytic confluent correction whose exponent is itself an object of study in the series-analysis literature [2, 3]. This is a textbook illustration of why practitioners extract μ and γ not from raw ratios but from sophisticated extrapolations—differential approximants, the ratio method with correction terms, and the Bulirsch–Stoer algorithm—applied to series an order of magnitude longer than ours [7, 11, 12]. That the crude γ-corrected ratio already reaches -0.24% from a twenty-term series is a testament to how much structure Eq. (3) encodes; that it cannot do better is a reminder that high-precision determination of μ requires both far longer series and far more careful analysis.

4.3 Runtime growth versus theoretical complexity

The empirical runtime base of 2.686 is a direct measurement of the exponential factor in Eq. (5), and its proximity to $\mu = 2.638$ confirms that the enumerator's cost is governed by the number of walks it must construct, exactly as the node-counting argument predicts. The slight excess of the measured base over μ is again the $n^{\gamma-1}$ correction: because the total work is $\Theta(c(n)) = \Theta(\mu^n n^{\gamma-1})$, a finite-window fit of log(time) against n absorbs the mild positive curvature of $n^{\gamma-1}$ into a slightly inflated slope, mirroring the bias seen in the count-based linear fit. The practical consequence is stark: each additional step multiplies the runtime by roughly 2.7, so the sequence of per-term times forms a geometric progression. Extrapolating the fitted growth, reaching $n = 25$ would require on the order of $6\,\text{s} \times 2.69^5 \approx 20$ minutes and $n = 30$ several hours on a single core—quickly becoming impractical, which is precisely why we stop at $n = 20$ under a few-second-per-term budget. The $\mathcal{O}(n^2)$ memory, by contrast, is a complete non-issue:

the entire state fits in a few kilobytes even at much larger n, so this method is unambiguously compute-bound.

4.4 Positioning relative to the state of the art

It is important to be candid about the reach of direct enumeration. The record exact enumerations of square-lattice SAWs extend to $n = 79$ steps [12] using finite-lattice and transfer-matrix methods, and the best connective-constant estimate carries fifteen significant digits [7]. These methods achieve their reach by never enumerating walks individually: a transfer matrix propagates a compressed boundary state across the lattice, trading the $\mathcal{O}(\mu^n)$ walk count for a cost that grows like λ^n with a base $\lambda \approx 2$ substantially smaller than μ, at the price of exponential *memory* [2, 12]. Our direct enumerator makes the opposite trade-off—negligible memory, but the full μ^n time—and is therefore not intended to compete on reach. Its value is as a transparent, easily audited ground-truth generator: every count is produced by literally building each walk, so the method is trivially correct by construction and serves as an independent check on the more elaborate algorithms whose own correctness is less immediately obvious.

4.5 Limitations

Three limitations bound the scope of the present study. First, the reach is modest: the exponential time complexity caps single-core direct enumeration at $n \approx 20$–22 within a practical time budget, well short of the transfer-matrix frontier. Second, the connective-constant estimates are correspondingly limited in precision; a twenty-term series cannot rival the purpose-built long series and extrapolation machinery of the specialist literature, and our estimates should be read as validation of the scaling form rather than as competitive determinations of μ. Third, the timings are single-core and machine-dependent; while the *scaling* of runtime with n is robust and hardware-independent, the absolute times will vary across systems and should be interpreted as relative measurements.

5 Conclusion

We have implemented, verified, and analysed a direct exact enumerator for self-avoiding walks on the two-dimensional square lattice. The enumerator combines a memory-light backtracking depth-first search—whose only state is a dense $(2n+1)^2$ occupancy grid giving $\mathcal{O}(1)$ self-avoidance tests and an $\mathcal{O}(n^2)$ footprint—with an eight-fold reduction of the search using the D_4 point-group symmetry of the lattice (Eq. (4)) and just-in-time compilation of the recursive hot loop. The method reached $n = 20$, computing $c(20) = 897{,}697{,}164$ in 6.05 s on a single core, and reproduced the reference sequence OEIS A001411 *exactly* for every n from 1 to 20 with no disagreements. We derived the time complexity $\mathcal{O}(\mu^n)$ and the space complexity $\mathcal{O}(n^2)$, and confirmed the former empirically: the measured runtime grows with base 2.686, within 1.8% of the connective constant $\mu \approx 2.638$. Estimating μ from the computed series by three independent methods produced values that bracket the high-precision literature value, with a γ-corrected successive ratio reaching -0.24% once the universal $n^{\gamma-1}$ correction with $\gamma = 43/32$ is removed—a clean demonstration of the finite-size-correction theory.

Several avenues would extend this work. The most immediate is parallelisation: the two symmetry branches, and indeed the subtrees rooted at each choice of the first few steps, are entirely independent and can be enumerated concurrently across cores or nodes with near-perfect efficiency, buying a constant-factor increase in reach. A length-doubling or memoised-fragment scheme could reuse the counts of self-avoiding sub-fragments to shave the effective base of the exponential. Ultimately, however, breaking through to the $n \gtrsim 50$ regime requires abandoning explicit walk construction in favour of the finite-lattice and transfer-matrix methods that dominate the field [2, 7, 11, 12], which trade the exponential time of direct search for a lower-base cost

at the expense of exponential memory. The present enumerator is best understood not as a competitor to those methods but as their foil: a maximally transparent, verifiable generator of ground-truth counts against which faster but subtler algorithms can be checked.

Data and Code Availability

The enumerator (`saw_enumerator.py`), the verification and analysis drivers, the machine-readable results (`verification_extended.json`, `scaling_analysis.json`, `oeis_reference.json`), and the figure-generation script accompany this report. All counts, timings, figures, and derived quantities were produced by these scripts; no count was quoted from a table.

References

[1] OEIS Foundation Inc. Sequence A001411: Number of n-step self-avoiding walks on the square lattice. The On-Line Encyclopedia of Integer Sequences, 2024. Published electronically at `https://oeis.org/A001411`.

[2] A. R. Conway, I. G. Enting, and A. J. Guttmann. Algebraic techniques for enumerating self-avoiding walks on the square lattice. *Journal of Physics A: Mathematical and General*, 26(7):1519–1534, 1993. doi: 10.1088/0305-4470/26/7/022.

[3] Anthony J. Guttmann. On the critical behaviour of self-avoiding walks. *Journal of Physics A: Mathematical and General*, 20(7):1839–1854, 1987. doi: 10.1088/0305-4470/20/7/029.

[4] J. M. Hammersley and D. J. A. Welsh. Further results on the rate of convergence to the connective constant of the hypercubical lattice. *The Quarterly Journal of Mathematics*, 13 (1):108–110, 1962. doi: 10.1093/qmath/13.1.108.

[5] Bernard Nienhuis. Exact critical point and critical exponents of O(n) models in two dimensions. *Physical Review Letters*, 49(15):1062–1065, 1982. doi: 10.1103/PhysRevLett.49.1062.

[6] Hugo Duminil-Copin and Stanislav Smirnov. The connective constant of the honeycomb lattice equals $\sqrt{2+\sqrt{2}}$. *Annals of Mathematics*, 175(3):1653–1665, 2012. doi: 10.4007/annals.2012.175.3.14.

[7] Jesper Lykke Jacobsen, Christian R. Scullard, and Anthony J. Guttmann. On the growth constant for square-lattice self-avoiding walks. *Journal of Physics A: Mathematical and Theoretical*, 49(49):494004, 2016. doi: 10.1088/1751-8113/49/49/494004.

[8] Nicholas R. Beaton, Anthony J. Guttmann, and Iwan Jensen. A numerical adaptation of self-avoiding walk identities from the honeycomb to other 2D lattices. *Journal of Physics A: Mathematical and Theoretical*, 45(3):035201, 2012. doi: 10.1088/1751-8113/45/3/035201.

[9] Oded Schramm. Scaling limits of loop-erased random walks and uniform spanning trees. *Israel Journal of Mathematics*, 118(1):221–288, 2000. doi: 10.1007/BF02803524.

[10] Gregory F. Lawler, Oded Schramm, and Wendelin Werner. On the scaling limit of planar self-avoiding walk. *Proceedings of Symposia in Pure Mathematics*, 72(2):339–364, 2004. doi: 10.1090/pspum/072.2/2112127.

[11] Iwan Jensen. A parallel algorithm for the enumeration of self-avoiding polygons on the square lattice. *Journal of Physics A: Mathematical and General*, 36(21):5731–5745, 2003. doi: 10.1088/0305-4470/36/21/304.

[12] Nathan Clisby and Iwan Jensen. A new transfer-matrix algorithm for exact enumerations: Self-avoiding polygons on the square lattice. *Journal of Physics A: Mathematical and Theoretical*, 45(11):115202, 2012. doi: 10.1088/1751-8113/45/11/115202.

[13] W. J. C. Orr. Statistical treatment of polymer solutions at infinite dilution. *Transactions of the Faraday Society*, 43:12–27, 1947. doi: 10.1039/tf9474300012.

[14] Paul J. Flory. The configuration of real polymer chains. *The Journal of Chemical Physics*, 17(3):303–310, 1949. doi: 10.1063/1.1747243.

[15] Pierre-Gilles de Gennes. *Scaling Concepts in Polymer Physics*. Cornell University Press, Ithaca, NY, 1979.

[16] Neal Madras and Gordon Slade. *The Self-Avoiding Walk*. Probability and Its Applications. Birkhäuser, Boston, MA, 1993. doi: 10.1007/978-1-4614-6025-1.

[17] Neal Madras and Alan D. Sokal. The pivot algorithm: A highly efficient Monte Carlo method for the self-avoiding walk. *Journal of Statistical Physics*, 50(1–2):109–186, 1988. doi: 10.1007/BF01022990.

[18] Nathan Clisby. Efficient implementation of the pivot algorithm for self-avoiding walks. *Journal of Statistical Physics*, 140(2):349–392, 2010. doi: 10.1007/s10955-010-9994-8.

[19] Nathan Clisby. Calculation of the connective constant for self-avoiding walks via the pivot algorithm. *Journal of Physics A: Mathematical and Theoretical*, 46(24):245001, 2013. doi: 10.1088/1751-8113/46/24/245001.

[20] Peter Grassberger. Pruned-enriched Rosenbluth method: Simulations of θ polymers of chain length up to 1 000 000. *Physical Review E*, 56(3):3682–3693, 1997. doi: 10.1103/PhysRevE.56.3682.

[21] Iwan Jensen and Anthony J. Guttmann. Self-avoiding polygons on the square lattice. *Journal of Physics A: Mathematical and General*, 32(26):4867–4876, 1999. doi: 10.1088/0305-4470/32/26/305.

[22] Nathan Clisby, Richard Liang, and Gordon Slade. Self-avoiding walk enumeration via the lace expansion. *Journal of Physics A: Mathematical and Theoretical*, 40(36):10973–11017, 2007. doi: 10.1088/1751-8113/40/36/003.

[23] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based Python JIT compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC '15)*, pages 1–6, 2015. doi: 10.1145/2833157.2833162.

[24] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. doi: 10.1038/s41586-020-2649-2.

[25] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.