Verifying Quantum Error Correction Codes with SAT Solvers

Pengyu Liu
Carnegie Mellon University, USA

Mengdi Wu
Carnegie Mellon University, USA

Abstract

Quantum error correction is essential for executing quantum algorithms under realistic noise. However, verifying the correctness of quantum error correction code implementations remains challenging due to the exponential size of the possible error patterns. In this paper, we present a SAT-based approach to formally verify quantum error correction codes by encoding the verification problem as a SAT problem. We apply our method to analyze surface code implementations and successfully identify bugs in a recently published paper, where codes claimed to correct k errors actually fail to do so for larger distances. Our approach demonstrates that SAT solvers can efficiently find counterexamples (bugs) in quantum error correction implementations, though verifying correctness (proving no bugs exist) remains computationally challenging due to the inherent difficulty of UNSAT problems combined with XOR constraints.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Hardware → Quantum error correction and fault tolerance

Keywords and phrases SAT solver, quantum error correction, surface code, formal verification

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Quantum computing promises to solve certain problems exponentially faster than classical computers, with potential applications ranging from quantum chemistry [2], cryptography [16], and machine learning [18], to finance [15]. However, quantum systems are inherently fragile—quantum bits (qubits) are susceptible to errors from decoherence, environmental noise, and imperfect gate operations [12]. Unlike classical systems where errors mainly occur during data transmission or storage, quantum errors occur *continuously during computation itself*, which intertwines quantum algorithms with quantum error correction, making the correctness of a code not only a static property but also a dynamic one [7].

Quantum error correction (QEC) codes address this challenge by spreading logical information across multiple physical qubits, so that local errors can not affect the logical information easily. The *distance* d of a code determines its error-correcting capability: a distance-d code can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors. Verifying that a code implementation achieves its claimed distance is crucial for ensuring fault tolerance, but exhaustively testing all possible error combinations is computationally infeasible for practical code sizes.

There are previous works using SMT solvers to verify the correctness of quantum error correction codes, but their speed is not satisfactory, for example, it takes 70 hours to verify a distance-7 code [6].

1.1 Contributions

In this paper, we make the following contributions:

1. We formulate quantum error correction verification as a SAT problem, enabling the use of highly optimized SAT solvers.

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2. We develop efficient encodings for XOR constraints arising from detector definitions using Tseitin transformation with both chain and tree structures.
3. We apply our method to verify surface code implementations and discover bugs in a recently published Nature paper [5], where codes claimed to achieve certain distances actually fail.
4. We analyze the performance characteristics of our approach, identifying the computational challenges that make verification (UNSAT problems) significantly harder than bug-finding (SAT problems).

2 Background

2.1 Quantum Computing Basics

A *qubit* (quantum bit) is the fundamental unit of quantum information. Unlike a classical bit that exists in state 0 or 1, a qubit can exist in a *superposition* $\alpha|0\rangle + \beta|1\rangle$, where α and β are complex amplitudes satisfying $|\alpha|^2 + |\beta|^2 = 1$ [14]. When measured, the qubit collapses to $|0\rangle$ with probability $|\alpha|^2$ or $|1\rangle$ with probability $|\beta|^2$.

2.2 The Surface Code

The surface code [9, 8] is one of the most promising quantum error correction codes due to: (1) local nearest-neighbor interactions compatible with many hardware platforms, (2) high error threshold ($\sim 1\%$), and (3) easy decoding. There are already many experimental demonstrations of the surface code, including superconducting qubits [1] and neutral atoms [5].

2.3 Stabilizer Formalism

The stabilizer formalism [11] enables error detection without measuring the encoded quantum state directly. A stabilizer code is defined by a set of commuting n-qubit Pauli operators $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$. Valid codewords $|\psi\rangle$ satisfy $S_i|\psi\rangle = |\psi\rangle$ for all $S_i \in \mathcal{S}$.

When an error E (a Pauli operator) occurs, the corrupted state $E|\psi\rangle$ may no longer be a $+1$ eigenstate of all stabilizers. The syndrome is determined by the commutation relations: measuring S_i yields $+1$ if $[E, S_i] = 0$ (commute), and -1 if $\{E, S_i\} = 0$ (anti-commute). Mathematically, if we define syndrome bit $s_i \in \{0, 1\}$ where $S_i E = (-1)^{s_i} E S_i$, then the syndrome vector $\mathbf{s} = (s_1, \ldots, s_m)$ can be used to determine the error.

A quantum code with parameters $[[n, k, d]]$ uses n physical qubits to encode k logical qubits with distance d, meaning any error affecting fewer than d qubits produces a non-trivial syndrome and can be detected.

2.4 Detectors and Decoders

A *detector* is a linear combination of measurement outcomes that is deterministic in the absence of errors. When errors occur, detectors may produce unexpected values, providing classical information about which errors likely occurred.

The *decoder* is a classical algorithm that uses detector information to infer the error pattern and apply corrections.

2.5 Detector Error Model (DEM)

We use Stim's Detector Error Model (DEM) format [10] to represent error mechanisms and their effects. A DEM file describes each error mechanism with its probability, affected detectors (D#), and affected logical observables (L#). Example: `error(0.027) D0 D1` triggers detectors D0 and D1 with probability 0.027; `error(0.101) D0 L0` triggers D0 and flips logical observable L0 with probability 0.101. Here, we only focus on the number of errors that error and will ignore the probability.

2.6 Zero-Detector Verification

Most quantum error correction codes are *linear codes*: if error patterns E_1 and E_2 each produce syndromes $\mathbf{s}_1$ and $\mathbf{s}_2$, then $E_1 \oplus E_2$ produces syndrome $\mathbf{s}_1 \oplus \mathbf{s}_2$. This linearity has an important consequence for code verification.

A *zero-detector logical error* is an error pattern that triggers no detectors (zero syndrome) but flips at least one logical observable. Such errors are undetectable and cause logical failures. Due to linearity, if E_1 and E_2 produce the same syndrome $\mathbf{s}_1 = \mathbf{s}_2$ but different logical outcomes, then $E_1 \oplus E_2$ triggers no detectors yet flips a logical observable—a zero-detector logical error.

The *code distance* d is defined as the minimum weight of any zero-detector logical error:

$$d = \min\{|E| : E \text{ triggers no detectors and flips a logical observable}\}$$

A code with distance d can reliably correct up to $t = \lfloor (d-1)/2 \rfloor$ errors. This is because any two correctable error patterns E_1 and E_2 with $|E_1|, |E_2| \le t$ must have distinct syndromes; otherwise $E_1 \oplus E_2$ would be a zero-detector logical error with weight at most $2t < d$, contradicting the definition of distance.

3 SAT Encoding Methodology

Given n error mechanisms, m detectors, and ℓ logical observables, we create boolean variable e_i for each error mechanism and add following constraints:

1. *Detector*: $\bigoplus_{i \in \text{affects}(D_j)} e_i = 0$ for each detector;
2. *Observable*: $\bigvee_k (\bigoplus_{i \in \text{affects}(L_k)} e_i = 1)$;
3. *Cardinality*: $\sum_i e_i \le k$.

If the SAT solver finds a solution, it is an undetectable logical error with $\le k$ errors.

3.1 XOR Encoding with Tseitin Transformation

XOR constraints must be converted to CNF using Tseitin transformation. For a base-b XOR gate $c = e_1 \oplus e_2 \oplus \cdots \oplus e_b$, we enumerate all 2^b input combinations and generate 2^{b-1} clauses enforcing $c = 1$ when an odd number of inputs are true. For the simplest case $b = 2$, the constraint $c = a \oplus b$ requires 4 clauses: $(\neg a \vee \neg b \vee \neg c) \wedge (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee c) \wedge (\neg a \vee b \vee c)$.

To encode $e_1 \oplus \cdots \oplus e_n = 0$, we recursively decompose it using base-b XOR gates as building blocks:

Chain Structure: Introduce auxiliary variables sequentially: $a_1 = e_1 \oplus \cdots \oplus e_b$, $a_2 = a_1 \oplus e_{b+1} \oplus \cdots \oplus e_{2b-1}$, etc. This produces a linear chain with depth $O(n/b)$.

Tree Structure: Reduce XORs in a balanced tree: first compute $a_i = e_{(i-1)b+1} \oplus \cdots \oplus e_{ib}$ for each group of b variables, then recursively combine a_i's using the same method. This achieves depth $O(\log_b n)$ for better unit propagation.

Higher base values reduce the number of auxiliary variables but increase clause complexity exponentially (2^{b-1} clauses per gate). We experiment with $b \in \{2, 3, 4\}$ to find the optimal trade-off.

3.2 Cardinality Constraints

To encode "at most k of n variables are true," the naive approach adds a clause for each $(k+1)$-subset, yielding $\binom{n}{k+1}$ clauses—exponential in k.

We use the *totalizer encoding* [3], which constructs a unary counting circuit via a binary tree. Each leaf represents an input variable e_i. Each internal node merges two sorted unary counters from its children: if the left child outputs $(l_1, \ldots, l_a)$ and the right outputs $(r_1, \ldots, r_b)$, the merged output $(o_1, \ldots, o_{a+b})$ satisfies $o_i = 1$ iff at least i inputs below are true. The merge operation uses clauses of the form $l_i \wedge r_j \Rightarrow o_{i+j}$. At the root, we enforce $o_{k+1} = 0$ to guarantee at most k variables are true. This encoding requires $O(n \log n)$ auxiliary variables and $O(nk)$ clauses, and provides strong unit propagation.

4 Evaluation

Our implementation uses Python with PySAT and CaDiCaL [4], a state-of-the-art CDCL solver. We use Stim [10] to generate detector error models from quantum circuits. Table 1 shows the problem scales for different code distances.

Table 1 Problem sizes for different code distances

Dist.	Errors	Detectors	CNF Vars

4.1 Bug Discovery in Nature Paper

We applied our method to surface code implementations from a Nature paper [5], where they claimed they implemented a variant of the surface code. Which can correct $\frac{d-3}{2}$ errors for distance d.

Table 2 shows our findings: **distances 11 and 13 fail to correct claimed errors**. The distance-11 code corrects only 3 errors (not 4), and distance-13 corrects only 4 (not 5).

Table 2 Verification Results: Claimed vs Actual Correctable Errors

Distance	Actual	Claimed
3	0	0
5	1	1
7	2	2
9	3	3
11	**3**	4
13	**4**	5

Our SAT solver found explicit counterexamples—error patterns triggering no detectors but flipping the logical observable. For distance-11, an 8-error pattern (vs. expected 10) demonstrates a "shortcut" through the code. These counterexamples provide valuable debugging information, identifying exactly which error mechanisms combine to defeat error correction. Pengyu:TODO: add counterexamples

4.2 Performance Analysis

Figure 1 and 2 compare SAT (bug-finding) vs UNSAT (verification) performance. Finding counterexamples is fast; proving correctness grows rapidly with problem size.

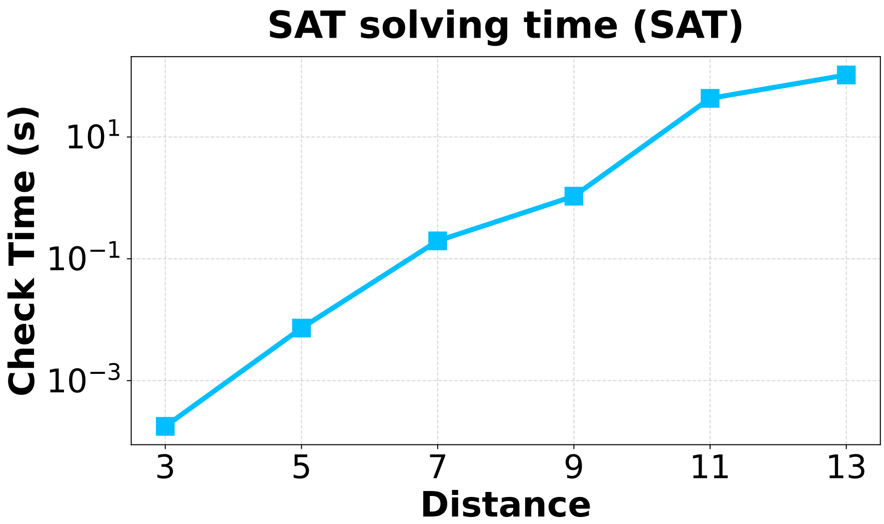

Figure 1 SAT performance: Finding bugs is fast

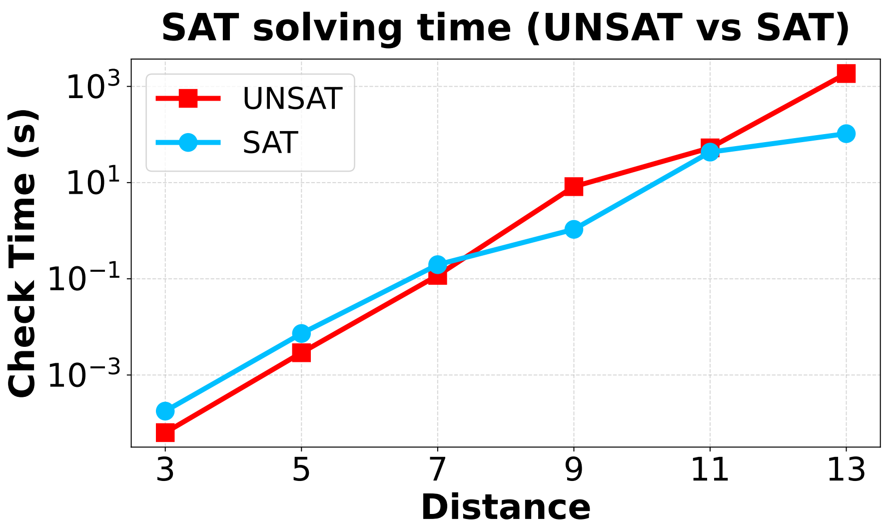

Figure 2 UNSAT performance: Verification is slow

Figure 3 compares XOR encoding strategies (chain vs tree, base-2 vs base-3). Tree-based encodings provide better propagation.

Table 3 shows timing results. Build time scales polynomially; SAT solve remains fast; UNSAT grows dramatically (distance-13 times out after 10 min).

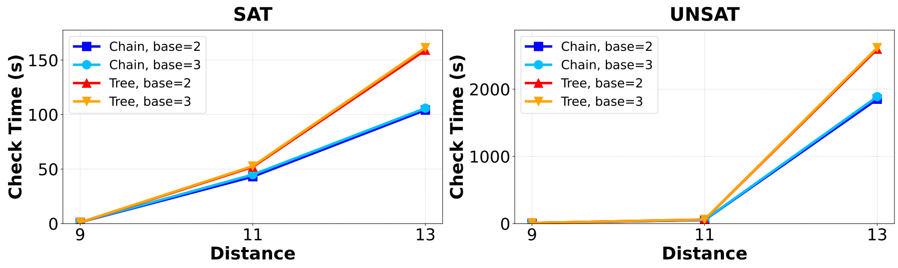

Figure 3 Comparison of XOR encoding strategies

Table 3 Verification timing (seconds)

Dist.	Err	Result	Build	Solve

4.3 Results using other solvers

4.3.1 Results using CryptoMiniSat

4.3.2 Results using Z3

4.3.3 Results using MAXSAT

5 Challenges and Limitations

We found that verification is significantly harder than bug-finding. We believe the following factors are the reasons:

UNSAT Problem Difficulty: Verification requires proving UNSAT—that no satisfying assignment exists. This is inherently harder than finding satisfying assignments because the solver must exhaustively rule out all possibilities. While SAT problems can often be solved quickly by finding a single witness, UNSAT proofs require exploring (and pruning) the entire search space.

XOR Constraints: SAT solvers are known to struggle with parity constraints [17]. Our XOR constraints, while encoded into CNF via Tseitin transformation, retain their underlying parity structure that causes difficulty for resolution-based proof systems. The inability of resolution to efficiently handle XOR is a fundamental limitation.

Cardinality Constraints: The "at most k errors" constraint resembles the pigeonhole principle, which is known to require exponentially long resolution proofs [13]. Combined with XOR constraints, this creates a particularly challenging problem structure.

Together, these factors make verification substantially harder than bug-finding, explaining the dramatic performance gap observed in our experiments.

6 Related Work

6.1 Quantum Error Correction Verification

6.2 SAT-Based Verification

Pengyu:TODO: add related work

7 Conclusion and Future Work

We presented a SAT-based approach to verifying quantum error correction codes, encoding the verification problem as boolean satisfiability with XOR constraints for detectors, cardinality constraints for error bounds, and disjunctive constraints for logical observables. Our method discovered bugs in a published Nature paper's surface code implementation, where distance-11 and distance-13 codes fail to achieve claimed error correction capability.

Our experiments reveal a fundamental challenge: SAT solvers efficiently find counterexamples in faulty implementations, but proving correctness (UNSAT) is significantly harder due to the combination of XOR constraints, cardinality constraints, and the need to exhaustively rule out all possibilities.

For future work, we propose a hybrid SAT + theorem prover approach. SAT solvers excel at bug-finding and search space pruning, while theorem provers (e.g., Lean) provide formal correctness guarantees. A hybrid approach could use SAT for rapid counterexample detection and pruning, then employ theorem provers to formally verify correctness.

References

1. Quantum error correction below the surface code threshold. *Nature*, 638(8052):920–926, 2025.
2. Ryan Babbush, Jarrod McClean, Dave Wecker, Alán Aspuru-Guzik, and Nathan Wiebe. Chemical basis of trotter-suzuki errors in quantum chemistry simulation. *Physical Review A*, 91(2):022311, 2015.
3. Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *International conference on principles and practice of constraint programming*, pages 108–122. Springer, 2003.
4. Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024. doi:10.1007/978-3-031-65627-9_7.
5. Dolev Bluvstein, Alexandra A Geim, Sophie H Li, Simon J Evered, J Pablo Bonilla Ataides, Gefen Baranes, Andi Gu, Tom Manovitz, Muqing Xu, Marcin Kalinowski, et al. A fault-tolerant neutral-atom architecture for universal quantum computation. *Nature*, pages 1–3, 2025.
6. Kean Chen, Yuhao Liu, Wang Fang, Jennifer Paykin, Xin-Chuan Wu, Albert Schmitz, Steve Zdancewic, and Gushu Li. Verifying fault-tolerance of quantum error correction codes. In *International Conference on Computer Aided Verification*, pages 3–27. Springer, 2025.
7. Nicolas Delfosse and Adam Paetznick. Spacetime codes of clifford circuits. *arXiv preprint arXiv:2304.05943*, 2023.
8. Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.
9. Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A—Atomic, Molecular, and Optical Physics*, 86(3):032324, 2012.
10. Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, 2021.
11. Daniel Gottesman. *Stabilizer codes and quantum error correction.* California Institute of Technology, 1997.
12. Daniel Gottesman. Surviving as a quantum computer in a classical world. *Textbook manuscript preprint*, 8(8.1):8–2, 2024.
13. Armin Haken. The intractability of resolution. *Theoretical computer science*, 39:297–308, 1985.

14 Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

15 Román Orús, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4:100028, 2019.

16 Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

17 Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM (JACM)*, 34(1):209–219, 1987.

18 Xin-Ding Zhang, Xiao-Ming Zhang, and Zheng-Yuan Xue. Quantum hyperparallel algorithm for matrix multiplication. *Scientific reports*, 6(1):24910, 2016.