applied sciences

IMPACT FACTOR 2.5

CITESCORE 5.3

Article

Rearrangement of Single Atoms by Solving Assignment Problems via Convolutional Neural Network

Kanya Rattanamongkhonkun, Radom Pongvuthithum and Chulin Likasiri

MDPI

https://doi.org/10.3390/app14177877

applied sciences

MDPI

Article

Rearrangement of Single Atoms by Solving Assignment Problems via Convolutional Neural Network

Kanya Rattanamongkhonkun [1], Radom Pongvuthithum [1] and Chulin Likasiri [2,*]

[1] Department of Mechanical Engineering, Faculty of Engineering, Chiang Mai University, Chiang Mai 50200, Thailand; kanya.r@cmu.ac.th (K.R.); radom.p@cmu.ac.th (R.P.)
[2] Department of Mathematics, Faculty of Science, Chiang Mai University, Chiang Mai 50200, Thailand
* Correspondence: chulin.l@cmu.ac.th

Featured Application: This work provides a systematic way to produce a defect-free atom array of neutral atom platforms for quantum information processing.

Abstract: This paper aims to present an approach to address the atom rearrangement problem by developing Convolutional Neural Network (CNN) models. These models predict the coordinates for atom movements while ensuring collision-free transitions and filling all vacancies in the target array. The process begins with designing a cost function for the assignment problem that incorporates constraints to prevent collisions and guarantee vacancy filling. We then build and train CNN models using datasets for three different grid sizes: 10×10, 13×13, and 21×21. Our models achieve high accuracy in predicting atom positions, with individual position accuracies of 99.63%, 98.93%, and 97.24%, respectively, for the aforementioned grid sizes. Despite limitations in training larger models due to hardware constraints, our approach demonstrates significant improvements in speed and accuracy. The final section of the paper presents detailed accuracy results and calculation times for each model, highlighting the potential of CNN-based methods in optimizing atom rearrangement processes.

Keywords: atom arrangement; convolutional neural network; collision-free path; defect-free arrangement

check for updates

Citation: Rattanamongkhonkun, K.; Pongvuthithum, R.; Likasiri, C. Rearrangement of Single Atoms by Solving Assignment Problems via Convolutional Neural Network. *Appl. Sci.* **2024**, *14*, 7877. https://doi.org/10.3390/app14177877

Academic Editor: Antonio Pagliaro

Received: 2 August 2024
Revised: 23 August 2024
Accepted: 26 August 2024
Published: 4 September 2024

1. Introduction

Over the past decade, substantial progress has been made in the neutral atom platform, which is one of the platforms used to construct a quantum system whose applications exist in many quantum branches, for instance, quantum information processing in Nielsen [1], quantum metrology in Giovannetti [2], and quantum simulation in Georgescu [3]. Arrays of cold neutral atoms are necessary to gain control of single qubits. After trapping atoms, rearrangement of neutral atoms is therefore crucial, and there have been several recent impressive results.

Most research on the problems of rearranging atoms from any initial array to any target array can be categorized into physical methods and mathematical algorithms. One frequently used physical technique for atom rearrangement is the optical tweezer. In the most recent work of 2024, by Pichard et al. [4], after the trapping of single atoms up to a 2000 site, atom-by-atom rearrangement is demonstrated with very few defects up to an 828-atom target array using this optical tweezer technique. While the task of trapping single atoms is crucial [5–7], creating defect-free atomic arrays is just as important as they have been demonstrated to be a platform for quantum simulations and quantum computations. There are methods in physics that focus on physically moving the atoms to defect-free target arrays [8] and more theoretical methods that try to find the best way to move the atoms under some specified conditions. The development of theoretical solutions will benefit those who try to physically move the atoms to the target arrays.

The following are instances of work using mathematical algorithms to choose the path of atom. In 2016, Lee et al. [9] proposed and demonstrated 3D rearrangements of single atoms. They considered 3D atom arrays as layers of 2D images and then presented a method for moving atoms in 2D into the right positions. The same year, Barredo et al. [10] relocated atoms to user-defined target arrays using a developed shortest-move heuristic. From the heuristic, the number of moves obtained is approximately $\frac{N}{2}$, where N is the number of atoms in the initial array. However, this heuristic does not necessarily minimize the total distance of atoms' movement. Moreover, to prevent collision of atoms, their method of atom moving depends on nearest-neighbor distance in the target array. Later that year, Endres et al. [11] showed moving atoms in 1D, which is a defect-free condition. Using this method, atoms can be moved simultaneously in one single movement and clusters of atoms can be generated in any form.

While atoms are being moved, some of them are lost from the target array. These empty traps are then filled using the reservoir atoms. In 2017, Lee et al. [12] constructed an optimization problem from an atom rearrangement problem. The Hungarian matching algorithm is used to plan collision-free paths through vacancy filling. They were concerned about the collision of atoms in multiple ways. In the end, they concluded that, if the cost function is appropriately defined, then all atom collisions can be avoided. In 2019, de Mello et al. [13] rearranged atoms by applying a shortest-move heuristic. In order to reduce the number of moves, the heuristic attempts to choose the paths with the fewest obstructing atoms to fill all vacancies in a pre-defined target array. The heuristic does not necessarily minimize the total distance of moving atoms but instead focuses on time efficiency. Later in the year of 2019, Brown et al. [14] presented 3D atom relocation. First, they choose atoms that they want to move to the target array and then rearrange these atoms in a single movement by switching rows and columns in 2D. This work does not focus on either the number of moves or the total distance of moving atoms.

The central point of this work is the rearrangement of neutral atoms using mathematical algorithms under the following conditions. First, all vacancies in the target arrays must be filled, and, second, the moves must avoid collision of atoms. It is known that atoms in optical traps have a finite trapping time and can also be lost during transportation with a certain probability given as a function of time and distance. The time is the total computation time of the algorithm and atom transportation time. In the case of a half-filled array, transportation time does not depend much on the initial arrays but changes significantly on choice of algorithm. Thus, the efficiency of atom rearrangement relies mostly on the chosen algorithm.

Rearranging atoms (pairing atoms from the trapping sites to the target sites) can be seen as a combinatorial optimization problem, to be more specific, an assignment problem with a carefully designed cost function. However, only a few tailored algorithms have been developed for choosing atoms and their corresponding target sites. Only the Hungarian matching algorithm, a well-known linear programming technique to solve an assignment problem, is frequently mentioned in the literature on rearranging atom formation. When compared to other developed methods for solving an assignment problem, such as greedy algorithm or other metaheuristic problems, the Hungarian method is relatively slow and inefficient, especially when the problem is quite large scale as in the atom rearrangement problem [15].

Therefore, in this paper, we are interested in developing a faster way to solve the atom rearrangement problem. The orientation of the paper starts with the process of designing the cost function for the assignment problem to prevent the collision of atoms, and all vacancies in target array will be filled by the constraints of the assignment problem. The model we developed can provide an optimal solution which is the optimal matching to move the atoms in one single movement to prevent atom losses in multiple atom-moving processes. As solving an assignment problem is slow, especially for a larger-size array, we offer Convolutional Neural Network (CNN) models to predict the optimal solutions of the assignment problems for three grid sizes of 10×10, 13×13, and 21×21. This drastically shortens the time to find the optimal solutions. Finally, we present the accuracy and calculation times for each model.

2. Methodology

We are interested in the relocation of atoms in two dimensions with defect-free atomic array formation while minimizing the total moving distance. For the defect-free condition, atoms will be moved from an initial array to the target array without collision, and no vacancies in the target array are allowed. Figure 1 shows one way to move the atoms to their target positions.

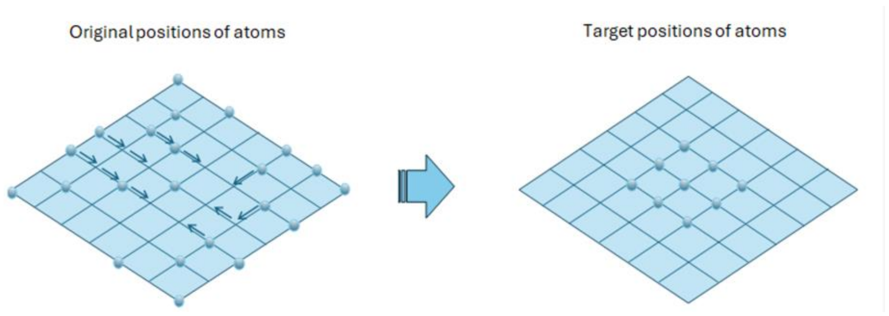


Figure 1. Rearranging atoms to their target positions.

Model Formulation

Matching the atoms and their target sites is obviously an assignment problem if we define the cost function to prevent atom collisions. We start with defining $x_{ij} = 1$ if an atom moves from position i to position j and 0 otherwise. Since we want the atoms to not collide, it is better to force them to move in a horizontal or vertical way. However, as shown in Figure 2, if we move the atom directly from position i to position k, atom collision will happen. So, if the designated position of the atom is as in Figure 2, the atom in position i should move to position j while the atom in position j simultaneously moves to position k.

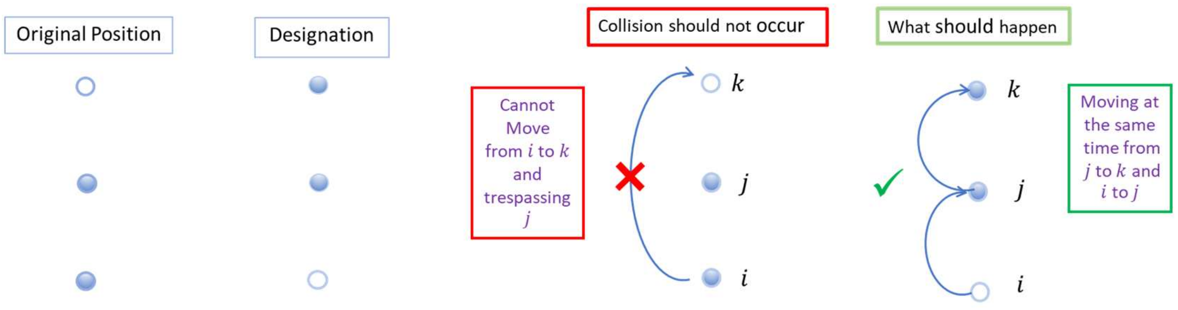

Original Position
Designation
Collision should not occur
Cannot Move from i to k and trespassing j
k
j
i
What should happen
Moving at the same time from j to k and i to j
k
j
i

Figure 2. How to design the cost function so that collisions do not occur.

If we define c_{ij} as a function of the Euclidean distance from position i to position j and let the distance between 2 connecting positions (both horizontally and vertically) be 1, we can see that c_{ik} should be greater than $c_{ij} + c_{jk}$.

Figure 3 shows how to prevent the atom from moving diagonally so that trespassing does not occur. In this case, we do not want an atom to directly move from position i to k, but rather an atom will move from position i to position j and, at the same time, the atom at the position j will move to position k. Hence, c_{ik} should still be greater than $c_{ij} + c_{jk}$.

In general, c_{ik} should be greater than $c_{ij} + c_{jk}$ since, in both cases, we do not want the atom to move directly from position i to position k but instead move from position i to position j while the atom that is originally in position j simultaneously moves to position k. Figure 4 explains how to define c_{ij}.

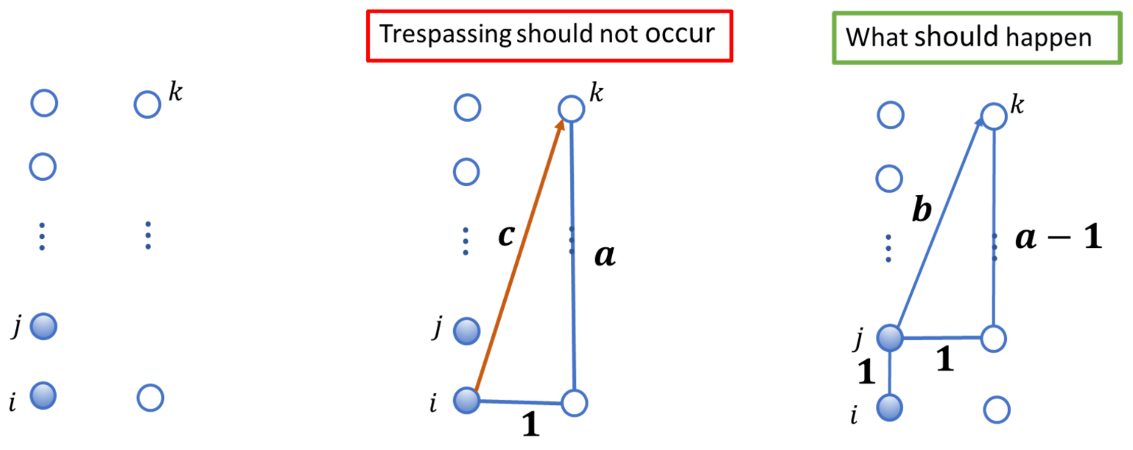


Figure 3. How to design the cost function so that trespassing is not allowed.

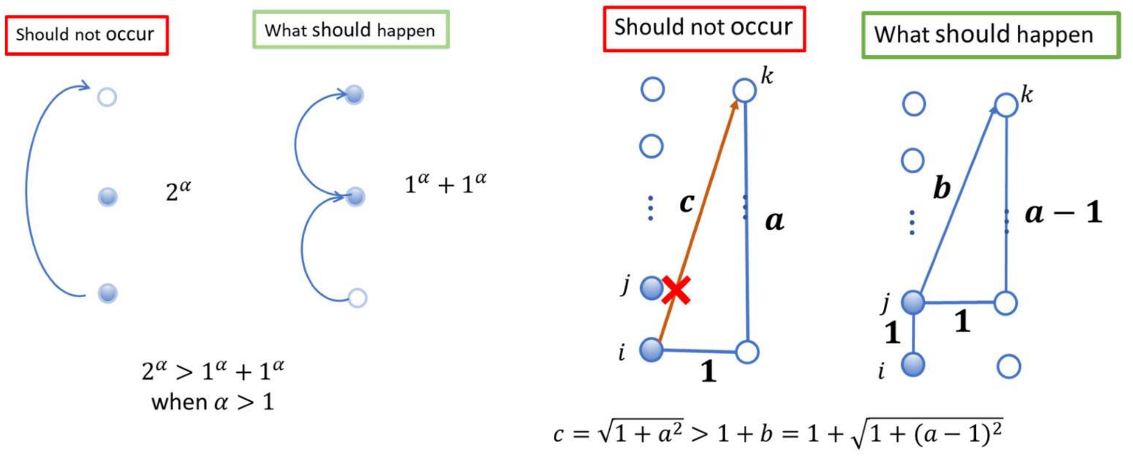


Figure 4. How to design the cost function so that atom collision and trespassing do not occur.

We then define c_{ij} as a function of the Euclidean distance from position i to position j by $(\text{the Euclidean distance from } i \text{ to } j)^\alpha$ as follows:

$$\begin{aligned} c^\alpha &= \left(1+a^2\right)^{\frac{\alpha}{2}} > 1^\alpha + b^\alpha = 1 + \left(1+(a-1)^2\right)^{\frac{\alpha}{2}} \\ &= 1 + \left(1 + a^2 - 2a + 1\right)^{\frac{\alpha}{2}} \\ &= 1 + \left(a^2 - 2a + 2\right)^{\frac{\alpha}{2}} \end{aligned}$$

We can see that when $a = 1$ and $\alpha = 2$, $c = 1 + 1 = 2$.

This implies that moving either from i to k directly or from j to k and i to j would be the same. One can solve for α. In fact, α was given in [8]. However, it is obvious that if $\alpha \geq 2$, then the above inequality holds so long as $a > 1$. We let $\alpha = 2$, and d_{ij} is the Euclidean distance from node i to node j, and we have $c_{ij} = {d_{ij}}^2$. However, this depends on the physical distance allowed for trespassing; if $\alpha = 2$, then the only non-horizontal/vertical move allowed will be when $a = 1$.

The assignment problem for atom rearrangement is then summarized as follows:

$$\min \sum_{\forall i,j} c_{ij} x_{ij}$$

subject to

$$\sum_{\forall i} x_{ij} = 1, \quad \forall j$$

$$\sum_{\forall j} x_{ij} = 1, \quad \forall i$$

or

$$\sum_{\forall i} x_{ij} = 1, \quad \forall j \in T$$

$$\sum_{\forall j} x_{ij} = 1, \quad \forall i \in S$$

where $c_{i,j} = \begin{cases} M, & \text{if } i \notin S \text{ and } j \notin T \\ d_{ij}^2, & \text{otherwise} \end{cases}$.

As stated in Lee et al. [8], the probability of single atom trapping per site is about 50%, that is, they can generate only half-filled atoms in an initial site.

This might look like a simple assignment problem with a cleverly designed cost function to avoid atom colliding and trespassing. However, as far as atom rearrangement is concerned, the number of variables blows up relatively fast. For the 10×10 trap array, the problem contains 10^4 variables. Therefore, for the n trap sites, the number of variables becomes n^2. The number of reasonable trap sites in quantum computing makes this a large-scale assignment problem. Along with the time limit in the atom rearrangement problem, solving this as a traditional LP standard routine or the Hungarian matching algorithm is not feasible. A tailor-made algorithm is thus needed to solve the problem.

3. Convolutional Neural Network Model Formulation

We have developed a Convolutional Neural Network (CNN) model to find the optimal matching for atom movement and fill vacancies in a grid. Since, from the experimental data [4], roughly only 50% of the loaded atoms remain, the number of atoms will fill only 50% of the grids. The model was tested on three types of grids: a 10×10 grid with 49 atoms, a 13×13 grid with 81 atoms, and a 21×21 grid with 169 atoms with the initial positions of the atoms randomly chosen. The process of building the CNN model involves three main steps.

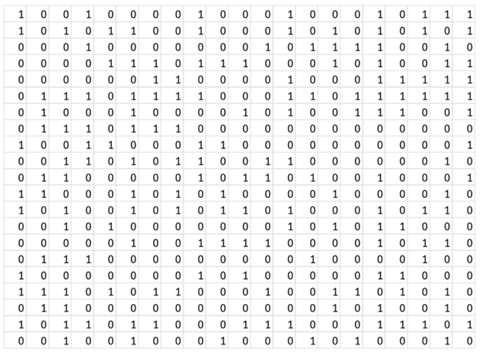
First, we prepare the dataset obtained in the above section. The input to the model is the initial state of the loaded atoms, while the output is the position each atom needs to move to, calculated using the previously mentioned algorithm. Since the output data represent positions in two–three-digit numbers where each digit holds equal significance, these outputs are converted into individual digits. This transformation increases the output dimension from its original shape (m) to either 2 m or 3 m, depending on the grid size. For example, for a 10×10 grid ($m = 100$), the output ranges from 0 to 99, resulting in a transformed output of 200. For a 13×13 grid ($m = 169$), the output ranges from 0 to 168 (three-digit numbers), leading to a transformed output of $3 \times 169 = 507$. The same method applies to the 21×21 grid.

Next, we construct the CNN model using PyTorch 2.4.0. The CNN model comprises convolutional (conv2d) and linear layers with varying numbers of nodes, which increase with larger grid sizes. We employ ReLU as the activation function for all models and incorporate dropout layers to prevent overfitting.

We treat this task as a regression problem using the Mean Squared Error (MSE) as the loss function and the Adam optimizer in the training process. During the training, the models are saved every 100 epochs. These saved models are then validated with a separate dataset (test set) to ensure accuracy. If a model does not meet the required accuracy, we return to the training step to refine it further.

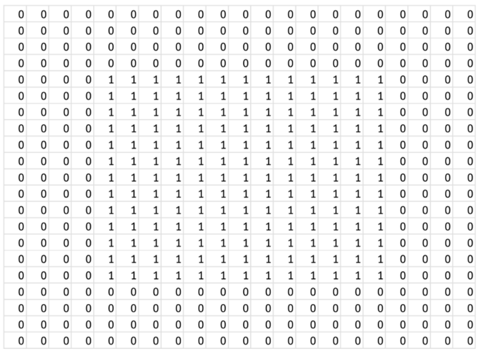
Understand the Data

The following example demonstrates a model specifically designed for rearranging atoms within a general $n \times n$ grid size. In this example, let us consider an $n = 21$, containing a total of $n^2 = 441$ positions, with a specific number of atoms k, as shown in Figure 5. Figure 5b represents the target atom arrangement, where '1' indicates the presence of an atom, confined to a central square of $m \times m$ dimensions, surrounded by zeros. Figure 5a illustrates an example of the input dataset or the initial state of loaded atoms, where '1's and '0's are randomly distributed, with k atoms and $n^2 - k$ empty spaces. Figure 5c depicts the output data, showing the positions where atoms need to be relocated. It consists of $k + 1$ distinct numbers, with k non-zero numbers denoting the coordinates of the central square of $m \times m$ atom positions, and zero signifying empty spaces. Notably, positions with '0's in Figure 5a align with corresponding positions in Figure 5c, while positions with '1's in Figure 5a correlate with non-zero numbers in Figure 5c. The model aims to predict atom movements based on the input, ultimately achieving the target configuration depicted in Figure 5b.



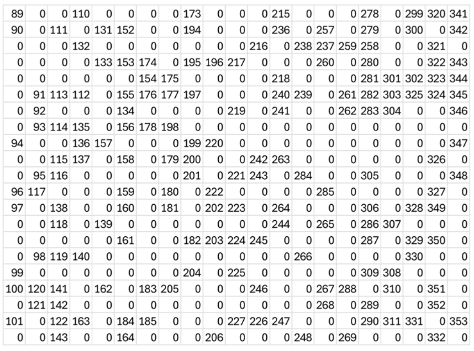
1	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	1	1	1
1	0	1	0	1	1	0	0	1	0	0	0	1	0	1	0	1	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	1	0
0	0	0	0	1	1	1	0	1	1	1	0	0	0	1	0	1	0	0	1	1
0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	1	1	1	1	1
0	1	1	1	0	1	1	1	1	0	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	1	1	1	0	0	1
0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
0	0	1	1	0	1	0	1	1	0	0	1	1	0	0	0	0	0	0	1	0
0	1	1	0	0	0	0	0	1	0	1	1	0	1	0	0	1	0	0	0	1
1	1	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1	0	1	1	0	1	0	0	0	1	0	1	1	0
0	0	1	0	1	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	0
0	0	0	0	0	1	0	0	1	1	1	1	0	0	0	0	1	0	1	1	0
0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0
1	1	1	0	1	0	1	1	0	0	0	1	0	0	1	1	0	1	0	1	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0
1	0	1	1	0	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1
0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0

(**a**)



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(**b**)



89	0	0	110	0	0	0	0	173	0	0	0	215	0	0	0	278	0	299	320	341
90	0	111	0	131	152	0	0	194	0	0	0	236	0	257	0	279	0	300	0	342
0	0	0	132	0	0	0	0	0	0	0	216	0	238	237	259	258	0	0	321	0
0	0	0	0	133	153	174	0	195	196	217	0	0	0	260	0	280	0	0	322	343
0	0	0	0	0	0	154	175	0	0	0	0	218	0	0	0	281	301	302	323	344
0	91	113	112	0	155	176	177	197	0	0	0	240	239	0	261	282	303	325	324	345
0	92	0	0	0	134	0	0	0	0	219	0	241	0	0	262	283	304	0	0	346
0	93	114	135	0	156	178	198	0	0	0	0	0	0	0	0	0	0	0	0	0
94	0	0	136	157	0	0	0	199	220	0	0	0	0	0	0	0	0	0	0	347
0	0	115	137	0	158	0	179	200	0	0	242	263	0	0	0	0	0	0	326	0
0	95	116	0	0	0	0	0	201	0	221	243	0	284	0	0	305	0	0	0	348
96	117	0	0	0	159	0	180	0	222	0	0	0	0	285	0	0	0	0	327	0
97	0	138	0	0	160	0	181	0	202	223	0	264	0	0	0	306	0	328	349	0
0	0	118	0	139	0	0	0	0	0	0	0	244	0	265	0	286	307	0	0	0
0	0	0	0	0	161	0	0	182	203	224	245	0	0	0	0	287	0	329	350	0
0	98	119	140	0	0	0	0	0	0	0	0	0	266	0	0	0	0	330	0	0
99	0	0	0	0	0	0	0	204	0	225	0	0	0	0	0	309	308	0	0	0
100	120	141	0	162	0	183	205	0	0	0	246	0	0	267	288	0	310	0	351	0
0	121	142	0	0	0	0	0	0	0	0	0	0	0	268	0	289	0	0	352	0
101	0	122	163	0	184	185	0	0	0	227	226	247	0	0	0	290	311	331	0	353
0	0	143	0	0	164	0	0	0	206	0	0	0	248	0	269	0	0	0	332	0

(**c**)

Figure 5. (**a**) Initial state of loaded atom (input of the model). (**b**) Target state of the atoms after movement. (**c**) Target coordinates of each atom that needs to move to achieve the target state (output of the model).

Step 1: Dataset preparation: Change the output shape (separate the output digits)

The output data (Figure 5c) represent the positions where atoms need to move. Each number in the output data indicates a specific point within the grid. Each digit (hundreds, tens, ones) in these numbers holds the same importance, meaning if any digit is predicted incorrectly, it affects the entire predicted point. To enhance prediction accuracy, each output is transformed from n^2 numbers to either $2n^2$ or $3n^2$ numbers, depending on whether the original numbers are two-digit or three-digit numbers. For instance, a 10×10 grid size with two-digit numbers would result in $2n^2$ numbers, while larger grids with three-digit numbers would result in $3n^2$ numbers. Each set of two or three numbers represents a single point, with the separation of digits explicitly indicated. The assumption is made that '0' is represented as '000'. The modified output data consist of $2n^2$ or $3n^2$ positions accordingly. Specifically, the first n^2 indices correspond to the hundreds (if applicable), followed by indicating the tens, and, finally, the last n^2 positions represent the ones of the original output data. The dataset is split into training and testing sets with an 80:20 ratio to evaluate the model's performance.

Step 2: CNN model construction

The Convolutional Neural Network (CNN) model utilized in this work is composed of several layers that, together, enable the network to learn and make accurate predictions from input data. The primary layer type is the 2D convolution layer, which is responsible for extracting features from input data. Each convolutional layer typically uses a kernel size of 3×3, with a stride of 1 and padding of 1. This configuration helps preserve the spatial dimensions of the input data while applying the filters to extract features. The number of input and output channels varies across the layers and is adjusted based on the complexity and size of the grid. In the early layers, the number of channels starts at a lower value and increases gradually in subsequent layers to capture more complex features. This increment continues up to a certain point, after which the number of channels is reduced in the later layers to decrease the model's complexity and computational load.

Following the convolution layers are the linear layers. After the convolutional layers have extracted features from the input, the feature maps are flattened into a one-dimensional vector. This flattened vector is then fed into the linear layers, which process the data to make final predictions. These layers operate similarly to a traditional neural network, where each neuron is connected to every neuron in the previous layer, enabling the model to learn complex representations of the data.

To introduce non-linearity into the model and allow it to learn more complex functions, the ReLU (Rectified Linear Unit) activation function is used throughout the network.

Additionally, to prevent overfitting, the model employs Dropout Regularization. During training, dropout works by randomly dropping a set percentage of neurons in the network with a probability of 0.005. This technique forces the model to learn redundant representations and thus generalize better to unseen data. This technique significantly improves the robustness and performance of the model, especially when working with limited datasets.

Step 3: Model Training

To train the model for finding the shortest path for atom movement, we treat the problem as a regression task using the Mean Squared Error (MSE) as the loss function and the Adam optimizer for training.

Training Process

1. Data Input and Prediction: We input the training data into the CNN model, which processes this data to generate predictions.
2. Loss Calculation: The model's predicted output, an array of either $2n^2$ or $3n^2$ size (depending on grid size), is compared to the actual modified output. The difference between the predicted and actual outputs is measured using the MSE loss function.
3. Backpropagation and Weight Updates: Through backpropagation, gradients are computed, and the Adam optimizer updates the model weights accordingly.
4. Training Loop: This process is repeated for a pre-defined number of epochs. To ensure model reliability, the model is saved regularly after every hundred epochs.

Evaluation Process

1. Model Evaluation: Post-training, we evaluate the saved models using the test set (20% of datasets). This involves calculating the accuracy of each saved model and counting the number of correct positions predicted in each data instance to ensure that the trained model is not overfitting and can predict well.
2. Retraining if Necessary: If the evaluated results do not meet our desired goals, we make adjustments to the CNN model parameters or structures and go back to step 3 to retrain again.

4. Results and Discussions

The optimal solutions of the assignment problem in Section 2 are solved by MATLAB R2019a on a computer with an INTEL I9-13900ks with 192 GB of RAM. The CNN models are trained and tested on a computer with an NVIDIA GeForce RTX 4090 with 24.0 GB of RAM. Of the datasets, 80% are used to train the model, and 20% of the datasets are used to test the accuracy of the CNN models. Figures 6–8 show how many positions the CNN models can predict correctly in a percentage for the 10 × 10 array, 13 × 13 array, and 21 × 21 array, respectively. The test results are also summarized in Table 1. It can be seen that the models can predict the optimal solutions reasonably well, especially for the 10 × 10 array. The models can predict correctly more than 97% of the total number of atom movements in all test datasets. When considering the correct prediction of each dataset, the models can predict correctly more than 87% of the number of test datasets with less than 2% error in each dataset (at least 98% accuracy in each dataset). While the model for the 10 × 10 array can achieve a very high percentage of defect-free dataset predictions, this percentage of the model for 21 × 21 is very low. This is due to our current hardware capabilities. The model for the 21 × 21 array we employ is the largest model that can fit in the GPU RAM. This limitation highlights the need for more advanced computational resources to fully realize the potential of our CNN model for larger arrays.

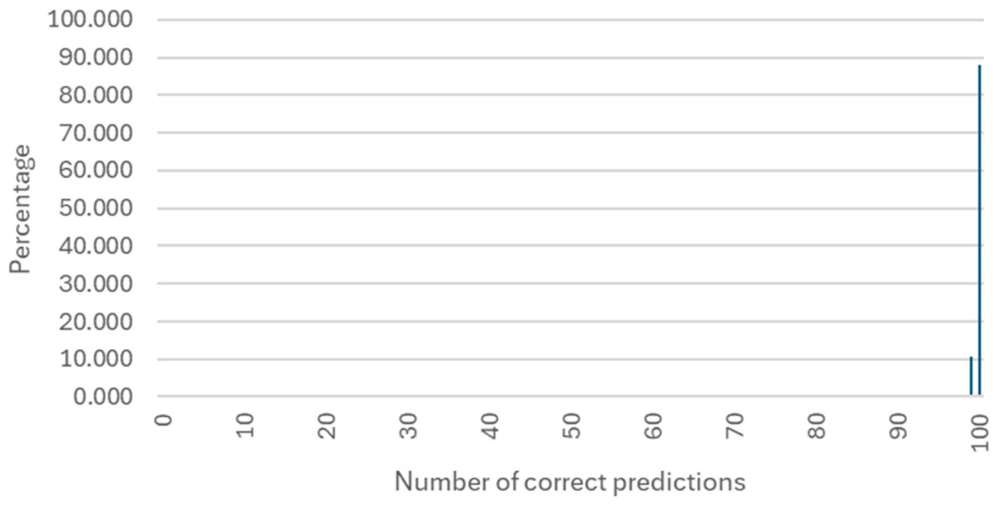


Figure 6. Number of correct predictions for 10 × 10 array.

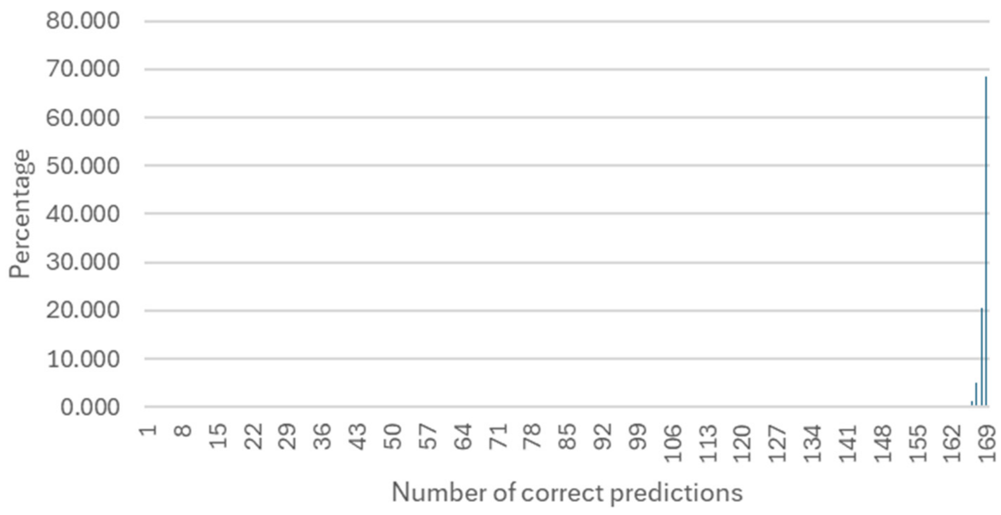


Figure 7. Number of correct predictions for 13 × 13 array.

Table 2 shows the calculation times of the optimal solutions from MATLAB and the prediction from the CNN models. Computation times are measured using “cputime” in MATLAB for the optimal solutions and Python 3.12’s “time” library for the CNN models. The times in Table 2 are the average times of the corresponding calculating times of

5000 datasets. It shows the benefit of using the CNN models. In the 10 × 10 array, the CNN model reduces the calculation time by at least 5 fold. As the size of the array increases, the benefit of the CNN models is even greater. It can reduce calculation time more than 200 fold in the 21 × 21 array. Hence, CNN models can provide a viable solution for the atom rearrangement problem.

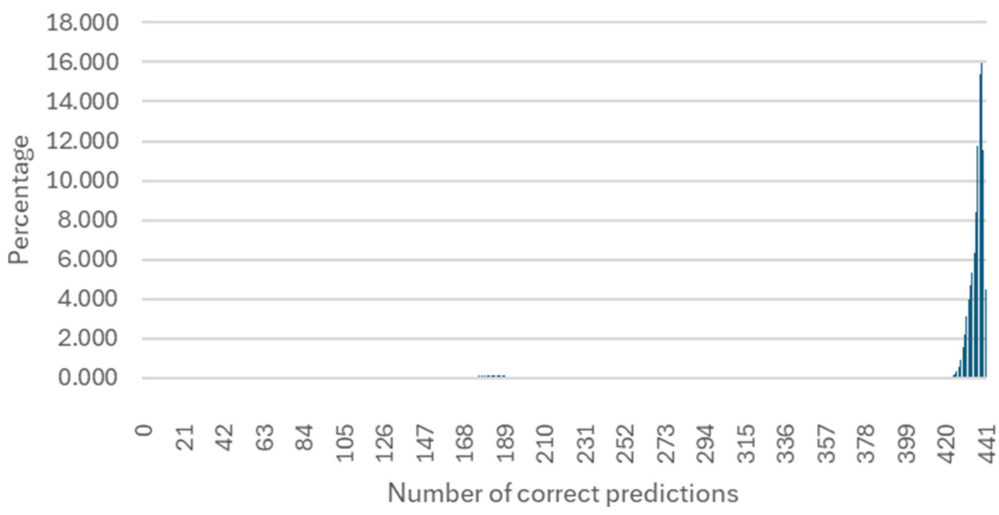


Figure 8. Number of correct predictions for 21 × 21 array.

Table 1. The models' accuracy.

Problem Size (Target Size)	Number of Datasets	Percentage of Overall Position Accuracy	Percentage of 98% Correct Arrangement	Percentage of 99% Correct Arrangement	Percentage of Defect-Free Arrangement
10 × 10 Grid (49 Atoms)	200,000	99.63	99.10	98.44	87.85
13 × 13 Grid (81 Atoms)	600,000	98.93	95.10	89.01	68.54
21 × 21 Grid (169 Atoms)	1,000,000	97.24	87.80	59.04	4.45

Table 2. Computational time.

Problem Size	Calculation Time per Dataset (Calculation Time 5000 Datasets)		% Time Saving (Time Reduction)
	Optimal Solution	CNN	
10 × 10 Grid	0.0159 (79.73) s	0.0031 (15.32) s	80.50 (5.12 folds)
13 × 13 Grid	0.0898 (449.14) s	0.0048 (24.38) s	94.65 (18.71 folds)
21 × 21 Grid	1.289 (6445.9) s	0.0061 (30.50) s	99.53 (211.3 folds)

Figures 9 and 10 compare the calculation times of the optimal solution and the CNN models. The calculation times of the CNN models are multiplied by 100 for clearer comparison. The calculation times per dataset for the optimal solutions increase up to $O\left(N^3\right)$ if the Hungarian matching method is used when the array sizes increase, while the calculation times per dataset of the CNN models increase less than linearly (see Figure 9). Interestingly, Figure 10 shows that, as the number of atoms increases, the calculation times per atom of the CNN models decrease. This shows the potential of the CNN approach for a very large array size since the larger the array size, the more efficient the CNN approach.

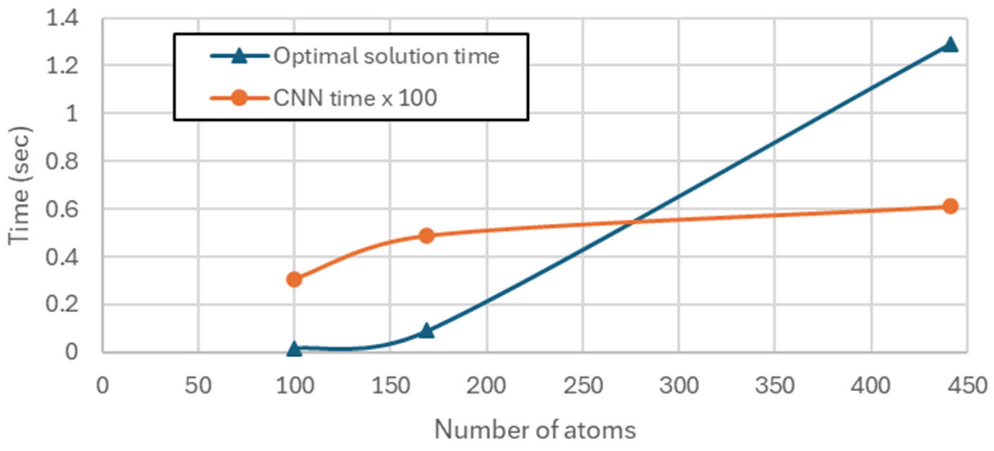


Figure 9. Calculation times per dataset.

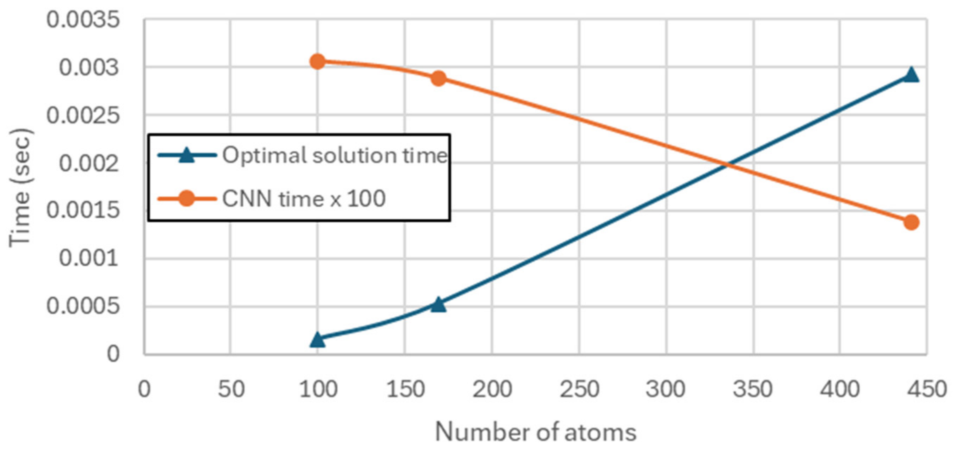


Figure 10. Calculation times per atom.

5. Conclusions

This study aims to predict coordinates for atom movements while ensuring collision-free transitions and filling all vacancies in the target array. The process begins with the design of a cost function for the assignment problem to prevent collisions and constraints to guarantee vacancy filling. The conventional Hungarian matching method to solve such an assignment problem is replaced by Convolutional Neural Network models that we built and trained using three datasets with different grid sizes: 10×10, 13×13, and 21×21. Our models achieve high accuracy in predicting atom positions, with individual position accuracies of 99.63%, 98.93%, and 97.24% and time savings of 5, 39, and over 200 fold, respectively. Despite hardware limitation effects in training larger models, our approach demonstrates significant improvements in computational time and prediction accuracy. It is worth mentioning that computational time tends to increase linearly when the grid sizes increase compared to potential exponential increases if the problem is solved as a traditional LP or a Hungarian matching method.

Author Contributions: Conceptualization, C.L. and R.P.; methodology, C.L. and K.R.; software, R.P. and K.R.; validation, K.R.; analysis, C.L. and R.P.; draft preparation, C.L. and K.R.; writing—review and editing, C.L. and R.P.; funding acquisition, C.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Fundamental Fund 2023 [grant no. FF66/024], Chiang Mai University.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available upon request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*; Cambridge University Press: Cambridge, UK, 2000.
2. Giovannetti, V.; Lloyd, S.; Maccone, L. Advances in Quantum Metrology. *Nat. Photonics* **2011**, *5*, 222–229. [CrossRef]
3. Georgescu, I.M.; Ashhab, S.; Nori, F. Quantum simulation. *Rev. Mod. Phys.* **2014**, *86*, 153–185. [CrossRef]
4. Pichard, G.; Lim, D.; Bloch, E.; Vaneecloo, J.; Bourachot, L.; Both, G.; Mériaux, G.; Dutartre, S.; Hostein, R.; Paris, J.; et al. Rearrangement of single atoms in a 2000-site optical tweezers array at cryogenic temperatures. *Phys. Rev. Appl.* **2024**, *22*, 024073. [CrossRef]
5. Kim, H.; Lee, W.; Lee, H.; Jo, H.; Song, Y.; Ahn, J. In situ single-atom array synthesis using dynamic holographic optical tweezers. *Nat. Commun.* **2016**, *7*, 13317. [CrossRef] [PubMed]
6. Covey, J.P.; Weinfurter, H.; Bernien, H. Quantum networks with neutral atom processing nodes. *npj Quantum Inf.* **2023**, *9*, 90. [CrossRef]
7. Srakaew, K.; Phrompao, J.; Anukool, W. Experimental apparatus and methods for synthesizing 1D single-atom array. *J. Phys. Conf. Ser.* **2019**, *1380*, 012059. [CrossRef]
8. Vala, J.; Thapliyal, A.V.; Myrgren, S.; Vazirani, U.; Weiss, D.S.; Whaley, K.B. Perfect pattern formation of neutral atoms in an addressable optical lattice. *Phys. Rev. A* **2005**, *71*, 032324. [CrossRef]
9. Lee, W.; Kim, H.; Ahn, J. Three-dimensional rearrangement of single atoms using actively controlled optical microtraps. *Opt. Express* **2016**, *24*, 9816–9825. [CrossRef] [PubMed]
10. Barredo, D.; De Léséleuc, S.; Lienhard, V.; Lahaye, T.; Browaeys, A. An atom-by-atom assembler of defect-free arbitrary two-dimensional atomic arrays. *Science* **2016**, *354*, 1021–1023. [CrossRef] [PubMed]
11. Endres, M.; Bernien, H.; Keesling, A.; Levine, H.; Anschuetz, E.R.; Krajenbrink, A.; Senko, C.; Vuletic, V.; Greiner, M.; Lukin, M.D. Atom-by-atom assembly of defect-free one-dimensional cold atom arrays. *Science* **2016**, *354*, 1024–1027. [CrossRef] [PubMed]
12. Lee, W.; Kim, H.; Ahn, J. Defect-free atomic array formation using the Hungarian matching algorithm. *Phys. Rev. A* **2017**, *95*, 053424. [CrossRef]
13. de Mello, D.O.; Schäffner, D.; Werkmann, J.; Preuschoff, T.; Kohfahl, L.; Schlosser, M.; Birkl, G. Defect-free assembly of 2D clusters of more than 100 single-atom quantum systems. *Phys. Rev. Lett.* **2019**, *122*, 203601. [CrossRef] [PubMed]
14. Brown, M.O.; Thiele, T.; Kiehl, C.; Hsu, T.W.; Regal, C.A. Gray-molasses optical-tweezer loading: Controlling collisions for scaling atom-array assembly. *Phys. Rev. X* **2019**, *9*, 011057. [CrossRef]
15. Malacky, P.; Madlenak, R. Transportation problems and their solutions: Literature reviews. *Transp. Res. Procedia* **2023**, *74*, 323–329. [CrossRef]