University of Piraeus

Evolutionary Higher-Order Extreme Learning Machine for Predicting Student Academic Success

MSc in e-Learning

Department of Digital Systems

Author: Dr. Vasileios Christou

Supervisor: Dr. Dimitrios Sampson

September 8, 2026

Statement of Authenticity

This dissertation is submitted in partial fulfillment of the requirements for the MSc programme in "E-Learning" at the Department of Digital Systems, University of Piraeus.

I hereby declare, under my own responsibility, that I am the author of this dissertation and that it is not submitted to, or evaluated as part of, any other postgraduate or undergraduate degree programme, either in Greece or abroad.

The dissertation represents my own views and work on the subject. All sources consulted during its preparation have been appropriately acknowledged and fully referenced, including sources obtained from the internet.

I understand that any violation of the above declaration of academic responsibility may constitute grounds for the revocation of my degree. In the event of a false or inaccurate declaration, I accept that I may be subject to the consequences provided for under Greek and European Union legislation concerning intellectual property rights.

The Declarant

Full Name: Vasileios Christou

Student ID Number: MHM25101

Signature: 

Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Dimitrios Sampson, for his guidance and support throughout the writing of this dissertation. I am also thankful to my family for their encouragement and spiritual support throughout the course of this dissertation.

Contents

List of Tables

List of Figures

Abstract

Early prediction of student academic performance can help higher education institutions identify students at risk of dropping out and provide timely support. This dissertation proposes an evolutionary higher-order extreme learning machine (EHO-ELM) algorithm for predicting student academic success and dropout. The proposed approach extends the traditional extreme learning machine (ELM) method by incorporating higher-order multi-cube units (MCUs) into a single-layer neural network (SLNN). The performance of ELM can be affected by the random initialization of hidden-layer weights and thresholds. In addition, ELM typically adopts low-order neurons, which limits the SLNN's ability to capture more complex interactions amongst input variables. MCUs allow the model to capture nonlinear relationships while avoiding the rapid increase in parameters that can occur with conventional higher-order neurons. EHO-ELM uses a modified genetic algorithm (GA) with self-adaptive parameters to circumvent the suboptimal generalization performance associated with hidden-layer initialization and to identify appropriate network structures. The evolutionary process generates, trains, and evaluates a population of MCU-based SLNNs and optimizes their hidden-layer weights, thresholds, and sub-cube configurations. The output weights are calculated analytically using the Moore–Penrose pseudoinverse, retaining the computational efficiency and structural simplicity that characterize the ELM algorithm. The proposed system is evaluated using a dataset containing a range of student-related variables, including demographic characteristics, higher-education pathways, family and financial circumstances, early academic performance, and macroeconomic conditions. Its performance is compared with 14 alternative machine-learning methods (ELM, online sequential ELM, MCU-ELM, a decision tree, and 10 backpropagation-based neural-network algorithms). The experimental results show that EHO-ELM achieves higher classification accuracy than the other methods in the student-outcome prediction task. The statistical significance of these differences is further examined using the Wilcoxon signed-rank test. Overall, the findings suggest that combining evolutionary optimization with higher-order neural architectures can improve the performance of ELM-based models in educational data mining applications.

Keywords: Extreme Learning Machine, Genetic Algorithm, Higher-Order Neurons, Hybrid Algorithm, Multi-Cube Units, Neural Networks

Chapter 1

Introduction

Overview

This chapter presents the problem of predicting student academic success and justifies its significance. Then it situates the study within educational data mining (EDM), emphasizing the value of early prediction in reducing dropout rates and improving institutional outcomes. Then it analyses the proposed evolutionary higher-order extreme learning machine (EHO-ELM) as a hybrid approach based on an adapted extreme learning machine (ELM) variant for higher-order neural networks (SLNNs). The chapter continues by explaining artificial neurons and neural networks, beginning with the basic structure of the biological neuron on which they are based. It continues by explaining the threshold logic unit (TLU), the simplest form of artificial neuron, and moves on to the low-order unit. Then, it explains the higher-order (cubic) and multi-cube unit (MCU) types, focusing on the advancements over traditional low-order ones. In methodological terms, it explains that although traditional ELM is efficient, its random initialization of hidden-layer parameters can limit performance. To address this issue, the proposed framework employs a modified genetic algorithm (GA) to optimize network structure and weights while preserving ELM's simplicity using self-adaptive parameters. At the end of the chapter, the motivation for the proposed research is analyzed, and a synopsis of the dissertation structure is presented.

1.1 Theoretical and Methodological Foundations

In the last few years, significant advancements in information technology have influenced a variety of sectors, including higher education (Wang et al. 2024). Academic institutions have accumulated extensive volumes of data that can be systematically analyzed to enhance students' academic trajectories and overall learning experiences. One significant research area that uses these data types is predicting students' future academic

performance from past data. In this area, the researcher tries to identify patterns that can affect future performance. Predicting student academic success is very important for academic institutions, as early interventions can reduce student dropouts and increase success rates. The interventions include targeted individual student support, which will enhance the student's performance (Burgos et al. 2018). The support includes tutoring plans for students in need, which have been shown to effectively reduce dropout rates (Tondeur et al. 2008). Recent studies have shown that an increasing number of higher education institutions recognize the importance of predicting student performance and taking proactive measures to improve student success rates (Boujmiraz et al. 2026).

Higher education institutions are interested in creating prediction models based on the characteristics and performance of their student data (El Aissaoui et al. 2020) by using educational data mining (EDM) and machine learning tools. EDM applies data mining in the educational sector to analyze and predict the future academic performance of students and instructors based on past data. Its purpose is to enhance the teaching and learning processes while reducing the institutions' dropout rates (Dol and Jawandhiya 2023). The use of these system types aims at improving the overall educational process (Christou et al. 2023; Cuevas et al. 2018; S. Li and T. Liu 2021).

The proposed method applies the evolutionary higher-order extreme learning machine (EHO-ELM) algorithm by Christou et al. (2025) for the prediction task. It is a hybrid approach based on an adapted version of the extreme learning machine (ELM) algorithm for higher-order single-layer neural networks (SLNNs). A modified GA is used to train a series of higher-order neural networks using a modified version of ELM and then evolve them through a repetitive process. At the end of the process, the best one is selected for the classification task.

The artificial neural networks (ANNs) are layered parametric models that map inputs to outputs by composing many simple computational units termed neurons interconnected together. The artificial neuron is a simplified equivalent of the biological neuron depicted in Fig. 1.1.

Neurons are specialized cells that generate and conduct electrical activity to carry information through the body. They use both electrical impulses and chemical signaling to communicate. Neurons connect with other neurons at synapses and with muscles, glands, or other target cells at neuroeffector junctions. A typical multipolar neuron has three main parts. These parts are the soma (cell body), the dendrites, and a single axon. In general, the axon carries outgoing, or efferent, signals, while the dendrites receive incoming, or afferent, input from the surrounding environment (Ludwig et al. 2023, July).

The artificial neuron is a simplified equivalent of its biological counterpart, termed a node or unit (a typical example illustrated in Fig. 1.2). Synapses are represented by real numerical values called weights, meaning each input is multiplied by a corresponding

weight before reaching the equivalent of a neuron's cell body. These weighted inputs are then combined through simple addition to produce the node's activation. In the model shown in Fig. 1.2, termed a threshold logic unit (TLU), this activation is compared against a predefined threshold (bias) (θ). If the activation surpasses the threshold, the unit outputs a high value (usually 1), otherwise, it outputs 0. The TLU is the earliest and most basic form of an artificial neuron, originally introduced by McCulloch and Pitts (1943). The term "network" is used to describe any such system having interconnected TLUs (Gurney 2018).

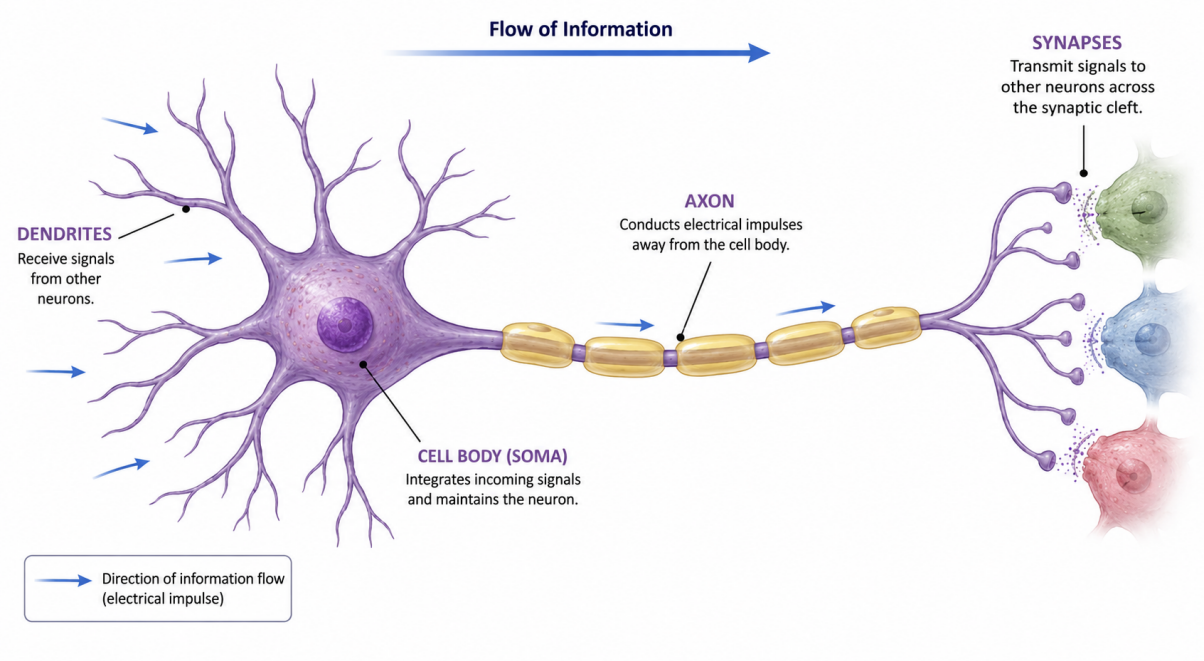

Fig. 1.1: The Biological Neuron. This figure depicts the structure of a typical biological neuron. Neurons are specialized cells that transmit information through electrical and chemical signals. A typical multipolar neuron consists of the soma, dendrites, and a single axon. The dendrites receive incoming input, while the axon carries outgoing, or efferent, signals. [The conceptual illustration was generated with the assistance of Perplexity artificial intelligence (AI) and subsequently edited by the author (Perplexity AI 2026).]

The TLU, although it is closer to its biological counterpart (the biological neuron is electrically polarized, and when it reaches the firing threshold, triggers an action potential), is restricted to binary values. An extension to real value outputs is the semi-linear or low-order unit type. Similarly, these neuron types receive an input vector x which is multiplied with a corresponding weight vector w, and an optional threshold θ is added. This part forms the activation, which in turn is added as input to the activation (transfer) function g, to produce the neuron's output y. The low-order unit activation is depicted in formula $a = \sum_{i=1}^{n} w_i x_i + \theta$ with the sigmoid function ($\frac{1}{1+e^{-x}}$) being a popular activation function choice (Gurney 2018).

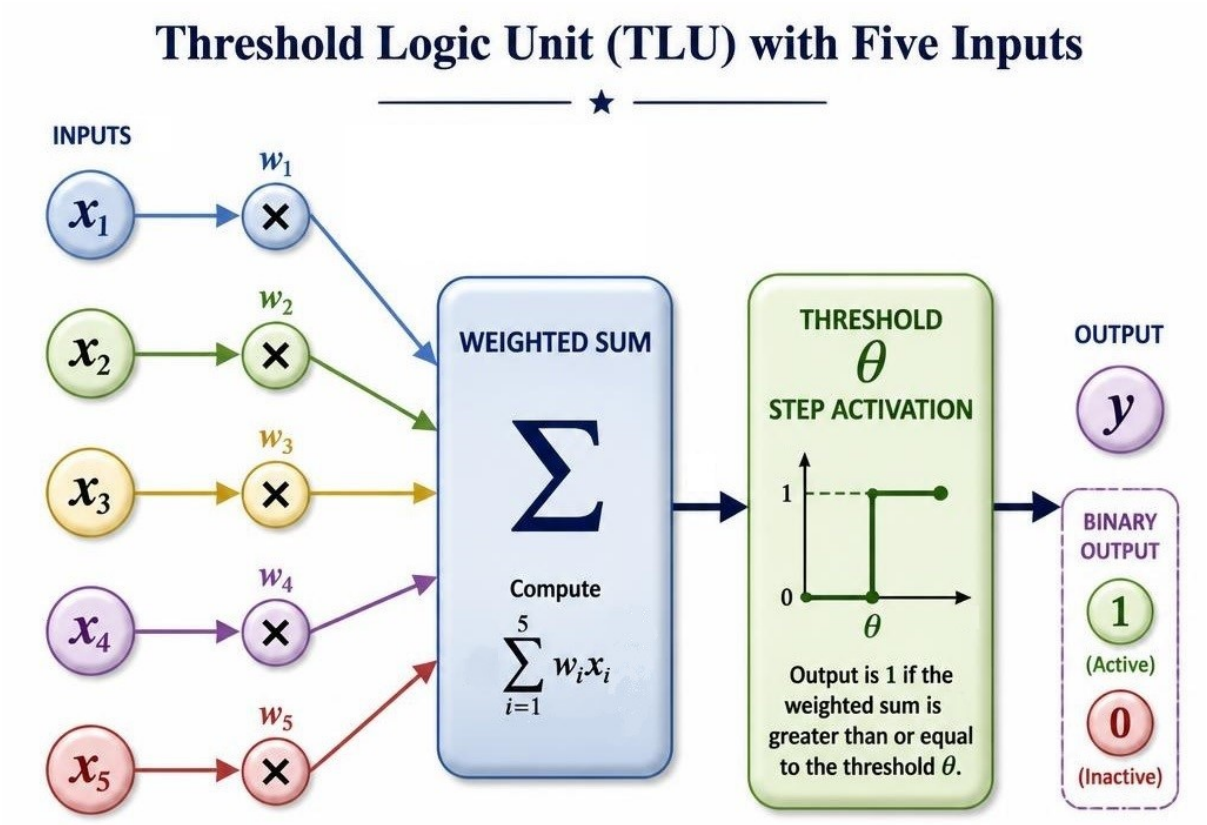

Fig. 1.2: The Threshold Logic Unit. This figure depicts the structure of a typical 5-input TLU. Each input $[x_1, x_2, x_3, x_4, x_5]$ is multiplied with a real valued weight $[w_1, w_2, w_3, w_4, w_5]$ and the outcome from each multiplication is summed ($\sum_{i=1}^{5} w_i x_i$). Then, this sum is compared to a threshold θ, and if it is equal to or greater than this value, the TLU outputs 1; otherwise, it outputs 0. [The conceptual illustration was generated with the assistance of Perplexity AI and subsequently edited by the author (Perplexity AI 2026).]

The TLU and the low-order unit types are restricted to linear separable problems. The higher-order (cubic) units proposed by Gurney (1989) deal with this problem by mapping a weight to a high-dimensional cube site. In these neuron types, an n-input neuron contains 2^n weights. Although these neuron types can solve non-linear separable problems, they introduce a significant issue: the exponential growth of cube weights. In order to circumvent this problem, Gurney (1989) proposed the multi-cube unit (MCU), which breaks a hyper-cube into smaller sub-cubes. Using this technique, the number of weights follows the formula $\sum_{j=1}^{q} 2^{d_j}$ where q denotes the number of sub-cubes and $d = [d_1, d_2 \dots d_q]$ is the vector containing the sub-cube dimensions, with j denoting the current sub-cube dimension. The combination of the number and sub-cube dimensions can affect the classification accuracy or regression error of the MCU. For this reason, proper selection of these parameters is needed. When these neuron types are interconnected, they form a neural network with the SLNN being its simplest form. In these network types, the information flows in one direction from the input layer, which contains the input vector

$x = [x_1, x_2, \ldots, x_n]$, where n is the number of inputs, to the hidden layer. The hidden-layer contains $h \in \mathbb{N}^*$ number of neurons and the outputs of these neuron types are sent to the output layer neurons which produce the neural network's output. The architecture of a 5-input network, with seven hidden-layer neurons and two exit nodes, is depicted in Fig. 1.3.

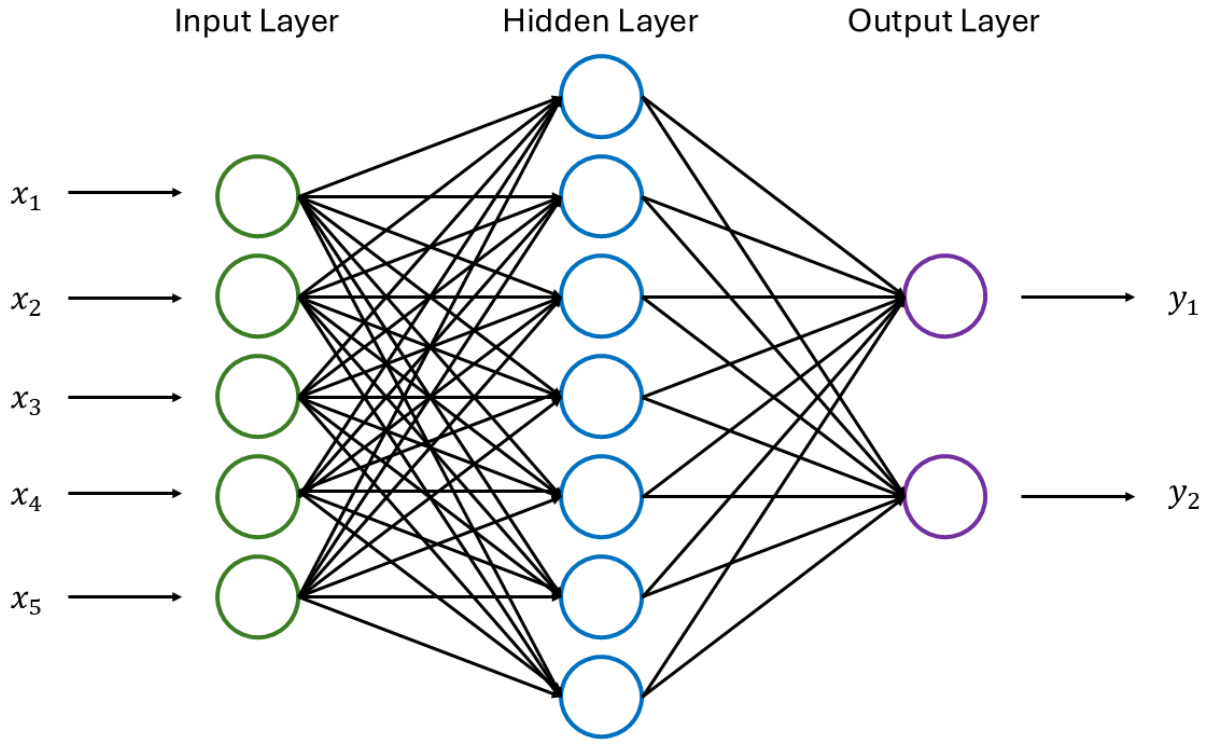

Fig. 1.3: The Single Layer Neural Network Architecture. This figure depicts the architecture of a typical SLNN with five inputs, seven hidden-layer units, and two outputs. It is the simplest form of neural network, with information flowing in one direction from left to right and no loops.

The low-order unit type SLNNs can be trained using the ELM algorithm by Huang et al. (2004, 2006b). ELM can train an SLNN in a very short time. The reason behind this speed advantage is that it does not rely on iterative methods such as backpropagation and its variants. It models an SLNN as a system of linear equations, with the hidden layer weights and thresholds randomly assigned. Then, it uses the Moore–Penrose pseudo-inverse to calculate the output weights analytically. Also, it doesn't fall into local minima, and it doesn't have any tunable parameters. The user only needs to specify the number of hidden-layer neurons. Besides the aforementioned advantages, it also has some significant disadvantages, including the random assignment of hidden-layer weights and thresholds. The randomization procedure may result in a suboptimal network configuration, thereby negatively affecting its generalization performance. Moreover, the traditional ELM algorithm works only with low-order unit types and does not consider the higher-order neurons proposed by Gurney (1989). An adaptation of ELM for these

unit types was proposed by Christou (2026), who demonstrated that both higher-order and MCU unit types can yield significant performance improvements in regression and classification datasets. EHO-ELM is based on this adaptation and uses a modified GA to optimize the hidden-layer weights and network structure.

The motivation behind using EHO-ELM contained two primary objectives. The first was to address the issue of random initialization of hidden-layer weights and thresholds, while the second was to identify an optimal configuration of sub-cube units for neurons in both the hidden and output layers. These challenges were addressed using a modified self-adaptive GA that iteratively generates, trains, and evolves a population of MCU-based SLNNs to improve generalization performance. Upon completion of the evolutionary procedure, the algorithm selects the most suitable network for classifying the "predict students' dropout and academic success" dataset. Furthermore, EHO-ELM preserves the structural simplicity of ELM by employing self-adaptive GA parameters. The method was adopted because its evolutionary mechanism produced more optimized SLNN architectures than ELM and MCU-ELM (a SLNN with MCU in both layers trained with a modified version of ELM). At the same time, it demonstrated superior performance relative to 14 other machine learning approaches in the experimental evaluation.

The dissertation is organized into six chapters and one appendix. Chapter 1 (Introduction) presents the problem statement, explains the motivation behind the study, and provides a brief overview of the proposed system architecture. Chapter 2 (Literature Review) reviews existing methodologies that are relevant to the proposed approach. Chapter 3 (Related Work) consists of five subsections covering neuron architectures, the traditional ELM algorithm, the MCU-ELM extension, the GA architecture, and the EHO-ELM hybrid approach. Chapter 4 (The System Architecture) describes the proposed system in detail. It includes the dataset's characteristics and the implementation of the prediction algorithm. Chapter 5 (Experimental Results) presents the experimental setup, compares the results obtained, and evaluates their statistical significance. Chapter 6 (Limitations and Future Work) discusses the study's limitations and outlines possible future research directions. Finally, Appendix A documents the use of artificial intelligence in preparing the dissertation.

Chapter 2

Literature Review

Overview

The current chapter presents a detailed literature review focused on popular and recent ELM variants. It begins by presenting a series of generic incremental, weight, and bias optimization ELM methods focused on optimizing the hidden layer neuron parameters. Then it continues with hidden parameter optimization methods that have been created for application in specific problem types. The literature review continues with ELM-based methods in the educational domain, including student-performance and dropout prediction, analysis of mentor–student relationships, and entrepreneurial-intention modeling. These studies employ ELM variants combined with optimization algorithms to address outliers, class imbalance, nonlinear relationships, and instability caused by random parameter assignment. This section also presents and compares alternative existing higher-order neuron types and compares them with MCUs. Finally, the literature review is completed with an analysis of the methods used to compare the proposed EHO-ELM in the "predict students' dropout and academic success" dataset.

2.1 Weight and Threshold Optimization Methods

Many existing methods focus on optimizing the hidden-layer parameters. The incremental extreme learning machine (I-ELM) can automatically determine neural network structure while maintaining rapid learning speed (Huang et al. 2006a). However, during hidden-node expansion, nodes with limited relevance to the target may be introduced, thereby unnecessarily increasing model complexity. To mitigate this issue, prior studies have proposed evaluating the relevance between hidden nodes and network outputs and then adding or removing nodes accordingly. Although random hidden nodes can promote pattern diversity, they may also generate nodes that are weakly related or unrelated to the target, leading to an excessive number of hidden nodes. In contrast to conventional

I-ELM approaches that rely on random hidden-node generation, the compact I-ELM initially introduces linear regression nodes and subsequently applies a strategy to ensure that newly added hidden nodes exhibit patterns distinct from those already present in the network. (Seo et al. 2024).

The paper by Zhang et al. (2022) introduces an ELM variant optimized with an enhanced salp search algorithm, called NSSA-ELM. The salp search algorithm (SSA) is a metaheuristic method. To improve its exploration ability and reduce the risk of getting trapped in local optima, the authors incorporate neighborhood centroid opposite-based learning into the optimization process. This helps preserve population diversity, which in turn supports better avoidance of local minima and faster convergence.

The SSA is a population-based, nature-inspired optimization method. It is inspired by the collective movement and foraging behavior of salps in the ocean. Salps are transparent, barrel-shaped organisms that form chain-like structures, referred to as salp swarms, when searching for food. A salp swarm is divided into two groups (a leader and a set of followers). The leader directs the swarm toward the food source, and its position is updated based on the target's location. In contrast, followers update their positions by moving relative to the salp immediately ahead of them. The position of the leader in the j-th dimension is computed in equation 2.1 where x_j^1 denotes the leader's position, F_j is the position of the food source, and ub_j, lb_j are the upper and lower bounds of the j-th dimension. The parameters c_2 and c_3 are random values in the interval $[0, 1]$, while c_1 controls the balance between exploration and exploitation.

$$x_j^1 = \begin{cases} F_j + c_1\Big((ub_j - lb_j)c_2 + lb_j\Big), & c_3 \geq 0 \\ F_j - c_1\Big((ub_j - lb_j)c_2 + lb_j\Big), & c_3 < 0 \end{cases} \tag{2.1}$$

c_1 is defined in formula 2.2 where l is the current iteration and L is the maximum number of iterations.

$$c_1 = 2\, e^{-\left(\frac{4l}{L}\right)^2} \tag{2.2}$$

For the followers (with $i \geq 2$), the position update rule is given in equation 2.3 where x_j^i represents the position of the i-th follower in the j-th dimension (Mirjalili et al. 2017).

$$x_j^i = \frac{1}{2}\left(x_j^i + x_j^{i-1}\right) \tag{2.3}$$

ELM is commonly applied to complex industrial problems, and its online sequential version, OS-ELM, is useful for industrial online modeling. However, OS-ELM must first be trained on a batch of samples to obtain initial weights, which can reduce its ability to respond promptly to new data. To address this limitation, the paper by Zha et al. (2022) introduces a new online regression model called recurrent-ELM. In this model, hidden-layer nodes are interconnected, so each hidden layer receives information from

both the current input and the previous hidden layer. The model's weights and biases are derived analytically rather than assigned randomly.

SLNNs trained with the ELM algorithm usually require a relatively large hidden layer, which may cause them to become ill-conditioned due to the random initialization of the input weights and hidden biases. To overcome these drawbacks, Han et al. (2013) introduces a hybrid learning method that integrates an improved particle swarm optimization (PSO) scheme with the Moore-Penrose generalized inverse. The improved PSO is applied to determine the input weights and hidden thresholds, while the output weights are obtained analytically by the Moore-Penrose inverse. The improved PSO seeks an optimal SLNN by reducing both the validation-set root mean squared error (RMSE) and the norm of the output weights. The experimental results indicate that the proposed approach offers better generalization than standard ELM and other evolutionary ELM variants. Also, it yields a better-conditioned SLNN.

PSO shows similarities to GAs in that it begins with a population of randomly generated candidate solutions. The difference between PSO and GAs is that each candidate solution in PSO is assigned a velocity vector, and these solutions (referred to as particles) move through the search space accordingly. Each particle maintains a record of its current position, and the best position it has achieved so far is based on a fitness evaluation. This personal best value is denoted by *pbest*, and a global best value (denoted *gbest*) is tracked. This corresponds to the best solution identified by any particle in the entire population. The velocity of each particle at each iteration is updated by directing it toward both its *pbest* and the *gbest* positions. These updates are influenced by stochastic components, while separate random factors control the attraction toward the personal and global best positions. In a local variant of PSO, each particle considers its *pbest* and the best solution found within its local neighborhood, referred to as *lbest*. Both the global and local variants follow similar principles but differ in how they utilize neighborhood information. The main parameter the user needs to specify is the maximum allowable particles' velocity. Although an acceleration coefficient is also defined, it is typically kept constant across different applications based on empirical observations (Eberhart and Kennedy 1995).

To improve ELM's generalization performance and produce more compact network structures, the paper by Eshtay et al. (2020) proposes a hybrid approach that integrates ELM with the competitive swarm optimizer (CSO). The proposed model, called CSONN-ELM, optimizes the weights and biases while dynamically determining the most suitable number of neurons.

CSO is a swarm intelligence-based optimization algorithm that adopts the general search framework of the original PSO. In fact, CSO is a PSO variant inspired by the same basic ideas. It was developed to overcome the premature convergence problem that is commonly observed in PSO. This issue often arises because PSO performs poorly on

problems with many local optima and on high-dimensional search spaces. Many PSO variants have been introduced in the literature to improve exploration by adding new mechanisms or extra operators. However, these modifications often increase algorithmic complexity. In addition, most of them still suffer from premature convergence because the global best position has a dominant influence on the search process. The key idea behind CSO is to eliminate both the global and personal best positions. Unlike PSO, where each particle is updated according to its own best position and the swarm's best position, CSO updates particles through pairwise competition between two swarms. In each competition, particles from the two swarms are compared, and one is assigned as the winner while the other becomes the loser. The winner is passed directly to the next generation, whereas the loser is updated by learning from the winner. A major difference between CSO and PSO is that CSO does not preserve memory of previous generations, since it does not store either the global best or the personal best. As a result, the search process in CSO is driven by random competition, and any particle may take the role of leader during optimization (Cheng and Jin 2015).

The self-adaptive evolutionary ELM (SaE-ELM) by J. Cao et al. (2012) for SLNNs is an improved learning approach that uses a self-adaptive differential evolution (DE) algorithm to optimize the network's hidden-node weights and thresholds. Within this framework, the trial-vector generation strategies and their associated control parameters are adaptively managed in a strategy pool, where they are refined based on prior experience in generating effective solutions. The network's output weights are then determined analytically using the Moore–Penrose pseudo-inverse. In addition, SaE-ELM can automatically choose appropriate DE control parameters and generation strategies.

DE is a heuristic method for minimizing continuous-space functions that may be nonlinear and non-differentiable. It has only a small number of control parameters, is robust and easy to implement, and is well suited to parallel computation (Storn and Price 1997)

Extreme Learning Machines (ELMs), which are single hidden layer feedforward neural networks (SLFNs), have gained significant interest because of their fast training speed and strong predictive performance. However, their generalization ability and stability are often reduced due to the random assignment of input weights and hidden layer biases.

The study from Ling et al. (2025) proposes a model called improved multi-objective PSO-ELM (IMOPSO-ELM). The proposed method overcomes the randomization issue of input weights and hidden-layer biases. The goal is to enhance both generalization performance and convergence stability by applying a multi-objective PSO method to determine the input weights and hidden biases of the SLNN. Unlike traditional approaches that rely on single-objective optimization, the proposed method considers two objectives at the same time (the accuracy on a validation dataset and the $L2$ norm of the output weights). Furthermore, an improved version of multi-objective PSO is introduced to

improve the diversity and convergence of the solution set. This method uses a new strategy for selecting the global best particles. The population is randomly divided into several subgroups, and each subgroup is guided by different information from an external archive. The archive also serves as a shared space for exchanging information among the subgroups.

Wu et al. (2017) proposed a hybrid learning method called the dolphin swarm algorithm ELM, which uses the dolphin swarm algorithm to efficiently optimize the input weights and hidden thresholds. In this method, each candidate set of input weights and hidden biases is encoded as a single vector, referred to as a dolphin. The dolphins are assessed using RMSE and updated through the four key phases of the dolphin swarm algorithm. In the end, an optimal set of input weights and hidden biases is obtained.

The dolphin swarm algorithm by Wu et al. (2016) is an efficient global search approach for solving a wide range of optimization problems. The method is inspired by the dolphin's natural hunting behavior and biological traits, including echolocation, communication, cooperation, and division of labor. By modeling these behaviors, DSA captures the interactions among dolphins and their collective dynamics, allowing the group to adapt at a higher level and work toward a shared objective. With effective control strategies, DSA has demonstrated several attractive properties, such as rapid periodic convergence, avoidance of local minima, and no strict dependence on the form of the cost function.

Rathod and Wankhade (2022) developed a hybrid algorithm that combines the cuckoo search and invasive weed optimization algorithms to optimize the hidden-layer parameters.

Cuckoo search draws on the parasitic brooding behavior of certain cuckoo species combined with the Lévy flight patterns observed in some birds and fruit flies. Cuckoos are birds with highly competitive breeding strategies. In some species, eggs are laid in collective nesting sites. Also, birds may take away other eggs to improve the chance that their own will hatch. Many cuckoo species practice brood parasitism, placing their eggs in other birds' nests, often belonging to different species. This parasitism generally appears in three forms. These forms are intraspecific brood parasitism, cooperative breeding, and nest takeover. Host birds may respond aggressively if they detect foreign eggs, either by discarding them or abandoning the nest and building elsewhere. Some cuckoos have evolved egg mimicry that closely matches the color and pattern of the eggs of selected host species. This helps reduce rejection and improves reproductive success. Their timing is also highly adapted, because they often lay eggs in nests where the host has just begun laying. Cuckoo chicks usually hatch slightly earlier than the host's eggs, and once hatched, the chick may eject the host eggs from the nest to secure more food. Some cuckoo chicks can even imitate the calls of host chicks to increase their feeding opportunities (Yang and Deb 2009).

The flight motifs of various insects and animals exhibit the characteristic properties

of Lévy flights. Fruit flies search their environment through sequences of straight flights interrupted by abrupt 90° turns, producing an intermittent, scale-free search pattern consistent with Lévy-flight behavior. This behavior has been applied to optimization and optimal search with positive results (Yang and Deb 2009).

The cuckoo search algorithm utilizes the following three rules:

1. Every cuckoo lays a single egg every time and places it in a nest selected in a random manner.

2. The nests containing the best eggs are carried forward to the succeeding generation.

3. The number of host nests is set, and a host bird discovers a cuckoo's egg using a specific probability ($p_a \in [0, 1]$). In this situation, the bird may discard the egg or leave the nest and construct a new one. This behavior can be estimated by assuming that a fraction p_a of the n nests are replaced by new ones, each initialized using a new random solution.

In a maximization problem, the solution's fitness quality can be taken as directly related to the objective function value (Yang and Deb 2009).

A cuckoo is defined as i and a new candidate solution as $x_i^{(t+1)}$. A Lévy flight is applied using formula 2.4 with $\alpha > 0$ denoting the step size, which is typically selected based on the problem scale. In many cases, an $\alpha = 1$ value is adequate.

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \text{Lévy}(\lambda) \tag{2.4}$$

The above expression represents a stochastic random walk process. A random walk can be interpreted as a Markov chain in which the next state depends only on the current position (first term) and a probabilistic transition component (second term). The operator $\oplus$ indicates element-wise multiplication. Although similar operations appear in Particle PSO, Lévy flights enable more effective exploration by producing longer step sizes over time.

The step lengths in Lévy flights follow a Lévy distribution given in equation 2.5, which has infinite mean and variance. Consequently, the resulting random walk exhibits a heavy-tailed, power-law distribution of step sizes.

$$\text{Lévy} \sim u = t^{-\lambda}, \quad (1 < \lambda \leq 3) \tag{2.5}$$

To balance exploration and exploitation, new solutions are generated near the current best solution using Lévy-based local search to accelerate convergence. Meanwhile, a considerable portion of solutions is created through long-range randomization, ensuring diversity and reducing the risk of convergence to local optima (Yang and Deb 2009).

The invasive weed algorithm by Karimkashi and Kishk (2010) is motivated by the natural colonization behavior of invasive weeds and is rooted in weed biology. Prior studies have demonstrated that capturing the essential properties of invasive weeds can yield an effective optimization technique. In this process, weeds disperse into a crop field and occupy the open spaces among plants. Each invading weed uses the available resources in the field, grows, and creates new weeds. The offspring of each flowering weed rely on its fitness inside the colony. The weeds that are better adapted to the environment can exploit more available resources. They can grow more rapidly and generate more seeds. The newly generated weeds are then randomly dispersed into the field and develop into flowering weeds. This process continues until the field reaches its maximum weed capacity due to limited resources. At that point, only the weeds with higher fitness survive and continue reproducing. Through the above repeated competition, the weeds gradually become better adapted over time.

The invasive weed optimization algorithm starts by identifying the decision variables to be optimized. For each variable within the N-dimensional search space, corresponding lower and upper bounds are defined. An initial population is generated by randomly distributing a fixed number of seeds throughout the solution space (each seed represents a potential solution). Every seed is evaluated using a fitness function that measures the solution's quality (it is considered a flowering plant). Based on their fitness values, the plants are ranked prior to reproduction. The number of seeds produced by each plant depends on its rank, ranging between a minimum $S_{\min}$ and a maximum $S_{\max}$. Plants with higher fitness values create more seeds. This way, they increase their influence on the next generation. The produced seeds are dispersed in the search space according to a normal distribution centered at the parent plant's position. The standard deviation (SD) of this distribution decreases over time to balance between exploration and exploitation. A large SD at the beginning of CSA promotes global exploration, while its gradual reduction enables fine-tuned local search around optimal regions. After dispersion, the new seeds grow into plants and are evaluated alongside the existing population. All individuals are ranked, with the best ones retained to ensure that the population size does not exceed the maximum allowed (P_{max}). On the other hand, lower-ranked plants are discarded (competitive exclusion). The remaining plants reproduce, and the cycle continues until either the maximum number of iterations is reached or a predefined fitness condition is achieved (Karimkashi and Kishk 2010).

L. Cao et al. (2021) proposed an improved crow search algorithm (CSA) for ELM optimization. By analyzing the limitations of the conventional CSA, a search strategy inspired by PSO is incorporated to strengthen global exploration. During the later stages of iteration, a Gaussian-based perturbation mechanism is introduced, where a penalty factor regulates the magnitude of local disturbances. This approach progressively decreases the search step size and adaptively tunes parameters to reduce the risk of being

trapped in local optima. The enhanced CSA is subsequently employed to optimize both the hidden layer structure and the connection weights of the ELM, leading to improved prediction accuracy.

Askarzadeh (2016) proposed CSA, a metaheuristic optimization method inspired by the social foraging behavior of crows. Crows are intelligent, group-living birds that store surplus food in hidden locations, referred to as memory, and retrieve it when necessary. They may also follow other crows to steal their hidden food. To counteract this, crows exhibit an awareness probability (AP), which determines their ability to detect and avoid being followed. CSA has been successfully applied in various domains such as network optimization and medical diagnostics. The algorithm is characterized by its simplicity, relying mainly on two parameters: flight length and awareness probability, while also offering fast convergence and competitive performance compared to other optimization techniques.

The algorithm begins by defining the maximum number of iterations as $iter$. The position of crow i in an n-dimensional space at iteration $iter$ is defined as $X^{i,iter} = x_1, x_2, \ldots, x_n$, and its memory is expressed as $m^{i,iter} = m_1, m_2, \ldots, m_n$, which represents the best solution found so far. A fundamental component of CSA is the interaction among crows through a chasing mechanism, which generates new candidate solutions depending on whether the target crow detects the follower. If crow i follows crow j and crow j is unaware of being tracked (state A), crow i updates its position toward the memory of crow j according to formula 2.6 where $r_i \in [0, 1]$ is a random variable and f_i^{iter} denotes the flight length of crow i. Smaller values of f encourage local exploitation, whereas larger values support global exploration.

$$X^{i,iter+1} = X^{i,iter} + r_i f l^{i,iter} \left(m^{j,iter} - X^{i,iter} \right) \tag{2.6}$$

On the other hand, if crow j becomes aware of being followed (state B), it attempts to deceive crow i by moving randomly within the search space. The position update is then given by equation 2.7.

$$X^{i,iter+1} = \begin{cases} X^{i,iter} + r_i f l^{i,iter} \left(m^{j,iter} - X^{i,iter} \right), & r_i \geq AP^{j,iter} \\ a \text{ random position}, & \text{otherwise} \end{cases} \tag{2.7}$$

In this equation, $AP^{j,iter}$ represents the awareness probability of crow j, which plays a significant role in the trade-off between convergence and diversity. Lower AP values encourage local search and faster convergence. On the other hand, higher values promote global exploration and increase population diversity. Despite its advantages, CSA has several limitations. The random movement during global search can make the optimization process inefficient and slow to converge. Additionally, the random initialization of

the population may lead to clustering. This way, it increases the possibility of falling into local optima. Another drawback is the use of a fixed flight length, which is not suitable for all stages of the search process. Smaller step sizes are preferable in later iterations for fine-tuning. To overcome these issues, an improved CSA-based extreme learning machine (PSO-CSA-ELM) is proposed by integrating particle swarm optimization strategies. The enhancements include initialization with tent chaotic mapping combined with opposition-based learning to improve population diversity. Adaptive adjustment of flight length, decreasing over iterations to balance exploration and exploitation. Finally, incorporation of PSO-inspired search mechanisms to expand the global search capability and improve convergence speed. These modifications significantly enhance CSA's performance when applied to optimize extreme learning machines (L. Cao et al. 2021).

Neural network ensembles are widely regarded as powerful and effective approaches for classification tasks. High performance of these network architectures requires the simultaneous optimization of the architectures and weight parameters of all constituent neural networks. This ensures an appropriate balance between accuracy and diversity among the base models. ELM is a fast, non-iterative training method that has attracted considerable interest due to its strong generalization ability. ELM is primarily suited to neural networks with fixed structures. To address these challenges, a novel ensemble learning framework combining multimodal differential evolution and ELM is proposed by Ma et al. (2025). This approach optimizes the structural configurations and weight parameters of all base networks within the ensemble in a single optimization process. In the proposed method, a specialized encoding strategy is employed to represent the architecture and input parameters of the base networks. The output weights of the ELM-trained network are computed analytically. Moreover, the multimodal differential evolution algorithm is further enhanced to better handle the high-dimensional optimization requirements inherent in neural network ensemble design. This enhancement leads to improved classification performance.

Mengcan et al. (2021) proposed an enhanced approach termed constrained voting ELM. This method constructs a set of difference vectors from samples belonging to different classes. Then, it uses these vectors to determine the hidden-layer weights and thresholds. Rather than relying on a single model, multiple independent ELMs are trained and integrated into an ensemble framework. Final predictions are obtained using a majority voting scheme. Weight vectors between the input and hidden layers are computed based on difference vectors formed from inter-class samples. This way, they reduce the dependency on randomly initialized hidden-layer parameters. An ensemble learning strategy is employed by integrating multiple independent ELM models. The decisions are made using a majority voting mechanism.

Although these methods demonstrate strong performance, they remain constrained to conventional low-order neuron models and do not use higher-order units.

2.2 Weight and Threshold Optimization Methods for Specific Problems

A series of papers proposed ELM optimization methods for specific problem types. Ahmad et al. (2026) introduced a hybrid forecasting approach that combines the artificial hummingbird algorithm (AHA) with traditional ELM to achieve high-resolution DC power prediction for retrofitted PV installations. The proposed AHA-ELM method employs foraging strategies. These strategies are inspired by biology to optimize ELM hidden-layer parameters, thereby mitigating issues associated with random initialization and unstable predictions. Precise short-term prediction of photovoltaic power generation is essential for maintaining a stable grid with effective power management and seamless inclusion of renewable energy sources. Retrofitted rooftop PV systems introduce significant forecasting challenges. These challenges are related to site-specific factors, including inconsistent tilt angles, partial shading, aging components, and irregular system configurations.

The AHA is an optimization method inspired by biology designed to replicate the searching behavior of hummingbirds. It is based on three primary flight patterns (axial, diagonal, and omnidirectional movements). It also includes territorial and migratory foraging strategies. The algorithm simulates how hummingbirds memorize food source locations and dynamically adjust their search behavior to locate optimal resources. These natural mechanisms are translated into an optimization framework for identifying global optima in complex search spaces.

Consider N artificial hummingbirds with every individual representing a candidate solution in a search space with specific dimension D. The positions at the algorithm's beginning are randomly initialized using formula 2.8 where X_i^0 denotes the position of the i-th hummingbird, LB and UB represent the lower and upper bounds of the search space, and rand(D) is a vector of uniformly distributed random numbers in $[0, 1]$.

$$X_i^0 = LB + (UB - LB)\text{rand}(D), \ldots i = 1, 2, \ldots, N \tag{2.8}$$

Axial flight represents movement along a single dimension d that is randomly selected, and the position update is defined in equation 2.9 where α is a scaling factor, randn() generates a normally distributed random number, and d is the randomly chosen dimension.

$$X_i^{t+1}(d) = X_i^t(d) + \alpha \cdot \text{rand}(UBd - LBd) \tag{2.9}$$

In diagonal flight, the hummingbird moves across all dimensions according to the following

formula, where β is a scaling parameter responsible for adjusting the flight range.

$$X_i^{t+1} = X_i^t + \beta \cdot \text{rand}(D), (UB - LB) \tag{2.10}$$

Omnidirectional flight enables global exploration of the search space through random perturbations according to equation 2.11 where γ determines the exploration intensity.

$$X_i^{t+1} = X_i^t + \gamma(\text{rand}(D) - 0.5), (UB - LB) \tag{2.11}$$

To enhance local exploitation around promising regions, the territorial foraging strategy is applied as follows, where X_{best}^t is the best solution identified so far, and δ is a small parameter controlling local search.

$$X_i^{t+1} = X_{\text{best}}^t + \delta \cdot \text{rand}(D), (UB - LB) \tag{2.12}$$

When the search process stagnates, a migration mechanism is triggered to encourage exploration (formula 2.13), where X_r^t is a randomly selected hummingbird from the population, and $\epsilon \in [0, 1]$ is a random control parameter.

$$X_i^{t+1} = X_i^t + \epsilon \left(X_r^t - X_i^t \right) \tag{2.13}$$

By emulating directed, regional, and relocation foraging behaviors, AHA effectively balances exploration and exploitation during optimization. Its governing equations range from position updates based on directions to probabilistic, memory-driven search, allowing the algorithm to adapt dynamically and converge toward the global optimum. Additionally, the use of uniform and Gaussian randomization enhances population diversity and reduces the risk of premature convergence (Ahmad et al. 2026; Zhao et al. 2022).

Rao et al. (2025) introduces a novel forecasting methodology aimed at improving prediction accuracy in future trends from past data. The proposed pelican-optimized ELM (PO-ELM) enhances the standard ELM by incorporating the pelican optimizer (PO) to determine key parameters, including hidden-layer weights and thresholds. In contrast to traditional ELM (where these parameters are randomly assigned), the PO-ELM approach optimizes them, resulting in more accurate and reliable predictions. This improvement substantially strengthens the model's forecasting performance, particularly when applied to highly complex time series data such as stock market signals.

PO by Trojovský and Dehghani (2022) is a recent metaheuristic recognized for its effective exploration capabilities and computational efficiency. Inspired by pelicans' hunting strategies, the algorithm emulates their systematic search for food by initiating the optimization process with a set of randomly generated candidate solutions. These initial

solutions are represented in matrix form, which serves as the basis for subsequent iterative refinement. The size of the solution matrix x_j^i is determined by the n parameter, which defines the population size of the PO algorithm, and the m parameter, which defines the number of decision variables in the cost function. The initial population generated by the PO algorithm is expressed in the following formula 2.14.

$$\begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_d^1 \\ x_1^2 & x_2^2 & \cdots & x_d^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^n & x_2^n & \cdots & x_d^n \end{bmatrix} \tag{2.14}$$

These candidate solutions are randomly generated within the predefined lower (l) and upper (u) bounds of the decision variables in the search space. The initialization process is described in equation 2.15 where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.

$$x_i^j = l_j + \text{rand}(u_j - l_j) \tag{2.15}$$

With the candidate solutions from equation 2.15, their fitness values are evaluated and the PO objective-function vector $Ojnew = \begin{bmatrix} Oj_1 \\ Oj_2 \\ \vdots \\ Oj_n \end{bmatrix}$ is formed.

Pelicans do not always obtain food during their initial movements, so they adjust their hunting strategy to locate a food source. This behavior is reflected in the PO mechanism, where the optimal solution is not guaranteed by the initial population. The positions x_j^i are updated in two stages that correspond to exploration and exploitation. In stage 1, pelicans move toward the prey location according to the following formula.

$$x_j^{i(p1)} = \begin{cases} x_j^i + \text{rand}\left(p_j - I x_j^i\right), & Oj_p < Oj_i \\ x_j^i + \text{rand}\left(x_j^i - p_j\right), & \text{else} \end{cases} \tag{2.16}$$

In formula 2.16, $x_j^{i(p1)}$ denotes the updated position of x_j^i after stage 1. The prey location corresponding to the jth decision variable is represented by p_j, while I is a binary value, taking either 1 or 2. This procedure is applied to all individuals to update X_i during stage 1. In stage 2, the position update equation is given by equation 2.17

$$x_j^{i(p2)} = x_j^i + R\left(1 - \frac{t}{T}\right)(2\text{rand} - 1)x_j^i \tag{2.17}$$

This procedure is applied to all individuals to update X_i in stage 2, using the parameters R (a constant set to 0.2), t (the current iteration number), and T (the total number

of iterations). The process terminates when the current iteration reaches the maximum number of iterations (Xue and Shen 2020).

Lian (2025) proposed a model based on an improved sparrow search algorithm (SSA) to optimize ELM's hidden parameters. The proposed system was used to achieve precise ultra-short-term wind speed forecasting, thereby enhancing the grid integration efficiency. Also lowers the costs of wind farms and ensures their stable operation. Building upon the standard SSA, three enhancement strategies are introduced (population initialization using a piecewise chaotic map, an improved position update mechanism for discoverers, and an enhanced optimal value search strategy). These modifications promote a more uniform initial population distribution. Also, they strengthen global search capability and enhance the algorithm for avoiding local optima.

The sparrow population in the SSA consists of three roles: discoverers, followers, and emergency responders. The process begins with population initialization and assumes the population contains N individuals, while the optimization problem has d decision variables. Assumong all the above, the total population X can be expressed with the following equation 2.18

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,d} \end{bmatrix} \tag{2.18}$$

Afterwards, it updates the discoverer position. The discoverer searches for food resources on behalf of the entire population. It also communicates food locations to followers. The update rule for the discoverer's position is given in formula 2.19, where t is the current iteration number, i is the sparrow index, j denotes the dimension ($j = 1, 2, \ldots, d$), and ϑ is the maximum allowed iterations. $X_{i,j}^{t+1}$ represents the updated position of the ith sparrow in the jth dimension. K_1 and K_2 are randomly chosen variables following the uniform distribution in $[0, 1]$, while K_3 is the early-warning threshold, typically a random value between 0.5 and 1. The variable q is also uniformly distributed, and L is a $1 \times D$ matrix where its entries are all 1. When $K_2 < K_3$, the population is regarded as being in a safe state, so the discoverers can search over a wider area. When $K_2 \geq K_3$, some sparrows have detected predators and alerted the rest of the population.

$$X_{i,j}^{t+1} = \begin{cases} X_{i,j}^{t} e^{-\frac{i}{K_1 \vartheta}}, & K_2 < K_3, \\ X_{i,j}^{t} + qL, & K_2 \geq K_3. \end{cases} \tag{2.19}$$

Besides the discoverers, the followers are the sparrows that remain in the population. When the discoverers locate additional food, the followers quickly move from their current positions to compete for the available food. Their position update rule is given in equation 2.20, with i denoting the ith follower, N denoting the total sparrows, and j rep-

resenting the dimension of followers. X_o^t is the discoverer'w optimal position at iteration t, and X_w denotes the poorest sparrow position. B is a $D \times 1$ dimension matrix, and $B^+ = B^T(BB^T)^{-1}$ is the pseudo-inverse of B. When $i > \frac{n}{2}$, the sparrow with the poorest position must move from its current location and search other places for food.

$$X_{i,j}^{t+1} = \begin{cases} q\, e^{\frac{X_w - X_{i,j}^t}{i^2}}, & i > \frac{n}{2}, \\ X_o^{t+1} + \left| X_{i,j}^t - X_o^{t+1} \right| B^+ L, & \text{otherwise.} \end{cases} \tag{2.20}$$

Predator attacks are more likely to happen when emergency responders are outside the population. Every time danger is perceived, the emergency responder continuously changes its position to avoid being captured by the predator. The emergency responder's position update rule is given in equation 2.21, where λ is drawn from a Gaussian distribution with mean 0 and variance 1. It also serves as the step-size control parameter. X_b represents the current global optimal position. The sparrow trajectory and step-length control parameter are governed by a random variable θ in the range $[-1, 1]$. In addition, h_w denotes the current global minimum fitness value, h_i is the fitness value of the ith sparrow, h_g is the current global maximum fitness value, and σ is a small constant introduced to prevent division by zero. When $h_i > h_g$, sparrows occupy the most peripheral region of the population, making them easier to detect and capture by predators. When $h_i = h_g$, the sparrows at the population's center sense imminent danger, so they move closer to other sparrows to reduce predation risk.

$$X_{i,j}^{t+1} = \begin{cases} X_b^t + \lambda \left| X_{i,j}^t - X_b^t \right|, & h_i > h_g, \\ X_{i,j}^t + \theta \left(\dfrac{\left| X_{i,j}^t - X_w^t \right|}{(h_i - h_w) + \sigma} \right), & h_i = h_g \end{cases} \tag{2.21}$$

The improved SSA version includes three enhancements to circumvent the local extremum issue (Tian and Chen 2021). The first strategy uses a piecewise chaotic map to generate the initial population. Since SSA relies on random generation to initialize the population, the resulting distribution is often uneven. This can significantly affect the quality of the final search results. A commonly used method for improving population diversity in optimization algorithms is opposition learning. Although opposition learning improves search diversity and global exploration by generating opposite solutions, its effectiveness depends heavily on the symmetry of the search space and the sensitivity of its parameters. Improper parameter settings may cause the search process to become trapped in a local optimum (Q. Liu et al. 2024). To overcome this issue, a piecewise chaotic map is employed for population initialization, thereby improving its initial distribution. The calculation equation is given below, with c denoting the constant 0.4 and j

representing every sparrow's dimension.

$$x(j+1) = \begin{cases} \dfrac{x(j)}{c}, & 0 \le x(j) < c, \\ \dfrac{x(j)-c}{-(c-0.5)}, & c \le x(j) < 0.5, \\ \dfrac{-(c-1)-x(t)}{-(c-0.5)}, & 0.5 \le x(j) < -(c-1), \\ \dfrac{1-x(t)}{c}, & -(c-1) \le x(j) < 1. \end{cases} \tag{2.22}$$

The second is the improved discoverer position update. During the optimization process, the standard SSA is more likely to become trapped in a local optimum. To improve its global search capability, a cubic function is incorporated. At the early stage of iteration, the discoverer takes a higher value, whereas it becomes smaller toward the end of the iteration, as seen in formula 2.23, where K_2 denotes the safety threshold, and K_3 represents the early-warning threshold, which follows a uniform distribution. q is a random number uniformly distributed in the interval, and L is a $1 \times d$ matrix whose entries are 1. The improved discoverer provides strong global search capability in the early iterations and enhances the exploration ability of the original SSA compared with the pre-improvement version.

$$X_{i,j}^{t+1} = \begin{cases} X_{i,j}^{t} e^{-15\left(\frac{i}{2\vartheta}\right)^3}, & K_2 < K_3, \\ X_{i,j}^{t} + qL, & K_2 \ge K_3. \end{cases} \tag{2.23}$$

The third and final improvement is the optimal value search strategy. The standard SSA is susceptible to becoming trapped in local optima, so an optimal-value search strategy is introduced to address this issue. After each iteration, the algorithm explores the neighborhood of the current optimum, and if a better value is found, it replaces the existing one. The updated rule is given in equation 2.24, where X_b denotes the current global optimal position, F_g is the current global best fitness value, and F_i is the fitness value of the ith sparrow. When $F_i > F_g$, the algorithm identifies a superior position and updates it accordingly. Otherwise, the individual's position remains unchanged.

$$X_g^{t+1} = \begin{cases} X_b^t \left(1 + 0.08 \sin\left(\dfrac{t}{\vartheta}\right)\right), & F_i > F_g \\ X_b^t, & F_i \le F_g \end{cases} \tag{2.24}$$

Although these methods demonstrate strong performance, they remain constrained to conventional low-order neuron models and do not use higher-order units.

2.3 ELM-based Methods in the Educational Domain

A series of ELM-based methods are used in the educational domain. The study from Viswakiran et al. (2024) proposes applying ELM to model and predict student outcomes to facilitate the development of decision-support tools for enhanced academic achievement and personalized learning pathways. Accurate prediction of student performance has emerged as a pivotal challenge in contemporary educational systems, requiring the adoption of sophisticated computational methodologies. Despite ongoing advancements, there is currently no universally optimal approach to forecast all variables associated with academic performance. Due to their computational efficiency and scalability in large datasets, ELM-trained methods are particularly advantageous for early preference modeling and adaptive educational support. The predictive modeling pipeline encompasses data acquisition, preprocessing, and feature engineering.

Similarly, the article from Sa'ad, Mustafa, et al. (2020) utilized the original ELM algorithm to predict student dropout in the education management PhD program at FKIP Mulawarman University. The utilized dataset comprises 110 student records from the 2012–2018 cohorts of the above PhD program. The prediction model incorporates five input variables. These variables are gender, third-semester IP value, age, employment, and family status. It also has two output classes (dropout and non-dropout).

The article from H. Gao et al. (2025) used data from a dataset of 873 full-time postgraduate students at universities in Zhejiang Province, China. It contains questionnaire response data and the students' grade point averages (GPAs) for one academic year. Within China's graduate education system, mentors are the faculty members who engage most frequently with postgraduate students and establish the closest academic relationships. In recent years, issues surrounding mentor–student interactions have become increasingly prevalent. This leads to a discernible decline in the quality of graduate education. The proposed study explores the impact of the mentor–postgraduate relationship on students' academic performance. It considers admission motivation and learning pressure as moderating variables. In practical data collection, outliers are common and can significantly affect both the convergence rate and predictive performance of machine learning models. The authors addressed this issue by proposing a novel robust kernel ELM (RK-ELM) method designed to handle outliers. The RK-ELM model can automatically identify data points that may be affected by uncertain disturbances, thereby improving its robustness and generalization.

The study investigates how the relationship between mentors and postgraduate students affects academic performance and incorporates enrollment motivation and learning pressure as moderating factors. During data acquisition, the presence of outliers not only slows the convergence of machine learning algorithms but also negatively impacts their predictive performance.

To mitigate the influence of such anomalies, the RK-ELM model extends the conventional kernel ELM (KELM) framework. KELM is a kernelized version of extreme learning machine that retains the fast-training philosophy of ELM while replacing the explicit hidden-layer mapping with a kernel function. This allows the model to compute predictions using kernel similarities between samples, so it can handle nonlinear patterns without tuning hidden-layer weights one by one (Huang et al. 2012). The RK-ELM can automatically identify outliers under uncertain perturbations, thereby enhancing both robustness and generalization.

The training dataset is defined as $X = \{x_n\}_{n=1}^{N} \in \mathbb{R}^{N\times D}$ and $Y = \{y_n\}_{n=1}^{N} \in \mathbb{R}^{N\times K}$, with kernel function $K(x, x')$. The RK-ELM objective function combines a regularization component with a robust error term, as seen in formula 2.25

$$\min_{\beta',\varepsilon} \frac{1}{2}\mathrm{Tr}\left(\beta'^{T}K(\tilde{X},\tilde{X})\beta'\right) + \frac{C}{2}\sum_{n}\mathbf{1}_{\mathrm{MSE}_n\leq\varepsilon}\left\|K(x_n,\tilde{X})\beta' - y_n\right\|_2^2 \tag{2.25}$$

β' denotes the output weight matrix, C is the regularization coefficient, and ε is a threshold determined by the prediction errors. The set $\tilde{X}$ represents the subset of training samples after excluding detected outliers. The prediction error for the n-th sample is defined in formula 2.26.

$$\mathrm{MSE}_n = \left\|K(x_n,\tilde{X})\beta' - y_n\right\|_2^2 \tag{2.26}$$

Assuming that a proportion $r\%$ of the samples are anomalous, the indicator function $\mathbf{1}_{\mathrm{MSE}_n\leq\varepsilon}$ determines whether a sample is considered normal or an outlier. Specifically, if $\mathrm{MSE}_n \geq \varepsilon$, then $\mathbf{1}_{\mathrm{MSE}_n\leq\varepsilon} = 0$, indicating an outlier; otherwise, $\mathbf{1}_{\mathrm{MSE}_n\leq\varepsilon} = 1$, indicating a valid sample. From the formulation above, two key observations can be made. First, compared to the standard KELM, the RK-ELM introduces an embedded anomaly detection mechanism, which enhances model performance and can be integrated into other KELM-based extensions. Second, the optimization process involves solving for β' and ε, where ε is adaptively estimated based on the predefined outlier ratio $r\%$ (H. Gao et al. 2025).

The article from Bharathi and Manavalan (2026) proposes a hybrid prediction framework that integrates the weighted ELM (WELM) with the coati optimization Algorithm (COA). Predicting student performance constitutes a fundamental task in educational data mining. Such predictive capabilities provide critical insights into students' mastery of course content and the effectiveness of instructional strategies. Beyond facilitating timely feedback for learners and educators, accurate performance prediction also enables institutional stakeholders to evaluate and enhance course quality. The proposed architecture extends the conventional ELM and its weighted variant by incorporating a systematic analysis of interactions between course-related and student-specific attributes. A key limitation of ELM-based models lies in the random initialization of hidden-layer parameters.

They can induce variability across training runs and degrade predictive performance. To mitigate this problem, the proposed system employs COA to optimize the hidden-layer parameters of WELM. This way, it improves model stability and accuracy.

WELM represents an enhanced variant of conventional ELM. It is designed to address limitations associated with noisy or improperly balanced datasets. WELM introduces a weighting scheme that assigns distinct importance values to individual data instances based on their reliability or relevance. This mechanism enables the system to prioritize complex or informative samples. This way, it improves its generalization capability and robustness. Instance-specific weights are computed for each sample in the imbalanced dataset and subsequently employed during the prediction of unseen instances. Furthermore, differential weighting strategies are applied across classes to address disparities in class distribution. The underlying objective of WELM is to minimize a weighted loss function, where the contribution of each training instance is modulated by its assigned weight (K. Li et al. 2014; Zong et al. 2013).

The COA by Dehghani et al. (2023) is a biologically inspired metaheuristic method developed to tackle a variety of optimization problems. COA emulates the foraging behavior of coatis, which are small mammals renowned for their acute olfactory capabilities and efficient food-source localization. By balancing exploration and exploitation within the search space, the method demonstrates strong performance in identifying high-quality optimal solutions.

The COA comprises three primary stages (initialization, exploration, and exploitation). In the first stage, each coati corresponds to an individual member of the population. Within the search space, the position of a coati encodes the values assigned to the decision variables. Each position corresponds to a candidate solution for the optimization problem. At the initialization stage of the COA, the positions of all coatis are randomly generated throughout the search space in accordance with equation 2.27, where:

- X_i is the ith positioned coati inside the search space.
- $x_{i,j}$ is the jth decision variable value.
- N corresponds to the total coatis population.
- m indicates the number of decision variables.
- r is a randomly chosen real number drawn between 0 and 1.
- lb_j and ub_j denote the lower and upper bounds, respectively, of the jth decision variable.

$$X_i : x_{i,j} = lb_j + r(ub_j - lb_j), \quad i = 1, 2, \ldots, N, \; j = 1, 2, \ldots, m, \tag{2.27}$$

The coati population is formally represented by the matrix depicted in formula 2.28.

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_i \\ \vdots \\ X_N \end{bmatrix}_{N\times m} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,j} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i,1} & \cdots & x_{i,j} & \cdots & x_{i,m} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,j} & \cdots & x_{N,m} \end{bmatrix}_{N\times m} \tag{2.28}$$

The placement of candidate solutions in the decision variable space yields distinct objective function values for the optimization problem. These values are formulated in formula 2.29, where F is the vector of evaluated objective function values and F_i corresponds to the objective function value associated with the ith coati.

$$F = \begin{bmatrix} F_1 \\ \vdots \\ F_i \\ \vdots \\ F_N \end{bmatrix}_{N\times 1} = \begin{bmatrix} F(X_1) \\ \vdots \\ F(X_i) \\ \vdots \\ F(X_N) \end{bmatrix}_{N\times 1} \tag{2.29}$$

A candidate solution's quality is evaluated according to the value of its objective function. As a result, the population member that produces the most favorable objective function value is chosen as the population's best member. Since candidate solutions are updated iteratively throughout the algorithm's execution, the best population member is similarly updated at each iteration.

The position update mechanism in the COA draws inspiration from two innate coati behaviors: their predatory hunting strategy when pursuing iguanas and their evasive escape response when confronted by predators. Consequently, the COA population is updated via two distinct phases (exploration and exploitation). The exploration phase emulates the hunting strategy, while the exploitation phase mirrors the escape strategy.

During the exploration phase, the coati population is partitioned into two subgroups: the first ascends trees to startle their prey, while the second remains stationed beneath the trees in anticipation of the prey's descent. This cooperative hunting strategy facilitates effective exploration of the search space. Within this framework, the prey is modeled as an iguana, with its position assumed to coincide with that of the best-performing coati. The climbing behavior of coatis is mathematically described by formula 2.30.

$$\begin{aligned} &X_i^{P1} : x_{i,j}^{P1} = x_{i,j} + r\left(Iguana_j - Ix_{i,j}\right) \\ &\text{for } i = 1, 2, \ldots, \left\lfloor \frac{N}{2} \right\rfloor \text{ and } j = 1, 2, \ldots, m \end{aligned} \tag{2.30}$$

Upon the iguana's fall to the ground, it is repositioned randomly within the search space. In response to this randomized location, the coatis positioned on the ground update their locations accordingly, a process modeled using equations 2.31 and 2.32.

$$Iguana^G : Iguana_j^G = lb_j + r\left(ub_j - lb_j\right), \quad j = 1, 2, \ldots, m \tag{2.31}$$

$$\begin{aligned} X_i^{P1} : x_{i,j}^{P1} &= \begin{cases} x_{i,j} + r\left(Iguana_j^G - Ix_{i,j}\right), & F_{Iguana^G} < F_i \\ x_{i,j} + r\left(x_{i,j} - Iguana_j^G\right), & \text{otherwise} \end{cases} \\ \text{for } i &= \left\lfloor \frac{N}{2} \right\rfloor + 1, \left\lfloor \frac{N}{2} \right\rfloor + 2, \ldots, N \text{ and } j = 1, 2, \ldots, m \end{aligned} \tag{2.32}$$

Every coati's updated position is retained only if it improves the objective function value. Alternatively, the coati retains its prior position. This acceptance criterion is applied across all $i = 1, 2, \ldots, N$ and is formally defined in formula 2.33. In this formulation, $X^{P1}i$ defines the candidate position generated for the ith coati, with $x^{P1}i, j$ representing its jth decision variable and F_i^{P1} its corresponding objective function value. The parameter r is a random number between 0 and 1 that follows the uniform distribution. The term $Iguana$ refers to the position of the best-performing individual in the population, where $Iguana_j$ indicates its jth dimension. Variable I is a randomly chosen integer from the $\{1, 2\}$ set. Furthermore, $Iguana^G$ denotes a randomly generated ground position for the iguana, with $Iguana_j^G$ representing its jth dimension and F_{Iguana^G} its associated objective function value. The operator $\lfloor \cdot \rfloor$ denotes the floor function Dehghani et al. (2023).

$$X_i = \begin{cases} X_i^{P1}, & F_i^{P1} < F_i \\ X_i, & \text{otherwise} \end{cases} \tag{2.33}$$

The second phase of the position update process for coatis within the search space is mathematically formulated by emulating their natural response to predator encounters. Upon detecting a predator, a coati rapidly relocates to evade the threat. This movement typically results in a new position in close proximity to the original location, reflecting the COA's exploitation capability for local search. To model this behavior, a stochastic position is generated in the vicinity of each coati's current location, as defined by formulas 2.34 and 2.34.

$$lb_j^{local} = \frac{lb_j}{t}, ub_j^{local} = \frac{lb_j}{t}, \text{ where } t = 1, 2, \ldots, T \tag{2.34}$$

$$\begin{aligned} X_i^{P2} : x_{i,j}^{P2} = x_{i,j} + (1 - 2r)\left(lb_j^{local} + r\left(ub_j^{local} - lb_j^{local}\right)\right), \\ i = 1, 2, \ldots, N, j = 1, 2, \ldots, m \end{aligned} \tag{2.35}$$

The updated position is retained only if it results in an improvement in the objective function value (equation 2.36). With this formulation, X_i^{P2} denotes the newly generated

position for the ith coati during the exploitation stage of the optimization algorithm. The $x_{i,j}^{P2}$ represents its jth dimension, and F_i^{P2} corresponds to the associated objective function value. Parameter r is a randomly chosen number between 0 and 1 that follows the uniform distribution. t indicates the current iteration, while lb_j^{local} and ub_j^{local} define the local search bounds for the jth variable. The global bounds are given by lb_j and ub_j (Dehghani et al. 2023).

$$X_i = \begin{cases} X_i^{P2}, & F_i^{P2} < F_i, \\ X_i, & \text{otherwise} \end{cases} \tag{2.36}$$

The article by Ye et al. (2025) optimized the KELM algorithm with an enhanced version of the fruit fly optimization algorithm (FOA). The FOA was augmented with the covariance matrix adaptation evolution strategy (CMA-ES) and also combined with quadratic approximation (QA). The research study utilized 3278 questionnaire responses collected from seven universities across four Chinese provinces. The aim of the questionnaires was to identify the principal factors affecting students' academic performance.

The FOA by Xing and W.-J. Gao (2013) is a population-based global optimization approach inspired by the food-searching behavior of fruit flies. It mimics their characteristic foraging strategy by integrating olfactory and visual search mechanisms. At the algorithm's initial stage, it employs olfactory sensing to perceive food-related odors. This procedure enables individuals to estimate movement direction based on the gradient of scent concentration, without requiring prior knowledge of the exact food location. Upon identifying the position with the highest scent intensity, a fruit fly effectively signals this optimal location with the purpose of attracting neighboring individuals. Subsequently, the surrounding population utilizes visual cues to move directly toward this position, resulting in convergence at the identified place.

The algorithm begins by determining the maximum number of allowed iterations and the initial population's size. The initialization phase involves defining the population size and the maximum number of iterations. Each individual in the fruit fly population is represented in a two-dimensional search space by the coordinate pair (X_{axis}, Y_{axis}). During the olfactory search phase, each fruit fly moves randomly in both direction and distance. The X and Y positions of the i-th individual are updated according to the following formulas.

$$\begin{aligned} X_i &= X_{axis} + \text{RandomValue} \\ Y_i &= Y_{axis} + \text{RandomValue} \end{aligned} \tag{2.37}$$

Since the precise location of the food source is unknown, the equation 2.38 calculates the distance D_i between each individual and the origin.

$$D_i = \sqrt{X_i^2 + Y_i^2} \tag{2.38}$$

The corresponding food odor concentration judgment value is calculated using formula 2.39.

$$S_i = \frac{1}{D_i} \tag{2.39}$$

The perceived odor concentration of each individual ($Smell_i$) is obtained by evaluating the following fitness function.

$$Smell_i = fitness(S_i) \tag{2.40}$$

The individual with the highest odor concentration is identified in equation 2.41.

$$[bestSmell, bestIndex] = \max(Smell) \tag{2.41}$$

The coordinates of the best-performing individual are retained, and the rest of the population is directed toward this position as seen in the equations below.

$$\begin{aligned} Smell_{best} &= bestSmell \\ X_{axis} &= X_{bestIndex} \\ Y_{axis} &= Y_{bestIndex} \end{aligned} \tag{2.42}$$

The procedure after the algorithm's initialization is repeated to evaluate whether the newly obtained odor concentration improves upon that of the previous generation. If an improvement is observed, the population is updated accordingly.

One limitation of FOA is its premature convergence, which may lead to becoming trapped in local optima. The limited and relatively rigid movement patterns of its agents hinder comprehensive exploration of the solution space. This way, it reduces the likelihood of identifying the global optimum. Moreover, the absence of adaptive mechanisms to dynamically adjust search strategies according to problem characteristics further limits its effectiveness in challenging environments.

To address these limitations, an enhanced position update mechanism is designed to improve the interaction between search strategies and the algorithm while maintaining a balance between global exploration and local exploitation. In this study, the CMA-ES and a QA-based strategy are incorporated to strengthen the performance of FOA. The integration of CMA-ES enables adaptive adjustment of the population distribution and covariance matrix. This way, it guides the search direction and accelerates convergence toward the global optimum. Additionally, the QA strategy enhances the algorithm's ability to accurately locate optimal solutions within the global search space (Ye et al. 2025).

Wei et al. (2020) proposed an intelligent predictive framework for assessing entrepreneurial

intention. It aims to support talent cultivation programs and provide informed guidance for fostering students' entrepreneurial tendencies. The proposed model is built upon a KELM optimized using an enhanced Harris hawk's optimization (HHO) algorithm. A Gaussian barebones (GB) strategy is incorporated into HHO to improve parameter tuning and feature selection. This strategy enhances its optimization capability to identify optimal parameter settings and compact feature subsets. Based on these optimized components, a hybrid model, termed GBHHO-KELM, is constructed for entrepreneurial intention prediction.

In the HHO algorithm, the prey represents the current best solution, while the remaining candidate solutions correspond to the positions of the hawks. The algorithm models two primary exploration strategies. In the first one, hawks update their positions based on the locations of other individuals and the prey. In the second, a hawk selects a random perch, which simulates observation from a random position. The choice between these strategies is governed by a random parameter q. he position update rule for HHO that integrates these mechanisms is given by formula 2.43. In the HHO framework, X_{t+1} denotes the updated position at the following iteration, X_t is the current position of Harris hawks, X_m is the mean position of the Harris hawks, and X_p is the position of the prey. The parameters r_1, r_2, r_3, r_4, and q are random variables uniformly distributed in the interval $(0, 1)$. The symbols lb and ub represent the lower and upper bounds for every dimension. The incorporation of stochastic components in the update rules enhances exploration capability and helps maintain population diversity. This way, it effectively mimics the cooperative hunting behavior of Harris hawks.

$$X_{t+1} = \begin{cases} X_{\text{rand}} - r_1 \left| X_{\text{rand}} - 2r_2 X_t \right|, & q \geq 0.5 \\ X_p - X_m - r_3 \left(lb + r_4(ub - lb) \right), & q < 0.5 \end{cases} \tag{2.43}$$

The average position of the population is denoted in equation 2.44 where X_i denotes the i-th hawk's position in the current iteration, and N is the total number of Harris hawks.

$$X_m = \frac{1}{N} \sum_{i=1}^{N} X_i \tag{2.44}$$

During the exploitation (hunting) phase, the algorithm selects different strategies based on the escaping energy of the prey, which decreases over iterations. The escaping energy is modeled in formula 2.45, where E represents the escaping energy, T is the maximum number of iterations, and $E_0 \in (0, 1)$ is the initial escaping energy, randomly generated and updated in each iteration.

$$E = 2E_0 \left(1 - \frac{t}{T} \right) \tag{2.45}$$

HHO employs multiple strategies to simulate various attack scenarios, depending on the

prey's escape behavior and the hawks' hunting tactics. A random variable r is used to determine whether the prey escapes. When $r < 0.5$, the prey escapes successfully. Otherwise, it is captured. The magnitude of the escaping energy $|E|$ governs the type of besiege strategy. A soft besiege is applied when $|E| \geq 0.5$, whereas a hard besiege is adopted when $|E| < 0.5$ (Heidari et al. 2019; Wei et al. 2020).

In this study, a GB strategy is incorporated into the traditional HHO approach to accelerate convergence and enhance solution accuracy. The GB strategy aims to guide population members toward promising search directions so that they do not get prematurely trapped into a local solution. In this proposed work, an update mechanism is determined by a probability coefficient C_T. When the probability generated is smaller than CT, the position of a Harris hawk will be updated via a Gaussian distribution. Otherwise, when the probability generated is larger than or equal to CT, a DE-based strategy will be used to produce a new candidate solution. The Gaussian-based updating strategy and the DE mechanism utilize information from the current position of a Harris hawk and the globally best solution identified thus far to determine a new search position. The proposed GB-enhanced HHO can thus make better trade-offs between exploration and exploitation, thus providing more efficiency and quality while offering fast computation and high accuracy (Wei et al. 2020).

Despite their strong performance, these approaches are still limited to conventional low-order neuron models and do not incorporate higher-order processing units.

2.4 Alternative Higher-Order Neuron Types

Higher-order neuron models include the sigma-pi neurons, introduced by Rumelhart et al. (1986a), which are closely related to the conjunctive units proposed by Feldman and Ballard (1982). These models extend conventional low-order neurons by enabling multiplicative interactions among input variables. In these neuron types, the input is computed as a weighted sum of products of individual inputs. Although such architectures enhance neural networks' ability to model complex input relationships, they suffer from a significant limitation. This limitation is that the number of weights increases exponentially with the number of inputs.

To address this limitation, pi-sigma units were proposed (Shin and Ghosh 1991). These units exploit higher-order correlations among input components to construct nonlinear discriminant functions within a single-layer structure. The underlying computational elements, referred to as higher-order processing units, incorporate input correlations of higher degree and serve as the foundation for these network architectures. In a pi-sigma network, a K-th order processing unit models its input as the product of K linear combinations of the original inputs, using product units as outputs. This design preserves the expressive power of higher-order models while significantly reducing the

number of weights and computational units required.

The multi-cube unit is more advanced compared to the pi-sigma unit because it allows inputs to be partitioned into groups, each assigned to a sub-cube unit. This structure enables selective adjustment of the number of weights associated with specific input groups, thereby offering greater flexibility in controlling the dimensionality of the weight vector and improving model efficiency (Rumelhart et al. 1986a; Shin and Ghosh 1991).

2.5 Alternative Machine Learning Methods

The experimental section of this dissertation evaluated EHO-ELM against ten backpropagation variants, the decision tree (DT), ELM, OS-ELM, and MCU-ELM algorithms. Although Paul Werbos originally introduced the core idea of backpropagation in his PhD thesis (Werbos 1974, August), the method gained broad recognition several years later through the work of Rumelhart et al. (1986b), titled "Learning representations by back-propagating errors".

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm is a widely used quasi-Newton method for solving unconstrained nonlinear optimization problems. Unlike basic gradient-based approaches, BFGS leverages curvature information by iteratively approximating the Hessian matrix rather than computing second derivatives directly. Beginning with an initial estimate of the parameters, the algorithm updates its approximation of the inverse Hessian matrix using changes in both the parameter values and their gradients at each iteration. It then determines an appropriate search direction and step size via a line-search procedure. By combining fast convergence with relatively low computational cost, BFGS often outperforms standard gradient descent while remaining more efficient than full Newton methods. These properties make it particularly well suited for medium-scale optimization tasks and have led to its widespread use in machine learning, numerical analysis, and other scientific computing applications (Broyden 1970a,b; Fletcher 1970; Goldfarb 1970; Shanno 1970).

The Powell–Beale conjugate gradient (PBCG) method is an improved variant of the nonlinear conjugate gradient approach. It is designed to enhance the efficiency, stability, and reliability of unconstrained optimization. Due to its low memory requirements and strong convergence characteristics, it can be used in large-scale optimization problems. The Powell–Beale method incorporates an adaptive restart strategy that monitors the optimization process and modifies the search direction when required. This gives it an advantage over traditional optimization methods that may suffer from slow convergence or reduced effectiveness when the search direction becomes less informative. It also allows the algorithm to discard outdated search information and regain an effective direction toward the optimum. By introducing strategic restarts during the optimization procedure, the method reduces unnecessary iterations and improves convergence performance

(particularly in complex and highly nonlinear objective functions). The integration of conjugate gradient principles with the Powell–Beale restart technique provides a practical compromise between computational efficiency and numerical stability. It allows the algorithm to achieve a more reliable performance in challenging optimization scenarios. Because of its simplicity, limited computational demands, and improved convergence characteristics, the Powell–Beale conjugate gradient method has been successfully applied in various areas (e.g., scientific computing, engineering design optimization, and machine learning). It is a useful optimization technique for obtaining accurate solutions in complex numerical problems because its adaptive nature enables it to respond effectively to changes during the search process (M. J. D. Powell 1977).

The Fletcher–Powell conjugate gradient (FPCG) algorithm is an iterative optimization method widely used for training neural networks. The difference between FPCG and traditional steepest-descent backpropagation is that this method improves the search process by combining the present gradient information with the direction taken in previous iterations. The steepest descent updates the network parameters only according to the current gradient. The algorithm avoids repeatedly exploring the same areas of the error surface by generating conjugate search directions. It also enables a more efficient path toward the optimum solution. A line-search strategy is then applied to determine a suitable step length for each update. This search strategy allows the objective function to decrease effectively. At the same time, it maintains stable convergence and helps reduce the slow progress and oscillations often associated with conventional gradient-based training methods. The Fletcher–Powell approach requires only first-order gradient information and avoids computing or storing the Hessian matrix. For this reason, it provides a practical and computationally efficient alternative to Newton-based optimization techniques (Fletcher and Michael J. D. Powell 1963).

The Polak-Ribiére conjugate gradient backpropagation method is an iterative optimization approach used for training neural networks. It combines gradient information obtained from backpropagation with an efficient conjugate-direction search strategy. In contrast to conventional gradient-descent techniques that rely only on the current gradient for parameter updates, the Polak-Ribiére method incorporates changes in values of gradients observed between successive iterations. Using this additional information allows the algorithm to determine more effective search directions that reflect the recent behavior of the optimization process. This way, it reduces inefficient updates and improves convergence performance. A line-search procedure is employed at each training iteration to determine an appropriate step size along the selected direction, enabling the network parameters to move toward better solutions. The method has lower computational requirements than Newton-based optimization approaches because it does not require computing second-order derivatives or storing the Hessian matrix. Also, it provides a practical compromise between convergence efficiency and computational complexity.

For this reason, makes a suitable optimization method for training medium-sized neural networks with differentiable activation functions (Polak and Ribière 1969).

The scaled conjugate gradient (SCG) algorithm is a supervised optimization technique widely used for training neural networks. It improves the traditional conjugate gradient approach by incorporating a scaling mechanism that uses approximate second-order information to guide the optimization process. SCG estimates the local characteristics of the error surface and adjusts the parameter updates accordingly. This differentiates SCG from standard conjugate gradient methods, which often rely on a line-search procedure to determine the appropriate step size at each iteration. Moreover, it eliminates the need for repeated function evaluations, reduces computational effort, and improves training efficiency. SCG uses gradients computed through backpropagation to generate conjugate search directions during the learning process. Thereby, it enables more efficient updates of the network parameters and reduces unnecessary movements across the error surface. Since the method does not require the direct calculation or storage of the Hessian matrix, it maintains relatively low memory requirements while still benefiting from curvature information. As a result, SCG provides a practical and efficient alternative to conventional gradient-based training algorithms, especially for medium-sized feedforward neural networks where a good balance between convergence rate and computational complexity is needed (Møller 1993).

Gradient descent (GD) is one of the most widely used first-order optimization algorithms for training artificial neural networks. Its main purpose is to minimize a predefined performance function by continuously adjusting the network weights and biases in the direction that reduces the error. The gradient is calculated using the backpropagation process and provides information about how the performance function changes with respect to the network parameters. Since the gradient points toward the direction of maximum increase in the error, the algorithm updates the parameters in the opposite direction to gradually decrease the training loss. GD has a simple implementation and requires a relatively low computational effort compared to other backpropagation variants. Its main drawback is slow convergence in complex error landscapes. The choice of learning rate in GD plays a crucial role in the algorithm's performance. A too small learning rate can lead to unnecessarily slow training, while a value that is too large can prevent the optimization process from reaching an optimal solution (Cauchy 1847).

GD with momentum (GDM) improves conventional gradient descent by incorporating a memory term that considers previous weight updates when computing the current update. The algorithm maintains part of the previous movement direction. This allows the optimization process to continue moving effectively along directions that have consistently reduced the error and helps accelerate convergence while reducing unwanted oscillations. It is also particularly useful in complex error surfaces. The momentum mechanism enables the algorithm to move efficiently through regions where the gradi-

ent changes slowly. For this reason, it helps it overcome shallow minima and flat areas that may slow down standard GD. However, the performance of this method depends on the appropriate selection of the learning rate and momentum factor because unsuitable parameter settings may lead to slow convergence or unstable training behavior (Polyak 1964).

A further improvement is achieved by combining momentum with an adaptive learning rate strategy (GDMALR). The learning rate is automatically adjusted during training and follows the behavior of the performance function. It can be increased to speed up convergence when the optimization process consistently moves toward lower error values. When the error begins to increase, the learning rate is reduced to prevent excessive parameter changes and improve stability. This method provides a more flexible and robust training procedure compared with conventional gradient descent by integrating momentum-based acceleration with adaptive control of the update step size.(Endah et al. 2017).

The one-step secant (OSS) backpropagation algorithm is a quasi-Newton-based training method developed to enhance the convergence performance of traditional GD algorithms. It also avoids the high computational and memory demands associated with full second-order optimization techniques. The algorithm uses error gradients obtained through backpropagation during training to determine the direction in which the network weights and biases should be adjusted. OSS does not require storage and continuous updating of a complete Hessian matrix and uses a simplified secant approximation. In this approximation, the previous Hessian estimate is replaced by the identity matrix at each iteration. This approach enables the algorithm to incorporate some curvature information from the error surface while maintaining lower computational complexity and avoiding large matrix operations. A line-search mechanism is applied to determine a suitable step size along the selected search direction and allows the performance function to be reduced efficiently during training. As a result, the OSS algorithm provides a balance between conjugate-gradient methods and more advanced quasi-Newton approaches. It typically involves greater computational effort than conjugate-gradient algorithms but requires considerably fewer resources than methods such as BFGS (Ginantra et al. 2021).

Resilient backpropagation (RProp) is a first-order supervised learning algorithm designed to improve the efficiency and stability of training neural networks with multiple layers. It differs from conventional gradient descent (GD) methods, which determine the size of parameter updates based on the magnitude of the error gradient by using only the sign of the partial derivative to determine the direction of each weight and bias adjustment. Rprop focuses solely on the gradient's direction and can achieve faster and more stable convergence during training. Each network parameter is associated with an independent adaptive update value that changes according to the consistency of the gradient direction across successive iterations. In RProp, if the gradient maintains the same sign,

the corresponding update value is increased to accelerate convergence. Alternatively, the update value is reduced to avoid overshooting and minimize oscillations near an optimum when the sign changes. Rprop mitigates the effects of extremely small, large, or uneven gradients that may arise across different regions of the error surface by decoupling the update magnitude from the gradient's size. This characteristic makes it less sensitive to variations in gradient magnitude and often leads to faster and more stable convergence than conventional GD methods. It is particularly well suited for batch training of feed-forward neural networks that employ differentiable activation functions (Riedmiller and Braun 1993).

A decision tree (DT) is a supervised learning method used to classify observations or make predictions by applying a set of logical rules based on input features. It analyzes the available data during training to determine which features are most informative and gradually partitions the dataset into more homogeneous subsets with respect to the target variable. These decisions are typically depicted as a tree-like form with internal nodes corresponding to feature-based tests. Tree branches represent the possible outputs of those tests, while terminal nodes give the final class label or predicted value. The partitioning procedure continues until a predefined stopping criterion is satisfied (e.g., reaching a maximum tree depth). One of the main advantages of decision algorithms is their interpretability. The resulting model can be expressed as a set of clear if–then decision rules, which makes it easier to understand how predictions are generated. However, if appropriate constraints or pruning techniques are not applied, the model may become overly complex and overfit the training data, reducing its ability to generalize effectively to previously unseen observations (Quinlan 1986).

The fourteen machine-learning methods discussed above do not consider the use of higher-order neurons. The only exception is the MCU-ELM method, which integrates MCUs into every neuron of an SLNN. However, this approach cannot automatically identify the most suitable combination of sub-cubes for each MCU.

Chapter 3

Related Work

Overview

The "Related Work" chapter reviews the previous research that forms the foundation of this study. It begins by examining the structure and architecture of semi-linear, cubic neurons and MCUs, with a focus on their mathematical formulation, operating principles, and advantages over conventional neuron models. Special attention is given to their ability to capture complex nonlinear relationships, which makes them well suited for challenging learning tasks. The chapter then introduces the traditional ELM algorithm by discussing its fundamental concepts, including its fast learning speed and good generalization ability. It also explores MCU-enhanced SLNNs, analyzing how integrating these units improves predictive performance. Next, the chapter reviews the fundamentals of GAs and their role in model optimization. It covers their essential components (selection, crossover, mutation) and convergence criteria, explaining how these mechanisms are applied to optimization problems. Finally, the chapter describes the EHO-ELM algorithm. It analyses how it is integrated with MCU-ELM networks to optimize the hidden-layer parameters and sub-cube combinations for each hidden neuron.

3.1 Neuron Architectures

Traditional artificial neuron models like the TLU and sigmoidal units, produce their outputs by applying an activation function to a weighted sum of their inputs. These models form the fundamental building blocks of conventional neural networks. However, when used on their own, they are limited in their ability to capture complex relationships among input variables. Higher-order neuron models like the cubic and MCUs, address this limitation by incorporating polynomial terms between inputs. This way, they can model nonlinear decision boundaries and higher-order feature dependencies more effectively, increasing their representational capacity without relying solely on deeper network

architectures. As a result, cubic and multi-cube units offer a useful alternative for tasks where interactions among predictors play an important role in achieving accurate classification or function approximation (Gurney 1989).

3.1.1 The Low-Order Neuron Architecture

A low-order artificial neuron is a computational unit that processes information by forming a weighted linear combination of its input variables. Given an input vector $x = [x_1, x_2, \ldots, x_n]^T$, each input x_i is associated with a synaptic weight w_i, which determines both the magnitude and direction of its contribution to the neuron's response. The weighted inputs are summed together with a bias term θ to produce the neuron's activation, defined in equation 3.1

$$a = \sum_{i=1}^{n} w_i x_i + \theta \tag{3.1}$$

The bias term allows the neuron to adjust its activation threshold independently of the input values, increasing the flexibility of the resulting decision function. The activation a^l (the l superscript denotes the low-order unit type) is then passed through an activation function g to generate the output y. The structure of the low-order neuron is visualized in Fig. 3.1

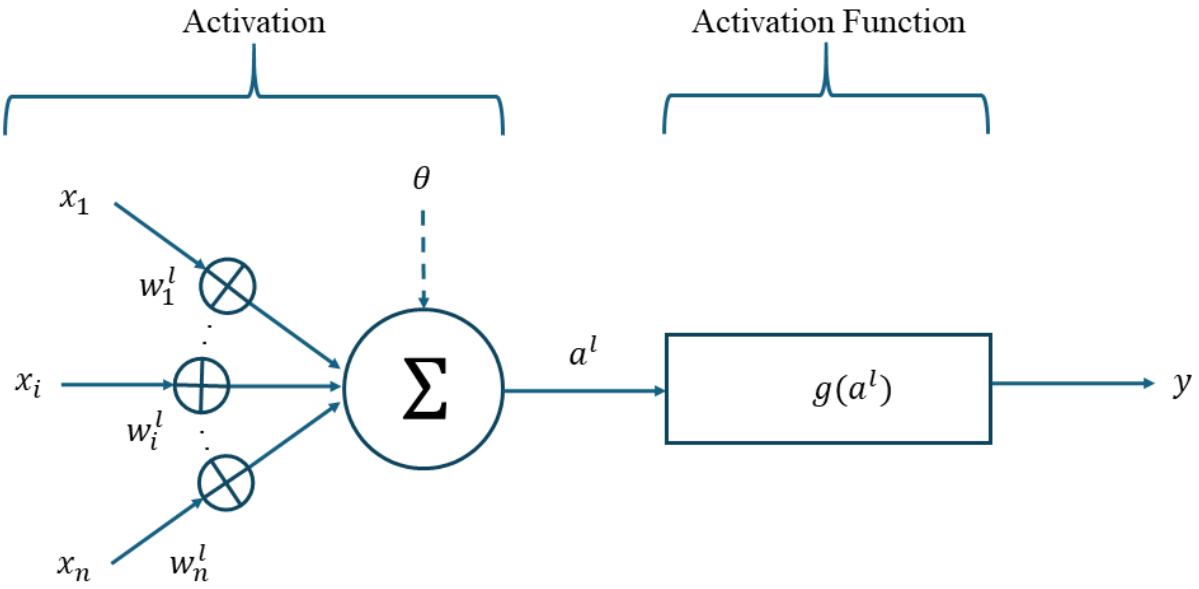

Fig. 3.1: The Low-Order Neuron Architecture. This figure depicts the structure of a typical low-order neuron. Each input $[x_1, x_2, \ldots, x_n]$ is multiplied with a corresponding real valued weight $[w_1, w_2, \ldots, w_n]$ and the outcome from each multiplication is summed $(\sum_{i=1}^{n} w_i x_i)$. Then, an optional threshold θ is added to the sum before being sent to the transfer function g to produce the neuron's output y. [Reproduced from the author's previous work (Christou 2026).]

The behavior of the neuron depends largely on the choice of activation function. A threshold function, for example, produces a binary output (such a neuron type is termed TLU) and is suited for linearly separable classification tasks. Other transfer function types, such as sigmoid, hyperbolic tangent, and rectified linear unit (ReLU), provide real-valued outputs. Although the activation function may introduce nonlinearity into the output, the underlying aggregation process remains a first-order linear combination of the inputs.

As a result, an individual low-order neuron defines a hyperplane decision boundary and can directly represent only linearly separable patterns. Capturing more complex relationships, such as nonlinear decision regions or interactions among input variables, typically requires networks of multiple interconnected neurons. Another solution to this issue is to use higher-order unit models that explicitly incorporate polynomial terms and products of the input features (Gurney 1989).

3.1.2 The Cubic Neuron Architecture

The higher-order neuron by Gurney (1989) differs from the conventional low-order neuron because it does not rely solely on a linear weighted sum of its inputs. An n-dimensional input vector is mapped to a polynomial representation of order n, with 2^n associated weights ($w_n^c = 2^n$, where the superscript c denotes the cubic neuron type). This structure can be interpreted geometrically as an n-dimensional hypercube, in which each vertex corresponds to one specific weight (a 4-dimensional hypercube is seen in Fig. 3.2). In this way, the neuron does not treat the input merely as a set of independent variables, because this representation provides a compact and systematic means of encoding higher-order interactions. For an input vector $x = [x_1, x_2, \ldots, x_n]$, the cubic neuron maps the n-dimensional input space onto an n-order polynomial structure whose weights correspond to the vertices of an n-dimensional hypercube. This mapping allows the neuron to capture not only the individual contributions of the inputs, but the interactions among them within a single computational framework. As a result, the model can represent complex nonlinear relationships that are difficult for a standard first-order neuron to capture.

A probability function determines each weight's contribution to the activation. Let $\mu = \mu_1, \mu_2, \ldots, \mu_n$, where each $\mu_i \in \{0, 1\}$. The corresponding term in the polynomial expansion is given by equation 3.2

$$P_\mu = \frac{1}{2^n} \prod_{i=1}^{n} (1 + \mu_i x_i) \tag{3.2}$$

The binary representation of μ determines how the input variables contribute to each product term $1 + \mu_i x_i$. A binary value of 0 represents a negative sign, whereas a value of 1 represents a positive sign. Before applying this formulation, the input variables are

normalized. The normalization procedure is typically done by dividing each feature by its maximum absolute value.

The cubic activation is computed as a normalized weighted average of these pattern-specific contributions. If w_μ^c denotes the weight associated with the binary index (μ), and ($w_{\max}^c$) is the maximum absolute value among the cubic weights, the activation is defined as seen in formula 3.3.

$$a^c = \frac{1}{|w_{\max}^c|} \frac{1}{2^n} \sum_{\mu=0}^{2^n-1} w_\mu^c \prod_{i=1}^{n} (1 + \mu_i x_i) \tag{3.3}$$

Unlike a conventional neuron, whose output is determined by a single linear projection of the inputs, this formulation aggregates the product terms. The normalization factor $|w_{\max}^c|$ bounds the magnitude of the activation. Once the cubic activation (a_c) has been computed, it is passed through a nonlinear activation function g to produce the final output.

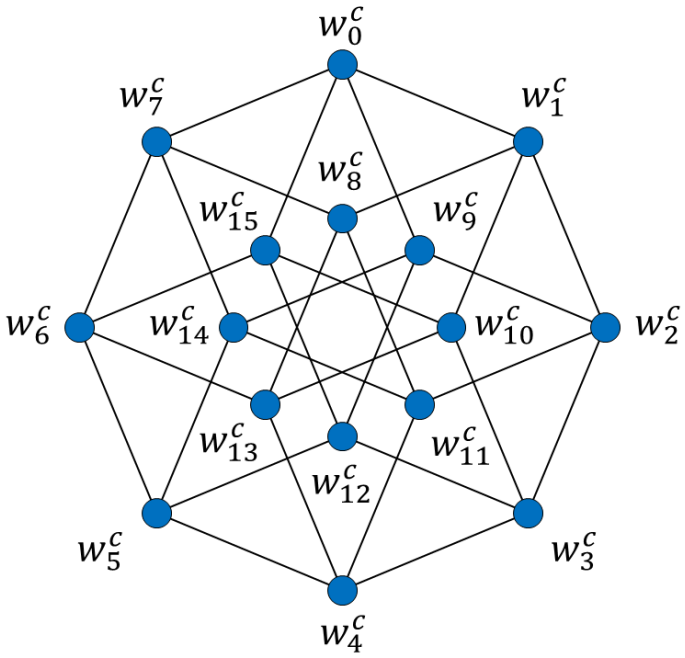

Fig. 3.2: A tesseract is used to illustrate the weight configuration of a four-input neuron. In this geometric representation, the neuron's weights are mapped onto the structure, with each vertex corresponding to a different input weight. [Reproduced from the author's previous work (Christou et al. 2026).]

As in traditional neural models, the nonlinear activation function determines the final response of the neuron. The key distinction from the low-order units lies in the richer internal representation that is constructed before the activation function is applied. By explicitly accounting for higher-order interactions among the inputs, Gurney's cubic neuron provides substantially greater representational flexibility than a classic neuron. This makes it particularly well suited for modeling complex nonlinear decision boundaries and

feature interactions, although these advantages come at the cost of increased computational complexity and a larger parameter space. The cubic neuron structure is depicted in Fig. 3.3

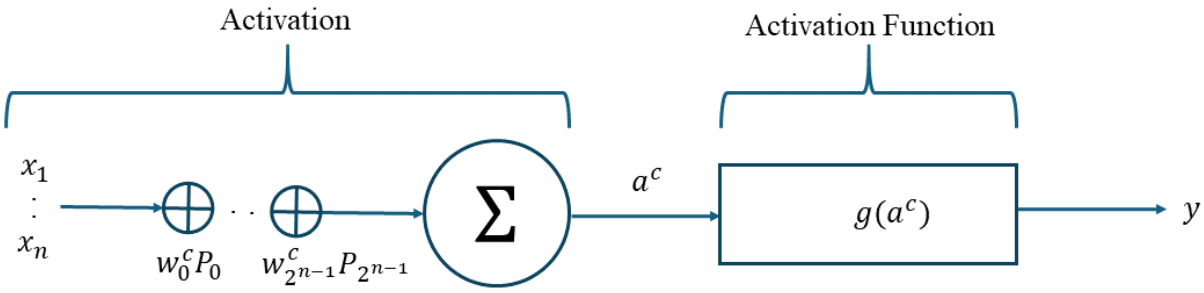

Fig. 3.3: The Higher-Order Neuron Architecture. This figure depicts the structure of a cubic neuron, where an input vector with n components is associated with 2^n weights. Each weight is multiplied by its associated probability term. These weighted contributions are then summed and normalized along with the inputs to produce the neuron's activation. The activation is subsequently passed through a transfer function, which generates the neuron's final output. [Reproduced from the author's previous work (Christou et al. 2025).]

3.1.3 The Multi-Cube Neuron Architecture

The MCU was introduced to overcome the scaling limitations of cubic neurons. While a cubic unit represents all higher-order interactions within a single large cube, the MCU architecture replaces this structure with a collection of lower-order sub-cubes. This substantially reduces the number of required weights and improves computational efficiency. For an n-dimensional input vector, the model is partitioned into q sub-cubes with the same or different dimensions, denoted by $d = [d_1, d_2, \ldots, d_q]$. The number of weights in the MCU is therefore given by equation 3.4 with $(j, d_j, q) \in \mathbb{N}$ and the superscript mc identifying the MCU type.

$$w_{\text{no}}^{mc} = p_{\text{no}} = \sum_{j=1}^{q} 2^{d_j} \tag{3.4}$$

The activation of the multi-cube unit is defined in formula 3.5, where q denotes the number of sub-cubes and d_j represents the dimension of the jth sub-cube. The resulting activation value is then passed through a nonlinear activation function g to produce the output.

$$a^{mc} = \frac{1}{|w_{\max}^{\text{mc}}|} \sum_{j=1}^{q} \frac{1}{2^{d_j}} \sum_{\mu=0}^{2^{d_j}-1} w_{\mu}^{\text{mc}} \prod_{i=1}^{d_j} (1 + \mu_i x_i) \tag{3.5}$$

This decomposition preserves the expressive capacity of higher-order modeling while making the architecture more practical for higher-dimensional input spaces. A visual depiction of the MCU is seen in Fig. 3.4 (Gurney 1989).

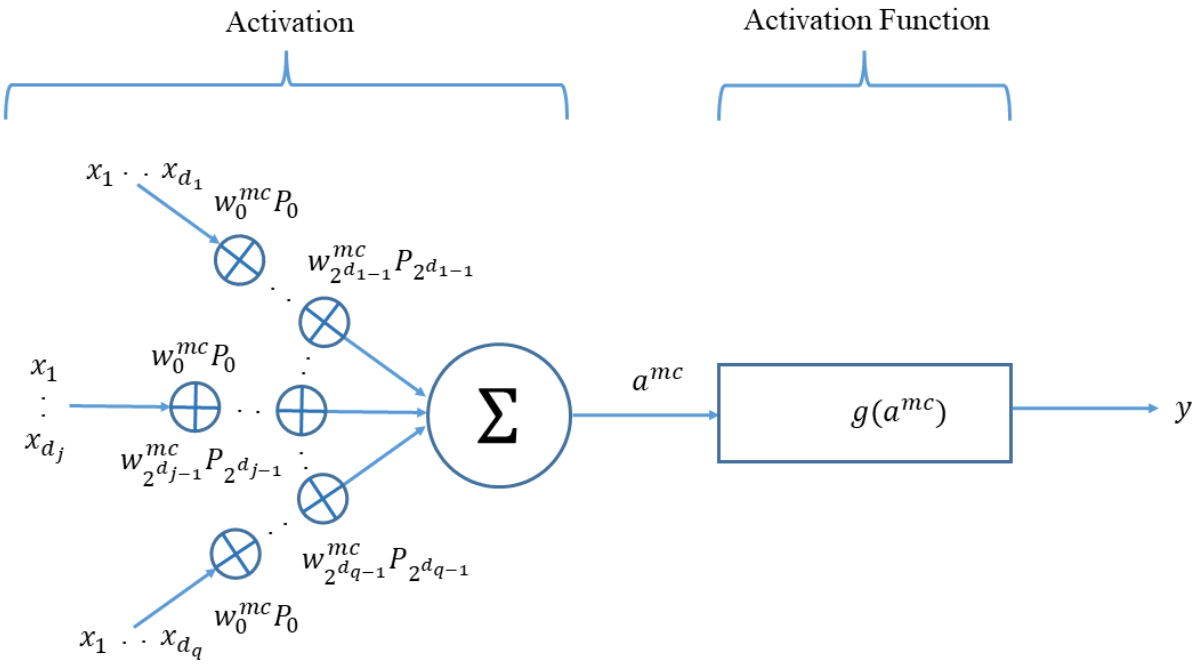

Fig. 3.4: The Multi-Cube Neuron Architecture. This figure illustrates the structure of the MCU, where an n-dimensional input vector is partitioned into q lower-dimensional sub-cubes with dimensions $d = [d_1, d_2, \ldots, d_q]$. Each sub-cube contributes its own set of weights, giving a total of $\sum_{j=1}^{q} 2^{d_j}$ weights. By distributing the input space across multiple lower-dimensional cubes, the model reduces parameter complexity compared with a single higher-order cube while retaining the ability to capture non-linear input relationships. The weighted contributions are then summed and normalized along with the inputs to produce the neuron's activation. The activation is subsequently passed through a transfer function to generate the final output. [Reproduced from the author's previous work (Christou et al. 2025).]

3.2 The Extreme Learning Machine Algorithm

The ELM algorithm introduced by Huang et al. (2004, 2006b) is a learning framework for SLNNs that differs from conventional gradient-based training methods. In the standard ELM architecture, the weights connecting the input and hidden layers, along with the hidden-layer thresholds, are assigned random values. Also, they have fixed values throughout the training procedure. Consequently, learning is confined to the output layer, whose weight matrix is determined analytically as a least-squares problem. The least-squares problem is solved with the use of the Moore-Penrose pseudo-inverse of the hidden-layer output matrix.

The Moore-Penrose pseudo-inverse is a generalisation of the conventional matrix inverse, extending matrix inversion to both square and rectangular matrices. In his seminal work, Penrose (1955) showed that, for every finite matrix A with real or complex entries, there exists a unique pseudo-inverse $A^{\dagger}$ satisfying the four Moore-Penrose conditions given

in formulas 3.6-3.9.

$$AXA = A \tag{3.6}$$

$$XAX = X \tag{3.7}$$

$$(AX)^* = AX \tag{3.8}$$

$$(XA)^* = XA \tag{3.9}$$

In these expressions, the * symbol denotes the conjugate transpose. Although this generalized inverse was first studied by Moore (1920) from a different perspective, the formulation established by Penrose is now commonly referred to as the Moore-Penrose pseudo-inverse (Ben-Israel and Greville 2003). Because of its uniqueness and favorable algebraic properties, it has become a fundamental tool in numerical linear algebra, least-squares estimation, and machine learning.

In the MATLAB programming language, the Moore-Penrose pseudo-inverse is computed using the `pinv` function. This function utilizes singular value decomposition (SVD), through which a matrix $A \in \mathbb{R}^{m \times n}, (m > n)$ is decomposed into the product of three matrices, namely U, S, and V^T, where V^T denotes the transpose of V. Since SVD applies to both square and nonsquare matrices, it is particularly suitable for rectangular systems in which the number of rows m differs from the number of columns n, as indicated in equation 3.10.

$$A_{m \times n} = U_{m \times m} S_{m \times n} V^T_{n \times n} \tag{3.10}$$

In the above formula, U and V are orthogonal matrices, while S is a rectangular diagonal matrix whose diagonal entries are the singular values of A. These singular values are conventionally arranged in descending order ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$) along the main diagonal. The S matrix is defined in equation 3.11

$$S = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & \sigma_n \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.11}$$

For a complex-valued matrix A, the SVD is written as shown in formula 3.12, where V^* denotes the conjugate transpose of V, and both U and V are unitary matrices (Kishore Kumar and Schneider 2017).

$$A_{m \times n} = U_{m \times m} S_{m \times n} V^*_{n \times n} \tag{3.12}$$

The numerical computation of the SVD is generally performed in two principal stages.

First, the matrix A is reduced to bidiagonal form, which requires $O(mn^2)$ floating-point operations. Second, the SVD of the bidiagonal matrix is calculated in an iterative method with $O(n)$ iterations, making the overall algorithm's complexity $O(mn^2)$ (Golub and Van Loan 2013; Kishore Kumar and Schneider 2017; Trefethen and Bau 2022). Once the factorization $A = USV^*$ has been obtained, the Moore-Penrose pseudo-inverse is computed as $A^\dagger = VS^\dagger U^*$ with $S^\dagger$ formed by taking the reciprocal of each nonzero value in the diagonal, transposing the resulting diagonal structure, and leaving zero singular values unchanged. In practical implementations, singular values below a prescribed tolerance are typically treated as zero to enhance numerical stability and prevent the amplification of round-off errors (MathWorks 2026).

ELM significantly reduces both computational cost and training time by eliminating the need for iterative backpropagation. Also, it has become an attractive approach for classification and regression problems, particularly in large-scale applications where fast model training is important.

The randomly generated hidden layer acts as a nonlinear mapping that projects the original input space into a higher-dimensional feature space. This allows the linear output layer to capture complex relationships in the data. This design offers a practical balance between expressive power and computational efficiency. Because the hidden-layer parameters are not optimized during training, the model may exhibit lower generalization performance than gradient-based methods that optimize all network parameters. Moreover, the pseudo-inverse-based solution can become numerically unstable or prone to overfitting when the hidden-layer output matrix is ill-conditioned, particularly when the input data contain noise or redundant features.

In the ELM training method, the transfer function is represented by g. Let $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n} \end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$ denote the input matrix, where N is the number of input patterns and n is the number of input variables. The hidden-layer weight matrix is denoted by $w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n\times h} \in \mathbb{R}^{n\times h}$, where the superscript l indicates the low-order unit type and h is the number of hidden neurons. The vector $\theta = [\theta_1 \dots \theta_h] \in \mathbb{R}^h$ contains the corresponding hidden-layer thresholds. The output-layer weight matrix is given by $\beta^l = \begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h\times m} \in \mathbb{R}^{h\times m}$ where m denotes the number of output

neurons. Finally, the matrix $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m} \in \mathbb{R}^{N \times m}$ is the target matrix containing the desired output values for all training patterns. The resulting SLNN can be written in matrix form as seen in formula 3.13, where g applies the activation function element-wise to the hidden-layer neurons.

$$\begin{bmatrix} g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) & \dots & g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \\ \vdots & \dots & \vdots \\ g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) & \dots & g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \end{bmatrix}_{N \times h} \cdot \begin{bmatrix} \beta^l_{1,1} & \dots & \beta^l_{1,m} \\ \vdots & \dots & \vdots \\ \beta^l_{h,1} & \dots & \beta^l_{h,m} \end{bmatrix}_{h \times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m} \tag{3.13}$$

This formulation shows that the network first transforms the input data via a nonlinear hidden layer, then linearly combines the hidden responses using the output weights to approximate the target matrix (Huang et al. 2004, 2006b).

The pseudo-code of Algorithm 1 summarizes the complete training procedure of an SLNN using the ELM method. The process begins with the random assignment of the input-to-hidden weight matrix $w^l \in \mathbb{R}^{n \times h}$ and the hidden-layer threshold vector $\theta \in \mathbb{R}^h$, where each column of w^l corresponds to a hidden neuron and each element of θ independently shifts its activation. Unlike conventional gradient-based methods, these parameters remain fixed throughout training, representing one of the defining characteristics of ELM.

The algorithm then constructs the hidden-layer output matrix $H \in \mathbb{R}^{N \times h}$ by computing, for each of the N training samples and each of the h hidden neurons, the weighted sum of the inputs together with the corresponding threshold. Then, applies the activation function g. This transformation projects the original input data into a nonlinear feature space induced by the random hidden-layer mapping, where complex relationships in the data may become more amenable to linear modeling.

The target matrix $T \in \mathbb{R}^{N \times m}$ is defined to represent the desired outputs for all training samples, either as encoded class labels in classification problems or as continuous values in regression tasks. Finally, the output-layer weight matrix is obtained analytically using formula 3.14, where $H^\dagger$ denotes the Moore-Penrose pseudo-inverse of the hidden-layer

output matrix.

$$\beta^l = H^\dagger T \tag{3.14}$$

The least-squares solution reduces training to a single matrix computation, thus eliminating the need for iterative backpropagation. ELM offers both computational efficiency and simplicity while retaining the universal approximation capability of SLNNs. Because the hidden-layer mapping is generated randomly and never optimized, the model's performance remains sensitive to factors such as the number and weight values of hidden neurons. These limitations have motivated the development of enhanced ELM variants for practical applications (Huang et al. 2004, 2006b).

Algorithm 1 : Extreme Learning Machine

1: $w^l = \begin{bmatrix} w^l_{1,1} & \dots & w^l_{1,h} \\ \vdots & \dots & \vdots \\ w^l_{n,1} & \dots & w^l_{n,h} \end{bmatrix}_{n \times h}$

2: $\theta = [\theta_1, \dots, \theta_h]$

3: $H = \begin{bmatrix} g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) \dots g\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \\ \vdots \dots \vdots \\ g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,1} \\ \vdots \\ w^l_{n,1} \end{bmatrix} + \theta_1\right) \dots g\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n} \end{bmatrix}^T \begin{bmatrix} w^l_{1,h} \\ \vdots \\ w^l_{n,h} \end{bmatrix} + \theta_h\right) \end{bmatrix}_{N \times h}$

4: $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N \times m}$

5: $\beta^l = H^\dagger T$

3.3 The Multi-Cube Unit Extreme Learning Machine Algorithm

The MCU-ELM is an extension of the ELM framework for SLNNs that uses the MCUs, introduced by Gurney (1989), as the fundamental computational elements throughout the network. It is one of the six higher-order ELM variants proposed by Christou (2026) in his study entitled "Higher Order Extreme Learning Machine."

Consistent with the original ELM paradigm, the hidden-layer parameters of MCU-ELM are randomly initialized and remain fixed throughout training. For this reason, learning is limited to estimating the output-layer weights only using the Moore-Penrose

pseudo-inverse. This approach avoids the iterative optimization procedures required by gradient-based training methods, substantially reducing computational cost while enabling the network to leverage the higher-order representational capabilities of the MCU probability functions. A MATLAB implementation of the proposed higher-order ELM methods, including MCU-ELM, is publicly available at https://github.com/bchristou1/HigherOrderELM.git.

The P symbol denotes the MCU probability function, which constructs higher-order feature representations from the transformed hidden-layer responses. The g function denotes the nonlinear activation function applied to the outputs of the MCU hidden units, introducing nonlinearity into the model and thereby enhancing its representational capacity. The matrix that contains the MCU activations is formally defined as follows:

$$a^{mc} \in \mathbb{R}^{N\times h} = \begin{bmatrix} a^{mc}_{1,1}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,1} \\ \vdots \\ w^{mc}_{p_h-1,1}\end{bmatrix}\right) & \dots & a^{mc}_{1,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,h} \\ \vdots \\ w^{mc}_{p_h-1,h}\end{bmatrix}\right) \\ \vdots & \dots & \vdots \\ a^{mc}_{N,1}\left(\begin{bmatrix} x_{N,1} \\ \vdots \\ x_{N,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,1} \\ \vdots \\ w^{mc}_{p_h-1,1}\end{bmatrix}\right) & \dots & a^{mc}_{N,h}\left(\begin{bmatrix} x_{1,1} \\ \vdots \\ x_{1,n}\end{bmatrix}^T, \begin{bmatrix} w^{mc}_{0,h} \\ \vdots \\ w^{mc}_{p_h-1,h}\end{bmatrix}\right) \end{bmatrix}_{N\times h}.$$

Within the matrix a^{mc}, the input data are represented by $x = \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \dots & \vdots \\ x_{N,1} & \dots & x_{N,n}\end{bmatrix}_{N\times n} \in \mathbb{R}^{N\times n}$, where each row corresponds to an individual input pattern and each column represents a distinct feature. The matrix $w^{mc} = \begin{bmatrix} w^{mc}_{0,1} & \dots & w^{mc}_{0,h} \\ \vdots & \dots & \vdots \\ w^{mc}_{p_h-1,1} & \dots & w^{mc}_{p_h-1,h}\end{bmatrix}_{p_h\times h} \in \mathbb{R}^{p_h\times h}$ defines the set of randomly initialized weight parameters associated with the hidden-layer MCUs. The mc superscript is used throughout to denote the MCU types. The n parameter denotes the number of input features, h represents the number of hidden units, and N is the total number of training patterns. In addition, p_h denotes the number of weights associated with the hidden layer, whereas p_o specifies the number of output-layer weights. The matrix $\beta^{mc} = \begin{bmatrix} \beta^{mc}_{0,1} & \dots & \beta^{mc}_{0,m} \\ \vdots & \dots & \vdots \\ \beta^{mc}_{p_o-1,1} & \dots & \beta^{mc}_{p_o-1,m}\end{bmatrix}_{p_o\times m} \in \mathbb{R}^{p_o\times m}$ represents the set of output-layer weights associated with the MCU-ELM model where m denotes the number of output units in the network. Finally, the matrix $T =$

$\begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m} \in \mathbb{R}^{N\times m}$ encapsulates the ground-truth targets used during the training process where each row contains the desired output values corresponding to a given input pattern (Christou 2026).

$$\begin{bmatrix} P_{1,0}\left(g(a_{1,1}^{mc}),\dots,g(a_{1,h}^{mc})\right) & \dots & P_{1,p_o-1}\left(g(a_{1,1}^{mc}),\dots,g(a_{1,h}^{mc})\right) \\ \vdots & \dots & \vdots \\ P_{N,0}\left(g(a_{N,1}^{mc}),\dots,g(a_{N,h}^{mc})\right) & \dots & P_{N,p_o-1}\left(g(a_{N,1}^{mc}),\dots,g(a_{N,h}^{mc})\right) \end{bmatrix}_{N\times p_o} \cdot \begin{bmatrix} \beta_{0,1}^{mc} & \dots & \beta_{0,m}^{mc} \\ \vdots & \dots & \vdots \\ \beta_{p_o-1,1}^{mc} & \dots & \beta_{p_o-1,m}^{mc} \end{bmatrix}_{p_o\times m} = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m} \tag{3.15}$$

Algorithm 2 outlines the training procedure of MCU-ELM. The hidden layer is initialized only once and remains fixed throughout training, while learning is performed exclusively at the output layer. The first step involves randomly initializing the MCUs' parameters. The second step involves constructing the hidden-layer output matrix. For each training sample, the input is propagated through all hidden units to compute their activations, which are then transformed by a predefined transfer function. The MCU probability functions generate higher-order feature representations of the input data. In the third step, the target output matrix is constructed from the desired output values associated with the training samples. Finally, the output-layer weights are computed analytically by applying the Moore-Penrose pseudo-inverse to the hidden-layer output matrix and multiplying the result by the target output matrix (Christou 2026).

Algorithm 2 :Multi-Cube Unit Extreme Learning Machine

1: $w^{mc} = \begin{bmatrix} w_{0,1}^{mc} & \dots & w_{0,h}^{mc} \\ \vdots & \dots & \vdots \\ w_{p_h-1,1}^{mc} & \dots & w_{p_h-1,h}^{c} \end{bmatrix}_{n\times h}$

2: $H = \begin{bmatrix} P_{1,0}\left(g(a_{1,1}^{mc}),\dots,g(a_{1,h}^{mc})\right) \dots P_{1,p_o-1}\left(g(a_{1,1}^{mc}),\dots,g(a_{1,h}^{mc})\right) \\ \vdots \dots \vdots \\ P_{N,0}\left(g(a_{N,1}^{mc}),\dots,g(a_{N,h}^{mc})\right) \dots P_{N,p_o-1}\left(g(a_{N,1}^{mc}),\dots,g(a_{N,h}^{mc})\right) \end{bmatrix}_{N\times p_o}$

3: $T = \begin{bmatrix} t_{1,1} & \dots & t_{1,m} \\ \vdots & \dots & \vdots \\ t_{N,1} & \dots & t_{N,m} \end{bmatrix}_{N\times m}$

4: $\beta^{mc} = H^{\dagger}T$

3.4 The Genetic Algorithm

GAs are a class of stochastic, population-based metaheuristic optimization methods inspired by the principles of Darwinian evolution, particularly natural selection and genetic inheritance (Holland 1992). Initially proposed by Holland in the 1970s as a theoretical framework for exploring adaptation in both natural and artificial systems, GAs were later developed into a broadly applicable optimization approach through the seminal work of Goldberg (1989). They have been adopted across numerous scientific and engineering fields, for tasks such as function optimization and combinatorial scheduling (Mitchell 1996).

A characteristic that distinguishes GAs from conventional gradient-based optimization methods is their ability to operate without relying on derivative information or assumptions about the continuity, convexity, or differentiability of the objective function. This advantage makes them well-suited to solve problems in discontinuous or noisy search spaces. A GA maintains a population of potential solutions, usually represented as fixed-length chromosomal strings having a specific encoding scheme. Each individual in the population is evaluated using a fitness function that measures its quality with respect to the optimization objective (Goldberg 1989).

Encoding defines how solutions in the problem domain are represented within a GA. The selected encoding scheme is highly dependent on the problem's characteristics. It directly affects the design of genetic operators and the overall performance of the algorithm. Common encoding approaches include binary encoding, real-valued encoding, permutation encoding, and tree-based encoding.

Holland (1992) in his original formulation of GAs introduced binary encoding. In this encoding, each candidate solution is represented as a fixed-length string of binary values. This representation is better suited to discrete optimization problems because it allows easy application of the crossover and mutation operators. Its limitations become more apparent when dealing with continuous or high-dimensional problems. In these cases, representing continuous variables as binary strings can add additional complexity. Real-valued encoding provides a more direct approach because it represents each individual as a vector of floating-point values. This makes the approach particularly suitable for numerical optimization because it allows continuous parameters to be optimized without converting them into a discrete form. It also supports specialized genetic operators, including arithmetic crossover and Gaussian mutation. Permutation encoding is another encoding type that is a popular choice in combinatorial optimization problems (e.g., the traveling salesman problem) where solutions are represented as ordered sequences of elements. Crossover and mutation operators must be designed carefully to ensure that the resulting offspring remain valid permutations, because the order of these elements determines whether a solution is valid. Tree-based encoding is primarily associated with

genetic programming, where individuals are represented as hierarchical tree structures. This representation enables GAs to evolve complex structures, including executable programs and mathematical expressions, by modifying and recombining different parts of the tree (Koza 1994). The evolutionary process is governed by the repeated application of three fundamental genetic operators (selection, crossover, and mutation). The selection operator determines which individuals are more likely to contribute to the next generation by favoring solutions with higher fitness values. Two popular choices are the tournament and roulette wheel selection operators.

Tournament selection is one of the most widely used selection mechanisms in GAs due to its simplicity, computational efficiency, and flexibility (Goldberg 1989). A k-size tournament is conducted by randomly selecting k individuals from the population. This is usually done with replacement, and by choosing the fittest individual for reproduction. This process is repeated N times to generate the mating pool. Several variants of tournament selection have been proposed. In deterministic tournament selection, the population is divided into k-size disjoint groups, and the fittest individual from each group is selected. Probabilistic tournament selection introduces additional randomness by selecting the winner with probability $p > 0.5$. This way, even the fitter individual in a tournament can be rejected (Mitchell 1996).

Tournament selection has many advantages. It can directly control selection pressure by adjusting the tournament size k. Larger k values result in stronger selection pressure and faster convergence because they increase the likelihood that highly fit individuals will win tournaments. Smaller tournament sizes promote diversity, because they give weaker individuals a greater chance for selection (Goldberg 1989). This flexibility allows researchers and practitioners to strike a balance between exploration and exploitation based on the requirements of the problem being addressed. Another advantage of tournament selection is its computational efficiency. The method relies solely on simple fitness comparisons and does not require fitness scaling, normalization, or the calculation of population-wide statistics (Mitchell 1996). It is particularly well suited to large-scale and parallel implementations, where avoiding costly global operations can significantly improve performance. Tournament selection is also robust across different fitness representations because it does not require fitness values to be strictly positive (Goldberg 1989). Empirical studies have shown that it often converges more rapidly than roulette-wheel selection on a variety of benchmark problems because it provides stronger and more consistent selection pressure (Blickle and Thiele 1996).

Despite the aforementioned advantages, tournament selection has several limitations. Since tournaments are formed through stochastic sampling, the resulting selection distribution may vary considerably (specifically when population sizes are small). Additionally, choosing an appropriate tournament size is critical. If k is too large, deterministic tournament selection can impose excessive selection pressure. This leads to premature

convergence and reduction in genetic diversity. The loss of diversity may limit the algorithm's ability to escape local optima and negatively affect its performance on complex optimization problems. A typical tournament selection operator is visualized in Fig. 3.5 (Mitchell 1996).

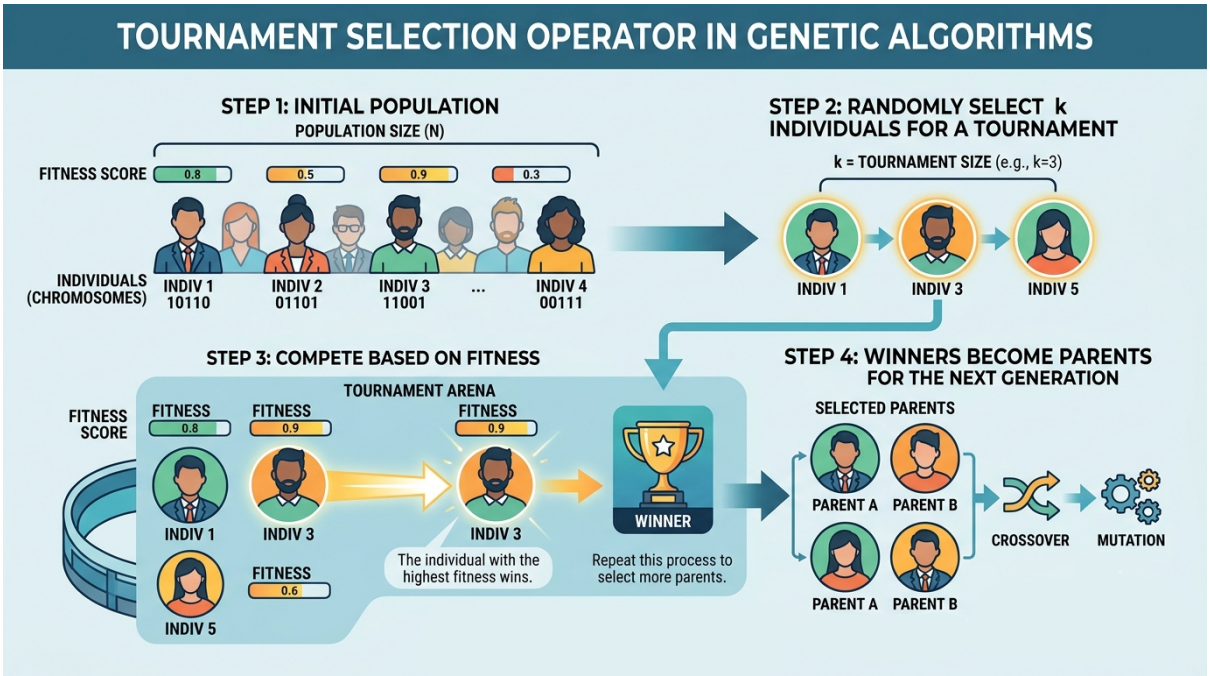

Fig. 3.5: Tournament Selection Operator. Tournament selection is a GA operator that randomly picks a small group of individuals and selects the one with the highest fitness to become a parent. The tournament size k controls selection pressure. Smaller k preserves diversity, while larger k makes the process more competitive and favors stronger candidates. [The conceptual illustration was generated with the assistance of Google Gemini AI and subsequently edited by the author (Google 2026).]

Roulette wheel (fitness-proportionate) selection, is a probabilistic selection mechanism widely used in GAs and other evolutionary optimization methods. It increases the likelihood that high-quality individuals contribute to subsequent generations by selecting candidate solutions for reproduction according to their relative fitness. It differs from deterministic selection strategies, since it doesn't always select the best-performing individuals. Instead, each individual is assigned a selection probability proportional to its fitness value. Fitter solutions are more likely to be selected, while lower-quality solutions retain a non-zero selection probability. This stochastic behavior helps preserve population diversity and promotes exploration of the search space (Goldberg 1989).

Consider a population of N individuals, where the fitness of the i-th individual is denoted by f_i. Assuming that all fitness values are non-negative, the probability of

selecting individual i is given by formula 3.16.

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3.16}$$

The resulting probability p_i represents the individual's contribution to the population's total fitness. For example, if an individual accounts for 20% of the total population fitness, it has a 20% chance of being selected during a single selection event. The term "roulette wheel" originates from the conceptual representation of the population as a circular wheel divided into segments, where each segment corresponds to an individual and is sized according to its selection probability. Spinning the wheel determines which individual is selected, making larger segments more likely to be chosen. In practice, roulette wheel selection is typically implemented using a cumulative probability distribution rather than an actual graphical wheel. The fitness values are first normalized to obtain the selection probabilities p_i, after which the cumulative probabilities are computed (Goldberg 1989).

A uniformly distributed random number $r \in [0, 1)$ is then generated, and the selected individual is the first one whose cumulative probability satisfies $q_i \geq r$. This procedure is repeated until the required number of parents has been selected. As an illustrative example, suppose that four individuals have fitness values $[10, 20, 30, 40]$. The total fitness is 100, resulting in selection probabilities of $[0.10, 0.20, 0.30, 0.40]$ and cumulative probabilities of $[0.10, 0.30, 0.60, 1.00]$. If the generated random value is $r = 0.26$, the second individual is selected because $0.10 < 0.26 \leq 0.30$. Similarly, if $r = 0.83$, the fourth individual is selected because $0.60 < 0.83 \leq 1.00$. Although the fourth individual has the highest probability of selection, it is not guaranteed to be chosen in every draw, allowing the remaining individuals to continue contributing to the evolutionary process.

Roulette wheel selection introduces selection pressure by increasing the expected reproductive opportunities of highly fit individuals. In a mating pool of size M, the expected number of times that individual i is selected is Mp_i. Individuals with above-average fitness enable favorable characteristics to propagate across generations and are expected to appear more frequently in the mating pool. At the same time, the stochastic nature of the method prevents the immediate elimination of lower-fitness individuals and helps retain potentially useful genetic material that might otherwise be lost(Goldberg 1989).

Despite its simplicity, roulette wheel selection is sensitive to the distribution and scaling of fitness values. When the differences between fitness values are very large, a small number of highly fit individuals may dominate the selection process. This can substantially reduce population diversity and increase the risk of premature convergence to a local optimum. On the other hand, when fitness values are very similar, the resulting selection probabilities become nearly identical. This produces weak selection pressure and potentially slows convergence. To address these issues, fitness scaling or transformation

techniques (e.g., linear scaling, sigma scaling, rank-based scaling, and fitness shifting) are applied before selection. These approaches help maintain an appropriate balance between exploring diverse candidate solutions and exploiting promising regions of the search space (Baker 1987).

Roulette wheel selection is well-suited to optimization problems that provide meaningful, non-negative fitness values that can be compared across the population. Special treatment is required for minimization problems, negative objective values, or situations in which the sum of all fitness values is zero. In minimization settings, the objective function is commonly transformed into a maximization-oriented fitness measure (e.g., through inverse and shifted transformations). Overall, roulette wheel selection remains one of the fundamental operators in evolutionary computation because of its conceptual simplicity, probabilistic behavior, and ability to favor high-quality solutions without entirely excluding weaker candidates A visualization of the roulette wheel selection mechanism can be seen in Fig. 3.6 (Lipowski and Lipowska 2012).

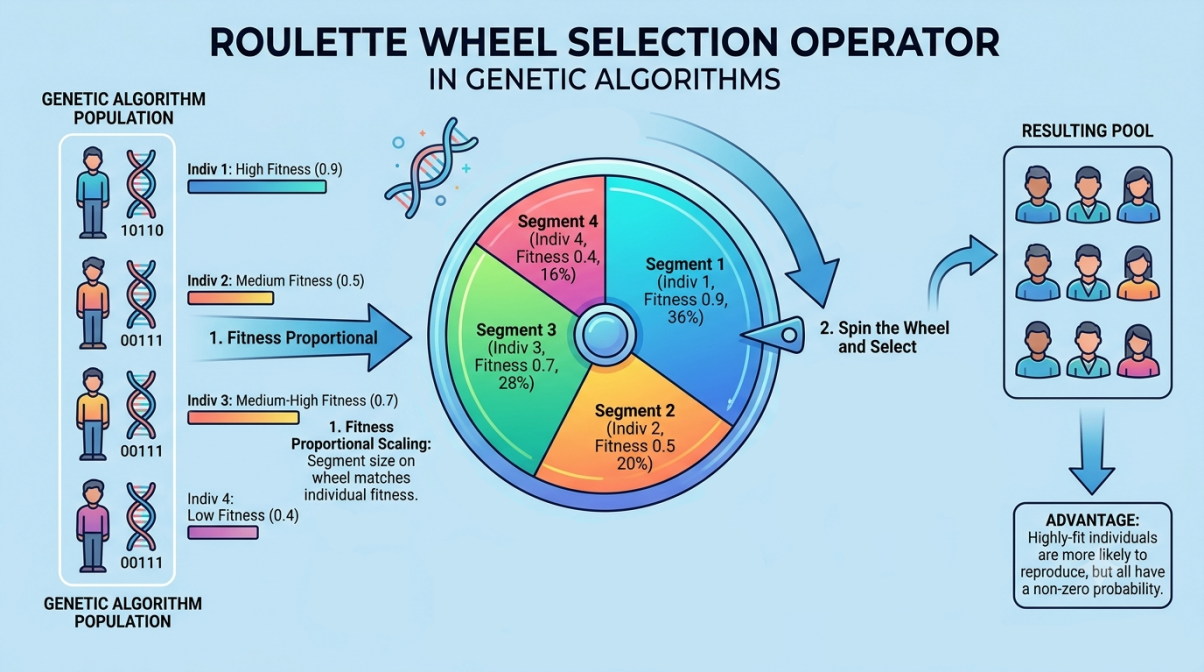

Fig. 3.6: Roulette Wheel Selection Operator. The roulette wheel selection is a fitness-based selection mechanism used in GAs. Individuals with higher fitness values are assigned larger portions of the selection wheel and have a greater probability of being chosen for reproduction. The process begins by calculating the total population fitness, followed by assigning selection probabilities proportional to each individual's fitness. Selection is then performed by simulating a spin of the wheel. Fitter individuals are more likely to be selected, but lower-fitness candidates still retain a chance of reproduction. This helps maintain population diversity while applying selective pressure toward higher-quality solutions. [The conceptual illustration was generated with the assistance of Google Gemini AI and subsequently edited by the author (Google 2026).]

The crossover (recombination) operator combines genetic information from selected

parent solutions to produce new offspring. Crossover preserves promising solution components and promotes exploring new regions within the search space by exchanging genetic material (Holland 1992).

One-point crossover is one of the most widely used recombination operators in GAs. It works by selecting a single random position along two parent chromosomes and exchanging the genetic material beyond that point to produce two offspring. Between two parent strings, the algorithm first chooses a cut point uniformly at random from the interval $[1, l-1]$, where l is the chromosome length. The first offspring is created by combining the first parent's prefix with the second parent's suffix, while the second offspring receives the complementary combination. A visualization of the one-point crossover operator can be seen in Fig. 3.7 (Holland 1992).

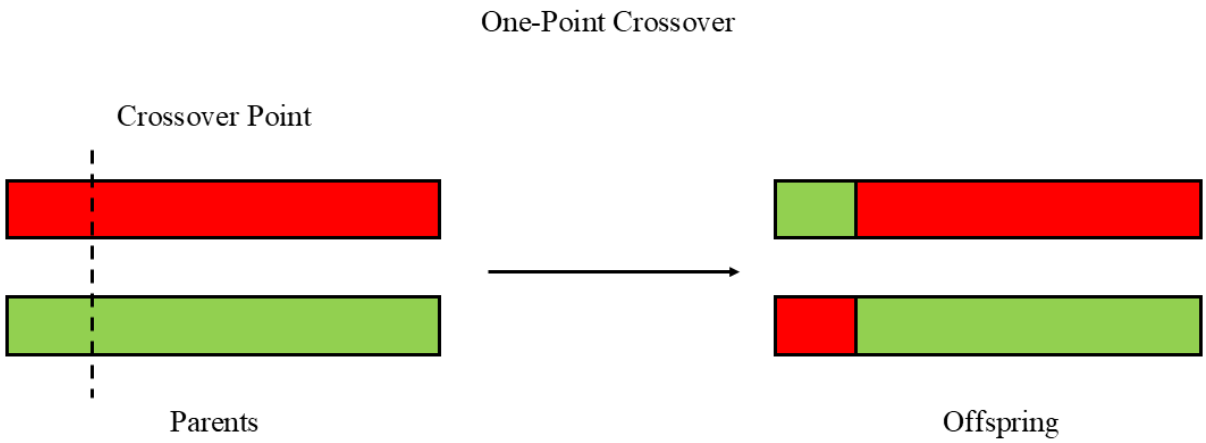

Fig. 3.7: One-Point Crossover Operator. This figure illustrates the one-point crossover operator used in GAs, where two parent chromosomes are recombined to generate two offspring. A single crossover point is selected along the length of the chromosomes, which divides each parent into two segments. The first offspring is produced by combining the left segment of the first parent with the right segment of the second parent. The second offspring is formed by combining the second parent's left segment with the first parent's right segment. It exchanges genetic material between the parents while preserving contiguous gene sequences. This way, it promotes genetic diversity and facilitates the exploration of new candidate solutions.

This operator is computationally efficient and easy to implement. It is a popular choice in many GA applications due to its tendency to preserve contiguous gene groups (building blocks). Thereby, it contributes positively to solution quality. One-point crossover is particularly well suited to binary-encoded representations and combinatorial optimization problems where the order of genes is important. It is less exploratory than multi-point or uniform crossover because it exchanges only a single segment of each chromosome. It remains a widely used baseline method due to its simplicity (Holland 1992).

The n-point crossover operator is a popular recombination mechanism in GAs that generates offspring by exchanging segments between two parent chromosomes at n ran-

domly selected crossover points. It combines genetic material from both parents while preserving useful traits inherited from each to produce new candidate solutions. This process helps maintain genetic diversity within the population and supports effective exploration of the search space. The parent chromosomes are divided into more segments as the n value increases. This leads to more frequent alternation of inherited genetic blocks and potentially greater variability in the resulting offspring. The n-point crossover operator is commonly applied to binary-coded and other discrete representations. It can be viewed as a generalization of one-point crossover, where the number of crossover points determines the extent of genetic material exchanged between the parents. A visualization of the n-point crossover operator is shown in Fig. 3.8 (De Jong and Spears 1992).

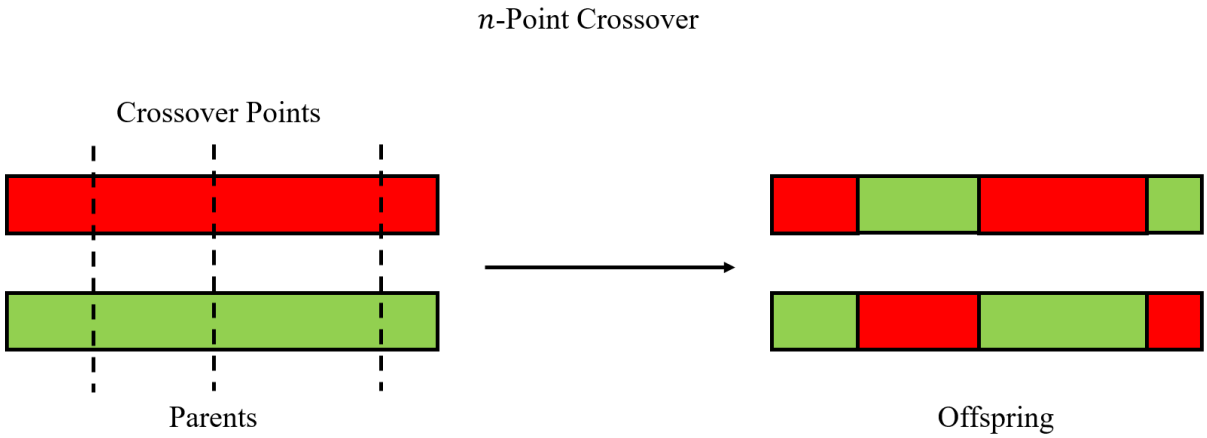

Fig. 3.8: n-Point Crossover Operator. The n-point crossover operator recombines two parent chromosomes by exchanging genetic material at multiple crossover points. It produces offspring that inherit alternating segments from both parents. The parent chromosomes are first divided at randomly selected crossover points, and the resulting segments are then recombined to generate two new offspring with a mixture of genetic information from each parent.

Uniform crossover is a recombination operator in GAs that generates offspring by independently selecting the value at each locus from one of two parent chromosomes. In biological terminology, a locus is a specific position on a chromosome, while an allele is the value or variant stored at that position. In computational terms, a locus corresponds to a particular gene (variable), and its allele represents the candidate value assigned to that gene. In a binary chromosome, the possible alleles at each locus are typically 0 and 1 (Morris-Rosendahl 2010). Uniform crossover does not exchange contiguous chromosomal segments (like one-point and multi-point crossover). Each gene is inherited independently, with the parent contributing that gene determined by a randomly generated binary mask.

Let the two parent chromosomes be represented as $P^1 = (P_1^1, P_2^1, \ldots, P_l^1)$ and $P^2 = (P_1^2, P_2^2, \ldots, P_l^2)$, where l is the chromosome length. The offspring chromosomes O^1 and O^2 are then constructed as seen in vector notation formulas 3.18 and 3.17, where $M = (M_1, M_2, \ldots, M_l)$ is a randomly generated binary mask and $\odot$ denotes element-wise

multiplication.

$$O^1 = (1 - M) \odot P^1 + M \odot P^2 \tag{3.17}$$

$$O^2 = M \odot P^1 + (1 - M) \odot P^2 \tag{3.18}$$

Each mask element is typically sampled independently from a Bernoulli distribution with parameter $p = 0.5$. For this reason, each gene has an equal probability of being inherited from either parent. A more general uniform crossover introduces a bias toward one parent by allowing $p \neq 0.5$.

Because every locus is treated independently, uniform crossover provides thorough mixing of parental genetic material and avoids positional bias. This same characteristic also makes it highly disruptive, as it can break apart co-adapted groups of alleles (referred to as building blocks). Consequently, its performance may be less effective for problems in which interactions between neighboring genes play an important role. A visualization of the uniform crossover operator is shown in Fig.3.9 (Syswerda 1989).

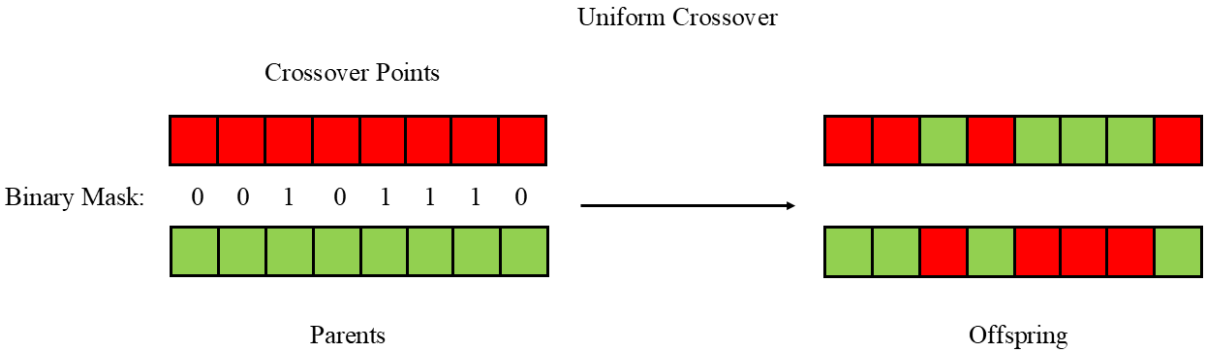

Fig. 3.9: Uniform Crossover Operator. The uniform crossover operator recombines two parent chromosomes (shown with red and green genes) independently at each of the eight loci according to the binary mask [0, 0, 1, 0, 1, 1, 1, 0]. For each offspring, a mask value of 0 indicates the gene is inherited from the current parent, while a value of 1 indicates it is inherited from the other parent. This process preserves the chromosome length while increasing genetic diversity by allowing genes to be exchanged independently at every position.

The mutation operator introduces random modifications to individual genotypes, which helps maintain genetic diversity and reduces the likelihood of premature convergence to suboptimal solutions (Holland 1992; Mitchell 1996).

The mutation operator in the binary-encoded GA proposed by Holland (1992) is applied independently at the gene level to chromosomes encoded using a finite alphabet. Each locus is subjected to mutation with a predefined probability p_m. This operation is equivalent to a bit-flip mutation, in which the allele at a selected locus is complemented. This way, an allele with value 0 is replaced by 1, whereas an allele with value 1 is replaced by 0. For a chromosome $x = (x_1, x_2, \ldots, x_l)$, with $x_i \in \{0, 1\}$, where l is the chromosome's

length, the mutated allele is given by equation 3.19.

$$x_i' = \begin{cases} 1 - x_i, & \text{with probability } p_m \\ x_i, & \text{with probability } 1 - p_m \end{cases} \tag{3.19}$$

The algorithm introduces variation while preserving most of the inherited chromosomal structure by applying this transformation with a relatively low probability. This mechanism further explores the search space and helps prevent the population from becoming prematurely confined to a suboptimal region (Holland 1992; Mitchell 1996).

Gaussian mutation is one of the most commonly used mutation operators for real-valued chromosome representations. It works by perturbing a selected decision variable with a random value drawn from a normal distribution with zero mean and standard deviation σ. For a real-valued gene x_i, where σ determines the typical size of the perturbation, the mutated value is given by formula 3.20.

$$x_i' = x_i + N(0, \sigma) \tag{3.20}$$

Smaller values of σ result in relatively small changes. This allows the search to focus on regions around promising solutions, while larger values produce larger perturbations and encourage exploration of a wider area of the search space. If the resulting value falls outside the predefined bounds of the variable, a boundary-handling method (like clipping or resampling) can be used to bring it back into the feasible range. Gaussian mutation introduces smooth changes around existing solutions while still helping to maintain diversity within the population (Hinterding 1995).

The algorithm progressively improves the population by repeatedly applying these operators over successive generations until a termination criterion is satisfied. Common termination criteria include reaching a predefined number of generations or achieving an acceptable fitness level. GAs do not provide a formal guarantee of reaching the global optimum despite their robustness and broad applicability. Their performance depends heavily on the appropriate selection of control parameters (population size, crossover probability, and mutation rate). Moreover, their ability to perform parallel global search of complex solution spaces without requiring detailed domain-specific knowledge has made GAs one of the most influential approaches in evolutionary computation. They have also provided the foundation for many modern metaheuristic and hybrid optimization techniques (Goldberg 1989).

3.5 The Evolutionary Higher-Order Extreme Learning Machine Algorithm

The EHO-ELM algorithm is a hybrid method that integrates the MCU-ELM approach with a modified GA. The algorithm starts from a randomly generated population of MCU-based SLNNs. This population evolves to obtain architectures having improved generalization performance for specific regression and classification tasks. A central design objective of EHO-ELM is to preserve the simplicity of the original ELM framework by allowing all user-specified GA parameters to adapt automatically. Algorithm 3 outlines the EHO-ELM procedure, while Fig. 3.10 illustrates its overall architecture. A MATLAB implementation is available at https://github.com/bchristou1/EHO-ELM.git.

Algorithm 3 : EHO-ELM

1: $Population \leftarrow create(Pop_{no})$
2: **loop**
3: $Population_{evaluate} = evaluate(Population)$
4: $solution_{best} \leftarrow best(Population_{evaluate})$
5: **if** $solution_{best}$ is unchanged for 5 generations **then**
6: **return** $solution_{best}$
7: **end if**
8: $Population_{select} \leftarrow select(\frac{Population_{evaluate}}{2})$
9: $Population_{crossover} \leftarrow crossover(Population_{select})$
10: $Population \leftarrow mutation(Population_{crossover})$
11: **end loop**

The EHO-ELM procedure begins by generating an initial population of SLNNs. MCUs are used across all network layers, as indicated in line 1 of the algorithm. Each candidate SLNN is represented by a chromosome, which allows the genetic algorithm to optimize both the network parameters and its structural configuration during the evolutionary process. Each chromosome contains two types of information. The first represents the weight parameters associated with the MCUs in the hidden layer. The second describes the structural configuration of the MCUs in both the hidden and output layers of the SLNN. This representation enables EHO-ELM to optimize not only the hidden-layer weights but also the arrangement of the sub-cubes that make up each MCU. The information that corresponds to an individual MCU (weight values, sub-cube sizes, and number) is encoded within the chromosome. The sub-cube size is limited to 2^5 for each MCU. This keeps the number of weights low inside each MCU. Pop_{no} denotes the population size, which is not specified manually. It is automatically determined using Equation 3.21, which considers

the number of hidden-layer neurons, denoted by h.

$$Pop_{no} = \begin{cases} 50, & 2h > 50 \\ 2h, & 0 > 2h \leq 50 \end{cases} \tag{3.21}$$

According to this strategy, the initial population size is adjusted according to the complexity of the SLNN architecture. This helps to maintain a balance between the diversity of candidate solutions and the computational cost of the evolutionary process.

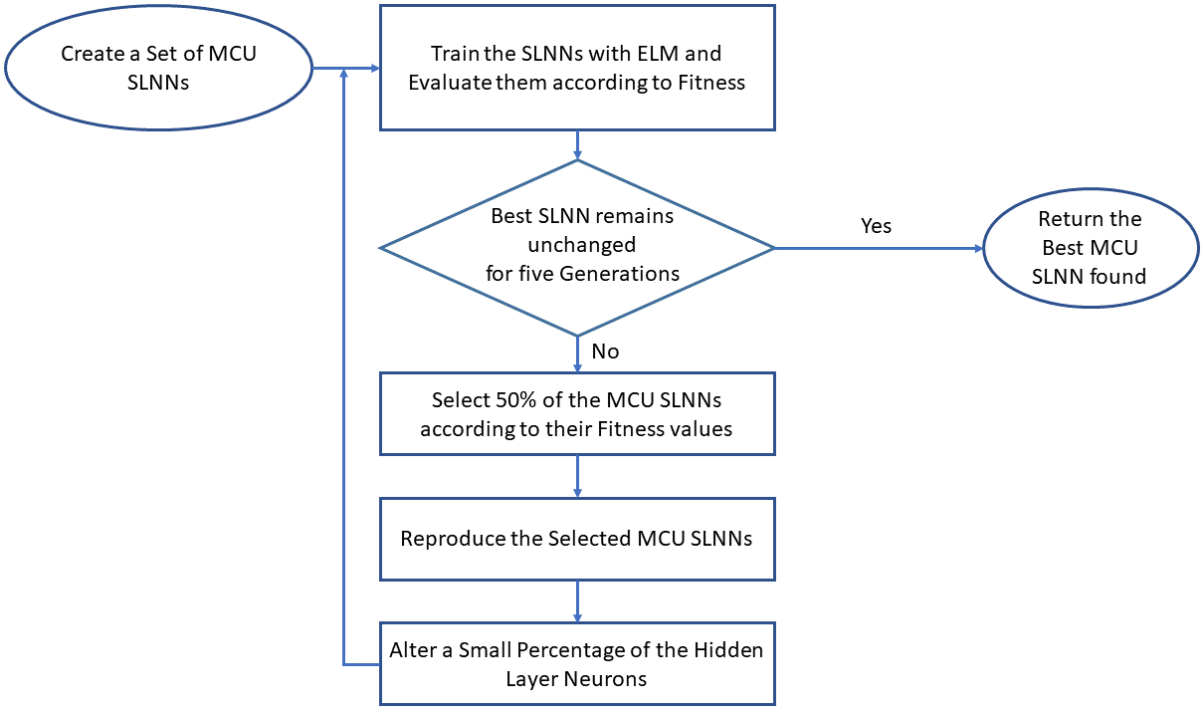

Fig. 3.10: Architecture of the EHO-ELM method. The figure illustrates the GA-based evolutionary framework used in EHO-ELM. The process begins by randomly generating a population of MCU-based SLNNs. Each network is then trained using the MCU-ELM algorithm and evaluated based on its generalization performance. The resulting population is ranked from the best to the worst (according to performance). The evolutionary process is terminated, and the best network found is returned if the best-performing SLNN remains unchanged for five consecutive generations. Otherwise, the algorithm proceeds to the selection stage, which involves selecting half of the population for reproduction according to their fitness values. New offspring are then generated and subjected to mutation, which modifies a small proportion of neurons in the hidden layer. This evolutionary process is repeated over successive generations until the specified stopping criterion is satisfied. [Reproduced from the author's previous work (Christou et al. 2025).]

The evolutionary procedure begins at line 2 with the first generation of the modified GA. In each generation, all chromosomes are evaluated to determine their suitability for the learning task. As indicated in line 3, a fitness value is assigned to each chromosome. Since each chromosome represents an MCU-based SLNN, its fitness is determined by

training the corresponding network using the MCU-ELM algorithm and evaluating its generalization performance.

Fitness is evaluated according to the problem type being addressed. In classification tasks, the classification accuracy (acc) is used. On the other hand, regression tasks utilize the mean squared error (MSE). An SLNN is considered fitter when its acc is higher in classification problems. In regression tasks, a low MSE value is considered better. This fitness criterion allows the GA to guide the evolutionary search toward network configurations that achieve better predictive performance.

The classification accuracy is calculated using equation 3.22, where k denotes the number of folds, pat represents the number of input patterns in each fold, and err denotes the number of incorrectly classified patterns. The final accuracy is obtained by averaging the accuracy across all folds.

$$acc = \frac{1}{k} \sum_{i=1}^{k} \left(1 - \frac{err_i}{pat_i} \right). \tag{3.22}$$

For regression problems, the performance of each SLNN is evaluated using the MSE defined in equation 3.23. In this formula, k denotes the number of folds, pat represents the number of input patterns evaluated in each fold, t_i^j is the target value of the j-th output pattern in the i-th fold, and y_i^j is the corresponding network output. The MSE is calculated as the average squared difference between the target and predicted values across all patterns and folds.

$$MSE = \frac{1}{k \cdot pat} \sum_{i=1}^{k} \left(\sum_{j=1}^{pat} \left(t_i^j - y_i^j \right)^2 \right). \tag{3.23}$$

The best solution of the current generation is identified in line 4, based on the fitness values (after all chromosomes are evaluated). The chromosome with the highest classification accuracy or lowest mean squared error is selected as the best candidate. The algorithm then checks in line 5 whether this solution has remained unchanged for five consecutive generations. If it is true, the evolutionary procedure terminates and returns the best SLNN found during the search (line 6). Otherwise, the process proceeds to the selection operation (line 8), where 50% of the population is selected for reproduction based on the fitness values obtained during evaluation. SLNNs exhibiting better generalization performance have a higher probability of being selected as parents. This selection mechanism directs the search toward more effective network structures while preserving high-quality candidate solutions.

Reproduction is performed using a custom crossover operator that generates offspring by exchanging sub-cubes between two selected parent chromosomes. This operation combines the MCU-related structural information of the parent SLNNs while enabling the

exploration of alternative MCU configurations and the generation of potentially improved network architectures. The crossover procedure is illustrated in Fig. 3.11.

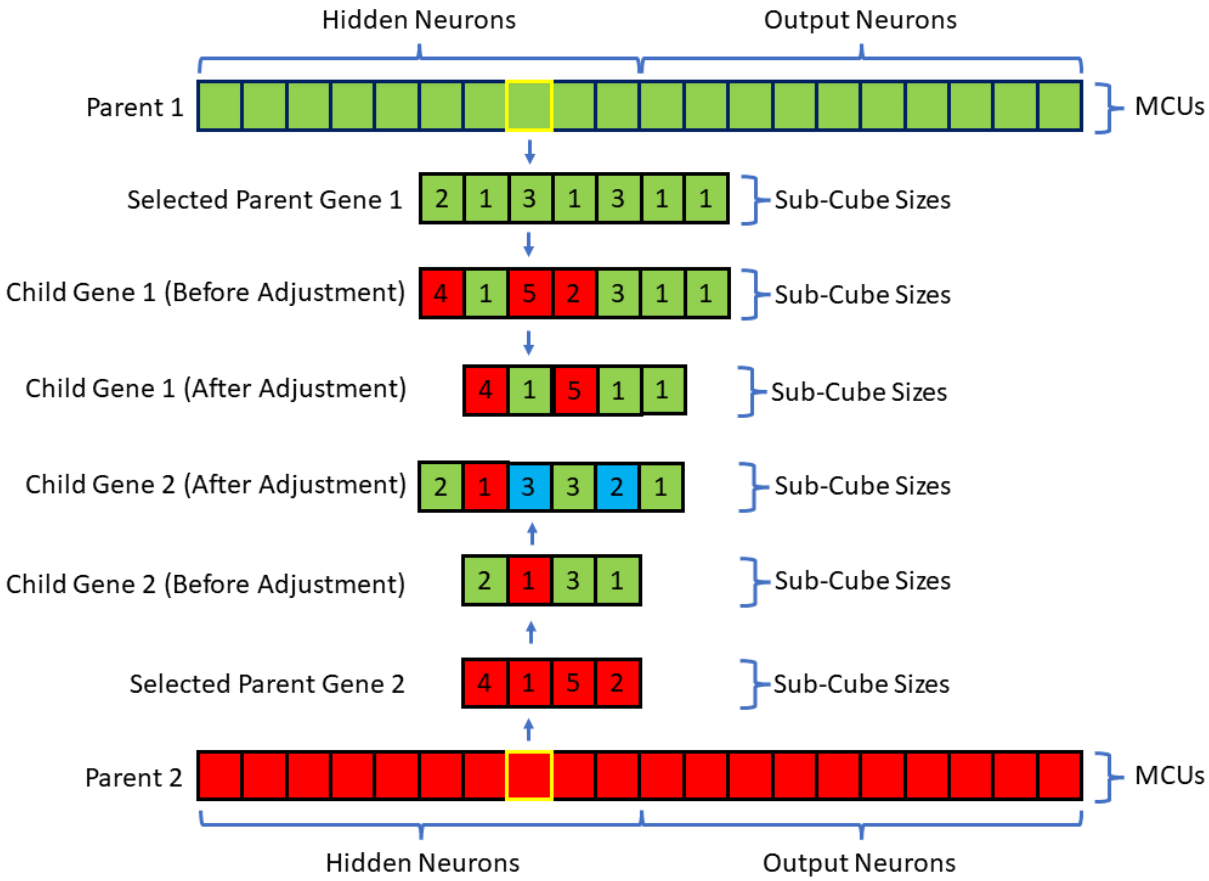

Figure 3.11: The EHO-ELM Crossover Operator. The proposed EHO-ELM crossover operator facilitates the exchange of genetic information between two parent chromosomes (represented in green and red). Each parent is selected with equal probability during the exchange process. Crossover may generate offspring with structurally incompatible MCU configurations, since the selected genes may encode MCUs with different input sizes. To maintain offspring validity, an adjustment mechanism automatically inserts (represented in blue) or removes genes as required. In the illustrated example, the maximum permitted sub-cube size is set to 5. [Reproduced from the author's previous work (Christou et al. 2026).]

The crossover operation selects sub-cubes from the two parent MCUs with equal probability. The resulting offspring MCU may have an input dimensionality that differs from that of the corresponding parent MCU, since the selected parent genes may contain different numbers of inputs. The proposed crossover operator incorporates an automatic adjustment procedure to maintain structural validity. When the offspring MCU contains more inputs than the corresponding parent MCU, sub-cubes are randomly removed from the offspring until its number of inputs is equal to or lower than that of the parent gene. This prevents the generation of offspring configurations that exceed the parent MCU's input dimensionality. The input difference is evaluated first when the offspring MCU

contains fewer inputs than the corresponding parent MCU. If the input difference exceeds the maximum permitted sub-cube input size, a new sub-cube is randomly generated and assigned a random input size with randomly initialized weight values. The new sub-cube is then inserted at a random position within the offspring MCU gene. This process is repeated until the remaining input difference is less than or equal to the maximum allowable sub-cube input size. A final sub-cube is then generated using the remaining number of inputs required to match the number of inputs in the parent MCU, and inserted at a randomly selected position within the offspring gene. This adjustment procedure ensures the structural validity of the resulting MCU-based SLNNs while introducing variation in both the hidden-layer weight values and sub-cube configurations.

Following crossover, a subset of hidden-layer neurons is randomly selected for mutation. The proposed self-adaptive mutation operator defined in formula 3.24 modifies the weights of a randomly selected sub-cube within each selected MCU. The mutation rate (denoted by μ_{rate}) is dynamically adjusted according to the fitness of the best chromosome in each generation. $fitness_{current}$ and $fitness_{previous}$ denote the fitness values of the best chromosomes in the current and previous generations, respectively. During the first generation, or when the current generation produces an improved best chromosome, the mutation rate is set to its minimum value of 10%. If no improvement is obtained, the mutation rate is increased by 20%, up to a maximum of 50%. This adaptive mechanism increases exploration when the evolutionary search stagnates while maintaining a lower mutation rate when improved solutions continue to be found (Christou et al. 2025, 2026).

$$\mu_{rate} = \begin{cases} \mu_{rate} = 10\%, & (generation = 1) \vee ((generation > 1) \\ & \wedge (fitness_{current} > fitness_{previous})) \\ \mu_{rate} = \mu_{rate} + 20\%, & fitness_{current} \leq fitness_{previous} \\ \mu_{rate} = 50\%, & \mu_{rate} \geq 50\% \end{cases} \tag{3.24}$$

Chapter 4

The System Architecture

Overview

This chapter applies the proposed EHO-ELM classifier to predict student academic outcomes using the "Predict Students' Dropout and Academic Success" dataset available from the University of California, Irvine (UCI) repository. The classification problem considers three possible outcomes (dropout, continued enrollment, and graduation), and the dataset contains a variety of information about students (background, admission pathway, family and financial circumstances, early academic performance, and broader macroeconomic factors).

The proposed EHO-ELM model combines an ELM variant with MCUs in the hidden and output layers. A modified GA is used to optimize the model by searching for suitable hidden-layer weights and suitable multi-cube structures. Useful structural components are preserved through sub-cube-based crossover, while a validity mechanism ensures that the generated offspring form valid SLNNs. Self-adaptive mutation is incorporated to maintain population diversity and reduce the likelihood of premature convergence. Once the hidden-layer structure and weights have been optimized, the output-layer weights are calculated analytically using the Moore–Penrose pseudo-inverse. This approach provides a flexible framework for capturing nonlinear relationships that may help distinguish dropout, remaining enrolled, and graduating students.

4.1 The Dataset Characteristics

The EHO-ELM method for predicting student academic success is evaluated using the "Predict Students' Dropout and Academic Success" dataset from the University of California, Irvine (UCI) repository (Realinho et al. 2021). The dataset provides a useful benchmark for educational data mining because it combines information available before enrollment, socioeconomic factors, and students' academic performance during their

first year to predict three possible outcomes (dropout, continued enrollment, and graduation). However, the main challenge is not simply to achieve high classification accuracy. A reliable prediction model must also account for the temporal nature of the available information. It must also consider the ethical issues involved in using student data for intervention purposes.

The UCI Predict Students' Dropout and Academic Success dataset contains 4424 student-level records collected from a higher-education institution. The students were enrolled in a range of undergraduate programs, including agronomy, design, nursing, journalism, management, social service, and technology-related fields. The dataset was compiled from several separate institutional databases and processed to identify and address anomalies, unexplained outliers, and missing values. The total number of records for each class, its percentage, and an interpretation of each class are summarized in Table 4.1. The majority class is Graduate, while Enrolled represents a considerably smaller proportion of the dataset. This results in a moderate but important class imbalance. A naïve classifier that always predicts the Graduate class could achieve nearly 50% accuracy. Such a model would have little practical value because it would fail to identify students at risk of dropping out. The dataset creators therefore describe the problem as an imbalanced three-class classification task.

Table 4.1: Class Distribution of Academic Outcomes

Outcome	**Records**	**Percentage**	**Interpretation**
Graduate	2,209	49.93%	Completed the program within its normal duration
Dropout	1,421	32.12%	Did not complete the program
Enrolled	794	17.95%	Still enrolled at the outcome-assessment point
Total	4,424	100%	Student-level analytical sample

The dataset contains 36 predictive attributes which can be organised in the following five groups:

- Student background
- Higher education pathway
- Family and financial context
- Early academic performance
- Macroeconomic conditions

This grouping is more meaningful than simply separating integer and continuous columns, because many integer-coded variables are nominal administrative categories rather than quantities with a true numeric order. One of its most analytically important

characteristics is the combination of static enrollment-time features with highly informative time-dependent semester-performance features. The provided documentation in the UCI repository identifies the data as integer, categorical, and real-valued attributes.

4.1.1 Student Background

The student-background group includes marital status, nationality, displaced status, gender, age at enrollment, and international-student status features. These features provide a broad picture of students' demographic backgrounds. Also describe the factors that influence their adjustment when they begin their studies. An example is marital status, which can be related to household responsibilities, employment, and financial independence. Moreover, its numerical codes represent categories and not ordered values. For this reason, it is treated as a categorical feature in the model. Age at enrollment is a numeric variable that helps distinguish students entering university at a conventional age from mature students. Its relationship with academic outcomes may not be linear. Older students may have greater maturity and a clearer sense of purpose, but they may also have additional workload (e.g., family responsibilities). Nationality and international-student status describe related but distinct aspects of student mobility. Nationality refers to citizenship, while international status reflects how the institution classifies the student. Both features may be linked to factors like language adaptation, accommodation, administrative requirements, social integration, and access to support networks. Displaced status indicates that a student has moved away from their home region to attend university. This could reflect greater independence and commitment, but it may also mean being further from family support and facing higher living costs. Gender is a sensitive demographic variable that may improve predictive performance (Realinho et al. 2021).

4.1.2 Higher Education Pathway

The higher education pathway group includes application mode, application order, course, daytime/evening attendance, previous qualification, previous qualification (grade), and admission grade. These variables describe how students entered the institution, their educational background, the programme they chose, and their academic standing at the time of admission.

Application mode is a high-cardinality categorical variable that captures different routes into higher education. This variable is informative because different admission routes can reflect different educational backgrounds, levels of prior experience, and challenges associated with transitioning into higher education. The differences between the numerical codes do not represent any meaningful distance or ranking. For this reason, the codes are treated as categorical rather than numerical values. Course identifies the student's programme of study and is treated as a nominal variable. Programme choice

may capture differences in subject area, grading practices, academic difficulty, student characteristics, and labor-market expectations. A strong association between course and dropout should not necessarily be interpreted as evidence that the programme itself causes students to leave. It may reflect differences in students' entry profiles or the specific context of each programme.

Application order has a clearer ordinal meaning because it indicates how highly the student ranked the selected programme (from their first choice to a later preference). Students who are admitted to a programme they ranked lower may have a weaker match between their interests and their eventual field of study, which could affect their engagement and persistence. Daytime/evening attendance can also provide useful information. Different schedules may reflect more traditional full-time study patterns or study arrangements that are more compatible with employment and family responsibilities. Its relationship with dropout is therefore likely to be more meaningful when considered alongside variables such as age, marital status, and financial circumstances.

Previous qualification records the type of educational credential a student held before admission. Previous qualification grade and admission grade measure academic performance on a scale from 0 to 200. These two grades may be correlated, and they refer to different stages of the student's educational pathway. The previous qualification grade reflects performance in the earlier qualification. The admission grade is the score used for entry into higher education. Both are particularly useful for an enrollment-time prediction model because they are available before the student begins his/her first semester. Their relationship with dropout may not be linear and could vary across programmes. This makes interactions between admission grade and course worth investigating (Realinho et al. 2021).

4.1.3 Family and Financial Context

The family and financial context group includes mother's qualification, father's qualification, mother's occupation, father's occupation, educational special needs, debtor, tuition fees up to date, and scholarship holder. These variables provide information about students' family backgrounds, socioeconomic circumstances, financial responsibilities, and access to institutional support.

Parental qualifications and occupations can serve as indicators of family educational and socioeconomic background. Students whose parents have experience with higher education may have greater access to information, guidance, and academic support. Occupational categories may provide an indirect indication of economic resources and employment stability. These variables can be viewed as contextual factors rather than measures of a student's ability or motivation. They may help identify broader inequalities that institutions could address through mentoring, better information, and targeted

support.

Financial variables are relevant when considering dropout prevention. Debtor status and tuition-fee status inform about financial difficulties that may affect a student's ability to continue, while scholarship-holder status indicates access to financial assistance. Students with debtor status may face financial pressure about continuing their studies. Scholarship status may reflect both financial need and institutional support. These variables can also be useful from an intervention perspective. Instead of using financial indicators to label students as being at high risk, institutions could use them to identify students who may benefit from financial counseling, payment-plan options, or information about available scholarships. The dataset documentation treats these variables as categorical institutional features that are available in the student record. Educational special needs is a sensitive feature. It requires additional care because it may indicate a need for additional institutional support (Realinho et al. 2021).

4.1.4 Early Academic Performance

The early-academic-performance group contains the most informative variables from a temporal perspective. These include curricular units 1st semester (credited), curricular units 1st semester (enrolled), curricular units 1st semester (evaluations), curricular units 1st semester (approved), curricular units 1st semester (grade), and curricular units 1st semester (without evaluations), together with the corresponding six variables for the second semester. These measures capture different aspects of a student's academic activity, including workload, engagement, participation in assessments, successful completion, and academic performance. The dataset provides performance information at the end of the first and second semesters along with information collected at enrollment.

Credited units refer to curricular units that have been recognized from previous studies or qualifications. They may provide some indication of a student's previous academic experience. On the other hand, enrolled units represent the amount of coursework a student takes in a given semester. A high number of enrolled units can reflect strong participation, but it may also indicate a workload that is difficult for the student to manage. Evaluations show how many enrolled units a student actually took part in for assessment. A noticeable difference between the number of enrolled and evaluated units may indicate disengagement from assessment, or other difficulties which affect participation.

Approved units provide a direct indication of successful academic progression and can be highly predictive of the student's final status. However, their predictive value raises an important methodological concern. These variables are not known when a student first enrolls. A model that uses first- or second-semester approval information should be described as an early-warning model operating at the end of a semester, rather than as an admission-time prediction model.

Semester grade provides another measure of academic performance, but it is more meaningful when considered alongside the number of enrolled and evaluated units. For instance, a high grade based on a small number of evaluated units may not indicate broad academic progress. On the other hand, a lower grade combined with participation across many units may describe a student who remains engaged but is struggling academically (Realinho et al. 2021).

4.1.5 Macroeconomic Conditions

The macroeconomic group includes the unemployment rate, inflation rate, and gross domestic product (GDP). These variables capture the broader economic conditions that students face during their period of enrollment. Their inclusion reflects the idea that academic persistence is shaped by more than just academic performance. Changes in the wider economy can influence household financial pressures, employment opportunities, and students' perceptions of affordability regarding their continued participation in higher education.

Students' decisions (stay in education or enter the labor market) may be influenced by the unemployment rate. Higher unemployment may reduce the opportunity cost of continuing university. On the other hand, lower unemployment makes employment a more attractive option. Inflation reflects changes in the cost of living and places additional financial pressure on students through various expenses (e.g., housing, transportation, study materials, and tuition-related costs). GDP provides a broader measure of economic activity and captures differences in overall economic conditions across enrollment periods. The above variables can vary over time rather than between individual students. Also, they may be correlated with one another and could partly act as proxies for the year of enrollment (Realinho et al. 2021).

4.2 Algorithm Implementation for Prediction

The preprocessed student records are represented as feature vectors and used as inputs to the proposed EHO-ELM classifier. The objective is to classify students into the three academic status categories (dropout, enrolled, and graduate). The input variables capture various aspects of the student profile (demographic characteristics, admission information, prior educational background, socioeconomic conditions, and first-year academic performance).

The EHO-ELM classifier uses an SLNN architecture in which the hidden and output neurons are implemented using MCUs. Each candidate SLNN is encoded as a chromosome within the evolutionary population. The chromosome contains the numerical parameters of the hidden-layer connections and the structural configuration of the MCUs

in both layers. This way, the evolutionary process does more than optimize connection weights. It also searches for an optimized neural architecture. The hidden-layer response matrix for each candidate architecture is obtained by passing the student feature vectors through the MCU-based hidden layer. The output-layer parameters are then calculated analytically using the Moore–Penrose pseudo-inverse. The evolutionary search aims not only to optimize the hidden-layer weights but also to find effective sub-cube combinations in all network layers.

The candidate networks are evolved using a modified GA. The better-performing models during this process are given greater opportunity to contribute to subsequent generations. The proposed method exchanges complete sub-cubes between parent MCUs rather than exchanging individual connection weights during crossover. This allows optimized sub-cubes to be preserved and transferred between generations. An additional adjustment mechanism is used to ensure that the resulting offspring retain valid MCU structures after crossover. A self-adaptive mutation mechanism is also introduced to maintain diversity within the population and limit premature convergence.

Once the evolutionary search is complete, the best-performing individual is selected as the final EHO-ELM classifier. The resulting model combines an optimized MCU structure in both layers and optimized hidden-layer weights. The proposed approach provides a flexible way to model the multiple factors associated with student dropout, continued enrollment, and graduation while retaining the simplicity associated with the ELM algorithm. The system architecture is visualized in Fig.4.1.

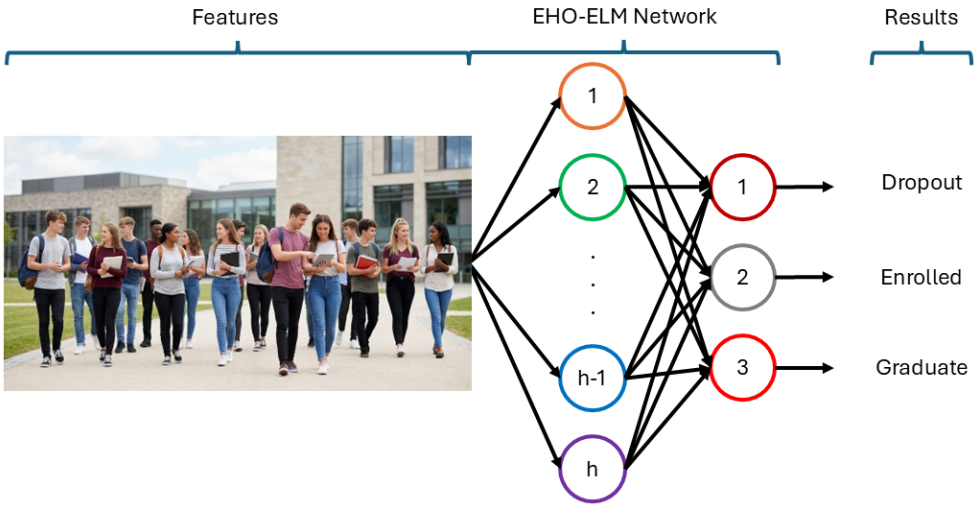

Fig. 4.1: The System Architecture. The EHO-ELM classifier is designed to predict students' academic performance using a range of preprocessed features. These features are provided to the network as feature-vector inputs. The EHO-ELM uses an SLNN with MCUs in the hidden and output layers to capture nonlinear relationships among the input features. A modified GA is then used to optimize the hidden-layer weights and the structure of the MCUs' sub-cubes. The evolved network with the highest classification *acc* classifies each student into one of three academic-status categories (dropout, enrolled, or graduate).

Chapter 5

Experimental Results

Overview

This chapter evaluates the proposed EHO-ELM algorithm for predicting student dropouts. Its performance was compared with 14 benchmark classifiers (ten backpropagation-based methods, a decision tree, ELM, OS-ELM, and MCU-ELM). The dataset was divided into separate training and test subsets, with the test data kept completely independent until the final evaluation. The training subset was further divided through cross-validation into training and validation sets. The validation results were used to limit overfitting in the backpropagation-based models and to assess candidate solutions during the evolutionary optimization process of EHO-ELM. All neural-network models used the same hidden-layer configuration and activation functions to ensure a fair comparison. The model parameters were initialized randomly, and each experiment was repeated ten times to reduce the effect of random initialization. The MCU-ELM method utilized one-dimensional sub-cubes, while EHO-ELM used multiple sub-cube dimensions (which increased its flexibility in constructing the hidden-layer structure). EHO-ELM achieved the highest classification accuracy among all evaluated methods, and the statistical significance of the results was evaluated using the Wilcoxon signed-rank test. The comparison between EHO-ELM and all competing methods produced statistically significant differences. The consistent performance observed across repeated experiments suggests that the proposed method is robust to the stochastic effects associated with both evolutionary optimization and neural-network initialization.

5.1 Experimental Setup

The proposed EHO-ELM algorithm for student academic success prediction was evaluated against a broad range of benchmark classifiers. These included ten backpropagation variants (BFGS, PBCG, FRCG, PRCG, SCG, GD, GDM, GDMALR, OSS, and RPROP),

DT, the traditional ELM, its online variant (OS-ELM), and the ELM adaptation for the MCUs (MCU-ELM). The "Predict Students' Dropout and Academic Success" dataset was initially divided into two partitions, with 80% of the instances allocated to the training set and the remaining 20% reserved as an independent test set. The test set was kept completely separate during model development and was used only for the final evaluation of generalization performance. This initial partitioning was applied to the DT, ELM, OS-ELM, and MCU-ELM methods. The 80% training portion was then further divided using 10-fold cross-validation to obtain the corresponding training and validation subsets. For the backpropagation-based neural network models, the training and validation sets were used to avoid overfitting. Validation performance was monitored throughout training, and the network configuration that achieved the best validation performance was retained. The same cross-validation procedure was also used with the proposed EHO-ELM method. The validation samples were treated as previously unseen data and were used to calculate the fitness of each candidate solution during the EHO optimization process. For this reason, the evolutionary search considered not only how well a candidate solution fitted the training data, but also how well it performed on unseen validation samples.

The main experimental parameters are reported in Table 5.1.

Table 5.1: Configuration of the Experimental Parameters

Parameter	**Notation**	**Setting**
Weights of Linear Neurons	w^l	$[-1,1]^n,\ n \in \mathbb{N}^*$
Weights of MCUs	w^{mc}	$[-1,1]^{p_{no}},\ p_{no} \in \mathbb{N}^*$
Input Vector	x	$[-1,1]^n,\ n \in \mathbb{N}^*$
Input-Layer Activation Function	g	*sigmoid*
Output-Layer Activation Function	g	*identity*
Number of Hidden-Layer Neurons	h	50
Number of Cross-Validation Folds	k	10
Number of Experimental Repetitions	$expNo$	10
Sub-cube Dimension of MCUs in MCU-ELM	$d_j^{\text{MCU-ELM}}$	$1,\ j \in \mathbb{N}^*$
Sub-cube Dimension of MCUs in EHO-ELM	$d_j^{\text{EHO-ELM}}$	$\{1,2,\ldots,5\},\ j \in \mathbb{N}^*$

For all neural-network-based methods, the weights and thresholds of conventional linear neurons and multi-cube units (MCUs) were initialized randomly. Their initial values were drawn from a continuous uniform distribution in the interval $[-1,1]$. Each neural network configuration used 50 hidden neurons, with the *sigmoid* function as the hidden-layer activation function. The output layer used the *identity* activation function, allowing the resulting output values to be passed directly to the classification decision mechanism. Each experiment was repeated ten times because random initialization can affect neural network performance. This provided a more reliable estimate of model performance than

relying on a single random initialization. In the MCU-ELM configuration, all MCUs were constructed using one-dimensional sub-cubes. On the other hand, EHO-ELM was allowed to optimize MCU structures with sub-cube dimensions ranging from 1 to 5. This gave the proposed method greater flexibility to explore different MCU topologies and identify more suitable structures for the classification problem.

5.2 Comparison Results

EHO-ELM was compared with 14 machine learning methods. Table 5.2 presents the classification results for EHO-ELM compared with these benchmark approaches. EHO-ELM achieved the highest classification accuracy among all evaluated methods (73.08%). This result demonstrates the strong predictive performance of the proposed approach for the classification task under consideration. The improvement can be attributed to the modified GA integration with an ELM architecture based on MCUs. EHO-ELM uses an evolutionary search process to identify more suitable network configurations, and it does not rely on randomly generated hidden-layer parameters.

Table 5.2: Classification Accuracy of the Compared Methods

Backpropagation-Based Methods		**Machine Learning and ELM-Based Methods**	
Method	**Accuracy**	**Method**	**Accuracy**
BFGS	45.61	DT	68.81
PBCG	56.40	OS-ELM	63.93
FRCG	50.78	ELM	72.01
PRCG	52.83	MCU-ELM	72.35
SCG	41.81	**EHO-ELM**	**73.08**
GD	37.98		
GDM	36.34		
GDMALR	42.34		
OSS	46.06		
RPROP	67.02		

The improved performance of EHO-ELM can be attributed to the joint optimization of the hidden-layer weights and the internal structure of MCUs. During the optimization process, EHO searches for suitable weight values and simultaneously identifies appropriate sub-cube configurations for the MCUs. This allows the hidden layer to generate more informative nonlinear representations of the input patterns. As a result, the network is better able to capture complex relationships among the selected features, which improves the separation of the different classes in the output space.

Another advantage of the proposed approach is that this improvement in represen-

tation does not require a substantial increase in the overall network architecture. The number of hidden-layer neurons remains fixed, while the evolutionary optimization process improves the structural diversity of neurons within that layer. The performance gain is associated with more effective parameter and MCU configuration selection rather than with an increase in the number of hidden neurons.

5.3 Assessment of Statistical Significance

The Wilcoxon signed-rank test was employed to determine whether the performance differences reported in Table 5.3 were statistically significant. This test is a non-parametric statistical procedure designed for analysing paired observations. It evaluates whether two related samples exhibit a systematic difference without the requirement that the underlying data follow a normal distribution. It was selected for comparing the performance of two machine-learning methods evaluated under identical experimental conditions (Conover 1999; Wilcoxon 1945). The test is based on the ranks of the absolute differences between paired observations, while retaining the sign of each difference. The resulting test statistic is subsequently used to calculate a p-value, which indicates the degree of evidence against the null hypothesis. In this context, the null hypothesis assumes that there is no systematic difference between the two methods. The p-value is described as the probability that the two populations are identical. A p-value lower than the selected significance level provides sufficient evidence to reject the null hypothesis.

Table 5.3: Wilcoxon Signed-Rank Test Results

Backpropagation-Based Methods		**Machine Learning and ELM-Based Methods**	
Method	***p*-value**	**Method**	***p*-value**
BFGS	3.8642×10^{-18}	DT	3.7604×10^{-18}
PBCG	3.8893×10^{-18}	OS-ELM	3.8742×10^{-18}
FRCG	3.8893×10^{-18}	ELM	1.0601×10^{-14}
PRCG	3.8933×10^{-18}	MCU-ELM	9.0547×10^{-9}
SCG	3.8926×10^{-18}		
GD	3.8862×10^{-18}		
GDM	3.8873×10^{-18}		
GDMALR	3.8812×10^{-18}		
OSS	3.8666×10^{-18}		
RPROP	3.8504×10^{-18}		

The results of the Wilcoxon signed-rank analyses are presented in Fig. 5.1. The EHO-ELM was compared with each of the 14 competing machine-learning methods. For all pairwise comparisons, the p-values were lower than 0.05. This indicates that the

performance differences between EHO-ELM and the considered methods were statistically significant at the 5% significance level. These findings strengthen the comparative results reported in Table 5.3, since the performance advantage of EHO-ELM is unlikely to be attributable solely to random experimental variation.

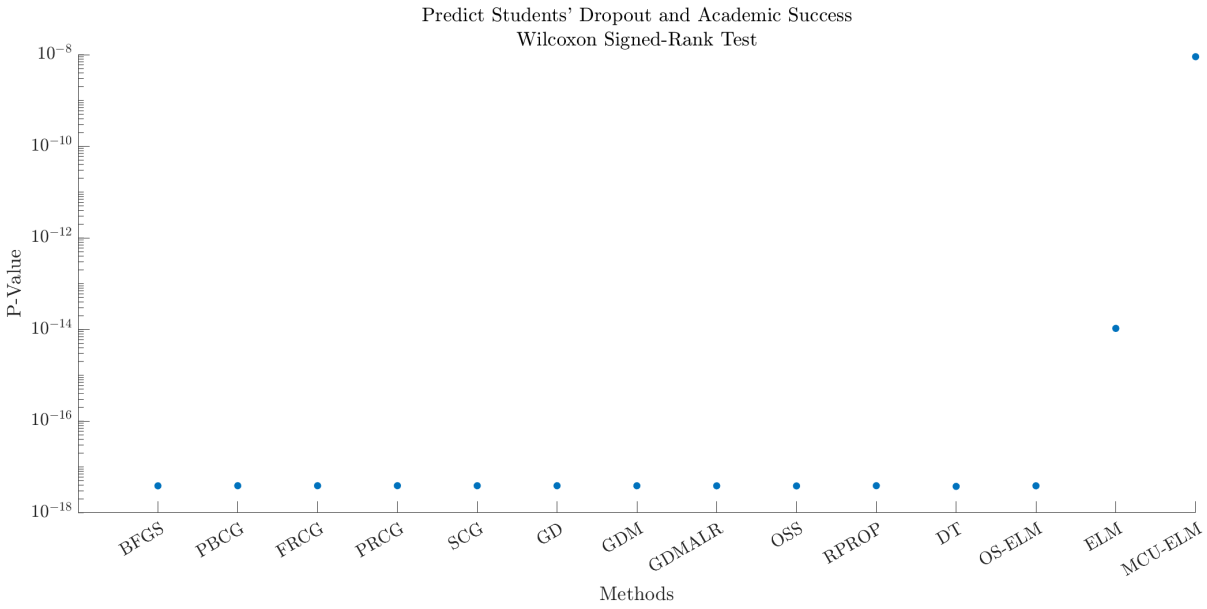

Fig. 5.1: Wilcoxon Signed-Rank Test Results. The Wilcoxon signed-rank test p-values obtained by comparing EHO-ELM with the 14 competing machine-learning methods on the student dropout and academic success dataset are shown in this figure. The p-values are presented on a logarithmic scale. In all comparisons, the p-values were below the 5% significance level, indicating statistically significant differences between EHO-ELM and the competing methods. These very small p-values provide strong evidence that the performance of EHO-ELM differs significantly from that of the evaluated alternatives.

The performance of EHO-ELM remained stable across the different experimental runs. This is important because evolutionary optimization and neural network training involve a degree of randomness. Different initial populations and random search paths can produce different optimization results. Also, the ELM component may be affected by the random initialization of its hidden-layer parameters. The small amount of variation observed across the repeated runs suggests that the proposed EHO-ELM model is robust to these sources of randomness. The statistical test results further support this conclusion. The Wilcoxon analysis shows that the differences between EHO-ELM and the competing methods are statistically significant. The consistent results across runs indicate that these differences can be reproduced under the same experimental conditions. These findings provide evidence of the proposed approach's reliability and stability.

Chapter 6

Limitations and Future Work

Overview

This chapter discusses the main limitations of the proposed EHO-ELM framework and outlines several possible directions for future research. Although EHO-ELM achieved better classification performance than the benchmark methods considered in this dissertation, it has a significant caveat. Its evolutionary optimization stage requires more computational effort than conventional ELM. This creates a trade-off between predictive performance and training efficiency. The use of multi-core CPUs or GPUs could reduce the time needed to evaluate multiple candidate network configurations and make the optimization process more efficient.

Several opportunities exist for further investigation. The proposed method could be tested on larger datasets from multiple institutions to assess its generalization performance across different educational environments. The prediction task could also be extended to distinguish between more detailed categories of academic outcomes. Another potential direction is to incorporate incremental versions of EHO-ELM. These variations enable the model to adapt over time by allowing predictions to be updated as new information about students becomes available.

Future research could also explore approaches for addressing class imbalance through multi-objective optimization and compare the modified GA with other evolutionary algorithms under comparable computational budgets. Ensemble versions of EHO-ELM could be investigated to determine whether combining several optimized models further improves prediction accuracy and stability. Overall, these directions would help provide a better understanding of the robustness, scalability, and practical value of EHO-ELM for predicting student dropout and academic success.

6.1 Limitations

Although EHO-ELM achieves better classification performance, this improvement comes with some additional computational cost. Most of this overhead is due to the evolutionary optimization stage. Several candidate ELM configurations are generated, trained, and evaluated during each generation. For this reason, the overall training process requires more computational resources than conventional ELM models that use a single network configuration.

This additional cost remains manageable because ELM uses a non-iterative training procedure. ELM does not repeatedly update the output weights, like conventional gradient-based neural networks. Once the hidden-layer parameters have been determined, the output weights can typically be obtained analytically. This reduces the computational effort required to evaluate each candidate solution during the evolutionary search. The independent evaluation of candidate networks makes the optimization process suitable for parallel computing. Distributing these evaluations across multiple central processing unit (CPU) cores or graphical processing units (GPUs) can significantly reduce execution time. This extra computational cost of EHO-ELM can be viewed as a reasonable trade-off for its improved predictive performance. This is significant in applications where classification accuracy and model reliability are more important than minimizing training time.

6.2 Future Work

The findings of this dissertation point to several opportunities for further development of the EHO-ELM framework. The following directions address aspects that were beyond the present study's scope and could help strengthen the theoretical foundation of the proposed approach.

A first area for future research is the theoretical analysis of the convergence behavior of EHO-ELM. GAs do not guarantee convergence to the global optimum, and their performance can be strongly affected by the choice of control parameters (Goldberg 1989; Mitchell 1996). Although the experimental results in this dissertation showed consistent performance across repeated runs, the proposed algorithm might fall into local optima if tested with more complex datasets. Additional research could examine the conditions that may lead to premature convergence (Holland 1992; Mitchell 1996) and investigate more flexible stopping criteria.

A second promising direction concerns the interpretability of EHO-ELM. DT algorithms are generally considered relatively easy to interpret because their predictions can be expressed through clear if–then rules (Quinlan 1986). The MCU-based hidden layer in EHO-ELM produces feature representations that are more difficult to interpret directly.

Future studies could therefore investigate methods for extracting feature-importance measures from the trained MCU structures. This would be particularly relevant in education, where institutions need not only reliable predictions but also meaningful explanations that can support decisions about appropriate interventions for students (Burgos et al. 2018).

Another important direction is the integration of richer and more diverse sources of student data. The increasing use of information technology in higher education has led institutions to generate large amounts of information beyond the enrollment-time and semester-level academic indicators considered in this dissertation (Wang et al. 2024). EDM techniques are already being used to analyze such information to improve learning and reduce student dropout (Dol and Jawandhiya 2023). Future versions of EHO-ELM could incorporate behavioral information from learning management systems, following approaches used in earlier studies for predicting student performance and dropout (Christou et al. 2023). Combining these different types of data would require appropriate modifications in the MCU encoding scheme to accommodate mixed data modalities.

The use of family, financial, and demographic attributes also raises important ethical considerations that should be examined in EHO-ELM's future applications. Financial and socioeconomic information should ideally be used to identify students who may benefit from support rather than to label them as high-risk students (Realinho et al. 2021). Another possible direction would be to develop EHO-ELM in a distributed privacy-preserving setting that allows institutions to contribute to model training without centralizing sensitive student records. This approach could help address privacy concerns and make the framework more suitable for use across different institutional settings.

Finally, an important step would be to evaluate EHO-ELM as part of a real-world early-warning system. Previous research suggests that early interventions contribute to the reduction of student dropout (Burgos et al. 2018; Tondeur et al. 2008), and an increasing number of institutions now recognize the significance of predictive information in supporting proactive intervention strategies (Boujmiraz et al. 2026). A natural extension of the present work would be to conduct a study examining whether interventions based on EHO-ELM predictions actually lead to improvements in student graduation rates.

Appendix A

Artificial Intelligence Use in the Dissertation

A.1 Purpose of the Appendix

This appendix provides transparent documentation of AI's use during the preparation of the present dissertation, entitled "Evolutionary Higher-Order Extreme Learning Machine for Predicting Student Academic Success." The use of AI was strictly limited to generating four conceptual images included in the thesis, with appropriate disclosures provided in their respective captions. AI was not used to write, rewrite, or translate the dissertation's text. It was also not used to search for or assess bibliographic sources, generate code, process data, conduct experiments, produce or interpret the research results.

The use of AI followed the human-in-the-loop (HITL) principle. AI worked as a supporting tool for producing initial visual representations, and not as an autonomous creator of the scientific work. Responsibility for selecting the topics, designing the figures, verifying their scientific accuracy, assessing their suitability, making the necessary modifications, and incorporating the final figures into the thesis rests exclusively with the author (Mosqueira-Rey et al. 2023).

A.2 Artificial Intelligence Usage Level

The present dissertation falls under AI Assessment Integrator (AIAS) Level 4 exclusively due to the creation of the four images. This level is appropriate because the AI-assisted images are part of the dissertation. This classification does not mean that AI contributed to the research methodology or conclusions. It applies only to the integration of visual material into the final document. The use of AI remained limited and controlled. The author determined the content and explanatory purpose of each image. He edited the generated graphics where necessary, and decided whether each result was suitable for

final use. No image was treated as evidence of scientific validity in itself.

A.3 Artificial Intelligence Use Summary

All AI-generated images were reviewed, edited, and approved by the author before being incorporated into the dissertation. The four images have captions stating that each conceptual graphic was generated with the assistance of either Perplexity AI or Google Gemini and subsequently edited by the author. This statement constitutes the direct documentation of AI use within the main body of the dissertation. These images are summarized in Table A.1.

Table A.1: Summary of AI Use

Stage	**AI tool**	**Purpose**	**AIAS**	**Human intervention**
Image 1	Perplexity AI	Biological neuron illustration	Level 4	Author review and editing
Image 2	Perplexity AI	TLU illustration	Level 4	Author review and editing
Image 3	Google Gemini AI	Tournament selection operator	Level 4	Author review and editing
Image 4	Google Gemini AI	Roulette wheel selection operator	Level 4	Author review and editing

A.4 Artificial Intelligence Detailed Description Use

AI was used solely to develop initial visual ideas for key concepts presented in the dissertation. These images were intended to serve an explanatory purpose for the theoretical background rather than a research purpose. The process involved four main stages. Initially, the author identified the concept to be illustrated and determined the elements that needed to appear in the figure. An AI tool was then used to generate an initial illustration using an appropriate prompt. The author subsequently reviewed the resulting image to verify the accuracy of its elements. Then, the author modified the appropriate result before incorporating it into the final document and adding the relevant disclosure to the figure caption. AI was not used to generate scientific claims, mathematical formulas, bibliographic references, code, or experimental data. The research and dissertation text were developed by the author independently of AI tools.

A.5 Illustrative Prompts

The prompts were descriptive and used solely to create the four images. Illustrative prompts included the following:

> "Create a clear conceptual diagram of a typical biological neuron in an academic style. It must include its main structures (dendrites, cell body, and axon) with labels. The diagram should have a simple and professional layout suitable for inclusion in a dissertation."

> "Create an academic-style diagram of a threshold logic unit with five inputs and their corresponding weights. Show the weighted inputs being summed, the threshold θ, and the resulting binary output. Use labels and a simple, easy-to-follow layout suitable for inclusion in a dissertation."

> "Create a clear conceptual diagram illustrating the tournament selection operator used in a genetic algorithm. Show several candidate individuals and their fitness values. They must be followed by the random selection of a subset to form a tournament. Illustrate their fitness values comparison and the selection of the fittest individual for reproduction. Use labels, directional arrows, and a simple, professional layout suitable for inclusion in a dissertation."

> "Create a clear conceptual diagram illustrating the roulette-wheel selection operator used in a genetic algorithm. Show a population of candidate individuals with different fitness values, represented as proportional sectors of a roulette wheel. Indicate a randomly selected point on the wheel and clearly show which individual is selected for reproduction. Use labels, directional arrows, and a simple, professional layout suitable for inclusion in a dissertation."

The prompts above are presented indicatively rather than as a complete record of the interactions. The final form of the figures was not adopted mechanically but resulted from an evaluation and editing procedure by the author.

A.6 Human Interventions Analysis

Human intervention was involved throughout the entire process. The author identified the concepts that would benefit from visual representation and revised the generated result. Each image was considered in relation to the accompanying text. AI did not determine the scientific interpretation of the concepts or decide where the images would be placed in the text. The final decision to accept, modify, or reject each generated image was made solely by the author.

A.7 Critical Evaluation

The AI utilization when developing visual representations of complex concepts was very helpful. On the other hand, AI-generated images have limitations because a generated figure does not ensure scientific accuracy. For this reason, every generated image required human review and editing before it could be included in the dissertation.

The AI's main contribution in this process was to support the creative stage and wasn't used as a tool to produce scientific knowledge. Its use required reviewing each generated element and determining whether it was consistent with the theoretical framework.

This experience demonstrated that AI tools can serve as useful supplementary design aids when used within a strict HILT framework. In this case, AI was limited to the creation of four images, with its use clearly disclosed in the relevant captions. The author remained fully responsible for reviewing, evaluating, and finalizing the images. This approach helps maintain both transparency and academic integrity.

Declaration of Responsibility

The author declares that AI was used only to generate the four images identified as AI-generated in their respective captions. No AI tools were used in the preparation of this dissertation. All other material is based on the author's own research. The author takes full responsibility for the content of the dissertation.

Bibliography

Ahmad, M. R., Razali, S., Jadin, M. S., & Sulaiman, M. H. (2026). A novel artificial hummingbird algorithm-optimized extreme learning machine for high-resolution dc power forecasting in rooftop retrofitted photovoltaic systems. *Franklin Open*, *16*, 100654. https://doi.org/10.1016/j.fraope.2026.100654

Askarzadeh, A. (2016). A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. *Computers & Structures*, *169*, 1–12. https://doi.org/10.1016/j.compstruc.2016.03.001

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In J. J. Grefenstette (Ed.), *Genetic algorithms and their applications: Proceedings of the second international conference on genetic algorithms* (pp. 14–21). Lawrence Erlbaum Associates.

Ben-Israel, A., & Greville, T. N. E. (2003). *Generalized inverses: Theory and applications* (2nd, Vol. 15). Springer. https://doi.org/10.1007/b97366

Bharathi, T., & Manavalan, R. (2026). Data-driven student performance prediction: A hybrid approach using weighted extreme learning machine and coati optimization algorithms. *SN Computer Science*, *7*(1), 72. https://doi.org/10.1007/s42979-025-04689-5

Blickle, T., & Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, *4*(4), 361–394. https://doi.org/10.1162/evco.1996.4.4.361

Boujmiraz, S., Darhmaoui, H., & El Maliani, A. D. (2026). Predicting student performance: A comprehensive review of machine learning, deep learning, and explainable AI approaches. *Computers and Education: Artificial Intelligence*, *10*(1), 100548. https://doi.org/10.1016/j.caeai.2026.100548

Broyden, C. G. (1970a). The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, *6*(1), 76–90. https://doi.org/10.1093/imamat/6.1.76

Broyden, C. G. (1970b). The convergence of a class of double-rank minimization algorithms 2. the new algorithm. *IMA Journal of Applied Mathematics*, *6*(3), 222–231. https://doi.org/10.1093/imamat/6.3.222

Burgos, C., Campanario, M. L., de la Peña, D., Lara, J. A., Lizcano, D., & Martínez, M. A. (2018). Data mining for modeling students' performance: A tutoring action plan to prevent academic dropout. *Computers & Electrical Engineering*, *66*, 541–556. https://doi.org/10.1016/j.compeleceng.2017.03.005

Cao, J., Lin, Z., & Huang, G.-B. (2012). Self-adaptive evolutionary extreme learning machine. *Neural Processing Letters*, *36*(3), 285–305. https://doi.org/10.1007/s11063-012-9236-y

Cao, L., Yue, Y., Zhang, Y., & Cai, Y. (2021). Improved crow search algorithm optimized extreme learning machine based on classification algorithm and application. *IEEE Access*, *9*, 20051–20066. https://doi.org/10.1109/ACCESS.2021.3054799

Cauchy, A.-L. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées [Séance du 18 octobre 1847]. *Comptes Rendus de l'Académie des Sciences*, *25*, 536–538.

Cheng, R., & Jin, Y. (2015). A competitive swarm optimizer for large scale optimization. *IEEE Transactions on Cybernetics*, *45*(2), 191–204. https://doi.org/10.1109/TCYB.2014.2322602

Christou, V. (2026). Higher-order extreme learning machine. *Cognitive Systems Research*, *97*, 101474. https://doi.org/10.1016/j.cogsys.2026.101474

Christou, V., Tsoulos, I., Loupas, V., Tzallas, A. T., Gogos, C., Karvelis, P. S., Antoniadis, N., Glavas, E., & Giannakeas, N. (2023). Performance and early drop prediction for higher education students using machine learning. *Expert Systems with Applications*, *225*, 120079. https://doi.org/10.1016/j.eswa.2023.120079

Christou, V., Tzallas, A. T., Gogos, C., Tsipouras, M. G., Tsoumanis, G., & Giannakeas, N. (2025). An evolutionary extreme learning machine algorithm for multi-cube unit single-layer neural networks. *Applied Soft Computing*, *171*, 112788. https://doi.org/10.1016/j.asoc.2025.112788

Christou, V., Tzallas, A. T., Tsoumanis, G., Tsipouras, M. G., Dimopoulos, D., Varvarousis, D., Gogos, C., Ploumis, A., & Giannakeas, N. (2026). An evolutionary higher-order extreme learning machine method for Parkinson's disease classification. *Brain Network Disorders*, *4*(4). https://doi.org/10.1016/j.bnd.2026.07.001

Conover, W. J. (1999). *Practical nonparametric statistics* (3rd). John Wiley & Sons.

Cuevas, R., Ntoumanis, N., Fernandez-Bustos, J. G., & Bartholomew, K. (2018). Does teacher evaluation based on student performance predict motivation, well-being, and ill-being? *Journal of School Psychology*, *68*, 154–162. https://doi.org/10.1016/j.jsp.2018.03.005

De Jong, K. A., & Spears, W. M. (1992). A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, *5*(1), 1–26. https://doi.org/10.1007/BF01530777

Dehghani, M., Montazeri, Z., Trojovská, E., & Trojovský, P. (2023). Coati optimization algorithm: A new bio-inspired metaheuristic algorithm for solving optimization problems. *Knowledge-Based Systems*, *259*, 110011. https://doi.org/10.1016/j.knosys.2022.110011

Dol, S. M., & Jawandhiya, P. M. (2023). Classification technique and its combination with clustering and association rule mining in educational data mining — a survey. *Engineering Applications of Artificial Intelligence*, *122*, 106071. https://doi.org/10.1016/j.engappai.2023.106071

Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 39–43. https://doi.org/10.1109/MHS.1995.494215

El Aissaoui, O., El Alami El Madani, Y., Oughdir, L., Dakkak, A., & El Allioui, Y. (2020). A multiple linear regression-based approach to predict student performance. *Advanced Intelligent Systems for Sustainable Development (AI2SD'2019) Volume 1 – Advanced Intelligent Systems on Artificial Intelligence, Software, and Data Science*, 9–23. https://doi.org/10.1007/978-3-030-36653-7_2

Endah, S. N., Widodo, A. P., Fariq, M. L., Nadianada, S. I., & Maulana, F. (2017). Beyond back-propagation learning for diabetic detection: Convergence comparison of gradient descent, momentum and adaptive learning rate. *2017 1st International Conference on Informatics and Computational Sciences (ICICoS)*, 189–194. https://doi.org/10.1109/ICICOS.2017.8276360

Eshtay, M., Faris, H., & Obeid, N. (2020). A competitive swarm optimizer with hybrid encoding for simultaneously optimizing the weights and structure of extreme learning machines for classification problems. *International Journal of Machine Learning and Cybernetics*, *11*(8), 1801–1823. https://doi.org/10.1007/s13042-020-01073-y

Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science*, *6*(3), 205–254. https://doi.org/10.1207/s15516709cog0603_1

Fletcher, R. (1970). A new approach to variable metric algorithms. *The Computer Journal*, *13*(3), 317–322. https://doi.org/10.1093/comjnl/13.3.317

Fletcher, R., & Powell, M. J. D. [Michael J. D.]. (1963). A rapidly convergent descent method for minimization. *The Computer Journal*, *6*(2), 163–168. https://doi.org/10.1093/comjnl/6.2.163

Gao, H., Xu, T., & Zhang, N. (2025). Robust kernel extreme learning machines for postgraduate learning performance prediction. *Heliyon*, *11*(1), e40919. https://doi.org/10.1016/j.heliyon.2024.e40919

Ginantra, N. L. W. S. R., Bhawika, G. W., Achmad Daengs, G. S., Panjaitan, P. D., Arifin, M. A., Wanto, A., Amin, M., Okprana, H., Syafii, A., & Anwar, U. (2021). Performance one-step secant training method for forecasting cases. *Journal of*

Physics: Conference Series, *1933*(1), 012032. https://doi.org/10.1088/1742-6596/1933/1/012032

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning* (1st). Addison-Wesley.

Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of Computation*, *24*(109), 23–26. https://doi.org/10.1090/S0025-5718-1970-0258249-6

Golub, G. H., & Van Loan, C. F. (2013). *Matrix computations* (4th). Johns Hopkins University Press.

Google. (2026). *Google gemini* [AI-powered assistant and multimodal model. Accessed: 2026-08-18.]. https://gemini.google.com

Gurney, K. (2018). *An introduction to neural networks* (1st). CRC Press. https://doi.org/10.1201/9781315273570

Gurney, K. (1989). *Learning in networks of structured hypercubes* [PhD thesis]. Department of Electrical Engineering, Brunel University [Available as Technical Memorandum CN/R/144].

Han, F., Yao, H.-F., & Ling, Q.-H. (2013). An improved evolutionary extreme learning machine based on particle swarm optimization. *Neurocomputing*, *116*, 87–93. https://doi.org/10.1016/j.neucom.2011.12.062

Heidari, A. A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., & Chen, H. (2019). Harris hawks optimization: Algorithm and applications. *Future Generation Computer Systems*, *97*, 849–872. https://doi.org/10.1016/j.future.2019.02.028

Hinterding, R. (1995). Gaussian mutation and self-adaption for numeric genetic algorithms. *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, *1*, 384–389. https://doi.org/10.1109/ICEC.1995.489169

Holland, J. H. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence* (1st MIT Press ed.). MIT Press.

Huang, G.-B., Chen, L., & Siew, C.-K. (2006a). Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, *17*(4), 879–892. https://doi.org/10.1109/TNN.2006.875977

Huang, G.-B., Zhou, H., Ding, X., & Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *42*(2), 513–529. https://doi.org/10.1109/TSMCB.2011.2168604

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2004). Extreme learning machine: A new learning scheme of feedforward neural networks. *2004 IEEE International Joint*

Conference on Neural Networks (IEEE Cat. No. 04CH37541), *2*, 985–990. https://doi.org/10.1109/IJCNN.2004.1380068

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006b). Extreme learning machine: Theory and applications. *Neurocomputing*, *70*(1–3), 489–501. https://doi.org/10.1016/j.neucom.2005.12.126

Karimkashi, S., & Kishk, A. A. (2010). Invasive weed optimization and its features in electromagnetics. *IEEE Transactions on Antennas and Propagation*, *58*(4), 1269–1278. https://doi.org/10.1109/TAP.2010.2041163

Kishore Kumar, N., & Schneider, J. (2017). Literature survey on low rank approximation of matrices. *Linear and Multilinear Algebra*, *65*(11), 2212–2244. https://doi.org/10.1080/03081087.2016.1267104

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, *4*(2), 87–112. https://doi.org/10.1007/BF00175329

Li, K., Kong, X., Lu, Z., Liu, W., & Yin, J. (2014). Boosting weighted ELM for imbalanced learning. *Neurocomputing*, *128*, 15–21. https://doi.org/10.1016/j.neucom.2013.05.051

Li, S., & Liu, T. (2021). Performance prediction for higher education students using deep learning. *Complexity*, *2021*(1), 9958203. https://doi.org/10.1155/2021/9958203

Lian, L. (2025). Improved sparrow search algorithm optimized extreme learning machine for ultra-short-term wind speed prediction. *Journal of Atmospheric and Solar-Terrestrial Physics*, *267*, 106693. https://doi.org/10.1016/j.jastp.2025.106693

Ling, Q., Tan, K., Wang, Y., Li, Z., & Liu, W. (2025). Evolutionary extreme learning machine based on an improved MOPSO algorithm. *Neural Computing and Applications*, *37*(12), 7733–7750. https://doi.org/10.1007/s00521-024-10578-4

Lipowski, A., & Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, *391*(6), 2193–2196. https://doi.org/10.1016/j.physa.2011.12.004

Liu, Q., Zhang, C., Li, Z., Peng, T., Zhang, Z., Du, D., & Nazir, M. S. (2024). Multi-strategy adaptive guidance differential evolution algorithm using fitness-distance balance and opposition-based learning for constrained global optimization of photovoltaic cells and modules. *Applied Energy*, *353*, 122032. https://doi.org/10.1016/j.apenergy.2023.122032

Ludwig, P. E., Reddy, V., & Varacallo, M. A. (2023, July). Neuroanatomy, neurons [PMID: 28723006; Bookshelf ID: NBK441977; Last updated: 2023 Jul 24]. In *Statpearls [internet]*. StatPearls Publishing. https://www.ncbi.nlm.nih.gov/books/NBK441977/

Ma, J., Li, H., Gao, W., & Xie, J. (2025). An ensemble learning algorithm based on multimodal differential evolution and extreme learning machine. *Memetic Computing*, *17*(2), 21. https://doi.org/10.1007/s12293-025-00461-7

MathWorks. (2026). Moore-penrose pseudoinverse [Accessed: 2026-07-28]. https://www.mathworks.com/help/matlab/ref/pinv.html

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *5*(4), 115–133. https://doi.org/10.1007/BF02478259

Mengcan, M., Chen, X., & Xie, Y. (2021). Constrained voting extreme learning machine and its application. *Journal of Systems Engineering and Electronics*, *32*(1), 209–219. https://doi.org/10.23919/jsee.2021.000018

Mirjalili, S., Gandomi, A. H., Mirjalili, S. Z., Saremi, S., Faris, H., & Mirjalili, S. M. (2017). Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*, *114*, 163–191. https://doi.org/10.1016/j.advengsoft.2017.07.002

Mitchell, M. (1996). *An introduction to genetic algorithms*. The MIT Press.

Møller, M. F. (1993). A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, *6*(4), 525–533. https://doi.org/10.1016/S0893-6080(05)80056-5

Moore, E. H. (1920). On the reciprocal of the general algebraic matrix. *Bulletin of the American Mathematical Society*, *26*(9), 394–395. https://doi.org/10.1090/S0002-9904-1920-03322-7

Morris-Rosendahl, D. J. (2010). A glossary of relevant genetic terms. *Dialogues in Clinical Neuroscience*, *12*(1), 116–120. https://doi.org/10.31887/DCNS.2010.12.1/dmrosendahl

Mosqueira-Rey, E., Hernández-Pereira, E., Alonso-Ríos, D., Bobes-Bascarán, J., & Fernández-Leal, Á. (2023). Human-in-the-loop machine learning: A state of the art. *Artificial Intelligence Review*, *56*(4), 3005–3054. https://doi.org/10.1007/s10462-022-10246-w

Penrose, R. (1955). A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, *51*(3), 406–413. https://doi.org/10.1017/S0305004100030401

Perplexity AI. (2026). *Perplexity ai* [AI-powered search and answer engine. Accessed: 2026-08-18.]. https://www.perplexity.ai

Polak, E., & Ribière, G. (1969). Note sur la convergence de méthodes de directions conjuguées. *Revue française d'informatique et de recherche opérationnelle. Série rouge*, *3*(R1), 35–43. https://doi.org/10.1051/M2AN/196903R100351

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods [Translated from Zh. Vychisl. Mat. Mat. Fiz. 4(5) (1964) 791–803]. *USSR Com-*

putational Mathematics and Mathematical Physics, *4*(5), 1–17. https://doi.org/10.1016/0041-5553(64)90137-5

Powell, M. J. D. [M. J. D.]. (1977). Restart procedures for the conjugate gradient method. *Mathematical Programming*, *12*(1), 241–254. https://doi.org/10.1007/BF01593790

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106. https://doi.org/10.1007/BF00116251

Rao, P. S., Varma, G. P., Chinta, D. P., Gottapu, K., Lakshmi, T. V. H., Naidu, K. A., & Saritha, M. (2025). Financial time series prediction using pelican optimized extreme learning machine with reduced weights. *Computational Economics*, *66*(6), 1–18. https://doi.org/10.1007/s10614-025-10869-5

Rathod, N., & Wankhade, S. (2022). Optimizing neural network based on cuckoo search and invasive weed optimization using extreme learning machine approach. *Neuroscience Informatics*, *2*(3), 100075. https://doi.org/10.1016/j.neuri.2022.100075

Realinho, V., Vieira Martins, M., Machado, J., & Baptista, L. (2021). Predict students' dropout and academic success. https://doi.org/10.24432/C5MC89

Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. *Proceedings of the IEEE International Conference on Neural Networks (ICNN'93)*, 586–591. https://doi.org/10.1109/ICNN.1993.298623

Rumelhart, D. E., Hinton, G. E., & McClelland, J. L. (1986a). A general framework for parallel distributed processing. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition, volume 1: Foundations* (pp. 45–76). MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986b). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Sa'ad, M. I., Mustafa, M. S., et al. (2020). Student prediction of drop out using extreme learning machine (ELM) algorithm. *2020 2nd International Conference on Cybernetics and Intelligent System (ICORIS)*, 1–6. https://doi.org/10.1109/ICORIS50180.2020.9320831

Seo, S., Jo, J., Hamza, M., & Kim, Y. (2024). Improving I-ELM structure through optimal addition of hidden nodes: Compact I-ELM. *Scientific Reports*, *14*(1), 22782. https://doi.org/10.1038/s41598-024-74446-w

Shanno, D. F. (1970). Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, *24*(111), 647–656. https://doi.org/10.1090/S0025-5718-1970-0274029-3

Shin, Y., & Ghosh, J. (1991). The pi-sigma network: An efficient higher-order neural network for pattern classification and function approximation. *Proceedings of the 1991*

International Joint Conference on Neural Networks (IJCNN), *1*, 61–66. https://doi.org/10.1109/IJCNN.1991.155142

Storn, R., & Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341–359. https://doi.org/10.1023/A:1008202821328

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the third international conference on genetic algorithms* (pp. 2–9). Morgan Kaufmann.

Tian, Z., & Chen, H. (2021). A novel decomposition-ensemble prediction model for ultra-short-term wind speed. *Energy Conversion and Management*, *248*, 114775. https://doi.org/10.1016/j.enconman.2021.114775

Tondeur, J., Van Keer, H., van Braak, J., & Valcke, M. (2008). ICT integration in the classroom: Challenging the potential of a school policy. *Computers & Education*, *51*(1), 212–223. https://doi.org/10.1016/j.compedu.2007.05.003

Trefethen, L. N., & Bau, I., David. (2022). *Numerical linear algebra, twenty-fifth anniversary edition* (25th). SIAM. https://doi.org/10.1137/1.9781611977165

Trojovský, P., & Dehghani, M. (2022). Pelican optimization algorithm: A novel nature-inspired algorithm for engineering applications. *Sensors*, *22*(3), 855. https://doi.org/10.3390/s22030855

Viswakiran, M., Padma, S., Hussain, G. S. S., Khan, A. S., & Naidu, J. R. (2024). Predictive modeling of academic success using extreme learning machine. In V. Sharmila, S. Kannadhasan, A. R. Kannan, P. Sivakumar, & V. Vennila (Eds.), *Challenges in information, communication and computing technology: Proceedings of the 2nd international conference on challenges in information, communication, and computing technology (iccicct 2024), april 26th & 27th, 2024, namakkal, tamil nadu, india* (pp. 732–737). CRC Press. https://doi.org/10.1201/9781003559085-126

Wang, S., Wang, F., Zhu, Z., Wang, J., Tran, T., & Du, Z. (2024). Artificial intelligence in education: A systematic literature review. *Expert Systems with Applications*, *252*, 124167. https://doi.org/10.1016/j.eswa.2024.124167

Wei, Y., Lv, H., Chen, M., Wang, M., Heidari, A. A., Chen, H., & Li, C. (2020). Predicting entrepreneurial intention of students: An extreme learning machine with gaussian barebone harris hawks optimizer. *IEEE Access*, *8*, 76841–76855. https://doi.org/10.1109/ACCESS.2020.2982796

Werbos, P. J. (1974, August). *Beyond regression: New tools for prediction and analysis in the behavioral sciences* [PhD thesis]. Harvard University [Committee on Applied Mathematics].

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, *1*(6), 80–83. https://doi.org/10.2307/3001968

Wu, T., Yao, M., & Yang, J. (2016). Dolphin swarm algorithm. *Frontiers of Information Technology & Electronic Engineering*, *17*(8), 717–729. https://doi.org/10.1631/FITEE.1500287

Wu, T., Yao, M., & Yang, J. (2017). Dolphin swarm extreme learning machine. *Cognitive Computation*, *9*(2), 275–284. https://doi.org/10.1007/s12559-017-9451-y

Xing, B., & Gao, W.-J. (2013). Fruit fly optimization algorithm. In *Innovative computational intelligence: A rough guide to 134 clever algorithms* (pp. 167–170, Vol. 62). Springer. https://doi.org/10.1007/978-3-319-03404-1_11

Xue, J., & Shen, B. (2020). A novel swarm intelligence optimization approach: Sparrow search algorithm. *Systems Science & Control Engineering*, *8*(1), 22–34. https://doi.org/10.1080/21642583.2019.1708830

Yang, X.-S., & Deb, S. (2009). Cuckoo search via lévy flights. *2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, 210–214. https://doi.org/10.1109/NABIC.2009.5393690

Ye, Z., Yang, Y., Chen, Y., & Chen, H. (2025). Predicting academic performance levels in higher education: A data-driven enhanced fruit fly optimizer kernel extreme learning machine model. *Journal of Bionic Engineering*, *22*(4), 1940–1962. https://doi.org/10.1007/s42235-025-00716-6

Zha, L., Ma, K., Li, G., Yang, J., & Fang, Q. (2022). An improved extreme learning machine with self-recurrent hidden layer. *Advanced Engineering Informatics*, *54*, 101736. https://doi.org/10.1016/j.aei.2022.101736

Zhang, X., Zhou, Y., Huang, H., & Luo, Q. (2022). Enhanced salp search algorithm for optimization extreme learning machine and application to dew point temperature prediction. *International Journal of Computational Intelligence Systems*, *15*(1), 98. https://doi.org/10.1007/s44196-022-00160-y

Zhao, W., Wang, L., & Mirjalili, S. (2022). Artificial hummingbird algorithm: A new bio-inspired optimizer with its engineering applications. *Computer Methods in Applied Mechanics and Engineering*, *388*, 114194. https://doi.org/10.1016/j.cma.2021.114194

Zong, W., Huang, G.-B., & Chen, Y. (2013). Weighted extreme learning machine for imbalance learning. *Neurocomputing*, *101*, 229–242. https://doi.org/10.1016/j.neucom.2012.08.010