





RULE-BASED EXPLAINING MODULE: ENHANCING THE INTERPRETABILITY OF RECURRENT RELATIONAL NETWORK IN SUDOKU SOLVING

Pimpa Cheewaprabkakit ^{1,2}, Timothy K. Shih ^{2,*}, Timothy Lau²,
Yu-Cheng Lin ³ and Chih-Yang Lin ⁴

¹Department of Information Technology, Asia-Pacific International University, Saraburi, Thailand

²Department of Computer Science and Information Engineering, National Central University,
Taoyuan, Taiwan

³Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan

⁴Department of Mechanical Engineering, National Central University, Taoyuan, Taiwan

*Corresponding author: Timothy K. Shih (timothykshih@gmail.com)

Abstract Computer vision has gained significant attention in the field of information technology due to its widespread application that addresses real-world challenges, surpassing human intelligence in tasks such as image recognition, classification, natural language processing, and even game playing. Sudoku, a challenging puzzle that has captivated many people, exhibits a complexity that has attracted researchers to leverage deep learning techniques for its solution. However, the reliance on black-box neural networks has raised concerns about transparency and explainability. In response to this challenge, we present the Rule-based Explaining Module (REM), which is designed to provide explanations of the decision-making processes using Recurrent Relational Networks (RRN). Our proposed methodology is to bridge the gap between complex RRN models and human understanding by unveiling the specific rules applied by the model at each stage of the Sudoku puzzle solving process. Evaluating REM on the Minimum Sudoku dataset, we achieved an accuracy of over 98.00%.

Keywords: rule-based explaining module, recurrent relational network, Sudoku puzzle solving, machine learning.

1. Introduction

Sudoku is one of the most popular intellectual puzzle games [26] that involves logical thinking to fill in numbers. It comprises a 9×9 grid, forming a numerical puzzle with nine rows and nine columns, totalling 81 cells. The grid is further divided into nine 3×3 subgrids, referred to as blocks, each containing nine cells. To initiate the game, a set of given numbers is provided as hints. These hints are placed in some of the cells of the Sudoku puzzle, providing clues to help the player solve the puzzle. In most cases, the more cells that are given, the easier the puzzle tends to be. Currently, to the best of our knowledge, the fewest clues required for a proper Sudoku puzzle is 17. This means that the most challenging Sudoku puzzles now are those with only 17 known cells. The goal is to fill the empty cells with the numbers 1 through 9, ensuring that each number appears only once in each row, column, and block [28]. An example of a Sudoku puzzle and its solution is shown in Fig. 1.

5	3		7					
6			1	9	5			
	9	8					6	
8			6					3
4			8	3				1
7			2					6
	6				2	8		
			4	1	9			5
			8			7	9	

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Fig. 1. Sudoku puzzle and its solution.

The rapidly evolving realm of computer vision has increasing in various aspects of our daily lives, encompassing domain such as image recognition [19], language translation [25], and critical medical applications like X-ray image analysis for disease diagnosis [10, 27], and game playing [8]. The challenging of Sudoku puzzle has attracted researchers to leverage deep learning techniques for its solution. The fascination lies not only in the puzzles' complexity but also in the diverse strategies required for their solution. Traditional rule-based methods have been prevalent, employing strategies such as elimination, naked singles, and hidden singles. The advent of deep learning has ushered in a revolution of puzzle-solving, introducing adaptive and data-driven approaches to tackle Sudoku's complexities. Despite the remarkable capabilities of deep learning, the reliance on black-box nature of neural network has raised concerns about inner workings and transparency of their decision-making processes, particularly in contexts where machine learning applications make critical decisions. Enhancing the transparency of black-box neural networks becomes particularly crucial in applications requiring abstract reasoning about objects and their interactions, enabling audiences to comprehend the rationale behind the decision process of machine learning. One direct method to achieve this transparency is through the addition of explanations [21]. Existing explanation methods for specific applications such as tracking feature extraction in image recognition to visualize the interpretation of input data [12]. Furthermore, logical methods that integrate logical reasoning into neural networks have been proposed to enhance interpretability throughout the entire process [7, 24]. However, these logical methods may face challenges in generalizing to new data or situations, often relying on hand-crafted rules or assumptions about the data. A similar approach, the expert system, is rule-based [4] but demands a substantial amount of knowledge to be encoded in its rules. This process can be time-consuming and expensive, particularly in complex domains. Macha et al. introduced RuleXAI [13], a tool designed to enhancing explainability in machine learning models. While currently limited to classification, regression, and survival analysis, RuleXAI leverages rule-based explanations and feature relevance to make

models more understandable. However, it may not be perfectly accurate for all model types, especially those with complex, black-box in neural network.

This study introduces the Rule-based Explaining Module (REM), a specialized tool designed to unveil the specific rules applied by the model at each stage of the puzzle-solving process. Therefore, the main contribution of our study is summarized as follows:

1. We present the Rule-based Explaining Module (REM), designed to offer comprehensive, step-by-step explanations of the decision-making processes employed by Recurrent Relational Network (RRN) in Sudoku puzzle solving.
2. We conducted experiments using the Minimum Sudoku and 1 million Sudoku games datasets. The results demonstrated that our model significantly contributes to the transparency and interpretability of the Sudoku solving process.

The remainder of this paper is structured as follows: Section 2 provides a review of related work, Section 3 introduces the proposed Rule-based Explaining module for solving Sudoku, and Sections 4 and 5 present experimental results and conclusions, respectively.

2. Related works

Sudoku is a wildly popular logic-based puzzle game, has captivated individuals of all ages for many years. Its deceptively simple rules and endless variations have sparked a worldwide fascination. The challenge of solving Sudoku puzzles lies in their ability to test both logical reasoning and strategic thinking, especially for more difficult puzzles that captivating players with their intricate patterns and hidden clues.

Over the years, various techniques have been explored for solving Sudoku puzzle. Classical methods like backtracking [20], constraint propagation [14], and genetic algorithms [11] have shed light on Sudoku solving strategies. For instance, the pencil-and-paper method, also known as the human solver approach, efficiently solves easier puzzles but faces difficulties with more challenging ones, especially in the absence of clear clues. In contrast, backtracking, though it guarantees a solution for every valid puzzle, is considerably slower [16]. Subsequently, a hybrid method for solving Sudoku puzzles, integrating traditional backtracking algorithms with pencil-and-paper techniques was introduced [26]. This approach initially utilizes pencil-and-paper strategies, followed by applying backtracking to specific sub-grids, and concludes with pencil-and-paper methods on the remaining cells. This method is designed to improve puzzle-solving efficiency. However, its complexity and computational demands could be a drawback. The integration of algorithmic and intuitive strategies might lead to redundant operations and increasing the time required to solve complex Sudoku puzzles. Musliu and Winter have integrated the structured approach of constraint programming with the iterative nature of local search methods in a hybrid solution [14]. This method leverages the strengths of both: the proficiency of constraint programming in solving constraint

satisfaction problems and the effectiveness of iterated local search in optimization tasks. However, a primary challenge arises in balancing the systematic nature of constraint programming with the adaptive strategy of local search. This balancing act could present difficulties in efficiently finding solutions, especially in the context of complex Sudoku puzzles. Das et al. present an evolutionary algorithm that employs genetic operators, such as crossover and mutation, to generate new candidate solutions [3]. This algorithm may require extensive computational resources and time to converge on a solution. Gaddam et al. propose a method for solving Sudoku puzzles using a combination of deep learning and image processing techniques [6]. This method first utilizes image processing techniques to extract the Sudoku grid from an image and then employs a deep learning model to solve the puzzle. It demonstrates the potential of deep learning for solving Sudoku puzzles. However, the accuracy of this method is dependent on the quality of the input image. If the image is blurry or distorted, the accuracy of the deep learning model may be compromised. These techniques, while laying the foundation for understanding the problem, often lacked the flexibility, solution explanation, and adaptability needed to tackle complex puzzles. Björnsson et al., introduced a search-based approach to generate explainable solutions to Sudoku puzzles [1]. This method involves modelling the perceived human mental effort of using different familiar Sudoku-solving techniques. This model serves as guidance for a search algorithm to identify the correct solutions and present them in a way that is easily understandable to human solvers. However, the method's dependence on a potentially inaccurate model of human mental effort could result in explanations that are not entirely accurate. Another approach, Demystify, introduced by Espasa et al., provides step-by-step explanations for solving various pen-and-paper puzzles, including Sudoku [5]. It utilizes Minimal Unsatisfiable Subsets (MUSes) to solve puzzles through logical deduction, identifying essential puzzle components for progress. While Demystify effectively explains puzzle solutions, it requires human input in the form of high-level logical descriptions. Additionally, its applicability may not be suitable for solving all types of pen-and-paper puzzles. Bogaerts et al. provide step-by-step explanations for constraint satisfaction problems (CSPs), focusing on logic grid puzzles as a specific instance of CSPs [2]. They propose a framework for generating step-wise explanations of the inference steps taken during puzzle-solving. However, this approach is particularly reliant on the availability of a formal rule representation for the CSP domain. Without a well-defined set of rules, the framework may struggle to generate meaningful explanations.

The introduction of neural networks, particularly Recurrent Relational Networks (RRNs), revolutionized the landscape of Sudoku solving. RRNs [17], a type of neural network well-suited for learning long-range dependencies in data, proved adept at capturing the intricate relationships between cells in a Sudoku grid. While RRN's capability to learn from large datasets of Sudoku puzzles enabled them to achieve remarkable accuracy, consistently outperforming classical approaches, their black-box nature makes

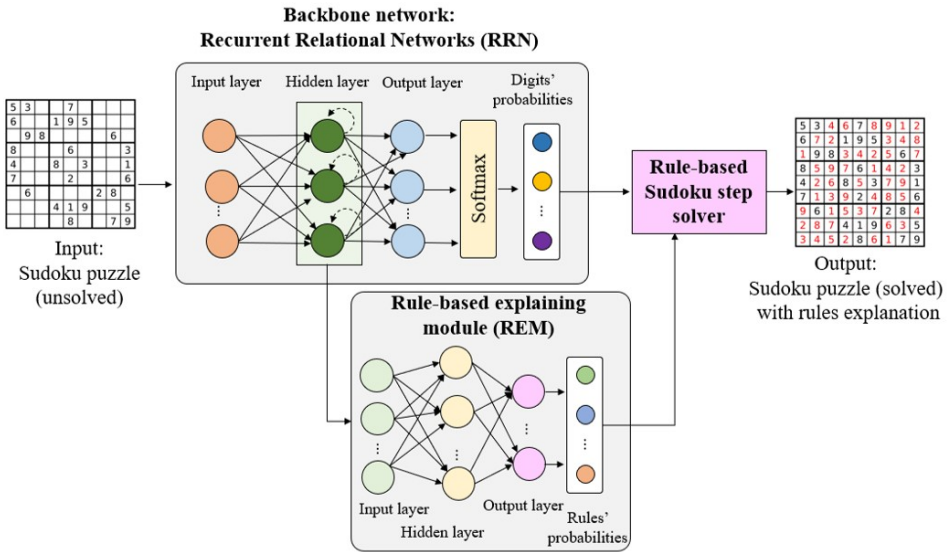


Fig. 2. Overview of the framework.

it challenging to comprehend their decision-making processes. Palm et al. introduced an RRN-based Sudoku solver that utilized a convolutional neural network (CNN) for feature extraction and an RRN for capturing relational dependencies [17]. However, the adoption of neural networks in Sudoku solving has raised concerns about transparency and interpretability, prompting the exploration of explain ability modules.

To address the lack of transparency in RRN-based Sudoku solving, we leveraged rule-based explanation techniques [21], inspired by prior research demonstrating their high accuracy. This integration enhances transparency and interpretability by generating human-readable rules that unveil the model’s decision-making process, these rules offer a more accessible way to understand the process compared to examining the raw model parameters.

3. Proposed method

Our proposed architecture incorporates the Recurrent Relational Network (RRN) [17] and Rule-based Explaining Module (REM), aiming to provide comprehensive, step-by-step explanations of the decision-making procedures in the context of Sudoku puzzle solving as shown in Fig. 2. The process begins by inputting an unsolved Sudoku puzzle

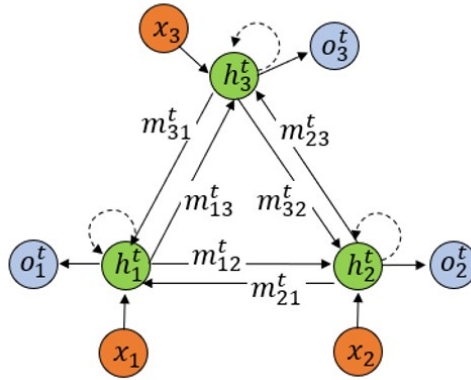


Fig. 3. A Recurrent Relational Network structure with 3 nodes.

zle into the backbone network, the Recurrent Relational Network (RRN). Each input node represents a feature vector (orange circles) corresponding to an individual Sudoku cell. Subsequently, a multi-layer perceptron (MLP) captures patterns and dependencies within the data. Recurrent computation updates all relevant information for each hidden state (green circles). The RRN then outputs probabilities for digits, representing the possible candidate digits for each cell. Following this, all hidden states (green circles) from the RRN are forwarded to the Rule-based Explaining Module (REM) using a multi-layer perceptron. This perceptron maps the hidden states to rules and outputs the probability of selected rules, offering explanations for the decision-making process. Finally, the Rule-based Sudoku step solver module receives output from both the RRN and REM modules. It identifies conditions that trigger specific rules and updates the knowledge base accordingly. This iterative process continues until the Sudoku puzzle is solved, ultimately providing both the solution and explanations for the decision-making steps involved.

3.1. Recurrent relational network (RRN)

RNN is a type of artificial neural network designed to capture long-range dependencies in sequential data. It is able to do this by learning to pass messages between nodes in a graph, which represents the relationships between the elements of the data. RRN is a powerful tool for a variety of tasks, including natural language processing, machine translation, and question answering. The RRN backbone consists of four main components, which are data input, message passing, node hidden state, and the output result as shown in Fig. 3.

3.1.1. Data input

In the context of Sudoku puzzles, there are input nodes, denoted as $i = 1, 2, \dots, 81$, each corresponding to a cell in the Sudoku grid. Each node i possesses an input x_i , representing the feature vector at that specific node.

3.1.2. Message passing

Message passing is responsible for communicating information between nodes. Each node sends a message to each of its neighbours at each iteration of the RRN. The message is a vector of numbers calculated based on the node's current state and its relationship to its neighbour. At each time step t , each node processes a hidden state vector h_i^t . During this process, each node sends a message m_{ij}^t from node i to node j at time step t , where node j represents a neighbouring node, utilizing message function f as illustrated in formula (1)

$$m_{ij}^t = f(h_i^{t-1}, h_j^{t-1}). \quad (1)$$

In Recurrent Relational Network (RRN), the message function f is implemented as a multiple-layer perceptron (MLP), enabling the network to learn the most effective types of messages to send for each situation. To incorporate all relevant information, each node must process all incoming messages, which are then summed together using formula (2). The combination of the MLP and the summation of messages enables RRN to learn complex patterns of communication and information exchange, making them powerful tools for solving tasks that require relational reasoning:

$$m_j^t = \sum_{i \in N(j)} m_{ij}^t, \quad (2)$$

where $N(j)$ represents all neighbouring nodes of node j , comprising nodes in the same row, column, and block as node j . Consequently, messages are currently computed for each node, allowing the model to progress to the next step in updating the network.

3.1.3. Recurrent nodes updates

Recurrent nodes are responsible for storing and updating the network's state, which represents the network's current understanding of the data. The state of a recurrent node is updated based on the messages it receives from its neighbours and the node's own internal state. The updates of recurrent nodes are key to the RRNs' ability to learn long-range dependencies in data. By repeatedly updating the state of each node based on the received messages, RRN can learn to capture the relationships between elements of the data that are separated by long distances in the input sequence. The formula for updating the state of a recurrent node is illustrated in (3):

$$h_j^t = g(h_j^{t-1}, x_j, m_j^t), \quad (3)$$

where g represents the node update function, functioning as a multiple-layer perceptron taking as input the hidden state from the previous iteration h_i^{t-1} , the feature vector of

input information x_j , and the message m_{ij}^t , the g function is trained to execute updates for the hidden state.

3.1.4. The output

After updating the hidden state, we can obtain the output at step t for node i by applying formula (4):

$$O_i^t = k(h_i^t), \quad (4)$$

where k denotes the output function, a multiple-layer perceptron trained to decode the hidden state into the output digit for the Sudoku. It converts the hidden state into output probabilities for a total of 10 different digits using the softmax function. The cross-entropy loss function, defined in formula (5), is used to optimize the model's performance during training. The target digits, represented by $y = y_1, y_2, y_3, \dots, y_81$ denote the correct digit at position i at step t .

$$l^t = - \sum_{i=1}^I \log O_i^t[y_i]. \quad (5)$$

3.2. Rule-based explaining module (REM)

Rule-based explaining is a technique in artificial intelligence employed to explain the reasoning behind decision-making by identifying the conditions that triggered specific rules and the conclusions reached by those rules. In our backbone network utilizing RRN, the message passing in the RRN network encompasses valuable information, including node relationships with its neighbours, which is highly valuable for examination. To extract the explanations from the message, we incorporate a multiple-layer perceptron that learns the rules from the hidden state after message passing and recurrent updating, as defined by formula (6).

$$R_i^t = r(h^t), \quad (6)$$

where R_i^t represents the output of the selected rules used to solve the Sudoku puzzle at step t . The hidden states h^t encompasses all of the RRN graph's hidden states, and the function r is a multiple-layer perceptron that maps the hidden state after message passing to rules at step t . The variable i represents the number of rules, where $i = 1, 2, \dots, n$. The selection of rules is guided by the tasks at hand. For Sudoku puzzles, we employ rules proposed by Hobiger [9] and Riley [22]. From their set of rules, we selected six rules for our experiments as they effectively solve the majority of the Sudoku puzzles in our dataset. Sudoku solving is divided into steps, with each step corresponding to filling in a single digit in the puzzle. Typically, multiple rules can be employed to determine a single digit. Consequently, the rule identification process generates more than one rule at each step. In this scenario, each Sudoku solving step can yield up to six different rule outputs.

4 5 9	2	8	4 5 6 9	1 5 7	4 6 9	3 6 4 6 9
4 5 9	1	6	4 5 9	8 3	4 2 9	7 4 2 9
4 7 9	4 7	4 9	4 6 9	2	4 8	5 1
1	3	7	2	9	5 8	4 5 6 8 6 4 5 6 8
4 5 6 8 9	4 5 6 8 9	4 5 9	7	3	1 5 8	4 2 5 6 8 9 1 2 6 4 5 6 8 9
5 8 9	5 8	2 5 9	1 5	4	6	3 1 2 8 9 7
2	9	1 3 4 5	4 5	7	1 4 5	5 6 3 6 5 8 8
7 5 7	5 5	5 3	8	6	2 5 9	1 4 2 3 5 9
4 5 6 8	4 5 6 8	4 5	3	1 5	1 2 4 5 9	7 2 6 5 8 9 8 9

Fig. 4. An example of Hidden Single.

3.3. Sudoku solving rules

We selected six rules from the rules proposed by Hobiger [9] and Riley [22]. These rules include Hidden Single, Naked Single, Locked Candidates Type 1, Locked Candidates Type 2, Naked Pair, and Hidden Pair. Although there are numerous rules beyond the six we chose, our decision was based on the observation that these specific rules already successfully solved over 98.00% of the most challenging Sudoku puzzles. In our study, our primary focus is on explaining the Sudoku solving process rather than improving accuracy. As such, we believe that utilizing these six rules is sufficient for our purposes.

3.3.1. Rule 1: Hidden single

A Hidden Single occurs when there is only one possible candidate number for a cell within a row, column, or 3×3 block, but that candidate number does not appear in any other cell within that row, column, or block. An example of the hidden single rule is presented in Fig. 4.

Examining row 3 (r3) in Fig. 4, it becomes evident that the cell at row 3, column 4 (r3c4) marked with a green 6, is the sole occurrence of the digit 6 within row 3. Consequently, we can confidently assign the digit 6 to cell r3c4 by eliminating other digit candidates.

3.3.2. Rule 2: Naked single

A Naked Single occurs when there is only one possible candidate number in a row, column, or 3×3 block that can contain a specific digit. An example of the naked single rule is presented in Fig. 5.

4	1	2	7	3	6	5	8	9
<small>3</small>	<small>4</small>	<small>5</small>	<small>2</small>	<small>2</small>	<small>2</small>	<small>1</small>	<small>2</small>	<small>6</small>
<small>7</small>	<small>9</small>	<small>7</small>	<small>9</small>	<small>9</small>	<small>4</small>	<small>8</small>	<small>4</small>	<small>5</small>
5	6	8	4	1	4	3	7	4
<small>3</small>	<small>6</small>	<small>4</small>	<small>3</small>	<small>8</small>	<small>5</small>	<small>2</small>	<small>1</small>	<small>3</small>
<small>9</small>	<small>9</small>	<small>9</small>	<small>9</small>	<small>7</small>	<small>7</small>	<small>6</small>	<small>4</small>	<small>9</small>
1	<small>4</small>	<small>5</small>	<small>4</small>	<small>5</small>	<small>4</small>	<small>6</small>	<small>4</small>	<small>6</small>
<small>2</small>	<small>3</small>	<small>8</small>	<small>7</small>	<small>1</small>	<small>2</small>	<small>3</small>	<small>1</small>	<small>2</small>
<small>2</small>	<small>3</small>	<small>8</small>	<small>7</small>	<small>1</small>	<small>2</small>	<small>3</small>	<small>4</small>	<small>5</small>
<small>2</small>	<small>9</small>	3	<small>1</small>	<small>4</small>	<small>9</small>	7	8	6
<small>4</small>	<small>2</small>	<small>1</small>	<small>4</small>	<small>6</small>	<small>1</small>	<small>2</small>	<small>3</small>	<small>2</small>
<small>8</small>	<small>4</small>	<small>4</small>	<small>6</small>	<small>4</small>	<small>6</small>	<small>4</small>	<small>5</small>	<small>7</small>
<small>2</small>	<small>6</small>	<small>2</small>	<small>5</small>	<small>5</small>	<small>6</small>	<small>9</small>	<small>2</small>	<small>3</small>
<small>7</small>	<small>7</small>	<small>7</small>	<small>9</small>	<small>2</small>	<small>6</small>	8	4	1

Fig. 5. An example of Naked Single.

Examining the cell at r6c7 in Fig. 5, it has only one possible digit candidate, which is 6. Therefore, we can assign the digit 6 to that cell.

3.3.3. Rule 3: Locked candidates Type 1

The third and fourth rules are more advanced compared to the first two. They employ an indirect method for eliminating potential candidates from a cell. In fact, all rules, except for the first two, are utilized to eliminate possible candidates, eventually leading to the condition where the first two rules can be applied to fill in a digit and complete a step. Locked Candidates Type 1 occurs when all candidates of a specific digit within a block are confined to a row or column, that digit cannot appear outside of that block in that row or column. Fig. 6 illustrates an example of Locked Candidates Type 1.

Observing block 1 (b1) in Fig. 6, digit 5 only appears in row 3 (r3). Consequently, there should not be another instance of digit 5 in row 3 outside of block 1. Therefore, the candidate 5 in cell r3c7 can be eliminated.

3.3.4. Rule 4: Locked candidates Type 2

Locked Candidates Type 2 is the opposite of Locked Candidates Type 1. It occurs when, in a row (or column), all candidates of a specific digit are confined to one block, allowing the elimination of that candidate from all other cells in that block. Fig. 7 provides an example of Locked Candidates Type 2.

Examining row 2 (r2) in Fig. 7, it is observed that all candidate positions for the digit 7 appear only within block 1 (b1). Consequently, the digit 7 must be present in

9	8	4						
		2	5				4	
		1	9		4			2
	6		9	7	2	3		
		3	6	2				
2		9	3	5	6	1		
1	9	5	7	6	8	4	2	3
4	2	7	3	5	1	8	9	6
6	3	8			9	7	5	1

Fig. 6. An example of Locked Candidates Type 1.

3	1	8			5	4		6
		7			3	8	1	
		6		8		5		3
8	6	4	9	5	2	1	3	7
1	2	3	4	7	6	9	5	8
7	9	5	3	1	8	2	6	4
		3		5		7	8	
						7	3	5
				3	9	6	4	1

Fig. 7. An example of Locked Candidates Type 2.

row 2 within block 1. As a result, the digit 7 candidates located outside row 2 in block 1 can be eliminated.

3.3.5. Rule 5: Naked pair

A Naked Pair occurs when there are exactly two candidate numbers for a cell within a row, column, or 3×3 block, and those two candidate numbers also appear together in

7	5 6	1 6	8	4 9	1 2 5	3	2 5
9	2 8	1 3 5	4 7	4 7	6		
4	5 3 1 3	2 6 7	1 5	8 9			
6	4 2	7 8 3	9 5 1				
3	9 7	4 5 1	6 2 8				
8	1 5	6 9 2	3 4 7 4 7				
2	7 8	4 5 1 6	7 8 9 3				
1	7	3 9 7	4 5 7 8 6 4 5				
5	3 6 3 3 2	4	2 2 1 2				

Fig. 8. An example of Naked Pair.

another cell within the same row, column, or block. This means that those two candidate numbers must be in these two cells, and cannot be appear elsewhere in that row, column, or block. Fig. 8 provides an example of naked pair.

Examining row 8 (r8), candidates 3 and 9 form a pair within a cell. Consequently, the candidate 3 in cell r8c2, (row 8 in this case), can be eliminated

3.3.6. Rule 6: Hidden pair

A Hidden Pair occurs when there are two candidate numbers for a cell within a row, column, or 3x3 block, and these two candidate numbers also appear together in another cell within that same row, column, or block. However, that other cell is already filled with another number. Consequently, all other candidates in those two cells can be eliminated. Fig. 9 provides an example of hidden pair.

Examining column 9 (c9), we observe a pair of candidate digits, 1 and 9, located in cells row 5 column 9 (r5c9) and row 7 column 9 (r7c9). Since 1 and 9 must occupy either of these two cells, any other candidate digits, such as the possible candidate 6 in r5c9, can be eliminated.

3.4. Rule-based sudoku step solver

Many Sudoku solving programs commonly eliminate candidates based on the given puzzle and search through all possibilities to identify the correct candidate digit. In contrast, our Sudoku solver takes a distinct approach. It solves Sudoku in a stepwise, rule-based, using specific rules in each action. The flow chart depicting our approach is presented

^{5 6}	4	9	1	3	2	^{7 6}	⁵	^{7 8}	^{7 8}
^{5 6}	8	1	4	7	9	^{2 3}	^{2 3}	²	⁶
	3	2	7	6	8	5	9	1	4
^{4 2}	9	6	³	5	1	8	⁴	^{2 3}	²
^{1 4}	7	5	³	2	8	^{1 3}	^{4 6}	³	^{1 6}
^{1 2}	3	8	^{7 9}	4	6	^{1 2}	²		5
	8	5	3	2	6	7	¹	⁴	^{1 6}
	7	1	2	8	9	4	5	6	3
	9	6	4	5	1	3	²	²	²

Fig. 9. An example of Hidden Pair.

in Fig. 10. The initial step involves assigning possible candidate digits to each cell in the provided Sudoku puzzle. Subsequently, we apply Sudoku rules by examining all potential candidate digits to identify any patterns that conform the established rules. As mentioned in the previous section, the first two rules form the foundation for solving Sudoku puzzles, and we can observe that the Sudoku puzzle can be solved by applying these two rules alone. Following the assessment of the first two rules, the remaining rules are examined one by one to determine if any patterns meet the criteria of each rule. If a pattern satisfying a rule is discovered, we revisit all the rules to ensure no other patterns exist. This process iterates until the step is resolved by either the first or second rule. Conversely, if no pattern satisfying the rule is found, that step cannot be solved, and the solver will cease attempting to solve it. In other words, the Sudoku puzzle cannot be solved within the framework of these six rules.

4. Experimental results

We conducted our experiments using a DGX station with a Nvidia V100 GPU with 32 GB of GPU RAM, employing a batch size of 128 and a learning rate of 2e-5. The training process involved 32 steps because the model stabilized at this step and took 5 days. The total number of trainable parameters was 518006. For testing, we used both the Minimum Sudoku dataset from Gordon Royle and 1 million Sudoku games (1M Sudoku) dataset [18]. Additionally, the Minimum Sudoku dataset [23] was used for training, divided into an 80% training set, a 10% validation set, and a 10% testing set.

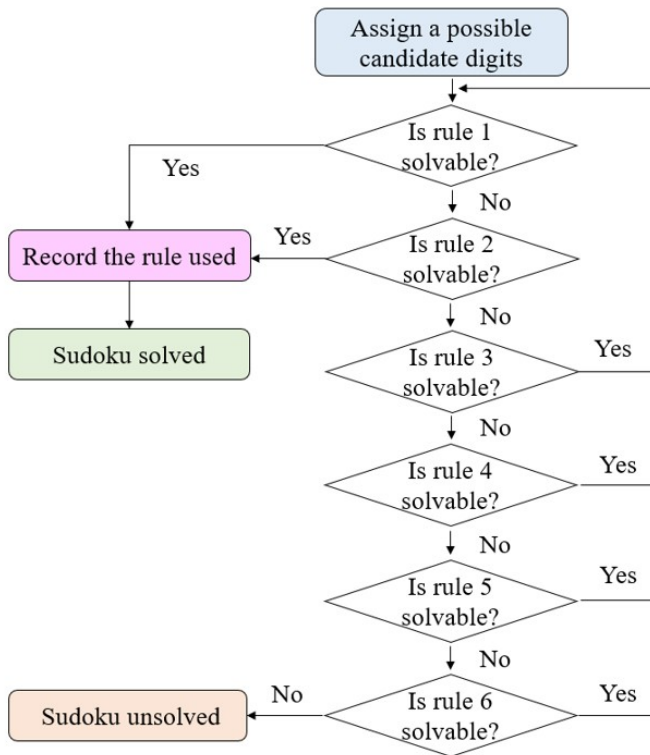


Fig. 10. Flow chart of the Rule-based Sudoku step solver.

Our method demonstrated the ability to solve Sudoku puzzles at a speed of 20 puzzles per second.

4.1. The Minimum Sudoku dataset from Gordon Royle

The Minimum Sudoku dataset [23] comprises 49 151 puzzles, each assigned the difficulty level of 17 given numbers. These puzzles are generated using a backtracking algorithm and is guaranteed to have a unique solution. Leveraging the complexity of these Sudoku puzzles with 17 given numbers during training enables our model to effectively handle a wide range of puzzles, from easy to difficulty levels.

We investigated the model's ability to apply six Sudoku-solving rules independently: Naked Single, Hidden Single, Locked Candidate Type 1, Locked Candidate Type 2, Naked Pair, and Hidden Pair. By testing the model on the Minimum Sudoku dataset,

we aimed to determine how effectively the model can apply each rule to solve Sudoku puzzles by applying 64 steps. The results are correctly verified using the Norvig Sudoku solver [15].

The results of solving the Sudoku puzzle are presented in Tab. 1. This table illustrates that, initially, the model assigns possible number candidates for the puzzle, enabling Sudoku solution with an accuracy of 72.32%. Subsequently, the model employs Rule 1, achieving a Sudoku accuracy of 99.00%. For cells in Sudoku puzzles that remain unsolved after Rule 1, the model applies Rule 2, achieving an accuracy of 98.98%. If any cells persistently resist resolution with Rule 2, the model turns to Rule 3, and so forth, up to Rule 6. The accuracy rates for Rules 3 to 6 are 98.62%, 98.60%, 98.79%, and 98.67%, respectively.

4.2. The 1 million Sudoku games (1M Sudoku) dataset

The 1M Sudoku dataset [18], available on Kaggle was developed by Kyubong Park. Using a computer program, he generated over a million Sudoku puzzles with their corresponding solutions. The dataset encompasses a variety of difficulty levels, ranging from easy to challenging. While several factors can influence a Sudoku puzzle’s difficulty, such as the pattern of given cells, the puzzle’s symmetry, and the existence of hidden singles or doubles, the number of given cells is a crucial factor. Sudoku puzzles with the fewest given cells are generally considered to be more difficult. The majority of puzzles in this dataset are of medium difficulty. We conducted experiments with our model using the 1M Sudoku dataset, performing 64 steps on each puzzle.

The results of solving these Sudoku puzzles are presented in Tab. 2. The contents of this table demonstrates that, in the initial stage, the model assigns possible number candidates for the Sudoku puzzle, enabling a 95.41% success rate in solving Sudoku puzzles. Subsequently, the model employs Rule 1, achieving a perfect accuracy of 100%. For any remaining unsolved cells after applying Rule 1, the model employs Rule 2, also achieving a perfect accuracy of 100%. Since Rule 2 successfully solves all remaining

Tab. 1. The results of solving the Sudoku puzzle on the Minimum Sudoku dataset.

Accuracy (%)	Rules used	Description
72.32	-	Model assigns possible numbers
99.00	1	Naked Single
98.98	2	Hidden Single
98.62	3	Locked Candidate Type 1
98.60	4	Locked Candidate Type 2
98.79	5	Naked Pair
98.67	6	Hidden Pair

Tab. 2. The results of solving the Sudoku puzzle on the 1 million Sudoku games dataset.

Accuracy (%)	Rules used	Description
95.41	-	Model assigns possible numbers
100.00	1	Naked Single
100.00	2	Hidden Single
100.00	3	Locked Candidate Type 1
100.00	4	Locked Candidate Type 2
100.00	5	Naked Pair
100.00	6	Hidden Pair

cells, Rules 3 to 6 become unnecessary. Therefore, Rules 3 to 6 consistently exhibit 100% accuracy when applied. Refer to Tab. 4 for more details. Since our model was trained on the Minimum Sudoku dataset, renowned for its difficulty, it excels in solving Sudoku puzzles, achieving an outstanding 100% success rate.

4.3. Rule-based explanation

Our rule-based explaining module allows us to understand how the RRN model solves Sudoku puzzles by breaking it down step by step based on established rules and inferences. This is demonstrated through examples, such as inputting a Sudoku puzzle from the Minimum Sudoku dataset, where each cell's candidate number represents the probability of it being the correct answer. Fig. 11 depicts a graph showcasing solving accuracy at various steps ranging from 0 to 60 using the Minimum Sudoku dataset. Additionally, Fig. 12 displays a graph representing rule accuracy employed at different steps with the same dataset.

Tab. 3 provides a comprehensive analysis and interpretation of the rule accuracy applied at various steps in the Sudoku-solving process. In the initial stage, the model achieved its highest accuracy near step 32. The model employed rules 1 through 6 to solve the puzzle. This indicates that the Sudoku puzzle is significantly complex, requiring the use of more than two rules to achieve a solution.

In another instance, we utilized input data from the 1M Sudoku dataset. As depicted in Fig. 13, a graph illustrates solving accuracy at various steps, ranging from 0 to 60. Fig. 14 presents a graph illustrating rule accuracy at different steps, with the model achieving its highest accuracy around step 32. Accompanying these figures is Tab. 4, where rules 1 and 2 were employed to solve the puzzle, achieving 100% accuracy. This observation sheds light on why rules 3 to 6 consistently show 100% accuracy. The reason behind this is that the model does not anticipate the utilization of rules 3 to 6, resulting in their consistent correctness as unused rules.

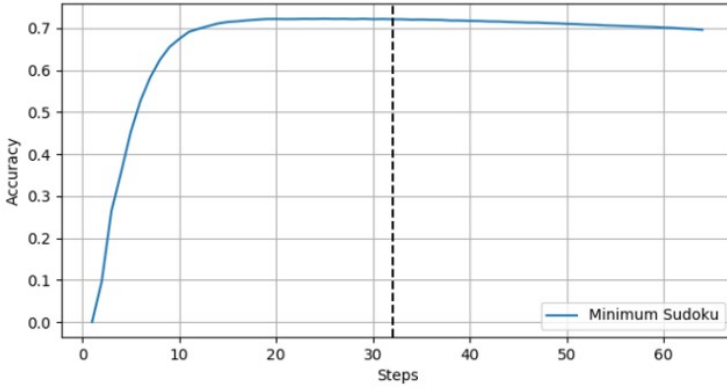


Fig. 11. A graph depicting solving accuracy at different steps using the Minimum Sudoku dataset.

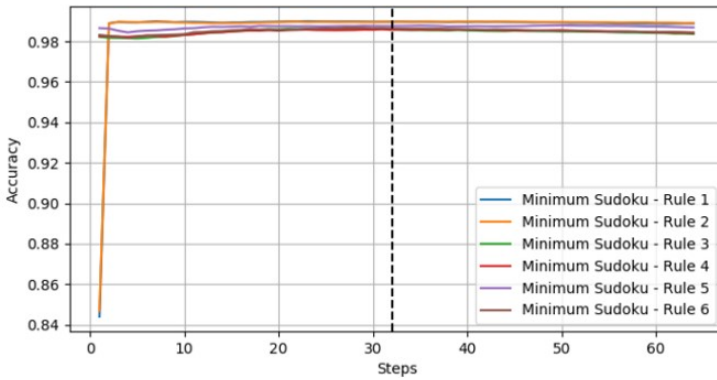


Fig. 12. Illustrating rule accuracy at different steps using the Minimum Sudoku dataset.

5. Conclusion

In this paper, we address concerns regarding the transparency and interpretability of machine learning applications, especially in critical decision-making domains. The opacity of neural networks, often labelled as black-boxes, has raised questions, particularly in Sudoku puzzle-solving scenarios. To tackle this challenge, we introduced the Rule-based Explaining Module (REM) as a tool to understand the complex decision-making processes of RRN during Sudoku puzzle-solving. While our REM has shown promise, there are opportunities for further exploration and improvement. Future research could explore broader applications of REM across diverse datasets. Additionally, extending

Tab. 3. The analysis and interpretation of rule accuracies across different steps in Minimum Sudoku dataset.

Steps (1-64)	1	16	32	48	64	Best Step/ accuracy (%)
Model assigns possible numbers	9.71	71.91	72.17	71.19	69.66	24/ 72.32
Rule 1	98.89	98.96	98.98	98.94	98.88	6/ 99.00
Rule 2	98.89	98.92	98.97	98.96	98.91	21/ 98.98
Rule 3	98.18	98.54	98.56	98.51	98.37	20/ 98.62
Rule 4	98.25	98.54	98.56	98.55	98.41	30/ 98.60
Rule 5	98.63	98.72	98.76	98.79	98.69	40/ 98.79
Rule 6	98.72	98.60	98.63	98.52	98.44	26/ 98.67

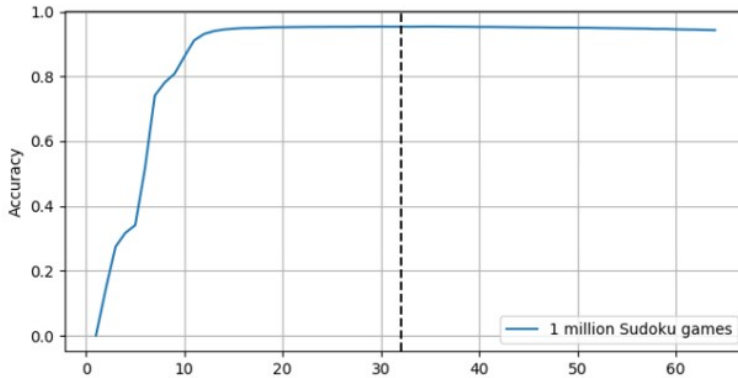


Fig. 13. An example of a graph depicting solving accuracy at different steps.

our approach to other puzzle types or complex decision-making tasks. The development of user-friendly interfaces and visualization techniques could facilitate the practical implementation of REM in real-world scenarios. This work represents a significant step in addressing transparency challenges posed by neural networks, offering a concrete solution in the form of the Rule-based Explaining Module. The success of our proposed method not only contributes to the field of explainable artificial intelligence but also paves the way for broader applications in various domains requiring interpretable decision-making.

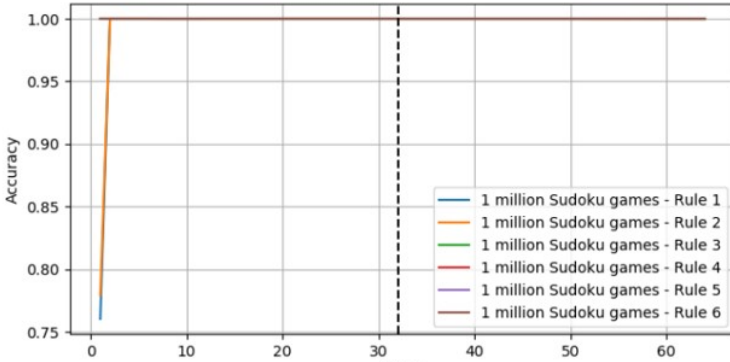


Fig. 14. Illustrating rule accuracy at different steps.

Tab. 4. An example of the analysis and interpretation of rule accuracies across different steps in the 1M Sudoku games dataset.

Steps (1-64)	1	16	32	48	64	Best Step/ accuracy [%]
Model assigns possible numbers	14.26	94.95	95.34	95.07	94.29	34/ 95.41
Rule 1	100.00	100.00	99.99	99.99	99.99	1/ 100.00
Rule 2	100.00	99.99	99.99	99.99	99.99	1/ 100.00
Rule 3	100.00	100.00	100.00	100.00	100.00	0/ 100.00
Rule 4	100.00	100.00	100.00	100.00	100.00	0/ 100.00
Rule 5	100.00	100.00	100.00	100.00	100.00	0/ 100.00
Rule 6	100.00	100.00	100.00	100.00	100.00	0/ 100.00

References

- [1] Y. Björnsson, S. Helgason, and A. Pálsson. Searching for explainable solutions in Sudoku. In: *2021 IEEE Conference on Games (CoG)*, pp. 1–8. Copenhagen, Denmark, 17-20 Aug 2021. doi:10.1109/CoG52621.2021.9618900.
- [2] B. Bogaerts, E. Gamba, and T. Guns. A framework for step-wise explaining how to solve constraint satisfaction problems. *Artificial Intelligence*, 300:103550, 2020. doi:10.1016/j.artint.2021.103550.
- [3] K. N. Das, S. K. Bhatia, S. Puri, and K. Deep. Solving Sudoku puzzle by evolutionary algorithm. In: *Proc. 21st Asian Technology Conference in Mathematics*. Mathematics and Technology, LLC, Pattaya, Thailand, 14-18 Dec 2016. https://atcm.mathandtech.org/EP2016/contributed/4052016_21261.pdf.
- [4] G. D. Engin, B. Aksoyer, M. Avdagic, D. Bozanli, U. Hanay, et al. Rule-based expert

- systems for supporting university students. *Procedia Computer Science*, 31:22–31, 2014. doi:<https://doi.org/10.1016/j.procs.2014.05.241>.
- [5] J. Espasa, I. P. Gent, R. Hoffmann, C. Jefferson, and A. M. Lynch. Using small MUSes to explain how to solve pen and paper puzzles. *ArXiv*, 2021. ArXiv:2104.15040. doi:10.48550/arXiv.2104.15040.
- [6] D. K. R. Gaddam, M. D. Ansari, and S. Vuppala. On Sudoku problem using deep learning and image processing technique. In: *Proc. 3rd Int. Conf. Communications and Cyber Physical Engineering (ICCCE) 2020*, vol. 698 of *Lecture Notes in Electrical Engineering*, pp. 1405–1417, 2020. doi:10.1007/978-981-15-7961-5_128.
- [7] O. Gerasimova, N. Severin, and I. Makarov. Comparative analysis of logic reasoning and graph neural networks for ontology-mediated query answering with a covering axiom. *IEEE Access*, 11:88074–88086, 2023. doi:10.1109/ACCESS.2023.3305272.
- [8] T. Guns, E. Gamba, M. Mulamba, I. Bleukx, S. Berden, et al. Sudoku assistant – an AI-powered app to help solve pen-and-paper Sudokus. In: *Proc. AAAI Conference on Artificial Intelligence*, p. 16440–16442. AAAI Press, 2023. doi:10.1609/aaai.v37i13.27072.
- [9] B. Hobiger. Sudoku for Java – HoDoKu, 2013. <https://sourceforge.net/projects/hodoku/>, [Accessed: 15 Oct, 2023].
- [10] A. Hussain, S. U. Amin, H. Lee, A. Khan, N. F. Khan, et al. An automated chest X-ray image analysis for Covid-19 and pneumonia diagnosis using deep ensemble strategy. *IEEE Access*, 11:97207–97220, 2023. doi:10.1109/ACCESS.2023.3312533.
- [11] B. Indriyono, N. Pamungkas, Z. Pratama, E. Mintorini, I. Dimentieva, et al. Comparative analysis of the performance testing results of the backtracking and genetics algorithm in solving Sudoku games. *International Journal of Artificial Intelligence and Robotics*, 5(1):29–35, 2023. doi:10.25139/ijair.v5i1.6501.
- [12] P. Linardatos, V. Papastefanopoulos, and S. B. Kotsiantis. Explainable AI: A review of machine learning interpretability methods. *Entropy*, 23(1):18, 2021. doi:10.3390/e23010018.
- [13] D. Macha, M. Kozielski, Ł. Wróbel, and M. Sikora. RuleXAI—A package for rule-based explanations of machine learning model. *SoftwareX*, 20:101209, 2022. doi:10.1016/j.softx.2022.101209.
- [14] N. Musliu and F. Winter. A hybrid approach for the Sudoku problem: Using constraint programming in iterated local search. *IEEE Intelligent Systems*, 32(2):52–62, 2017. doi:10.1109/MIS.2017.29.
- [15] P. Norvig. Solving every Sudoku puzzle, 15 Jan 2012. <https://norvig.com/sudoku.html>, [Accessed: 15 Oct, 2023].
- [16] E. Onokpasa, D. Bisandu, and D. Bakwa. A hybrid backtracking and pencil and paper Sudoku solver. *International Journal of Artificial Intelligence and Robotics*, 181(47):39–43, 2019. <https://dspace.unijos.edu.ng/jspui/handle/123456789/2769>.
- [17] R. B. Palm, U. Paquet, and O. Winther. Recurrent relational networks. In: *Advances in Neural Information Processing Systems 31 – Proc. 32nd Int. Conf. Neural Information Processing Systems (NeurIPS) 2018*, vol. 31 of *NIPS’18*, p. 3372–3382, 2018. <https://proceedings.neurips.cc/paper/2018/hash/b9f94c77652c9a76fc8a442748cd54bd-Abstract.html>.
- [18] K. Park. 1 million Sudoku games, 2017. <https://www.kaggle.com/datasets/bryanpark/sudoku>, [Accessed: 15 Oct, 2023].
- [19] X. Pengcheng, H. Zhenlin, Z. Liuqi, W. Ning, Z. Hanghang, et al. A realtime image recognition method of Power AI based on quadtree algorithm. In: *Proc. 2023 2nd Int. Conf. Innovation in Technology (INOCON)*, pp. 1–6. Bangalore, India, 3-5 Mar 2023. doi:10.1109/INOCON57975.2023.10101145.

- [20] M. Prabha, S. Radha, P. M. Priya, and B. S. Dhivya. Sudoku solver using minigrid based backtracking algorithm. *International Journal of Research in Engineering, Science and Management*, 5(6):138–140, 2022. <https://journal.ijresm.com/index.php/ijresm/article/view/2180>.
- [21] G. P. Reddy and Y. V. P. Kumar. Explainable AI (XAI): Explained. In: *Proc. 2023 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pp. 1–6. Vilnius, Lithuania, 27–27 Apr 2023. doi:10.1109/eStream59056.2023.10134984.
- [22] G. Riley. CLIPS rule based programming language code, jan 2016. <https://sourceforge.net/p/clipsrules/code/HEAD/tree/branches/63x/examples/sudoku/>, [Accessed: 15 Oct, 2023].
- [23] G. Royle. Good at Sudoku? Here’s some you’ll never complete. *The Conversation*, 12 Feb 2012. [Accessed: 15 Oct, 2023]. <https://theconversation.com/good-at-sudoku-heres-some-youll-never-complete-5234>.
- [24] S. Shi, H. Chen, W. Ma, J. Mao, M. Zhang, et al. Neural logic reasoning. In: *Proc. 29th ACM Int. Conf. Information & Knowledge Management (CIKM '20)*, p. 1365–1374. Boise, ID, USA, 21–25 Oct 2020. doi:10.1145/3340531.3411949.
- [25] Y. Singh, P. Kumar, S. Goel, P. Garg, T. Srivastava, et al. Anuvadak: Language system using machine learning techniques. In: *Proc. 2023 Int. Conf. Artificial Intelligence and Smart Communication (AISC)*, pp. 742–745. Greater Noida, India, 27–29 Jan 2023. doi:10.1109/AISC56616.2023.10085373.
- [26] A. A. Suha Binta Wadud and M. Abdullah-Al-Wadud. An improved hybrid method combining backtracking with pencil and paper for solving sudoku puzzles. In: *Proc. Int. Symp. Electrical, Electronics and Information Engineering (ISEEIE) 2021*, p. 438–441. Association for Computing Machinery, Seoul, Republic of Korea, 19–21 Feb 2021. doi:10.1145/3459104.3459176.
- [27] Y. Tian. Artificial intelligence image recognition method based on convolutional neural network algorithm. *IEEE Access*, 8:125731–125744, 2020. doi:10.1109/ACCESS.2020.3006097.
- [28] P.-S. T. P.-S. Tsai, T.-F. W. P.-S. Tsai, J.-Y. C. T.-F. Wu, and J.-F. H. J.-Y. Chen. Integrating of image processing and number recognition in Sudoku puzzle cards digitation. *Journal of Internet Technology*, 23(7):1573–1584, 2022. doi:10.53106/160792642022122307012.

