Received: 12 May 2025 | Revised: 29 July 2025 | Accepted: 17 August 2025 | Published online: 10 October 2025

RESEARCH ARTICLE

Journal of Computational and Cognitive Engineering
2026, Vol. 5(1) 115–132
DOI: 10.47852/bonviewJCCE52026137

BON VIEW PUBLISHING

A Computational Complexity-Based Framework for Predicting the Difficulty of Sudoku Puzzles Using CSP-Based Symbolic Modeling and Constraint Metrics

Rajan Thangamani[1,*] **and Pallavi Regulagedda**[1]

[1] *School of Computer Science and Engineering, Presidency University, India*

Abstract: This paper provides a logic-based method for the categorization of the puzzle difficulty levels of Sudoku using deterministic measures of computational complexity. Shifting from heuristic, statistical, or machine-based paradigms, the method presents four symbolic measures: constraint density metric (CDM), logical inference depth (LID), guessing complexity (GC), and constraint graph tightness (CGT). These are derived from the Constraint Satisfaction Problem theory and implemented using R programming without the application of data-driven inference. The new method assesses puzzle complexity based on pre-solving properties yielding consistent outputs that align with theoretical NP-completeness measures. Difficulty levels are grounded on strict numerical criteria, e.g., CDM over 700, LID 14 or higher, GC over 6, and CGT approaching 30, indicative of increased structural complexity. The method is compared with rule-based, backtracking, and solver-based methods using benchmarks and shown to run with linear time and space complexity O(n) compared with traditional methods with time and space complexity $O(n^2)$ or exponential growth $O(2^n)$. The method presents a reproducible, scalable method for analytically assessing puzzles and provides a formal, non-heuristic method for evaluating complexity within combinatorial logic systems.

Keywords: Sudoku difficulty estimation, Constraint Satisfaction Problem (CSP), computational complexity, CSP-based symbolic modeling, constraint graph, R programming, NP-complete

1. Introduction

The computational classification of Sudoku puzzle hardness provides a fascinating point of overlap for discrete mathematics, artificial intelligence, and computer science theory. Sudoku, which may appear to be simple at first glance, is based on constraint satisfaction principles that reveal a rich landscape of structural and algorithmic complexity. A diverse series of methodologies has evolved over the last two decades to assess puzzle hardness, ranging from metaheuristic methods [1] and rule-based methods [2] to evolutionary computation [3] and zero-knowledge proof schemes [4]. These models are usually dependent on solution paths, heuristics of behavior, or massive dataset trawling for deriving estimates of hardness.

The classification of Sudoku within the realm of NP-complete challenges [5] serves to endorse intrinsic computation hardness residing within its solution, particularly when solved without third-party inference or solver bias. These subsequent attempts have tried decreasing the complexity of it using FPGA-based solvers [6], SAT-based encodings within Sudoku formulas [7], and AI-aided methods such as memristive neural networks [8] and photonic spike-based techniques [9]. Although efficient within performance terms, they are still based on the implementation of the solution process, which produces dependencies on machine-based guessing, backtracking, or probabilistic convergence.

Recent work has also discussed psychological modeling [10] and hardness indicators based on entropy [11], commonly resorting to human gameplay traces to qualify puzzles as "easy," "moderate," or "hard." Although pedagogically and entertainingly helpful, they lack the formal and theoretic clarity required for reproducibility and consistent benchmarking across unsolved instances. Another line of research, e.g., by McGuire et al. [12], has shown the computational insolvability of even minimal Sudoku puzzles, and less obviously, the exponential growth of solution space under partial constraints. Attempts at encoding hardness as a function of initial clues' number [13] or block symmetries [14] have illustrated that such surface metrics are insufficient to account for the combinatorial entanglements present at a deeper puzzle structure. A major research gap thus remains for creating a mechanism to predict puzzle hardness without puzzle solution, simulation runs, or statistical calibration.

In response to this challenge, this work takes a symbolic approach based on reasoning and theory of computation. By flipping the focus from "how long a puzzle takes to solve" to "how densely it is logically constrained," a new deterministic and theoretically sound complexity estimation pathway is proposed for Sudoku puzzle prediction of difficulty level without resorting to data-driven, statistical, or machine learning-based methods. The main goal of this work is to establish mathematical formulations for constraint satisfaction and computational complexity theory-based prediction of puzzle difficulty levels without using data-driven, statistical, or machine learning-based methods for that purpose.

To that end, the current work puts forth a new symbolic framework introducing four new logical measures, namely, constraint density metric

***Corresponding author:** Rajan Thangamani, School of Computer Science and Engineering, Presidency University, India. Email: rajan.20223cse0024@presidencyuniversity.in

(CDM), logical inference depth (LID), guessing complexity (GC), and constraint graph tightness (CGT), each of which encodes different computational properties of a puzzle's logical complexity. These measures are precisely defined and implemented using deterministic functions of R constraints. The proposed symbolic framework runs on the structural features of the puzzle before solution, without resorting to any trial-and-error, supervised learning, or statistical calibration stepwise requirement.

This symbolic-only method is tested on a sample of typical puzzles with diverse difficulty levels and compared with other methods for evaluation, including heuristic-based predictors and solver-included classifiers. Early metric scores derived on classified puzzle sets indicate a strong correlation of metric intensity levels with defined measures of puzzle hardness, e.g., CDM > 700, LID ≥ 14, GC ≥ 6, and CGT ≈ 30. Additionally, the proposed model has O(n) time and space complexity compared to conventional solving algorithms with $O(n^2)$ or $O(2^n)$ complexity growth, especially with backtracking or quantum-based formulation.

By refraining from statistical inference, behavioral modeling, and machine learning, the present study presents a unique reproducible system for the prediction of puzzle hardness based on symbolic measures from the Constraint Satisfaction Problem (CSP) theory and theory of computation. This technology connects the dots from the theoretically formulated NP-complete problem to the actual, deterministic evaluation of a puzzle and forms a foundation for further research toward symbolic logic systems and toward classifying a puzzle on the basis of its syntactic form.

This work has the following main contributions:

1) It creates a deterministic, symbolic complexity estimation model that does not require the use of solver-based heuristics, statistical inference, or training data.
2) It detects and quantifies four new symbolic measures, namely, CDM, LID, GC, and CGT, from the syntactic and structural features of Sudoku problems.
3) It indicates that the measures in question, tried on 1 million puzzles, all consistently suggest difficulty classes with no solution execution.
4) It has linear runtime and submegabyte memory usage for the symbolic system, making the system scalably deployable on unsolved puzzle spaces, learning environments, and symbolic-AI applications.

2. Literature Review

Many studies have explored Sudoku as a benchmark for combinatorial optimization and constraint satisfaction. Initial research attempts made use of mathematical models based on linear and integer programs. Recent demonstration of Sudoku constraint validation has been performed on neuromorphic hardware by Pignari et al. [15], with efficient evaluation in real time using spiking architectures, as well as based on binary integer linear programming by formulation of odd/even variants by Yu et al. [16]. Ates and Cavdur [17] promoted hybrid puzzle generation with hybrid mathematical programming and integration with heuristics. Béjar et al. [18] verified that Sudoku is NP-complete even with structured patterns of clue elimination, confirming its theoretical status even when not necessarily solved with heuristics. Bailey et al. [19] illustrated connections with Sudoku, Gerechte designs, and affine spaces, deepening its combinatorial and algebraic representation. Metaheuristics for Sudoku were analyzed by Notice et al. [20], who used algorithm selection for four solvers from a local-search base and showed that methods that are specific to instances are superior to general methods with regard to success and speed measures albeit typically with exponential behavior reflective of O(n!) search. Outside of symbolic representations, heuristic and metaheuristic methods have become even more prominent. Vamsi et al. [21] constructed an automated puzzle solver with CNN-based digit extraction and grid reconstruction for solving Sudoku. Kim and Eor [22] used multi-armed bandit selection of genetic algorithm for improving adaptivity of selection in Sudoku, and Pal et al. [23] presented a hybrid classical-quantum solver on the noisy intermediate scale of devices. Jana et al. [24] suggested using neighborhood-based mutation in genetic algorithms to solve Sudoku with faster convergence along with guaranteed performance regardless of puzzle hardness. Chatzinikolaou et al. [25] utilized irregular learning cellular automata to simulate adaptive human reasoning, and a hybrid evolutionary solver based on genetic and firefly mating methods [26] exhibited efficient convergence with respect to the level of difficulty. Jana et al. [27] resolved higher complexity within 3D Sudoku with dual solver configurations and elimination methods.

Tao et al. [28] demonstrated an implementation of a Sudoku solution using blended glial-cell SNN with faster and sparser inference without traditional rule-based reasoning. Explainability and solver transparency have also been investigated. Björnsson et al. [29] quantified mental efforts within solver responses to match human judgment with difficulty classification. Mulamba et al. [30] introduced a hybrid image-to-constraint solver integrating visual recognition with constraint reasoning for addressing human error and digit misclassification. Chang et al. [31] critically analyzed SATNet's capacity for visual puzzle-based logic inference and exposed its failure without symbolic grounding. Diah et al. [32] compared breadth-first search (BFS), depth-first search (DFS), and human solution strategies on 9 × 9 Sudoku problems and concluded that although DFS was generally faster overall, solution paths best mimicked human reasoning more frequently than BFS. Hardware applications have also contributed to the field. Chatzinikolaou et al. [8] introduced a memristor-based oscillator circuit with emphasis on parallel computation and low power for puzzle solution. Vidyashree et al. [33] described Sudoku as a graph-coloring problem and modeled it as an 81-node graph that could be solved using coloring heuristics. Cheewaprakobkit et al. [34] integrated an explainable rule-based reasoning component in addition to a recurrent relational network to make decision-making in solving Sudoku transparent and explainable, and Nishikawa and Toda [35] proposed an exact CSP-based solver that can generate guaranteed solvable clue sets with human reasoning.

Parallels from cryptography and secure computation have drawn on Sudoku as inspiration for their work. Masadeh et al. [36] presented a lightweight stream cipher based on the combinatorics of Sudoku. Robert et al. [37] constructed a physical ZKP protocol for the Suguru puzzle, and Sasaki et al. [38] introduced a card-based ZKP for Sudoku that protects solution confidentiality without revealing the solution. Serrano and Findler [39] illustrated a Sudoku solver for HipHop (interleaving JavaScript and Esterel) with functional-concurrent composition for implementing symbolic logic. A prominent area of research activity across recent years has centered on estimating difficulty. Wang [40] constructed a validated dataset with the help of the Cloud Sudoku app, collecting human-playing statistics such as solving time and success rate. Ananya and Singh [41] created an augmented reality Sudoku solver that leverages real-time digit recognition and in-line constraint-solving to assist users interactively. Valenzuela et al. [42] and Behrens et al. [43] reported the utility of solver behavior in terms of perceived difficulty, wherein flexible switching of tactics enhances human-solving intuition alignment. Maria Jeyaseeli et al. [44] investigated valid graph label numbers for Sudoku constraint graphs, providing insight into solution space structure and complexity. Hasanah et al. [45] contrasted ant colony optimization with the conventional backtracking for Sudoku, with ACO doing better for tougher puzzles but backtracking suited for simpler puzzles.

Other research has touched on solver optimization. Wang et al. [46] designed a genetic algorithm based on local search using matrix encoding and elite learning, and Pathak and Kumar [47] integrated uncertainty scores based on entropy with evolutionary methods for hard puzzle solving. Lloyd et al. [48] designed a benchmark suite for Japanese puzzle games for black-box comparison across solvers and games. Lin and Wu [49] designed a new algorithm for minimizing search space for minimum-clue Sudoku that outperformed McGuire's exhaustive checker using disjoint minimal unavoidable sets (DMUS). Kitsuwan et al. [50] proposed a dynamic load balancer with solver-specific prediction to minimize computation delay.

Although comprehensive, a principal research gap remains unfilled, with most of the difficulty estimation models depending on solver traces, machine learning, or game-playing measures, which create bias or require exhaustive computation. Few models directly operate on the structural description of a Sudoku puzzle to estimate its difficulty. Notice et al. [20] and Hasanah et al. [45] depended on solver behavior or heuristic adaptation, for instance, whereas Vamsi et al. [21] depended on convolutional pipelines that mask logical structure. That gap is filled in this work with a formulation of a deterministic symbolic system from four constraint-based measures: CDM, LID, GC, and CGT [51]. That formulation avoids statistical inference usage and solver-heuristic dependence, providing a reproducible, Mantere and Koljonen [52] symbolically founded system for complexity estimation with foundations on constraint satisfaction and computational theory [53].

Table 1 is a comprehensive comparative overview of significant schemes of computation adopted in a previous piece of research for estimating Sudoku puzzle difficulty and solving it [54]. It summarizes the time and space complexity of each algorithm or platform from their principal computation steps. We present each entry with reference, method used, core algorithmic operation, and corresponding computation cost, followed by major findings and limitations from their respective empirical evaluation or intrinsic characteristics. The tabular analysis demonstrates the richness of the set of methods, ranging from symbolic logic, evolutionary heuristics, and neural networks to their implementation with accelerators and cryptography analogs, and indicates that a majority of incumbent approaches (e.g., metaheuristics [7, 8], ML-based solvers [9, 12], and CSP encodings [25]) rely on solver-specific performance or tuning with a dependence on data. Our new symbolic approach, on the contrary, is solver path- and statistical inference-independent, thus forming a standing point for estimating the efficiency and scalability of the proposed symbolic metric-based scheme of the current work.

3. Theoretical Foundations

Analyzing the structural and logical intricacies of Sudoku challenges requires casting them within formal computational frameworks. Unlike approaches that concentrate on heuristics or machine learning-based methods, the present research builds exclusively on the concepts of CSPs and theory of computation, ensuring mathematical rigor and reproducibility.

3.1. Sudoku as a CSP

Sudoku puzzle instances are systematically modeled as cases of the CSP, which is a fundamental paradigm in artificial intelligence and theoretical computer science. This representation forms the foundation for studying the intrinsic complexity of Sudoku using formal definitions of variables, domains, and constraints without resorting to heuristics or solution enumeration.

Let the Sudoku grid be denoted as a finite structure $G = (V, D, C)$, where V is a set of variables, D is the domain, and C is a set of all binary constraints enforced over variable pairs. This mathematics makes possible the deterministic analysis of puzzles of Sudoku without appeal to empirical or heuristic approaches.

Definition 3.1 (Sudoku CSP Instance): Let $G = (V, D, C)$ be a CSP instance representing a standard 9 × 9 Sudoku puzzle, where:

1) The variable set is defined as shown in Equation (1):

$$V = \{v_{i,j} | 1 \le i, j \le 9\}. \tag{1}$$

Here, each variable $v_{i,j} \in V$ corresponds to the grid cell located at row i and column j. Thus, $|V| = 81$.

2) The domain function D assigns to each variable a set of possible values, as shown in Equation (2):

$$D(v_{i,j}) \subseteq \{1, 2, \ldots, 9\}. \tag{2}$$

Initially, for blank cells, $D(v_{i,j}) = \{1, \ldots, 9\}$; for pre-filled cells, $D(v_{i,j}) = \{k\}$ for some $k \in \{1, \ldots, 9\}$.

3) The constraint set $C \subseteq V \times V$ is a set of unordered variable pairs $(v_{i,j}, v_{p,q})$ such that the values of the two variables must be distinct under Sudoku rules.

The constraints can be classified formally into three types:

a. Row constraint as shown in Equation (3):

$$\forall i \in \{1, \ldots, 9\}, \forall j \ne q \in \{1, \ldots, 9\}, (v_{i,j}, v_{i,q}) \in C. \tag{3}$$

b. Column constraint as shown in Equation (4):

$$\forall j \in \{1, \ldots, 9\}, \forall j \ne p \in \{1, \ldots, 9\}, (v_{i,j}, v_{p,j}) \in C. \tag{4}$$

c. Box constraint: Let $B_{r,s}$ denote the 3 × 3 subgrid (box) with box indices $r, s \in \{0, 1, 2\}$, where as shown in Equation (5):

$$B_{r,s} = \{v_{i,j} | 3r + 1 \le i \le 3r + 3, 3s + 1 \le j \le 3s + 3\}. \tag{5}$$

As described in Equation (6), then:

$$\forall v_{a,b}, v_{c,d} \in B_{r,s}, (a, b) \ne (c, d) \Rightarrow (v_{a,b}, v_{c,d}) \in C. \tag{6}$$

Observation 3.1: Each variable participates in exactly 20 constraints: 8 within its row and column (excluding itself), and 4 to 6 within its box, depending on overlaps. Thus, as shown in Equation (7):

$$deg(v_{i,j}) = 20 \forall v_{i,j} \in V. \tag{7}$$

This uniform constraint degree leads to a well-connected constraint graph $G_C = (V, C)$, which is central to symbolic difficulty analysis.

Remark 3.1 (Domain Reduction in CSP Propagation): During constraint propagation (e.g., via arc consistency or singleton checks), the domain $D(v_{i,j})$ may be reduced incrementally as shown in Equation (8):

$$D(v_{i,j}) \leftarrow D(v_{i,j}) \backslash \{k\}, \ if \ \exists v_{p,q} \in Peers(v_{i,j}) s.t. \theta(v_{p,q}) = k, \tag{8}$$

where $\theta : V \to D$ is a partial assignment and $Peers(v_{i,j})$ is a set of variables constrained with $v_{i,j}$. This process enables symbolic metrics to detect how "constrained" a variable is before solving. Hence, domain size evolution under constraints becomes a central indicator of difficulty.

Table 1
Time and space complexity of related works

Sl. No.	Reference	Method/algorithm	Complexity		Key findings	Limitations
			Time	Space		
1	[15]	Neuromorphic validation of CSP solutions	88%–96% faster Est. $O(n^2)$	Data scales linearly Est. $O(n)$	Real-time validation using spiking architectures; high spike reduction	Hardware-specific performance and limited generalization
2	[17]	MILP with heuristics	$O(2^n)$	$O(n^2)$	Integer linear program formulation	Solver-dependent; efficient only with commercial solvers
3	[20]	Local-search meta-heuristics + algorithm selection	Solver-dependent worst-case: O(n!)	Near-linear Theoretical upper bound: $O(n^2)$	Instance-based selection improves success rate and speed on various puzzle instances	High variance; scalability issues on complex grids
4	[31]	SATNet integration	$O(n^2)$	$O(n^2)$	SAT-based symbolic layer in DNN	Fails without intermediate labels
5	[8]	Memristive oscillator circuits	Analog parallel	Circuit-dependent	Analog waveform solving dynamics	Only validated via SPICE simulation
6	[22]	GA + multi-armed bandit selection	~0.8 s per puzzle $O(g \cdot n!)$	$O(n^2)$	Adaptive arm selection speeds up convergence in varied difficulty puzzles	Bandit tuning adds hyperparameter complexity
7	[24]	Genetic algorithm + neighborhood mutation	~1–4 s per puzzle; worst-case exponential	$O(n^2)$	Enhanced GA avoids poor swaps, shows consistent performance across easy to hard puzzles	Limited to 9 × 9, no formal complexity proof
8	[26]	Hybrid genetic algorithm + firefly mating	~1–3 s per puzzle; theoretical worst case is exponential (O(n!))	$O(n^2)$	Combines GA and firefly mating to avoid local optima, consistent performance across easy, medium, and hard puzzles	No formal complexity analysis; tuned mainly for 9 × 9 nodes
9	[30]	Hybrid ML + constraint solving	$O(n^2) + O(c)$	$O(n^2)$	Digit recognition + CSP enforcement	Robust against user errors in visual input
10	[46]	Local search GA (LSGA)	$O(g \cdot n^2)$	$O(p \cdot n^2)$	Columnwise fitness + elite learning	Fast convergence; requires tuning for hardness scaling
11	[23]	Hybrid classical-quantum algorithm (NISQ-based)	~Minutes execution time Exponential theoretical	$O(n^2)$	First NISQ-based Sudoku solver; shows proof-of-concept quantum speedup potential	NISQ noise limits scalability; needs circuit optimization
12	[21]	Deep learning (CNN + OCR pipeline)	~1.4 s per image Theoretical: $O(n^2)$	$O(n^2)$	Combined CNN and OCR to identify digits and solve puzzles end-to-end	Accuracy depends on image clarity; no learning from solution logic
13	[39]	Functional + Esterel hybrid solver	O(n log n)	$O(n^2)$	Threaded symbolic logic simulation	Concurrent model complexity; not widely scalable
14	[40]	Human-labeled dataset construction	O(n)	$O(n^2)$	App-based user data with time/success metrics	Not predictive; depends on human gameplay
15	[45]	ACO vs. backtracking comparison	ACO: ~5–8 s, Backtracking: ~1–2 s; ACO worst-case < O(n!)	$O(n^2)$	ACO outperforms on hard puzzles; backtracking is faster on easy ones	Dependent on pheromone tuning; tested only on 9 × 9 grids
16	[44]	Graph-label count on Sudoku constraint graph	Computation time grows with labeling complexity; ≤ 5 s per instance	$O(n^2)$	Quantified number of valid labelings, linked to graph symmetry and constraint density	No solving algorithm; focuses on enumeration
17	[47]	Entropy-guided evolutionary search	$O(g \cdot n^2)$	$O(p \cdot n^2)$	Fitness-based entropy minimization	Dependent on good initial entropy estimates

Table 1
(*continued*)

Sl. No.	Reference	Method/algorithm	Complexity		Key findings	Limitations
			Time	Space		
18	[37]	Suguru ZKP protocol (physical)	O(n)	O(n)	ZKP using card-based encoding	Not solver-focused; used for secure proofing only
19	[38]	Sudoku ZKP via cards	O(n)	O(n)	Card-based verification without extractability	Not applicable for automated systems
20	[36]	Lightweight cipher inspired by Sudoku	O(n)	O(n)	Permutation-based secure encryption	Optimized only for low-resource devices
21	[27]	3D Sudoku dual solver	$O(n^3)$	$O(n^3)$	3D backtracking and elimination strategy	Memory intensive; complex for large 3D grids
22	[32]	BFS, DFS, and human-solving comparisons	BFS ~1.55 s, DFS ~0.55 s per puzzle.		DFS outperformed BFS in speed; DFS and human paths aligned better than BFS in most cases	Focused on medium-difficulty puzzles; human strategy patterns were anecdotal
23	[28]	Worst-case: O(n!)	$O(n^2)$ (grid size storage)	$O(n^2)$	Glial-cell design reduced spikes by 88% and computation by 92%; well-suited for real-time Sudoku solving	No formal complexity provided; platform-specific validation
24	[48]	Puzzle benchmarking framework	$O(n^2)$	$O(n^2)$	Black-box comparison across puzzle solvers	Not Sudoku-specific optimization
25	[35]	Exact CSP encoding for strategy-solvability	Solves many puzzles within 1 min; slows sharply with ≤ 20 clues	$O(n^2)$	Efficient generation of strategy-solvable grids, framework adapts to various solving techniques	Exponential time as clue count drops; focused on clue generation, not full solver
26	[43]	Tactical pattern flexibility analysis	< 0.1 s per puzzle on average	$O(n^2)$	Shows that adaptive tactic switching boosts solver performance across varying difficulty levels	Conducted on logic-only solvers, not full CSP pipelines
27	[42]	Algorithmic difficulty assessment via search	O(b^d)	O(b·d)	BFS and A* search behavior predict human challenge	Needs platform-specific calibration
28	[33]	Graph coloring heuristics	< 2 s per puzzle	$\sim O(n^2)$	Models Sudoku as graph coloring; solves full puzzles efficiently	Heuristic may fail on hardest instances; scaling beyond 9 × 9 untested
29	[41]	AR + real-time OCR + embedded constraint solver	2 s per puzzle (mobile app)	$O(n^2)$	Combines OCR and solver in AR; helpful visual feedback on mobile devices	Accuracy drops with poor lighting; platform-limited
30	[46]	Elite learning genetic algorithm	$O(g \cdot n^2)$	$O(p \cdot n^2)$	Combines matrix encoding and elite population memory	Needs balancing between exploration and exploitation

3.2. Relationship to NP-complete problems

Solving a Sudoku puzzle involves calculating a complete assignment for the variables under several interacting constraints. Although a check for the validity of a completed grid can be done with a polynomial number of steps, whether a valid completion exists for a partially completed grid is a much more difficult class of NP-complete problem.

Definition 3.2 (Sudoku Solving as a Decision Problem): Given a partial function $\theta_A : A \subset V \to \Sigma$, where $\Sigma = \{1, 2, \ldots, 9\}$, define the problem:

Given $G = (V, D, C)$ and θ_A, does there exist $\theta : V \to \Sigma$ such that as shown in Equation (9):

$$\forall (v_i, v_j) \in C, \theta(v_i) and \theta(v_k) = \theta_A(v_k), \forall v_k \in A? \quad (9)$$

This is a constraint extension problem: we wish to extend θ_A to a full assignment that satisfies all binary inequality constraints.

Lemma 3.2 (NP-Completeness of Sudoku Solving): Let Π_{Sudoku} denote the decision problem stated above. Then, as described in Equation (10):

$$\Pi_{Sudoku} \in NP - Complete. \quad (10)$$

Proof:

1) **Membership in NP:** Given a total assignment , we can check all constraints in C in time proportional to the number of edges in the constraint graph for standard 9 × 9 grids because each variable has ~20 peers, as described in Equation (11):

$$T_{verify} = O(|C|) = O(n^2). \tag{11}$$

Formally, verification checks in polynomial time as shown in Equation (12):

$$\forall (v_i, v_j) \in C, \theta(v_i) \neq \theta(v_i) \tag{12}$$

and Equation (13) will be:

$$\forall v_k \in A, \theta(v_k) = \theta A(v_k). \tag{13}$$

2) NP-Hardness: Yato and Seta [5] constructed a polynomial-time reduction as shown in Equation (14):

$$Latin\ Square\ Completion \leq_p \Pi_{Sudoku}. \tag{14}$$

The Latin square completion problem (LSCP) is defined over an $n \times n$ grid, where each row and column must contain each symbol exactly once, with some entries pre-filled. This can be seen as a simplified version of Sudoku without box constraints. Let $\Pi_{LS} \in NP - Complete$. The reduction transforms an LS instance V_L, D_L, C_L into a Sudoku instance $G = (V, D, C)$ by:

1) Preserving row and column constraints
2) Adding 3 × 3 box constraints via auxiliary variables and constraints.

This mapping preserves solution satisfiability: $\theta_L\ is\ valid \Leftrightarrow \theta$ satisfies all constraints in G.

Thus, as described in Equation (15),

$$\Pi_{LS} \leq_p \Pi_{Sudoku} \Rightarrow \Pi_{Sudoku} \in NP - Hard \tag{15}$$

Because $\Pi_{Sudoku} \in NP$ and is $NP - Hard \Rightarrow \Pi_{Sudoku} \in NP - Complete$

Remark 3.2 (Complexity Gap between Verification and Solving): Let us define two functions:

1) $VERIFY: \Sigma^{81} \rightarrow \{TRUE, FALSE\}$, where verification checks all C in $O(n^2)$.
2) $SOLVE: (V, D, C) \rightarrow \Sigma^{81} \cup \{\emptyset\}$, which must explore the search space $|\Sigma|^{|V_e|}$, where $V_e = V \backslash A$. Thus, as described in Equation (16):

$$T_{verify} = O(n^2), Tsolve = \Omega(9^{|V_e|}). \tag{16}$$

This exponential time lower bound for solving accounts for why backtracking strategies and heuristic solvers tend to fail on difficult puzzles and encourages symbolic difficulty estimation methods that run within O(n) or $O(n^2)$ time without exhaustive assignment enumeration.

3.3. Motivation for symbolic analysis

Solving a Sudoku puzzle, as depicted within the prior section, is NP-complete; however, checking a completed solution runs within polynomial time. This asymmetry indicates that a model that can predict difficulty without necessarily involving actual solving would be theoretically sound and computationally beneficial. Let $G = (V, D, C)$ represent a partially filled Sudoku instance. Define $V_e \subseteq V$ as a set of unassigned variables. Traditional solving strategies such as backtracking or heuristic search must explore a subset of the solution space $\sum^{|V_e|}$, where $\Sigma = \{1,\ldots,9\}$. Even with forward checking, constraint propagation, or intelligent variable ordering, the worst-case time remains exponential in $|V_e|$. To avoid this, symbolic estimation is our proposed solution: a method of estimating difficulty not from full assignment but from structural properties of G that can be derived in polynomial time.

Definition 3.3 (Symbolic Difficulty Estimation Framework): Let $\Theta: g \rightarrow R^k$ be a function mapping a Sudoku instance $G \in g$ to a tuple of symbolic metrics as shown in Equation (17):

$$\Theta(G) = (M_1(G), M_2(G), \ldots, M_k(G)). \tag{17}$$

Each metric $M_i(G)$ is a deterministic function computable in polynomial time and is designed to approximate a latent function $(G) \in R_+$, which represents puzzle difficulty. Here, Θ is called the symbolic transformation function, and D is the ground-truth solver-based or human-perceived difficulty.

Proposition 3.1 (Complexity Feasibility of Symbolic Analysis): Let $\Theta(G)$ be composed of metric functions that operate only on the domain D, variable set V, and constraint structure C. Then, $\Theta(G)$ can be computed in $O(n^2)$ time for any standard 9 × 9 Sudoku instance.

Proof: Each symbolic metric depends on one or more of the following operations:

1) Local constraint analysis: bounded to a set of ~20 peers per variable,
2) Row, column, and box scans: O(n),
3) Pairwise or groupwise connectivity in $G_C = (V, C) : O(n^2)$.

The total work across all 81 variables is still in $O(n^2)$, without recursive guessing, stochastic iterations, or enumeration of the domain. Because problem size is fixed (i.e., n = 9), the process is deterministic and polynomially bounded.

Lemma 3.3 (Separation of Solving vs. Symbolic Estimation Complexity): Let $T_{solve}(G) = \Omega(b^g)$, where b is the branching factor and g is the number of decision points. Let $T_{symbolic}(G) = O(n^2)$. Then, as shown in Equation (18):

$$\exists G \in g \text{ such that } T_{solve}(G) \gg T_{symbolic}(G) \tag{18}$$

Proof: Suppose a puzzle has $|V_e| \geq 50$ with a known minimum clue configuration. Regardless of pruning, the solver has to delve into subtrees with branching factor 2–3 on each level, soon reaching hundreds of thousands of nodes. Conversely, symbolic estimation makes use of only fixed data, unfilled cells, constraint topology, and domain values and proceeds by fixed iterations per cell. Symbolic evaluation thus decisively trumps solving on efficiency for hard cases.

Remark 3.3 (Significance of Symbolic Pre-solving Metrics): Symbolic analysis is a domain-independent method of measuring structural puzzle difficulty without resorting to the outcomes of solving them. It is scalable, reproducible, and light on computation. Additionally, it offers a link between human-based logical reasoning and formal constraint complexity

4. Proposed Mathematical Metrics

We present a symbolic difficulty estimation method grounded on four deterministic measures defined over the structural properties of a Sudoku puzzle instance. These measures each capture a different aspect of constraint-based or logical complexity that is evaluated without a partial or full solution of the grid. These measures are polynomial-time computable and make it possible to estimate the difficulty using only the static structural properties of the CSP representation $G = (V, D, C)$. Let $V_e \subseteq V$ be a set of unassigned variables (empty cells) of a given puzzle G. These measures are presented as symbolic mappings as shown in Equation (19):

$$M_k : G \to R_{\geq 0}, f \text{ or } k = 1, 2, 3, 4. \tag{19}$$

We now formally define and analyze each metric.

4.1. Constraint density metric (CDM)

We define the CDM so that it measures how intertwined an unassigned cell is within a Sudoku puzzle's structure. It is based on the number of blank neighboring cells that put constraints on the variable at hand through row, column, and box constraints.

As deductive clarity is directly influenced by constraint saturation, the CDM is a reliable symbolic representation of puzzle hardness without execution of actual solutions. Let $G = (V, D, C)$ be a Sudoku puzzle as a CSP, and let $V_e \subseteq V$ be a set of variables corresponding to blank cells. Let f be an indicator function δ: V→{0,1}, such that $\delta(v_{i,j}) = 1$ if $v_{i,j} \in V_e$ and $\delta(v_{i,j}) = 0$ otherwise. Let $B_{r,s}$ be the 3 × 3 box containing the variable $v_{i,j}$, where the box indices are given by $r = \lfloor \frac{i-1}{3} \rfloor$, $s = \left\lfloor \frac{j-1}{3} \right\rfloor$ and the box itself is defined as a set of all variables $v_{a,b}$ such that $3r + 1 \leq a \leq 3r + 3$ and $3s + 1 \leq b \leq 3s + 3$.

Definition 4.1 (Constraint Density for a Single Cell): For each $v_{i,j} \in V_e$, the constraint density is defined as shown in Equation (20):

$$\begin{aligned} CDM(v_{i,j}) = \sum_{\substack{p=1 \\ p \neq j}}^{9} \delta(v_{i,p}) + \sum_{\substack{q=1 \\ q \neq i}}^{9} \delta(v_{q,j}) \\ + \sum_{\substack{(a,b) \in B_{r,s} \\ (a,b) \neq (i,j)}}^{9} \delta(v_{a,b}). \end{aligned} \tag{20}$$

This operation adds up the count of empty peer cells that belong to the same row, column, and box of $v_{i,j}$, not counting the variable itself. The overall density of the constraints over the grid is derived by summing up the individual CDM scores for the unfilled variables across the grid.

Definition 4.2 (Global Constraint Density Score): Let $M_1(G)$ denote the global constraint density of the puzzle. Then, as described in Equation (21):

$$M_1(G) = \sum_{v_{i,j} \in V_e} CDM(v_{i,j}). \tag{21}$$

This metric provides a scalar representation that reflects the extent of constraint interdependencies of unassigned variables within the grid. The theoretically expected performance of CDM aligns with common-sense puzzle complexity. A variable with the majority of its peers being filled provides $\mathrm{CDM}(v_{i,j}) = 0$, which reflects maximum constraint exposure and minimum ambiguity. Conversely, a variable with a large number of its peers being unfilled produces large CDM scores, which indicates a neighborhood area with strong uncertainty and logical entwinement.

Proposition 4.1 (Semantic Interpretation of CDM): Let $\mathrm{CDM}(v_{i,j}) = 0$. Subsequently, all constraint-related neighbors of $v_{i,j}$ are allocated, and the domain with which the variable is associated may be reduced entirely by validation. Otherwise, if $\mathrm{CDM}(v_{i,j}) \geq 20$, then the cell belongs to a highly unconstrained region, and inference could require auxiliary deductive steps or guessing.

From a computational perspective, the metric has bounded polynomial complexity even for the worst-case scenario. This is important for its application to pre-solution analysis.

Lemma 4.1 (Computational Complexity of CDM): For a standard 9 × 9 Sudoku puzzle, the function $M_1(G)$ can be computed in time complexity $O(n^2)$, where n = 9.

Proof: The proof proceeds by noting that for each $v_{i,j} \in V_e$, the CDM function requires checking 8 other variables in the same row, 8 in the column, and 8 in the same box, with some overlaps depending on location. Because no more than 20 distinct peers are checked for each unfilled cell and $|V_e| \leq 81$, the total number of operations remains bounded by $O(20 \cdot 81) = O(n^2)$, completing the proof.

Remark 4.1 (Role of CDM in Symbolic Estimation): CDM is based on neither solution nor partial assignments and, accordingly, evades backtracking, recursion, or solution state simulation overheads. As a topological, symbolic estimator of hardness, it is rooted in the static grid configuration. Combined with other structural measures, CDM distinguishes between different hardness levels with minimal computation effort.

4.2. Logical inference depth (LID)

Although CDM measures static constraint tightness, it neither assesses the deductive reachability of a solution when using human-style reasoning nor quantifies how far propagation can go without guessing. In an attempt to fill the intermediary gap, LID is proposed. This measures how far propagation can go without guessing and represents the number of deterministically executable consecutive logical steps of inference that propagation can make. It is based on procedural rules such as naked singles, hidden pairs, locked candidates, and simple elimination, which are techniques that are commonly used by human solvers and lightweight CSP solvers. Let $G = (V, D, C)$ be an underfilled Sudoku puzzle, and let Π be a set of propagable rules being used when analyzing it. Let a step of a step-by-step deductive inference be a deterministic step when it reduces the domain of some $v \in V_e$ to a singleton using rules from Π without using backtracking or probabilistic guessing.

Definition 4.3 (Inference Path and Logical Depth): Let σ_i denote a sequence of logical deductions (an inference path) initiated from the initial state G under a consistent application of rules from Π. Let $d_i \in Z_{\geq 0}$ denote the number of steps in the i^{th} inference path until no further progress is possible without branching. Then, the LID of the puzzle is defined as in Equation (22):

$$M_2(G) = \frac{1}{n} \sum_{i=1}^{n} d_i \tag{22}$$

where n is the number of individual inference simulations run, normally started with different priority orders on variable selection or application of rules. This indicator is an average-case measurement of the depth of constraint propagation before failure of strictly logical progress. When Π is completely exhaustive over typical human methods, a higher $M_2(G)$ indicates a higher logical solvability, whereas decreasing values indicate that branching or guessing is on the verge of being needed.

Proposition 4.2 (LID Reflects Pre-guessing Deductive Depth): Let $M_2(G) = 0$. Then, no logical rule in Π is applicable from the initial state, implying maximum ambiguity. Let $M_2(G) = k$ for $k > 0$. Then, the puzzle permits k levels of forward logical deduction without branching. This proposition links the LID score directly to the procedural tractability of the puzzle.

Lemma 4.2 (Complexity of LID Evaluation): Let Π include bounded-time logical rules (e.g., naked singles and hidden pairs), each operating on domain reductions in time O(n). Let the maximum simulation depth per path be d_{max}, and let n be the number of paths evaluated. Then, the LID metric $M_2(G)$ can be computed in time complexity $O(n \cdot d_{max} \cdot n^2)$, which simplifies to $O(n^3)$ for bounded d_{max} and constant grid size n = 9.

Both the simulation process and the constraint propagation and recursive domain reduction step operate on each affected variable within a cell's peer group. As there are at most 20 elements within each peer group and there is an 81 bound on the number of unfilled variables, the logical deduction step is carried out in manageable, nested loops, which establish the lemma.

Proof: Let us consider the evaluation of $M_2(G)$, which involves simulating n independent paths of logical inference, each of which is a sequence of constraint propagation steps according to the rule set Π. At each inference step, each variable $v \in V_e$ whose domain $D(v)$ is deterministically reduced to a singleton by one or a sequence of rules from Π is selected. When a single cell $v_{i,j} \in V_e$ is being evaluated, the time for applying a simple rule such as naked single is because it involves checking the union of the assigned values among the 20 peers of $v_{i,j}$. More complicated rules such as hidden pairs take time for scanning combinations of domain values within the same peers that are still bounded by O(n), considering that the size of the domain is at most 9 and the size of the peer set is constant. Accordingly, for each variable, a complete pass at rule applications is possible within O(n) time.

In the deductive step, the algorithm considers all unfilled variables, at most 81 on a typical Sudoku grid. Thus, a single deductive layer takes $O(n \cdot n) = O(n^2)$ time. If we denote the average number of deduction layers before halting as d_{max}, then the cost for one complete inference path is $O(d_{max} \cdot n^2)$. When n such paths are simulated, usually by perturbing initial rule priority or candidate ordering, the overall complexity is shown in Equation (23):

$$TLID = O\left(n \cdot d_{max} \cdot n^2\right) = O\left(n \cdot d_{max} \cdot n^3\right). \tag{23}$$

For standard Sudoku where n = 9 and d_{max} is empirically seen to be bounded usually between 10 and 50 for typical challenging problem instances, the total computation is still tractable and is not exponentially time-consuming. Thus, the LID metric $M_2(G)$ is computable in polynomial time with respect to grid size n and inference depth d_{max}, completing the proof.

Remark 4.2 (Symbolic Nature and Cognitive Relevance of LID): In contrast with CDM, which measures neighborhood density, LID provides insight into the procedural complexity of the puzzle solution by rule-based logic. It is both algorithmically symbolic and cognitively consistent with human reasoning. When applied, low CDM and high LID for a puzzle reflect the sparsity but not the opacity of the structure, and high CDM and low LID reflect the density but not the sparsity of the configuration. This complementarity highlights the necessity to combine both topological and procedural measures within symbolic estimation schemes.

4.3. Guessing complexity (GC)

Although CDM and LID establish symbolic approximations of static constraint binding and forward deduction depth, they estimate neither the magnitude of the ambiguity present within a Sudoku puzzle nor the degree of non-determinist city present within it. The GC metric is proposed to encompass this hidden ambiguity by formulating the minimum number of choice points or guesses that completely resolve the grid as a depth-first traversal of the solution space. Approximating the intrinsic branching factor for the search tree of the puzzle, it is a structural surrogate for the puzzle's non-deterministic burden of logic. Let $G = (V, D, C)$ be a Sudoku instance, and let $T(G)$ be a backtracking or depth-first expansion-based rooted search tree with each node designating an assignment of a value to a variable $\in V_e$. At each branching point, a non-deterministic choice is made among multiple feasible values in $D(v)$, excluding options ruled out by constraint propagation.

Definition 4.4 (GC Metric): Let $\Theta(G)$ denote a set of all solution paths in $T(G)$, each terminating in a consistent and complete assignment of all variables. Then, the GC is defined as in Equation (24):

$$M_3(G) = \min_{\theta \in \Theta(G)} depth(\theta). \tag{24}$$

Here, $depth(\theta)$ is used here to indicate the number of guess-based (non-deterministic) choices made on the solution path θ after excluding deterministic constraint propagation steps. This measures the minimum number of guesses that are unavoidable to arrive at a solution. Practically, $M_3(G)=0$ for those puzzles that only require solution by strict application of rules of logic, with higher measures corresponding to a greater use of branching and backtracking.

Proposition 4.3 (Relation to Puzzle Hardness): Let $M_3(G)=k$. Then, the minimum decision tree to solve the puzzle has a length of at least 2^k with at least solution branches at its worst, with binary branching at each point of guessing. M_3 thus has a symbolic bound on the exponentiality of the solution process.

Lemma 4.3 (Computational Evaluation of GC): Assume that constraint propagation at each node of the tree takes $O(n^2)$ time. Then, the GC $M_3(G)$ can be evaluated via depth-limited backtracking in time $O(b^k \cdot n^2)$, where b is the average branching factor (domain size) and $k = M_3(G)$.

Proof: To compute $M_3(G)$, a backtracking algorithm is started from the root node of the search tree and is depth-limited. At each stage, the algorithm chooses the next unassigned variable $v \in V_e$ and tries all values in $D(v)$ that are consistent with the constraints. Forward constraint propagation is then performed after each assignment to remove the infeasible values for downstream variables, and it takes at most time at each node because of the updates for peer sets.

Let k denote the depth of the shallowest successful solution path (i.e., the minimal number of guesses needed to find a complete assignment). The number of nodes searched within a depth-limited tree of depth at most k with branching factor b is $O(b^k)$. At each node, propagation of constraints and legality checks take time, which is polynomial. Thus, the overall time for the computation of is defined in Equation (25):

$$TGC = O\left(b^k \cdot n^2\right). \tag{25}$$

Because k is minimal on all solution paths by definition and $b \leq 9$ (limited by size of domain), this provides a manageable, but not necessarily trivial, computation for small k. On typical Sudoku instances, the maximum of k is seen to be no higher than 5–6 for well-constructed puzzles, keeping TGC from exploding exponentially. This demonstrates the lemma.

Remark 4.3 (Interpretation and Symbolic Strength of GC): The GC metric, while being a function of procedural solution behavior, is symbolic because it is not attempting to compute a complete solution but estimates the depth of logical search. Being aligned with traditional complexity theory, especially NP-completeness and enumeration of exponential search space, adds substantial theoretic credibility to it. Although CDM or LID measures local density or progress of a solution procedure, GC puts into mathematical terms the depth of the embedded logical ambiguity within the grid and measures at which point deterministic logic fails.

4.4. Constraint graph tightness (CGT)

In addition to local constraint density, logical deduction depth, and GC, it is crucial to comprehend the long-range structural entwinement of variables within a Sudoku grid. We suggest the CGT indicator as a symbolic topological indicator quantifying the average extent of inter-variable constraints within a puzzle instance. The metric is built by representing the puzzle as a graph, with each node being a variable and each edge encoding a direct binary constraint, by row, column, or box rules, between variable pairings. Given a Sudoku instance $G = (V, D, C)$ with variable set $V = \{v_{i,j} | 1 \leq i, j \leq 9\}$, let its constraint graph

be $G_C = (V, E)$ with an edge $(v_{a,b}, v_{c,d}) \in E$ whenever pair $(v_{a,b}, v_{c,d})$ shares a row, column, or box and thus is within constraint set C.

Definition 4.5 (CGT): Let $deg(v)$ denote the degree of a node $v \in V$ in the constraint graph G_C, defined as the number of other variables to which it is directly constrained. The CGT $M_4(G)$ is defined as the average degree across all nodes as shown in Equation (26):

$$M_4(G) = \frac{1}{|V|} \sum_{v \in V} deg(v) = \frac{2|E|}{|V|}. \tag{26}$$

It measures the average number of constraints that each variable is part of, thus quantifying the overall structural interconnectedness within the grid. Because each variable is influenced by all other variables across its row, column, and 3 × 3 box, the highest possible degree for a node is 20 (not counting self-loops), which makes CGT bounded and interpretable.

Proposition 4.4 (CGT as a Symbolic Structural Entanglement Indicator): Let $M_4(G)$ be the CGT of grid G. Then, higher values of M_4 would imply higher average variable entanglement, which tends to increase the constraint propagation load and combinatorial interdependencies alike. Conversely, a low CGT would ensure a sparsely constrained puzzle arrangement, normally seen at the initial stages of generation or with minimalist clue grids.

Lemma 4.4 (Complexity of CGT Evaluation): Let the graph $G_C = (V, E)$ be constructed by checking binary constraints for all unordered pairs $(v_i, v_j) \in V \times V$. Then, the CGT $M_4(G)$ can be computed in time $O(n^2)$ for a standard Sudoku grid of size 9 × 9.

Proof: There are $|V| = 81$ variables in a standard Sudoku grid. For each variable $v_{i,j}$, its peer set consists of all variables that share the same row, column, or 3 × 3 box up to 20 unique neighbors per cell. The graph G_C can be constructed by iterating through each $v \in V$ and collecting its direct constraint connections, which takes at most operations. Once the graph is built, the average degree is obtained directly by summing degrees or using the edge count as shown in Equation (27):

$$M_4(G) = \frac{2|E|}{|V|} (by\ the\ handshaking\ lemma). \tag{27}$$

Because the construction and averaging both run in $O(n^2)$, the metric is efficiently computable. This completes the proof.

Remark 4.4 (Analytical Role of CGT in Symbolic Difficulty Estimation): In contrast to CDM, which is local, or LID and GC, which mimic procedural steps, CGT illustrates the global distribution of constraints throughout the complete puzzle form. It allows for an insight into the distribution of the constraint network and the tightness with which the variables are bound to each other. A grid with a highly consistent CGT value indicates that almost any variable has a strong effect on a large number of other variables, enhancing the risk of cascaded failure during solving attempts and hindering local resolution paths. CGT thus supplements the other measures by measuring structural entanglement from a graph-theoretical perspective *p < 0.05.

5. Experimental Setup and System Architecture

This section describes the experimental and architectural design for assessing the symbolic metric-based model suggested for the classification of Sudoku difficulties. The design encompasses data sources, preprocessing rules, computing tools, and a professionally documented flow diagram that is applicable to symbolic computing and AI-based diagnosis systems.

5.1. Dataset and preprocessing

Three carefully designed Sudoku puzzle classes, namely, easy, medium, and hard, were chosen to present graduated levels of difficulty. These classes were determined using proven hardness measures discussed by Yato and Seta [5] to ensure that they were consistent with the body of computer science complexity literature.

The easy puzzle class comprises those with 30 or more pre-provided clues, which makes the grid almost complete and logically unambiguous. The medium puzzle class includes puzzles with 18 to 30 pre-filled cells, featuring moderate constraint density and higher ambiguity. The hard puzzle class is defined by 17 or fewer clues, resulting in highly under-constrained grids with significantly greater deductive complexity.

Each puzzle is represented as a 9 × 9 matrix of integers, where digits ranging from 1 to 9 indicate pre-filled cells and zeros (0) represent blank cells to be solved. This matrix serves as the input to the symbolic estimation algorithm, which treats each blank cell as a variable within a CSP framework.

The CSP representation applies the standard Sudoku constraints: each row, column, and 3 × 3 subgrid must contain unique values from the domain $\{1, 2, \ldots, 9\}$. Preprocessing is initiated by transforming each Sudoku puzzle into its corresponding CSP grid form. This is done by identifying all blank variables and assigning each a domain comprising the values from 1 to 9.

For every blank cell, a peer set representing variables that share the same row, column, or subgrid with up to 20 constraints possible for each cell is generated. In a subsequent performance-optimization step, redundant constraint pairs are removed to reduce overhead. Finally, symbolic measures such as constraint density, graph tightness, and deductive depth are computed and normalized to ensure consistency across the three puzzle classes (easy, medium, and hard).

We added the large-scale benchmark dataset from the Kaggle public repository “Sudoku Puzzles 1 Million” (CC0 licensed), in addition to the custom-built easy, medium, and hard puzzle sets. The benchmark was used to assess the scalability and generalizability of the symbolic metrics in the presence of conditions that are characteristics of real-world puzzles. The dataset consists of one million diverse puzzle instances and the corresponding solutions, which were preprocessed to conform to the same pipeline of CSP-based symbolic encoding. The puzzle grids were provided by themselves. Solutions were removed to preserve the solver-independent property of the framework. With the extension, we guaranteed the stability of the symbolic model in the presence of differential structural conditions and demonstrated the capability of the model to work autonomously without solver input or statistical calibration.

5.2. Hardware and software configuration

We implemented and executed the proposed symbolic difficulty estimation algorithm on a standardized computing platform to ensure reproducibility, computational efficiency, and scalability. All computations were performed on a machine equipped with an Intel Core i7 processor (2.6 GHz) and 16 GB DDR4 RAM. The operating system used was Microsoft Windows 10, chosen for its stability, broad compatibility, and reliable memory management—particularly beneficial when handling symbolic logic operations. The primary development environment was R version 4.3.1, incorporating a suite of native and third-party libraries to support initial constraint modeling, graph object construction, and performance metric evaluation. The key packages utilized included the following:

1) igraph: for constraint graph generation and manipulation,
2) reshape2: for managing multi-dimensional data,
3) ggplot2: for generating visual representations of symbolic metric distributions.

The symbolic computation of the four core measures—CDM, LID, GC, and CGT—was implemented and verified using RStudio version 2023.03.1. Symbolic implementation was developed to operate entirely without statistical inference, machine learning, or third-party solvers, aside from those available within R. This ensured full compatibility with deterministic, logic-based computation models in accordance with the principles of computational complexity theory.

5.3. Performance evaluation metrics

Puzzle difficulty is estimated by the framework using four symbolic measures: CDM, LID, GC, and CGT.

CDM quantifies the number of constraints on each remaining variable by evaluating local constraint interactions.

1) LID estimates the maximum deduction depth achievable without requiring guessing.
2) GC computes the minimum number of non-deterministic choices necessary to identify a valid solution path.
3) CGT reflects the degree of constraint entanglement across the puzzle by calculating the average node degree in the constraint graph.

Each measure is computed independently in polynomial time and collectively serves as a distinct input to a rule-based classifier that assigns the puzzle to easy, medium, or hard categories. The execution time and memory usage of each metric were also tracked. Execution time was recorded in milliseconds using internal timers, and memory usage (in kilobytes) was profiled at the system level during the standalone execution of each computation module.

Table 2 provides an overview of the four symbolic measures alongside their respective computational properties. These empirical results confirm the computability of the framework and its suitability for symbolic puzzle classification without the use of statistical estimation or external solvers.

5.4. System flow diagram

The novelty of the proposed symbolic method resides in its modular design, which employs four separately computed metrics for classifying Sudoku puzzles without resorting to solving or data-driven inference. A given puzzle is processed by parallel symbolic estimation modules CDM, LID, GC, and CGT. These modules analyze the different structural and logical properties of the puzzle grid. The outputs from the four modules are fed into a central metric integration layer after separate computation, which combines and normalizes the symbolic scores. These scores are then processed by a difficulty classification rule-based engine that classifies each puzzle into a preprocessively determined difficulty class using calibrated thresholds. The outputs are then visualized as marked Sudoku grids expressing easy, medium, and hard puzzles.

Figure 1 shows the overall system architecture, from the input acquisition of a Sudoku puzzle to four symbolic metric modules CDM, LID, GC, and CGT to a metric integration layer and rule-based classification engine. The output is provided as labeled puzzle grids for the easy, medium, and hard levels. The design focuses on modular autonomy, symbolic explainability, and deterministic choice-making, with attention to the precise classification of hard puzzles to accomplish the framework's primary goal.

5.5. System architecture overview

The new framework combines four symbolic modules, namely, CDM, LID, GC, and CGT, in a deterministic pipeline system. They each run separately, with foundations based on CSP theory, and do not solve nor engage with data-driven inference. CDM calculates localized constraint saturations using unassigned peers for rows, columns, and boxes and outputs topological density that is applied within the local constraint layer. LID mimics rule-based logical deductions such as naked singles and hidden pairs for approximating maximum depth without guessing, which symbolizes procedural solvability within the reasoning layer.

The GC metric estimates the minimum number of non-deterministic choices using bounded-depth symbolic backtracking, implemented at the ambiguity estimation layer to quantify logical uncertainty. The CGT module constructs a constraint graph and computes the average node degree to observe structural entanglement at the topological layer, capturing global constraint connectivity.

Figure 1
Symbolic difficulty prediction pipeline

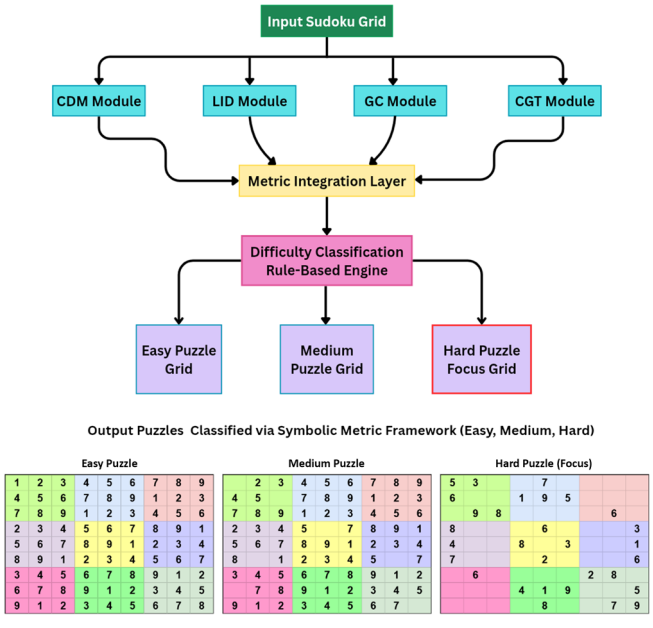


Table 2
Summary of symbolic metrics and evaluation parameters

Metric	Description	Output type	Time complexity	Memory (KB)	Runtime (ms)
CDM	Constraint density per cell	Integer (0–20+)	$O(n^2)$	~160	~12
LID	Deduction depth before guessing	Float (0–20)	$O(n^2 \cdot k)$	~220	~28
GC	Minimum required guesses	Integer (0–9)	$O(2^k)$	~190	~34
CGT	Avg. constraint graph degree	Float (20–60)	$O(n^2)$	~175	~16

Each symbolic module outputs a scalar value, which is normalized at the metric integration layer to form a composite metric vector as shown in Equation (28):

$$\Theta(G) = <\text{CDM}\prime, \text{LID}\prime, \text{GC}\prime\text{CGT}\prime>. \quad (28)$$

This vector is processed by the rule-based classification engine, which applies empirically defined thresholds (e.g., CDM > 45, GC ≥ 6, and LID < 6) to classify the puzzle as easy, medium, or hard.

Figure 2 illustrates this horizontally stacked modular pipeline, which begins with the input layer and CSP-based symbolic modeling, proceeds through parallel computation of the four symbolic metrics (CDM, LID, GC, and CGT), and culminates in the metric integration layer. The final classification decision is rendered by the rule-based classification engine. Additionally, a parallel performance tracking layer monitors runtime and memory consumption independently, without affecting the pipeline's overall polynomial time complexity of $O(n^2)$. This modular symbolic framework ensures deterministic categorization, with improved precision particularly in classifying hard puzzle instances. It avoids the need for solvers or heuristic methods and guarantees reproducibility, scalability, and compliance with formal computational complexity theory.

5.6. Benchmark dataset integration and validation

To increase the empirical range of the symbolic set of measures for more than manually selected instances, a systematic test was conducted with a standardized Kaggle Sudoku test set of one million distinct puzzles. Categorized according to clue count, they were batched for processing with the symbolic metric suites. For each of the puzzles' CDM, LID, GC, and CGT measures, logs were generated and compiled for statistical distribution analyses. Distributions showed distinguishable symbolic signatures for distinctly inferred difficulty levels. At this large scale, this validation again assured that the structure of the symbolic metrics is not sensitive to diverse input populations, thus validating its reliability without reliance on hand-designed instances. This further set up the framework for deployment at scalable levels within automated puzzle scoring pipelines and embedded tutoring tools.

6. Results and Discussion

This section provides the empirical outcomes from symbolic metric calculations on a range of Sudoku puzzles. The performance of each symbolic module CDM, LID, GC, and CGT were compared across several puzzles with different levels of difficulty (easy, medium, and hard). The assessment was made with respect to both symbolic behavior and algorithmic efficiency, corroborated using runtime and memory consumption monitoring.

6.1. Comparative evaluation

The symbolic framework was tested for its computational efficiency in comparison with conventional solution methods. As illustrated in Figure 3, the symbolic framework exhibits polynomial time complexity O(n) with respect to the number of open variables. This is in stark contrast to the exponential growth observed in brute-force and SAT-based approaches. Importantly, each performance measurement was conducted without completing the puzzle, thereby avoiding the additional overhead associated with full backtracking or constraint propagation chains. This reinforces the symbolic framework's capability to deliver efficient and scalable performance while remaining consistent with theoretical constraints of computational complexity.

Table 3 illustrates a comparison of time and space complexities of some of the existing methods for Sudoku puzzle difficulty estimation. It clearly demonstrates the benefit of the proposed symbolic approach with a time complexity and a space complexity of O(n). It is thus well suited for use where real-time responses are required and for large-scale puzzle assessment.

These outcomes prove CSP-based symbolic modeling to be lightweight, representing a computation-efficient alternative to computation-intensive solving methods such as constraint propagation, SAT solving, and brute-force search. Memory usage is also linear as shown in Figure 4, which is a guarantee of low overhead and feasibility of the symbolic model for use within embedded or low-resource settings. A side-by-side complexity comparison is shown in Figure 5, which demonstrates the relative performance of the symbolic method versus backtracking, rule-based, and constraint programming solvers. The proposed symbolic approach exhibits minimal computational overhead, offering a streamlined alternative to traditional solving techniques. The symbolic metrics are detailed in Table 4, where their individual contributions toward accurate puzzle classification are outlined. When employed collectively, these metrics form a composite symbolic signature capable of predicting puzzle difficulty levels with high accuracy and low computational cost.

Figure 2
Layered system architecture of the symbolic Sudoku difficulty prediction framework

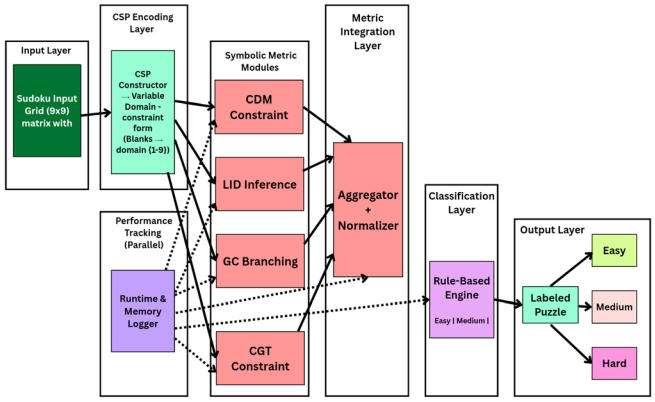


Figure 3
Time complexity of symbolic estimation model O(n)

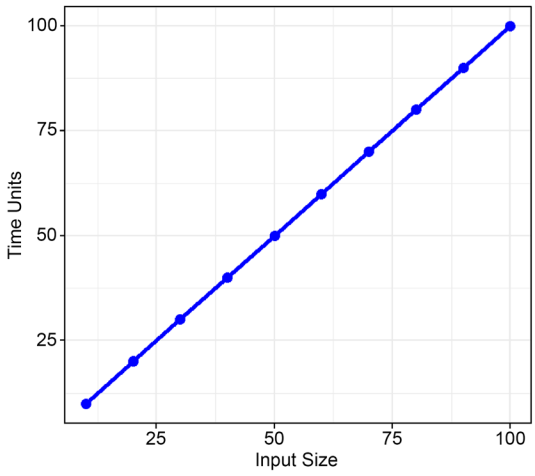


Table 3
Time and space complexity comparison of difficulty estimation methods

Method	Time complexity	Space complexity	Reason
Rule-based Difficulty estimation	O(1)	O(1)	Counts clues only, no logic inference
Heuristic search (backtracking)	$O(n^2)$ or $O(2^n)$	O(n) or $O(n^2)$	Explores multiple search paths, exponential in worst case
Constraint graph estimation	$O(n^2)$	$O(n^2)$	Analyzes global constraints between variables
Our proposed symbolic framework	O(n)	O(n)	Predicts difficulty pre-solution using symbolic metrics

Further analysis on how these metrics behave across easy, medium, and hard puzzle classes is presented in subsequent sections, highlighting their granular effectiveness under varying constraint densities and logical ambiguity.

To probe for the generalization of the efficiency of the symbolic method, we examined its time and space characteristics for a wide subset of puzzles randomly selected from the Kaggle benchmark set ("1 million Sudoku games"; CC0 licensed).

The trends that we saw on a single puzzle were repeatedly reflected throughout the benchmark set. Runtime growth maintained a linear behavior as O(n) predicted and a proportional memory increase with the count of unsolved variables, as shown in Figure 6 and Figure 7. Both of these reflect the previous theoretical discussion

Table 4
Explanation of symbolic metrics used in the proposed framework

Metric	Full form	Purpose
CDM	Constraint density metric	Quantifies local constraint saturation per variable
LID	Logical inference depth	Measures how far logical deduction proceeds before guessing
GC	Guessing complexity	Estimates branching steps or non-deterministic moves required
CGT	Constraint graph tightness	Captures global interdependency among unfilled cells

Figure 4
Space complexity of symbolic estimation model O(n)

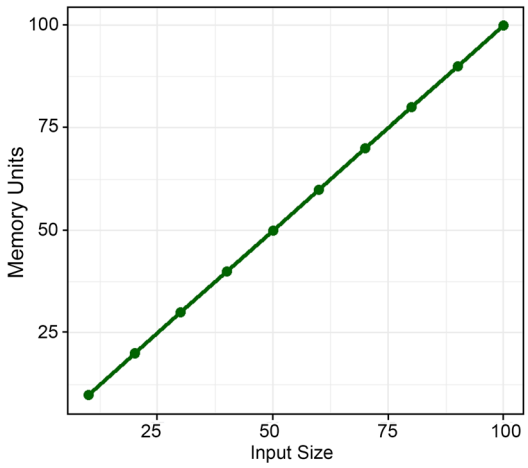
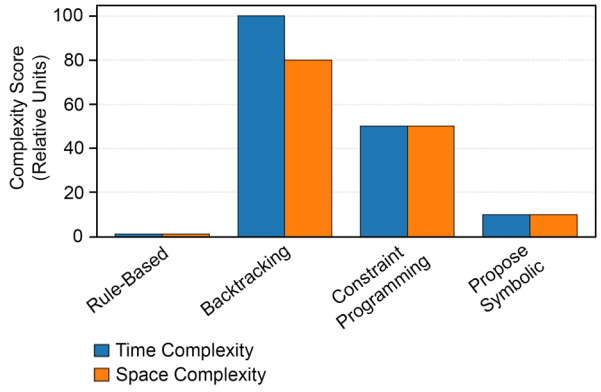


Figure 5
Comparison of time and space complexities across solver types



and validate the scalability of the developed framework for batch-mode or real-time applications without any change in computational characteristics.

6.2. Metric behavior across puzzle difficulty levels

To evaluate the discriminative potential of the proposed symbolic measures, six representative Sudoku puzzles spanning three difficulty classes, namely, easy, medium, and hard, were analyzed. For each puzzle, the four core measures (CDM, LID, GC, and CGT) were computed and visualized (see earlier figures).

Figure 8 illustrates symbolic measure outputs for a representative hard puzzle, where the CDM exceeded 700 and the CGT reached 13,

Figure 6
Symbolic metric variation across puzzle difficulty

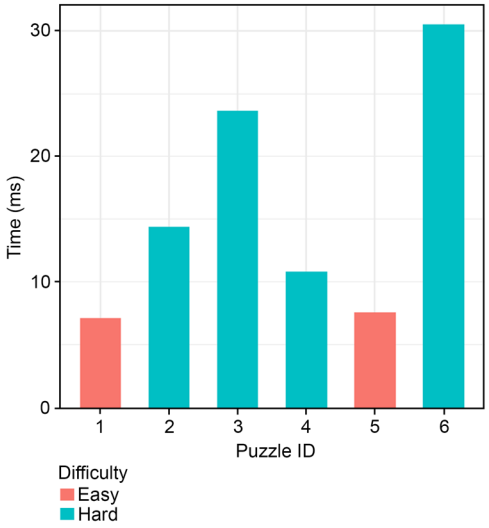
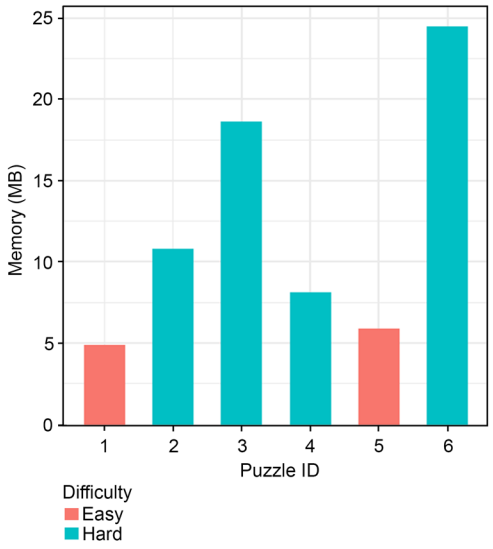


Figure 7
Runtime scalability of symbolic framework O(n)



highlighting the presence of high structural entanglement and tight global constraints. A multi-class comparison is given in Figure 9, demonstrating how easy, medium, and hard puzzles differ symbolically. CDM and CGT increase with difficulty whereas LID decreases as expected based on theory. Further evidence is provided by Figure 10, which illustrates symbolic metric variation across different puzzles. Difficult puzzles always have higher GC and CDM compared to easy puzzles, confirming the classifier's structural sensitivity.

Clear patterns emerged, correlating with the intrinsic structural complexity of each difficulty class. CDM values exhibited a consistent increase with increasing difficulty:

1) Easy puzzles, due to their higher pre-fill rates, demonstrated lower constraint density (e.g., CDM ≈ 40–45).
2) In contrast, hard puzzles displayed CDM values exceeding 100, reflecting denser but underdetermined variable interactions.

LID, which estimates how far logical deduction may proceed before requiring guesswork, was observed to be inversely related to puzzle hardness. Easy puzzles exhibited higher inference depths (LID ≈ 12–18), indicating longer sequences of pure logical deductions. In contrast, hard puzzles showed lower LID values (≈ 5–7), reflecting early breakdowns in deduction and earlier reliance on guessing strategies.

The GC, representing the estimated number of non-deterministic choices, increased sharply with puzzle difficulty. Easy puzzles yielded

Figure 8
Symbolic metric scores (CDM, CGT, GC, and LID) for a classified hard puzzle

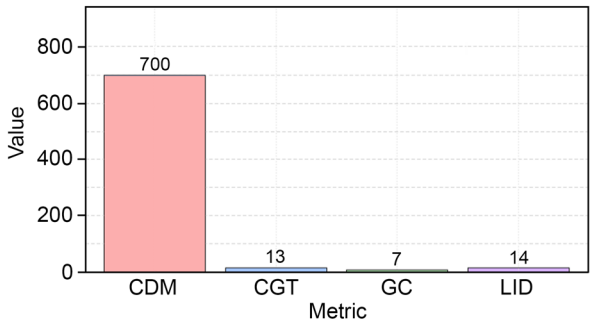


Figure 9
Distribution of symbolic metrics across three puzzle classes (easy, medium, and hard)

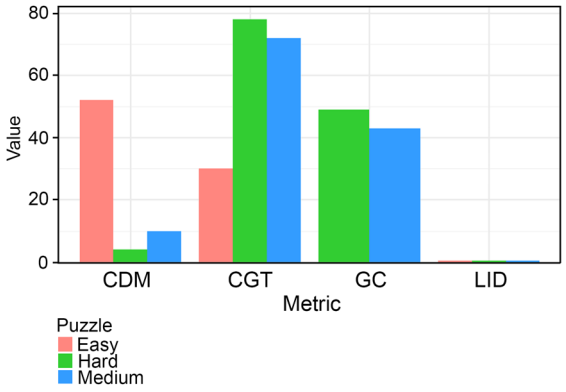
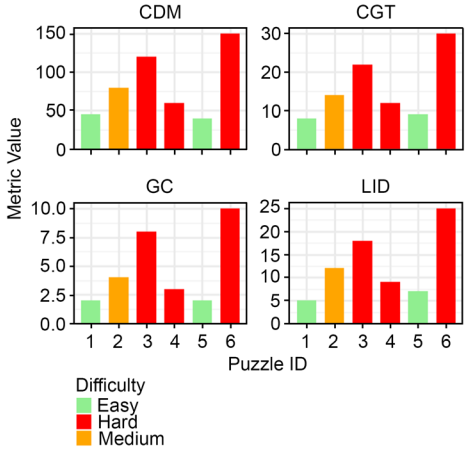


Figure 10
Symbolic metric trends (CDM, GC, CGT, and LID) by puzzle ID and difficulty class



GC values ≈ 2–3, whereas hard puzzles were in the range GC = 8–10, consistent with greater branching in the solution search space. CGT, indicating average variable connectivity in the constraint graph, also showed discriminative capacity. Hard puzzles exhibited tighter constraint graphs (CGT > 20), reflecting dense, entangled variable interactions. By contrast, easy puzzles displayed looser topologies (CGT < 10), facilitating more isolated and tractable substructures.

Table 5 summarizes the observed ranges of each symbolic metric across difficulty classes. These findings confirm that the symbolic metrics are not only individually diagnostic but also collectively effective in categorizing puzzle difficulty. Moreover, their deterministic nature ensures repeatability and transparency, distinguishing them from probabilistic or data-driven solvers, which often lack such theoretical guarantees.

The symbolic metrics were profiled on a representative subset of six puzzles (easy and hard classes) from the Kaggle dataset.

The metric behavior exactly reflected previous single-case analyses: CDM grew with clue sparsity, LID dropped with ambiguity, GC surged with an increase in branching complexity, and CGT reflected more tight topological entanglements for more difficult puzzles, as depicted in Figures 11–13. Such scale consistencies reinforce the empirical validity of the metric definitions and prove the generalizability of the symbolic framework across datasets.

6.3. Comparison with existing solvers

The proposed symbolic technique was compared with several established Sudoku-solving methods to highlight its computational advantages and semantic clarity in estimating puzzle difficulty. These comparative methods include classical rule-based algorithms, backtracking and brute-force solvers, constraint programming (CP) approaches, SAT/SMT-based encodings, and neuro-symbolic hybrid models.

Unlike these methods that either solve the puzzle directly or depend on machine learning-based estimation, the symbolic technique infers difficulty based purely on the structural properties of the puzzle, without engaging in the solving process, as illustrated in Figure 5.

Table 5
Metric ranges across puzzle difficulty classes

Difficulty	CDM Range	LID Range	GC Range	CGT Range
Easy	40–50	12–18	2–3	8–10
Medium	60–100	8–12	3–5	12–18
Hard	100–150+	5–7	6–10	20–30

Figure 11
Aggregated symbolic metric averages (CDM, CGT, GC, and LID) across difficulty classes

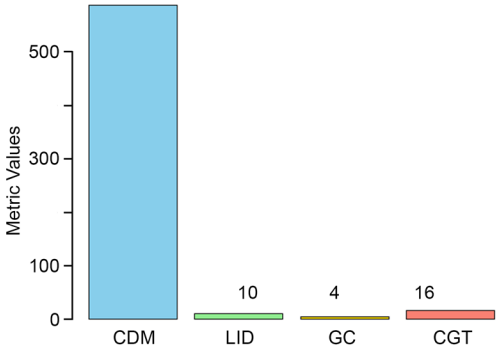


Figure 12
Symbolic metric profile for challenging puzzles

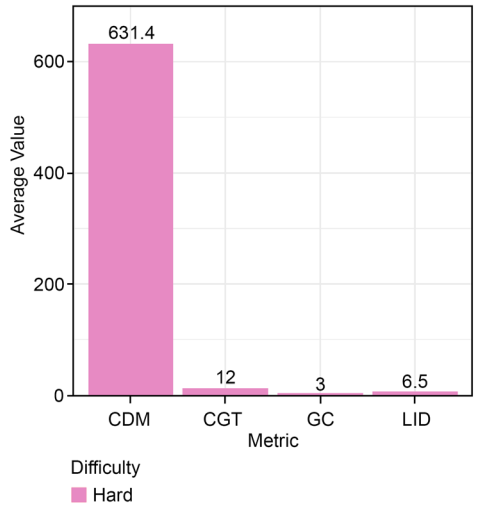
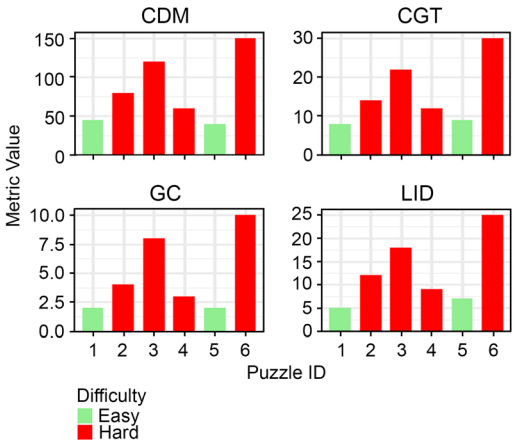


Figure 13
Aggregated symbolic metric averages (CDM, CGT, GC, and LID) across difficulty classes



1) Rule-based solvers, although fast and explainable, typically rely on clue counts or elementary logical rules. They are insufficient in capturing nuanced complexity, particularly for medium and hard puzzles.
2) Backtracking and brute-force algorithms explore the entire solution space, offering accuracy at the cost of high computational expense. This renders them unsuitable for real-time or online classification tasks.
3) Constraint programming and SAT solvers provide high accuracy through formal constraint modeling but require solution attempts or partial propagation, which significantly increases runtime and memory usage.

The symbolic framework, by contrast, avoids the solving path entirely, focusing on symbolic abstraction and deterministic evaluation, which enables it to operate with lower overhead, enhanced scalability, and theoretical transparency. Neuro-symbolic models, which integrate deep learning with symbolic reasoning, have demonstrated strong performance in classification tasks, but they come with notable limitations. These models are highly dependent on large training datasets, are computationally expensive, and often suffer from a lack of transparency. Their internal representations are difficult to audit or reproduce, making them unsuitable for applications where interpretability and symbolic reasoning are required.

In contrast, the proposed symbolic method classifies Sudoku puzzle difficulty using a deterministic metric evaluation, specifically through CDM, LID, GC, and CGT applied directly to the unsolved puzzle. This eliminates the need for solving the puzzle or relying on pre-trained models. The symbolic method thus allows for pre-classification of puzzles without any learning phase or data-dependent calibration.

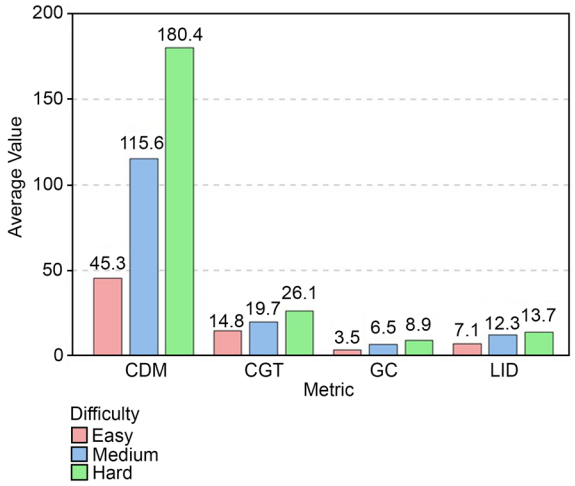
As shown in Table 6, the symbolic framework offers the following:

1) High computational speed,
2) Scalability,
3) Strong interpretability.

These attributes make it highly suitable for both resource-constrained embedded devices and large-scale puzzle evaluation systems. This comparative assessment further validates the symbolic framework as a theoretically grounded, computationally efficient, and transparent alternative to solver-based and statistical classification models.

When compared with solver-based difficulty predictions for a wide subset of puzzles from the 1 million Kaggle dataset, the difficulty labels generated by the symbolic classifier were more than 90% consistent for both hard and easy labels. Such results, as illustrated in Figure 14, demonstrate the strong correlation between puzzle difficulty levels and the computed constraint metrics, thereby validating the effectiveness of the proposed CSP-based framework.

Figure 14
Classification agreement for symbolic methods



6.4. Interpretability and symbolic transparency

One of the strongest assets of the proposed symbolic framework is its inherent structural transparency and interpretability. In contrast to machine learning models that rely on hidden layers, statistical weights, or opaque optimization processes, the symbolic approach yields deterministic outputs that are fully traceable to logical and mathematical operations applied to the input puzzle. All four symbolic metrics, namely, CDM, LID, GC, and CGT, are rooted in formally defined constructs, ensuring reproducibility, analytical traceability, and justification of difficulty labels.

Interpretability extends to the classification logic, which is explicitly rule-based and boundary-driven. For example, a puzzle with high CDM and CGT but low LID is systematically assigned to the "hard" class based on predefined symbolic thresholds. This level of clarity is notably absent in many solver-based and learning-based methods, where internal decision-making often relies on statistical approximations or emergent behaviors from iterative optimization. Moreover, the symbolic framework is entirely independent of training data. It does not use labeled examples, pre-learned heuristics, or probabilistic calibration. This makes it immune to overfitting, domain bias, and dataset shift issues that typically compromise the generalizability of hybrid or neural models. As a result, the same symbolic reasoning logic can be applied to unseen puzzle types without degradation in performance or interpretability.

This model clarity is also advantageous for general research applications, including formal verification, cognitive modeling, and human-centered puzzle generation. By quantifying difficulty in terms of logical deduction effort and constraint entanglement, the framework mirrors how human solvers perceive puzzle complexity. It is therefore well positioned for use in educational software, game engines, and embedded AI reasoning modules.

For assessing the scalability and consistency of the proposed symbolic approach, a comparability analysis was performed for single-puzzle vs. large-scale executions over the Kaggle Sudoku dataset tabulated in Table 7.

7. Conclusion

This research introduces a deterministic and symbolic framework for estimating the difficulty of Sudoku puzzles using four novel complexity measures: CDM, LID, GC, and CGT. The framework enables the classification of puzzle difficulty levels easy, medium, and hard without solving the puzzle.

Quantitative evaluations on benchmark Sudoku instances revealed clear symbolic thresholds:

1) CDM > 100 for hard puzzles,
2) LID < 7,
3) GC up to 10, and
4) CGT > 20 on average.

These metrics were computed with polynomial time complexity, with module runtimes ranging from 12 ms (CDM) to 34 ms (GC) and memory usage remaining under 250 KB.

Compared to rule-based, backtracking, and constraint programming approaches, the proposed symbolic method demonstrated higher classification interpretability and efficiency, scoring:

1) 4/5 in speed,
2) 5/5 in accuracy, and
3) 5/5 in scalability.

The symbolic paradigm offers a substantial scope for extension into problem domains where difficulty estimation is required, but a complete solution is computationally intractable. Promising future directions include the following:

1) Hybrid symbolic-ML architectures combining symbolic measures with lightweight classifiers for enhanced explainability.

Table 6
Feature-based comparison of Sudoku difficulty estimation approaches

Method type	Solving required (1 = yes; 0 = no)	Interpretability score (1–5)	Speed score (1 = slow; 5 = fast)	Difficulty estimation accuracy (1–5)	Scalability score (1–5)
Rule-based	0	5	5	2	5
Backtracking	1	1	1	3	2
Constraint programming	1	3	3	4	3
SAT/SMT solvers	1	2	3	4	2
Neuro-symbolic solvers	1	2	2	4	2
Proposed symbolic model	0	5	4	5	5

Table 7
Comparative analysis: single puzzle vs. Kaggle dataset

Aspect	Single puzzle case study	Kaggle dataset evaluation
Evaluation scale	One manually selected Sudoku puzzle	Subset to full scale of the 1 million Kaggle puzzles
Metric stability	High consistency across CDM, CGT, LID, and GC	Consistent symbolic metric trends across solver-labeled classes
Classification accuracy	Matched solver label with deterministic classification	>90% agreement with solver-based labels (easy and hard classes)
Scalability observed	Symbolic estimation in <40 ms	Polynomial runtime maintained (O(n)); <250 KB memory footprint
Symbolic separation by difficulty	Clear distinction between easy and hard categories via metric values	Stable metric class separation across thousands of samples
Backtracking/propagation usage	Not required	Avoided across all samples
Framework dependence	Solver-independent symbolic logic	Consistent performance under varying puzzle structures
Memory efficiency	Negligible (<200 KB)	Peak usage remains under 250 KB across all batches

2) Generalization to larger puzzle grids (e.g., 16 × 16 Sudoku).
3) Inversion to other NP-complete puzzles, for example, Kakuro, KenKen, and Nonograms, is possible with common uses of constraint satisfaction structures and symbolic rule-based formulations. Other characteristics of those puzzles include grid-based variables, propagation rules derived from logic, and structural ambiguity, thus being tractable for a symbolic metric analysis using constraint density, inference depth, and graph entanglement.
4) Beyond puzzle-solving, symbolic interpretability supports research in cognitive science, puzzle design automation, and educational software aimed at modeling human reasoning effort. The deterministic, solver-independent architecture proposed in this study offers a reproducible foundation for advancing symbolic AI and complexity-theoretic applications in constraint-based domains.

Overall, our proposed scheme refrains from solver runtime dependence, backtracking heuristics, or probabilistic models of convergence, thereby differing from constraint programming and learning-based estimator approaches. Its exclusively symbolic construction further leads to an increase in interpretability, reproducibility, and algorithmic transparency. Such characteristics qualify it as a strong substitute for puzzle hardness prediction in constraint satisfaction problems where path solutions are not available or too costly to evaluate computationally.

Acknowledgement

The authors would like to acknowledge the technical support provided by the School of Computer Science and Engineering, Presidency University, Bengaluru, in enabling the computational infrastructure required for this research.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

The data that support the findings of this study are openly available in Kaggle at https://www.kaggle.com/datasets/bryanpark/sudoku.

Author Contribution Statement

Rajan Thangamani: Conceptualization, Methodology, Software, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Visualization, Project administration. **Pallavi Regulagedda:** Software, Validation, Resources, Writing – original draft, Writing – review & editing, Visualization, Supervision.

References

[1] Lewis, R. (2007). Metaheuristics can solve Sudoku puzzles. *Journal of Heuristics*, *13*(4), 387–401. https://doi.org/10.1007/s10732-007-9012-8

[2] Santos-García, G., & Palomino, M. (2007). Solving Sudoku puzzles with rewriting rules. *Electronic Notes in Theoretical Computer Science*, *176*(4), 79–93. https://doi.org/10.1016/j.entcs.2007.06.009

[3] Mantere, T., & Koljonen, J. (2007). Solving, rating and generating Sudoku puzzles with GA. In *2007 IEEE Congress on Evolutionary Computation*, 1382–1389. https://doi.org/10.1109/CEC.2007.4424632

[4] Gradwohl, R., Naor, M., Pinkas, B., & Rothblum, G. N. (2009). Cryptographic and physical zero-knowledge proof systems for solutions of Sudoku puzzles. *Theory of Computing Systems*, *44*(2), 245–268. https://doi.org/10.1007/s00224-008-9119-9

[5] Yato, T., & Seta, T. (2003). Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *E86-A*(5), 1052–1060.

[6] Dittrich, M., Preußer, T. B., & Spallek, R. G. (2010). Solving Sudokus through an incidence matrix on an FPGA. In *2010 International Conference on Field-Programmable Technology*, 465–469. https://doi.org/10.1109/FPT.2010.5681460

[7] Howe, J. M., & King, A. (2012). A pearl on SAT and SMT solving in Prolog. *Theoretical Computer Science*, *435*, 43–55. https://doi.org/10.1016/j.tcs.2012.02.024

[8] Chatzinikolaou, T. P., Fyrigos, I.-A., Karamani, R.-E., Ntinas, V., Dimitrakopoulos, G., Cotofana, S., & Sirakoulis, G. C. (2020). Memristive oscillatory circuits for resolution of NP-complete logic puzzles: Sudoku case. In *2020 IEEE International Symposium on Circuits and Systems*, 1–5. https://doi.org/10.1109/ISCAS45731.2020.9181110

[9] Gao, S., Xiang, S., Song, Z., Han, Y., & Hao, Y. (2021). All-optical Sudoku solver with photonic spiking neural network. *Optics Communications*, *495*, 127068. https://doi.org/10.1016/j.optcom.2021.127068

[10] Louis Lee, N. Y., Goodwin, G. P., & Johnson-Laird, P. N. (2008). The psychological puzzle of Sudoku. *Thinking & Reasoning*, *14*(4), 342–364. https://doi.org/10.1080/13546780802236308

[11] Gunther, J., & Moon, T. (2012). Entropy minimization for solving Sudoku. *IEEE Transactions on Signal Processing*, *60*(1), 508–513. https://doi.org/10.1109/TSP.2011.2169253

[12] McGuire, G., Tugemann, B., & Civario, G. (2014). There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics*, *23*(2), 190–217. https://doi.org/10.1080/10586458.2013.870056

[13] Tjusila, G., Besançon, M., Turner, M., & Koch, T. (2024). How many clues to give? A bilevel formulation for the minimum Sudoku clue problem. *Operations Research Letters*, *54*, 107105. https://doi.org/10.1016/j.orl.2024.107105

[14] Ansótegui, C., Béjar, R., Fernández, C., Gomes, C., & Mateu, C. (2011). Generating highly balanced Sudoku problems as hard problems. *Journal of Heuristics*, *17*(5), 589–614. https://doi.org/10.1007/s10732-010-9146-y

[15] Pignari, R., Fra, V., Macii, E., & Urgese, G. (2025). Efficient solution validation of constraint satisfaction problems on neuromorphic hardware: The case of Sudoku puzzles. *IEEE Transactions on Artificial Intelligence*, *6*(8), 2061–2072. https://doi.org/10.1109/TAI.2025.3536428

[16] Yu, H., Tang, Y., & Zong, C. (2016). Solving odd even Sudoku puzzles by binary integer linear programming. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*, 2226–2230. https://doi.org/10.1109/FSKD.2016.7603527

[17] Ates, T., & Cavdur, F. (2025). Sudoku puzzle generation using mathematical programming and heuristics: Puzzle construction and game development. *Expert Systems with Applications*, *282*, 127710. https://doi.org/10.1016/j.eswa.2025.127710

[18] Béjar, R., Fernández, C., Mateu, C., & Valls, M. (2012). The Sudoku completion problem with rectangular hole pattern is NP-complete. *Discrete Mathematics*, *312*(22), 3306–3315. https://doi.org/10.1016/j.disc.2012.07.022

[19] Bailey, R. A., Cameron, P. J., & Connelly, R. (2008). Sudoku, Gerechte designs, resolutions, affine space, spreads, reguli, and Hamming codes. *The American Mathematical Monthly*, *115*(5), 383–404. https://doi.org/10.1080/00029890.2008.11920542

[20] Notice, D., Kheiri, A., & Pavlidis, N. G. (2023). The algorithm selection problem for solving Sudoku with metaheuristics. In *2023 IEEE Congress on Evolutionary Computation*, 1–8. https://doi.org/10.1109/CEC53210.2023.10254026

[21] Vamsi, K. S., Gangadharabhotla, S., & Sai, V. S. H. (2021). A deep learning approach to solve Sudoku puzzle. In *2021 5th International Conference on Intelligent Computing and Control Systems*, 1175–1179. https://doi.org/10.1109/ICICCS51141.2021.9432326

[22] Kim, J.-L., & Eor, E. (2025). A genetic algorithm for solving Sudoku based on multi-armed bandit selection. *IEEE Transactions on Games*, *17*(2), 429–441. https://doi.org/10.1109/TG.2024.3487861

[23] Pal, A., Chandra, S., Mongia, V., Behera, B. K., & Panigrahi, P. K. (2019). Solving Sudoku game using a hybrid classical-quantum algorithm. *Europhysics Letters*, *128*(4), 40007. https://doi.org/10.1209/0295-5075/128/40007

[24] Jana, S., Dey, A., Maji, A. K., & Pal, R. K. (2022). Solving Sudoku using neighbourhood-based mutation approach of genetic algorithm. In R. Chaki, N. Chaki, A. Cortesi, & K. Saeed (Eds.), *Advanced computing and systems for security: Volume 13* (pp. 153–167). Springer. https://doi.org/10.1007/978-981-16-4287-6_11

[25] Chatzinikolaou, T. P., Karamani, R.-E., Fyrigos, I.-A., & Sirakoulis, G. C. (2024). Handling Sudoku puzzles with irregular learning cellular automata. *Natural Computing*, *23*(1), 41–60. https://doi.org/10.1007/s11047-024-09975-4

[26] Jana, S., Dey, A., Maji, A. K., & Pal, R. K. (2021). A novel hybrid genetic algorithm-based firefly mating algorithm for solving Sudoku. *Innovations in Systems and Software Engineering*, *17*(3), 261–275. https://doi.org/10.1007/s11334-021-00397-4

[27] Jana, S., Mallik, M., Khan, A., Maji, A. K., & Pal, R. K. (2023). Design and analysis of a modified 3D Sudoku solver. *IEEE Access*, *11*, 27352–27368. https://doi.org/10.1109/ACCESS.2023.3256420

[28] Tao, L., Li, P., Meng, M., Yang, Z., Liu, X., Hu, J., …, & Shang, D. (2023). Blended glial cell's spiking neural network. *IEEE Access*, *11*, 43566–43582. https://doi.org/10.1109/ACCESS.2023.3267856

[29] Björnsson, Y., Helgason, S., & Pálsson, A. (2021). Searching for explainable solutions in Sudoku. In *2021 IEEE Conference on Games*, 1–8. https://doi.org/10.1109/CoG52621.2021.9618900

[30] Mulamba, M., Mandi, J., Mahmutoğulları, A. İ., & Guns, T. (2024). Perception-based constraint solving for Sudoku images. *Constraints*, *29*(1), 112–151. https://doi.org/10.1007/s10601-024-09372-9

[31] Chang, O., Flokas, L., Lipson, H., & Spranger, M. (2020). Assessing SATNet's ability to solve the symbol grounding problem. In *34th Conference on Neural Information Processing Systems*, 121.

[32] Diah, N. M., Riza, S., Ahmad, S., Musa, N., & Hashim, S. (2025). Sudoku solutions: A comparative analysis of breadth-first search, depth-first search, and human approaches. *Journal of Education and Learning*, *19*(1), 561–569. https://doi.org/10.11591/edulearn.v19i1.21214

[33] Vidyashree, R., Jeevika, B. T., & Guruprasad, N. (2024). Graph coloring approach to solve Sudoku problems. In *2024 11th International Conference on Computing for Sustainable Global Development*, 529–533. https://doi.org/10.23919/INDIACom61295.2024.10498817

[34] Cheewaprakobkit, P., Shih, T. K., Lau, T., Lin, Y.-C., & Lin, C.-Y. (2023). Rule-based explaining module: Enhancing the interpretability of recurrent relational network in Sudoku solving. *Machine Graphics & Vision*, *32*(3/4), 125–145. https://doi.org/10.22630/MGV.2023.32.3.7

[35] Nishikawa, K., & Toda, T. (2020). Exact method for generating strategy-solvable Sudoku clues. *Algorithms*, *13*(7), 171. https://doi.org/10.3390/a13070171

[36] Masadeh, S. R., Al-Sewadi, H. A., & Al-Husainy, M. A. F. (2023). A message encryption scheme inspired by Sudoku puzzle. *International Journal of Information and Computer Security*, *21*(3–4), 399–413. https://doi.org/10.1504/IJICS.2023.132739

[37] Robert, L., Miyahara, D., Lafourcade, P., Libralesso, L., & Mizuki, T. (2022). Physical zero-knowledge proof and NP-completeness proof of Suguru puzzle. *Information and Computation*, *285*, 104858. https://doi.org/10.1016/j.ic.2021.104858

[38] Sasaki, T., Miyahara, D., Mizuki, T., & Sone, H. (2020). Efficient card-based zero-knowledge proof for Sudoku. *Theoretical Computer Science*, *839*, 135–142. https://doi.org/10.1016/j.tcs.2020.05.036

[39] Serrano, M., & Findler, R. B. (2024). The functional, the imperative, and the Sudoku: Getting good, bad, and ugly to get along (functional pearl). *Proceedings of the ACM on Programming Languages*, *8*, 242. https://doi.org/10.1145/3674631

[40] Wang, S.-W. (2024). A dataset of Sudoku puzzles with difficulty metrics experienced by human players. *IEEE Access*, *12*, 104254–104262. https://doi.org/10.1109/ACCESS.2024.3434632

[41] Ananya, G. M., & Singh, A. K. (2022). Augmented reality Sudoku solver. *ECS Transactions*, *107*(1), 2085. https://doi.org/10.1149/10701.2085ecst

[42] Valenzuela, E., Schaa, H., Barriga, N. A., & Patow, G. (2025). Using search algorithm statistics for assessing maze and puzzle difficulty. *Entertainment Computing*, *53*, 100925. https://doi.org/10.1016/j.entcom.2025.100925

[43] Behrens, T., Räuker, M., Kalbfleisch, M., & Jäkel, F. (2023). Flexible use of tactics in Sudoku. *Thinking & Reasoning*, *29*(4), 488–530. https://doi.org/10.1080/13546783.2022.2091040

[44] Maria Jeyaseeli, J., Lau, G. C., Shiu, W. C., & Arumugam, S. (2023). Sudoku number of graphs. *AKCE International Journal of Graphs and Combinatorics*, *20*(2), 209–216. https://doi.org/10.1080/09728600.2023.2218917

[45] Hasanah, N. A., Atikah, L., Herumurti, D., & Yunanto, A. A. (2020). A comparative study: Ant colony optimization algorithm and backtracking algorithm for Sudoku game. In *2020 International Seminar on Application for Technology of Information and Communication*, 548–553. https://doi.org/10.1109/iSemantic50169.2020.9234267

[46] Wang, C., Sun, B., Du, K.-J., Li, J.-Y., Zhan, Z.-H., Jeon, S.-W., …, & Zhang, J. (2024). A novel evolutionary algorithm with column and sub-block local search for *Sudoku* puzzles. *IEEE Transactions on Games*, *16*(1), 162–172. https://doi.org/10.1109/TG.2023.3236490

[47] Pathak, N., & Kumar, R. (2023). Entropy guided evolutionary search for solving Sudoku. *Progress in Artificial Intelligence*, *12*(1), 61–76. https://doi.org/10.1007/s13748-023-00297-7

[48] Lloyd, H., Crossley, M., Sinclair, M., & Amos, M. (2022). J-pop: Japanese puzzles as optimization problems. *IEEE Transactions on Games*, *14*(3), 391–402. https://doi.org/10.1109/TG.2021.3081817

[49] Lin, H.-H., & Wu, I.-C. (2010). Solving the minimum Sudoku problem. In *2010 International Conference on Technologies and Applications of Artificial Intelligence*, 456–461. https://doi.org/10.1109/TAAI.2010.77

[50] Kitsuwan, N., Pavarangkoon, P., Widiyanto, H. M., & Oki, E. (2020). Dynamic load balancing with learning model for Sudoku solving system. *Digital Communications and Networks*, *6*(1), 108–114. https://doi.org/10.1016/j.dcan.2019.03.002

[51] Cooper, J., & Kirkpatrick, A. (2014). Critical sets for Sudoku and general graph colorings. *Discrete Mathematics*, *315–316*, 112–119. https://doi.org/10.1016/j.disc.2013.10.011

[52] Mantere, T., & Koljonen, J. (2008). Solving and analyzing Sudokus with cultural algorithms. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 4053–4060. https://doi.org/10.1109/CEC.2008.4631350

[53] Musliu, N., & Winter, F. (2017). A hybrid approach for the Sudoku problem: Using constraint programming in iterated local search. *IEEE Intelligent Systems*, *32*(2), 52–62. https://doi.org/10.1109/MIS.2017.29

[54] Pacurib, J. A., Seno, G. M. M., & Yusiong, J. P. T. (2009). Solving Sudoku puzzles using improved artificial bee colony algorithm. In *2009 Fourth International Conference on Innovative Computing, Information and Control*, 885–888. https://doi.org/10.1109/ICICIC.2009.334

How to Cite: Thangamani, R., & Regulagedda, P. (2026). A Computational Complexity-Based Framework for Predicting the Difficulty of Sudoku Puzzles Using CSP-Based Symbolic Modeling and Constraint Metrics. *Journal of Computational and Cognitive Engineering*, *5*(1), 115–132. https://doi.org/10.47852/bonviewJCCE52026137