Robust Energy Consumption Prediction with a Missing Value-Resilient Metaheuristic-based Neural Network in Mobile App Development

Seyed Jalaleddin Mousavirad[1] and Luís A. Alexandre[2]

[1]Universidade da Beira Interior, Covilhã, Portugal, jalaleddin.mousavirad@ubi.pt

[2]NOVA LINCS, Universidade da Beira Interior, Covilhã, Portugal, lfbaa@di.ubi.pt

Abstract

Energy consumption is a fundamental concern in mobile application development, bearing substantial significance for both developers and end-users. Moreover, it is a critical determinant in the consumer's decision-making process when considering a smartphone purchase. From the sustainability perspective, it becomes imperative to explore approaches aimed at mitigating the energy consumption of mobile devices, given the significant global consequences arising from the extensive utilisation of billions of smartphones, which imparts a profound environmental impact. Despite the existence of various energy-efficient programming practices within the Android platform, the dominant mobile ecosystem, there remains a need for documented machine learning-based energy prediction algorithms tailored explicitly for mobile app development. Hence, the main objective of this research is to propose a novel neural network-based framework, enhanced by a metaheuristic approach, to achieve robust energy prediction in the context of mobile app development. The metaheuristic approach here plays a crucial role in not only identifying suitable learning algorithms and their corresponding parameters but also determining the optimal number of layers and neurons within each layer. To the best of our knowledge, prior studies have yet to employ any metaheuristic algorithm to address all these hyperparameters simultaneously. Moreover, due to limitations in accessing certain aspects of a mobile phone, there might be missing data in the data set, and the proposed framework can handle this. In addition, we conducted an optimal algorithm selection strategy, employing 13 metaheuristic algorithms, to identify the best algorithm based on accuracy and resistance to missing values. The comprehensive experiments demonstrate that our proposed approach yields significant outcomes for energy consumption prediction.

Keywords: Energy Consumption, Evolutionary Computation, Swarm Intelligence, Neural Network, Neuro-evolution

1 Introduction

In contemporary mobile application development, developers express profound concerns regarding the impact of their Apps on the battery life of smartphones. Many apps are harassed in App stores because of their tendency to consume too much power [5, 12]. Consequently, developers diligently acknowledge this energy issue and actively seek assistance to address it, albeit finding satisfactory guidance remains a rarity [19, 28]. To promote enhanced energy efficiency, mobile device manufacturers offer comprehensive guidelines for developers. Moreover, the energy consumption factor significantly influences the overall satisfaction of mobile device consumers. Notably, recent research has underscored the pivotal role of battery life as a decisive criterion shaping the smartphone purchasing decisions of 1898 surveyed users in the US [5, 12]. This mounting concern over energy consumption has been corroborated by staggering statistics, revealing that a substantial majority of consumers, approximately 90%, struggle with the phenomenon of low battery anxiety [19, 28].

Considering sustainability, finding ways to make mobile devices use less energy is crucial. The billions of phones we use today also significantly impact the world. Remarkably, the ecological impact of our digital device utilisation, encompassing smartphones and tablets, is projected to surpass that of the aviation industry in terms of its contribution to global warming [9]. Therefore, prioritising energy efficiency in mobile devices is crucial in addressing environmental concerns and fostering sustainable practices.

Analysing and optimising energy consumption in mobile devices pose a challenging and time-consuming endeavour for both end-users and developers. Effectively monitoring an application's energy usage necessitates conducting numerous tests under diverse conditions and across multiple devices, which demands considerable time and effort. Developers frequently employ multiple monitoring tools, leading to contextually bound findings. Moreover, the Android platform demonstrates substantial diversity, accommodating a vast population of approximately 3.3 billion Android smartphone users in 190 countries globally.[1]. However, developers' understanding of hardware performance remains confined to their devices. Without access to suitable tools and expertise, they encounter challenges in effectively comparing energy consumption patterns among different Apps or comprehending App behaviour across diverse devices and scenarios. Additionally, the energy dynamics of an App exhibit variations contingent on its usage, including factors such as operating system versions and the activation status of various hardware components. Such distinctions must be thoughtfully considered during comparative analyses to ensure accurate evaluations.

Over the past few years, numerous research studies [2, 10, 11, 24, 33, 38] have explored energy-aware programming trends in the Android platform and sought to identify more efficient alternatives. However, automatic energy consumption prediction has received limited attention. For instance, [25] investigated the potential of proxy metrics like CPU usage and memory write operations as predictors for energy consumption in a music streaming application on mobile devices. Their experiments involved two Apps, namely

[1]Source: http://www.bankmycell.com/blog/how-many-android-users-are-there

Spotify music and podcast App, to understand the correlation between the selected proxies and energy consumption. In another study [26], researchers conducted a large-scale user study to measure the energy consumption characteristics of 15500 BlackBerry smartphone users. They compiled a substantial data set to create the Energy Emulation Toolkit (EET), facilitating developers in comparing their Apps' energy requirements with real user energy traces. Additionally, [18] leveraged machine learning (ML) techniques and environment data to predict a smartphone's next unlock event. Through a 2-week field study involving 27 participants, they demonstrated the feasibility of accurately predicting unlock events, leading to energy savings by optimising software-related background processes. They suggested reducing energy consumption using short-term predictions to avoid unnecessary executions or initiating resource-intensive tasks, such as OS updates, when the phone is locked. By predicting the next phone unlock event, smartphones can pre-collect sensor data or prepare content to enhance the user experience during subsequent usage. Moreover, a recent study by [22] identified two image-based proxies for energy consumption: image file size and image quality. They found that increasing image file size and quality directly impacts energy consumption. To address this issue, they proposed a multi-objective approach to optimise image size and quality, while maintaining a small energy footprint. However, this approach needed to have consideration of user opinions. As a result, [21] incorporated user preferences by altering the encoding representation to refine the energy consumption prediction process further. In another recent research, [23] presents a machine-learning technique for predicting smartphone energy consumption. Their approach utilises the Histogram-based Gradient Boosting Classification Machine (HGBC) for the modelling phase. However, an essential aspect of smartphones, namely permissions, is overlooked in their algorithm. As a result, some users may only grant some of the required permissions to an App. Consequently, certain features may be absent from the feature set due to the unavailability of corresponding permissions.

This paper proposes a robust energy consumption prediction with a missing value-resilient metaheuristic-based artificial neural networks (ANNs) approach in mobile App development. ANNs are popular and influential machine learning models inspired by the human brain's structure and functioning. However, to achieve optimal performance with ANNs, it is essential to tune hyperparameters carefully. For ANNs, hyperparameters include the number of layers, the number of neurons in each layer, learning rate, solver, and momentum, among others. Tuning hyperparameters is critical because it significantly impacts the neural network's performance and generalisation ability. If the hyperparameters are not set appropriately, the network may suffer from overfitting, where it performs well on the training data but fails to generalise to new, unseen data. On the other hand, setting hyperparameters too conservatively may result in underfitting, where the network fails to capture the underlying patterns in the data. Also, finding the correct number of layers and neurons is essential for controlling the model's complexity. Deep neural networks with multiple layers and neurons can potentially learn intricate patterns in the data but also require more data and computational resources. Shallower networks may not capture complex relationships adequately. Therefore, finding the optimal balance is crucial for achieving the best performance. As a result, this paper proposes a novel missing value-resilient metaheuristic-based ANN for finding the optimal hyper-parameters.

More precisely, this paper presents several contributions, encompassing the following key aspects:

- We introduce a missing value-resilient machine learning-based approach for predicting energy consumption in mobile App development. Notably, this work is the first study to address energy prediction in this domain while accounting for missing values in the data.
- Our machine learning algorithm leverages a tuned artificial neural network (ANN) during the modelling phase. By carefully adjusting the ANN's hyperparameters, we enhance its performance and predictive capabilities, ensuring more accurate and reliable energy consumption forecasts.
- We propose a general metaheuristic-based approach for hyperparameter tuning in ANNs. The utilisation of metaheuristic techniques facilitates an efficient search for optimal hyperparameter configurations, contributing to the overall effectiveness of our energy prediction model.
- As our proposed metaheuristic approach remains algorithm-independent, we applied our proposed method to 13 basic and advanced metaheuristic algorithms to identify the most robust and effective strategy for hyperparameter tuning.
- To the best of our knowledge, most of the employed metaheuristic algorithms have not been previously utilised in hyperparameter tuning for ANNs, nor in other contexts. Consequently, this paper represents the first endeavour to investigate the effectiveness of these algorithms in addressing this specific problem.
- A pivotal goal of this paper involves benchmarking various metaheuristic algorithms for hyperparameter tuning in ANNs. Through rigorous evaluation, we aim to identify the most suitable and efficient metaheuristic technique for our energy prediction model.
- Lastly, from a sustainability perspective, our approach is significant in green computing. By optimising energy consumption predictions in mobile App development, this methodology can contribute to more energy-efficient practices, aligning with sustainable computing.

The remainder of this paper is organised as follows. Section 2 briefly explains metaheuristic-based Multi-layer Neural Network construction, while the proposed algorithm is described in Section 3. Section 4 validates the proposed algorithm in the different conditions. Finally, we present the conclusions in Section 5.

2 Metaheuristic-based Multi-layer Neural Network Construction

In this section, we delve into the Multi-layer Neural Networks (MLNN) structure and its significance in energy consumption prediction for mobile app development. The MLNN is a fundamental component of our proposed framework, serving as a powerful tool for learning complex patterns and making accurate predictions. The MLNN is a versatile and powerful learning model comprising multiple layers of interconnected neurons, each contributing to the overall learning process. The key components of the MLNN architecture include:

- Input Layer: This is the first layer that receives the features extracted from the (transformed) data set. The dimensionality of the input data determines the number of neurons in this layer.
- Hidden Layers: The hidden layers are responsible for capturing the intricate relationships and representations of the input features. The number of hidden layers and neurons within each layer is a crucial hyperparameter significantly impacting the model's performance.
- Output Layer: The final layer in the MLNN produces the prediction based on the learned features and relationships from the hidden layers. The number of neurons in the output layer corresponds to the number of desired classes.

In the context of MLNN, finding good hyperparameters holds utmost importance as it directly impacts the performance and effectiveness of the neural network in learning and making predictions. Hyperparameters are configurations or settings predetermined before training the model and not learned from the data. Finding a well-optimised set of hyperparameters is crucial, as outlined below:

1. Model Performance: the choice of hyperparameters significantly affects the MLNN's ability to learn complex patterns and relationships within the data. Properly tuned hyperparameters can enhance model performance, resulting in more accurate and reliable predictions.
2. Avoiding Overfitting and Underfitting: properly tuning hyperparameters helps strike a balance between overfitting and underfitting. Overfitting occurs when the MLNN becomes overly complex and memorises the training data, leading to a poor generalisation of unseen data. Conversely, underfitting happens when the model is too simplistic to capture the underlying patterns, resulting in low predictive performance. Finding the correct hyperparameters helps prevent these issues and ensures the model generalises well to new data.
3. Generalisation: properly tuned hyperparameters enable the MLNN to generalise effectively to unseen data. This robustness allows the model to perform well in real-world scenarios and various applications.

Metaheuristic-based MLNN construction represent a powerful fusion of two cutting-edge techniques by integrating the flexibility and optimisation capabilities of metaheuristic algorithms with the predictive prowess of MLNN. Metaheuristic algorithms exhibit a remarkable ability to explore vast solution spaces and escape local optima, making them well-suited for finding optimal hyperparameters and network architectures. Leveraging these metaheuristic algorithms to guide the learning and optimisation process of

MLNN allows for enhanced performance, improved generalisation, and the ability to tackle diverse and challenging tasks.

3 The Proposed Algorithm

This research introduces a universal approach using metaheuristic techniques to discover optimal hyper-parameters within MLNN for mobile app development. This methodology can be seamlessly combined with various optimisation algorithms. The process involves integrating our suggested method into 13 distinct algorithms. Initially, we outline the data set we employ, then explain how we represent solutions and define the objective function. Subsequently, by incorporating the proposed approach into 13 foundational optimisation algorithms, we generate 13 new and distinct algorithms.

3.1 Data set

In this study, we utilised the GreenHub data set [27] as the foundation of our research. This comprehensive data set comprises over 23 million instances, spanning 900 brands and 5,000 models and covering 160 countries. It encompasses three distinct types of information: the sample data set, device data set, and App processes data set. The sample data set presents various details on different smartphone settings, while the device data set includes features related to smartphone brands and specifications. On the other hand, the App processes data set contains information about the installed applications on each smartphone.

Our research focused solely on the sample data set, deliberately disregarding the App processes and device data sets. This decision proposes a predictive algorithm that remains independent of the installed applications. Incorporating App processes would introduce challenges due to the regional variations in commonly used applications. For instance, while ”WeChat” may be prevalent in China, ”WhatsApp” may hold that position in Europe and in the US. Consequently, including such features would lead to sparsity in the data set, making the predictive process complex and less reliable. By excluding this data, we can ensure the robustness and geographical independence of the results. Similarly, the device data set also presents similar challenges, which led us to focus solely on Android settings. This approach ensures that our results are independent of varying device specifications across regions.

The sample data set contains seven distinct features, from which we selected 32 relevant features for our study. These features encompass various aspects such as battery details (charger status, health, voltage, and temperature), CPU states (usage, uptime, and sleep time), network details (network type, mobile network type, mobile data status, mobile data activity, roaming enabled, wifi status, wifi signal strength, and wifi link speed), samples (battery state, battery level, memory free, memory user, network status, screen brightness, and screen on), settings (Bluetooth enabled, location enabled, power saver enabled, flashlight enabled, NFC enabled, and developer mode), and storage details (free and total memory, memory active, and memory inactive). Additional information can be referenced from [27] for further specifics on these features.

3.2 Preprocessing

This particular step in our study involves transforming and encoding the existing data set to ensure compatibility with machine learning algorithms. First, we systematically removed instances labelled as "battery-state=charging" from the data set, as our primary objective is to predict energy consumption, specifically in the discharging state. In addition, we introduced a novel metric, Energy Consumption Per Minute (ECPM), designed to quantify the energy consumed per minute by a smartphone. The ECPM is defined as

$$ECPM = \frac{BT_{state_1} - BT_{state_2}}{TS_{state_1} - TS_{state_2}} \times 60 \tag{1}$$

where BT_{state_i} represents the battery level in state i, and TS_{state_i} refers to the timestamp in state i. State$_1$ and State$_2$ are consecutive states, capturing the current settings of the smartphone, including features such as Bluetooth and Wifi states (on/off). The key assumption here is that the smartphone settings remain unchanged between consecutive states. ECPM is a valuable metric for evaluating energy consumption, with lower ECPM values indicating reduced energy usage. To ensure the metric's accuracy, instances where the 'battery-state=charging' state is positioned between two 'battery-state=discharging' states, along with the instance immediately following it, are excluded from the ECPM calculations.

As the present study focuses on energy prediction viewed as a classification problem, each instance has been categorised into one of three classes: safe, warning, or critical status, based on the Energy Consumption Per Minute (ECPM) metric. The assignment of each sample to a class is contingent upon the mobile App developer's discretion. For instance, an ECPM value of 0.5 might be considered high for one developer, while another could consider it safe. Since the developer's preferences are unknown, a histogram analysis has been employed to determine the class assignments. Figure 1 shows the histogram for ECPM for 120000 randomly-selected instances. We tried to distribute the samples to different classes so that the classes remain almost balanced. However, it can be changed according to the developer's preference. According to the histogram, instances with ECPM values less than 0.5 are assigned to the first class. At the same time, those exceeding 1.5 are allocated to a separate class, with the remaining instances falling into the third class.

3.3 Proposed Neuroevolution Model

This section explains our proposed neuroevolution model for predicting energy consumption in mobile App development. For this, three issues to be determined are the representation of each candidate solution, the objective function, and the search strategy, which we define in the following before describing our proposed method step-by-step.

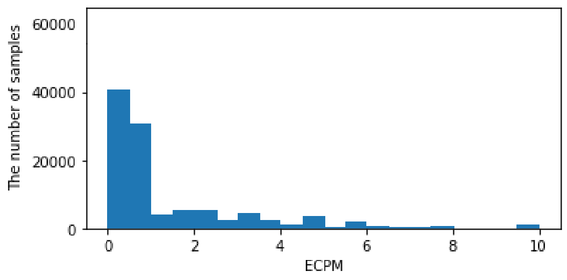

Figure 1: Histogram of randomly-selected instances

3.3.1 Representation of the candidate solutions

Our proposed algorithm aims to find all hyper-parameters in a MLNN. To the best of our knowledge, in none of the previous research studies, this set of hyper-parameters has been optimised simultaneously. Our proposed neuro-evolution model can optimise the parameters and the number of layers and neurons in each layer. As a result, we propose a two-segment representation $N_D + N_L$. N_D decision variables are responsible for finding the parameters (except the number of neurons in each layer). In contrast, the second part is assigned to the number of neurons in each layer. It is crucial to note that the initial segment in our proposed representation has a constant length, while the subsequent segment exhibits a variable length.

The first segment of our proposed representation is allocated explicitly to all hyper-parameters except the number of neurons, which is a string with a length of 9. The hyper-parameters assigned to each decision variable, their types and values are given in Table 1. We can see that all hyper-parameters in the first segment are real values except the solver. A solver, here, refers to an optimisation algorithm used to train an MLNN, and it is responsible for finding the optimal set of weights and biases. However, there are different solvers in the literature. As a result, the proposed representation also fulfils the task of finding the proper solver among distinct solvers (here are 10). An integer value is assigned to this decision variable, ranging from 1 to 10, with each number denoting a specific solver. The solvers include adaptive moment estimation (Adam) [13], Adam with adaptive learning rate method (Adadelta) [40], Adam with decoupled weight decay regularisation (AdamW) [17], infinity norm-based ADAM (Adamax) [13], accelerated stochastic gradient descent (ASGD) [29], Nesterov momentum into Adam (NAdam) [4], rectified Adam (RAdam) [16], root mean square propagation (RMSprop) [7], resilient backpropagation algorithm (Rprop) [31] and stochastic gradient descent method (SGD) [1], and each corresponds to a unique number from 1 to 10. Every solver within the experiment requires specific hyperparameters while disregarding others. For instance, the Adam algorithm requires the learning rate, β_1, β_2, and weight decay. Consequently, we employed a technique referred to as "**Selective Exclusion**" to calculate the objective function. Selective Exclusion entails the deliberate omission or exclusion of certain elements, a process contingent upon the chosen solver.

The subsequent segment of the proposed representation relates to allocating neurons in each hidden layer. It is important to note that the appropriate number of layers is controlled in the main algorithm.

Table 1: Hyper-parameters of the first segment of the proposed representation.

Hyper-parameter	type	values
Learning rate (lr)	Real	[0,1]
Weight decay	Real	[0,0.2]
ρ	Real	[0,1]
$\beta 1$	Real	[0.8,1]
$\beta 2$	Real	[0.8,1]
λ	Real	[0,1]
momentum	Real	[0,1]
Solver (optimiser)	Integer	{1,2,...,10}

As a result, the size of the second segment varies according to the number of layers. For instance, if the number of layers is 4, the second segment will have a size of 4, with each decision variable in this segment corresponding to a distinct layer.

3.4 Concealing Strategy

In real-world mobile Apps, if a user does not grant permissions for certain features, those features may be missing from the feature set. In other words, in a typical situation, it is likely that there will be missing values in the data. Therefore, our proposed method integrates a concealing (mask) strategy to the MLNN for handling missing values. The concealing strategy employs binary masks to address missing values within the input data. These binary masks serve as indicators, where 1 represents a known or observed feature, while 0 indicates a missing or unobserved feature.

Mathematically, we define $x \in \mathbb{R}^p$ as a vector representing an input sample comprising p features. We assume that the features in x can be divided into two distinct sets: q observed features denoted as $x^o \in \mathbb{R}^q$, and r missing features denoted as x^m, where the sum of observed and missing features equals the total number of features, p $(q + r = p)$. In our concealing strategy, the activation function of the k-th $(k = 1, 2, ..., s)$ neuron in the first hidden layer of an MLNN is defined as

$$a_k^{(1)} = f\left(\sum_{i=1}^{q} w_{ik}^{(1)} x_i^o + \sum_{j=1}^{r} w_{jk}^{(1)} x_j^m + b_k^{(1)}\right) \tag{2}$$

where$a_k^{(1)}$ denotes the activation of the k-th neuron in the first hidden layer, f represents the chosen activation function (e.g., sigmoid, ReLU, etc.), $\sum_{i=1}^{q} w_{ik}^{(1)} x_i^o$ is the weighted sum of observed features from the input, where x_i^o is the i-th observed feature, and $w_{ik}^{(1)}$ is the weight associated with the connection between the i-th observed feature and the k-th neuron in the first hidden layer. $\sum_{j=1}^{r} w_{jk}^{(1)} x_j^m$ is the weighted sum of missing features from the input, where x_j^m is the j-th missing feature, and $w_{jk}^{(1)}$ is the weight associated with the connection between the j-th missing feature and the k-th neuron in the first hidden layer. Also, $b_k^{(1)}$ represents the bias term for the k-th neuron in the first hidden layer.

In our proposed approach, the missing values in the data set are represented by a binary mask matrix

$M \in \{0,1\}^{n \times p}$, where n is the number of samples. The binary mask matrix M is defined as follows: if the j-th feature of the i-th sample is missing, M_{ij} takes the value 0, else it takes the value 1. Hence, for each sample i, $M_i = (M_{i1}, M_{i2}, ..., M_{ip})$ represents the binary mask vector indicating missing (0) and observed (1) features. To apply the masking approach, the original feature vectors are multiplied element-wise with the binary mask matrix: $x_i^m = x_i \odot M_i$, where $x_i^m \in \mathbb{R}^p$ represents the masked feature vector for the i-th sample. By performing the element-wise multiplication with the binary mask, the missing values are effectively masked or set to zero, while the observed values remain unchanged. This allows subsequent analysis, such as the calculation of activation function, since the activation function cannot be calculated in the state where there are missing values.

We have used the concealing strategy here since 1) it preserves the data structure by keeping track of missing values while retaining the remaining observed values, 2) it retains the sample size since by masking missing values, the original sample size is maintained, ensuring that all available data points are utilised in the analysis, and 3) it helps to prevent biased estimates or erroneous conclusions that could arise from ignoring missing values, leading to more accurate predictions. It is worth mentioning that the concealing strategy is similar to imputing missing values by 0. Although in concealing method, the network knows what values have been missed, which may be useful for future applications.

3.5 Objective Function

The objective function in this study indicates the quality of a specific architecture designed for energy consumption in our MLNN model. The objective function takes into consideration the classification error, which is computed as

$$\text{Classification error} = \frac{100}{P} \times \sum_{p=1}^{P} \xi(x_p) \tag{3}$$

with

$$\xi(x_p) = \begin{cases} 1 & \text{if } o_p \neq d_p \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where P is the number of samples, and for each input vector in training data, the corresponding desired output is d_p, and o_p is the predicted output. To ensure reliable and robust results, the objective function is evaluated using k-fold cross-validation.

3.6 Search Strategies

For search strategies, there is flexibility in selecting various types of population-Based Metaheuristic (PBMH) techniques. However, it is impractical to analyse every PBMH technique available in the literature due to its vast and diverse collection. Therefore, we selected several well-established and state-of-the-art variants commonly utilised in evolutionary and swarm computing, leading to 13 algorithms. This wide range of choices not only aids in choosing the best algorithm for our search strategy but also can be used as a benchmark study.

3.6.1 Base Algorithms

- **Genetic algorithm (GA)** [39]: The GA is the most established algorithm that consists of three key components: crossover, mutation, and selection. Crossover merges information from parent solutions, while mutation introduces random alterations to one or multiple elements of a candidate solution. The selection process in GA determines which solutions are carried forward to subsequent iterations based on the principle of favouring the most successful (”fittest”) solutions. There are several ways for the selection process, while this paper focuses on tournament selection due to some advantages like less bias towards the fittest.
- **Differential Evolution (DE)** [34]: DE employs three primary operators: mutation, crossover, and selection. The mutation operator generates candidate solutions by calculating a mutant vector using a scaled difference among candidate solutions. The most famous mutation is ”DE/rand/1”, which represented as

  $$v_i = x_{r1} + F(x_{r2} - x_{r3}), \tag{5}$$

  where F denotes a scaling factor, and x_{r1}, x_{r2}, and x_{r3} represent three distinct randomly chosen candidate solutions from the existing population. The crossover operator combines the mutant vector with a target vector selected from the current population. Finally, the selection operator determines the candidate solution to be retained.
- **Memetic Algorithm (MA)** [20]: MA is a search strategy that incorporates both a population-based algorithm, such as GA and a local search approach. In the particular version we employed, each agent is assigned a probability indicating whether or not a local search operation should be performed.
- **Particle Swarm Optimisation (PSO)** [32]: PSO is a PBMH inspired by swarm behaviour, where the updating process relies on the best position of each candidate solution and a global best position. The velocity vector of a particle is updated as

  $$v_{t+1} = \omega v_t + c_1 r_1 (p_t - x_t) + c_2 r_2 (g_t - x_t), \tag{6}$$

  where t denotes the current iteration, r_1 and r_2 represent random numbers drawn from a uniform distribution in the range [0,1], p_t corresponds to the personal best position, and g_t represents the global best position. Also, c_1 and c_2 are called cognitive coefficient and social coefficient, respectively, while ω signifies inertia weight.
- **Covariance Matrix Adaptation Evolution Strategy (CMA-ES)** [8]: CMA-ES algorithm is another PBMH employed in this paper that belongs to the class of evolution strategies. It utilises a prob-

abilistic model to adaptively update the search distribution and guide the search towards promising regions of the search space. At each iteration, CMA-ES maintains a population of candidate solutions represented as a set of multivariate Gaussian-distributed samples. The algorithm dynamically adjusts this distribution's mean vector and covariance matrix based on the success or failure of the sample solutions. The fitness evaluations of the candidate solutions guide the update of the mean vector. The algorithm calculates the weighted average of the successful solutions to obtain a new mean vector estimate. This allows CMA-ES to explore regions of the search space with higher fitness values. To update the covariance matrix, CMA-ES employs a rank-one update mechanism. It calculates the outer product of the mean-centred successful solutions, which is then used to update the covariance matrix estimate. This update ensures that the covariance matrix aligns with the successful search directions and adapts to the underlying problem landscape.

3.6.2 Advanced Algorithms

- **Self-organising Hierarchical PSO with Jumping Time-varying Acceleration Coefficients (HPSO)** [30]: The key features of HPSO can be summarised as follows:

  1. Mutation is incorporated into the PSO algorithm.
  2. A new approach, the self-organising hierarchical particle swarm optimiser with TVAC, is introduced. It focuses on the social and cognitive aspects of the particle swarm strategy when determining each particle's updated velocity. Additionally, particles are reset to their initial state if they become stagnant in the search space.
  3. The PSO algorithm includes a time-varying mutation step size.

- **Chaos PSO (CPSO)** [15]: CPSP incorporates two techniques to enhance its performance: an adaptive inertia weight factor (AIWF) and a chaotic local search (CLS). The AIWF dynamically adjusts the inertia weight, denoted as ω, in the original PSO algorithm based on the objective function value as

$$\omega = \begin{cases} \omega_{min} + \frac{(\omega_{max}-\omega_{min})(f-f_{min})}{f_{avg}-f_{min}} & f \leq f_{avg} \\ \omega_{max} & f \geq f_{avg} \end{cases}, \tag{7}$$

where ω_{max} and ω_{min} represent the maximum and minimum values of ω, respectively. The current objective function value of a candidate solution is denoted as f, while f_{avg} and f_{min} represent the average and minimum values of all candidate solutions, respectively.

To further improve the effectiveness of CPSO, the CLS operator is employed as a local search mechanism around the best position. The CLS equation is as follows:

$$cx_i^{k+1} = 4cx_i^k(1 - cx_i) \tag{8}$$

where cx_i represents the i-th chaotic variable, while k denotes the iteration number. The chaotic variable cx_i is distributed between 0 and 1. The equation exhibits chaotic behaviour when the initial value cx_0 lies in the range (0, 1), and the corresponding value x_0 is not equal to 0.25, 0.5, or 0.75.

- **Comprehensive Learning PSO (CLPSO)** [14]: To prevent premature convergence, a comprehensive learning (CL) strategy is proposed for particle learning. Instead of relying solely on their own best positions (pbest), all particles' pbest can be utilised to adjust the velocity of each particle. The updating scheme in CLPSO is defined as follows:

$$v_{t+1}^i = \omega v_t^i + c_1 r(pbest_{fi(d)}^f - x_t^i), \tag{9}$$

where $fi(d)$ determines which particles' $pbest$ a specific particle i should follow. The decision to learn from nearby particles is based on the comprehensive learning probability, PC. For each dimension, a random number is generated from a uniform distribution. If the generated random number exceeds $PC(i)$, the particle updates based on its pbest. Otherwise, it updates by incorporating information from nearby particles.

- **DE with Self-Adaptation Populations (SAP-DE)** [37]: It introduces self-adaptive features for population size, crossover rates, and mutation rates. Two variants, $SAP{-}DE{-}ABS$ and $SAP{-}DE{-}REL$, are proposed to determine the population size parameter π.

  In $SAP - DE - ABS$, π is calculated by rounding the sum of an initial population size NP_{ini} and a random value drawn from a standard normal distribution. On the other hand, $SAP - DE - REL$ initialises the population size parameter by sampling a value from a uniform distribution ranging from -0.5 to +0.5. During each stage, the π parameter needs to be updated. In $SAP - DE - REL$, the current population size is adjusted based on a growth rate, either increasing or decreasing by a certain percentage. In $SAP - DE - ABS$, the population size for subsequent generations is determined as the average population size attribute from all individuals in the current population.

- **Adaptive Differential Evolution with Optional External Archive (JADE)** [41]: JADE adjusts the likelihood of generating offspring by either mutation operators based on the success ratio over the previous 50 generations, leading to select, gradually, the best mutation strategy for a specific problem. Also, JADE employs a normal distribution for setting the scale factor for each candidate solution, leading to maintaining both local (with small F_i values) and global (with large F_i values) capabilities simultaneously.

- **Success-History-based Parameter Adaptation for Differential Evolution (SHADE)** [35]: The SHADE algorithm is a variant of DE that incorporates adaptive mechanisms to enhance its performance in solving optimisation problems. It focuses on adapting the control parameters of DE during the evolution process. The main idea behind SHADE is to maintain a success-history archive that stores information about previous successful solutions. This archive guides the search towards promising regions of the search space. In the archive updating strategy, SHADE uses a memory-based mechanism to update the success-history archive. Only the best solutions, which outperform their parents, are considered for archive inclusion. This ensures that the archive contains high-quality solutions that have exhibited better performance. Another characteristic of SHADE is the parameter adaptation strategy that aims to dynamically adjust the control parameters of DE, such as the crossover rate and the mutation factor, based on the information stored in the success-history archive. By considering the performance of previously successful solutions, SHADE adapts these parameters to suit better the characteristics of the optimisation problem being solved.

- **SHADE using Linear Population Size Reduction (LSHADE)** [36]: LSHADE is an extension of the SHADE algorithm, which aims to further improve the performance of SHADE by introducing a linear population size reduction mechanism. LSHADE introduces two key components: the population size reduction factor and the memory size factor. The population size reduction factor determines the rate at which the population size is decreased over iterations. By gradually reducing the population size, LSHADE allocates more computational resources to the most promising solutions, thereby enhancing search efficiency. On the other hand, the memory size factor determines the size of the success-history archive used in LSHADE. This archive stores information about previous successful solutions and guides the search towards promising regions of the search space. By adjusting the memory size factor, LSHADE controls the balance between exploration and exploitation during optimisation.
- **Phasor PSO(PPSO)** [6]: PPSO suggests using control parameters based on a mathematical concept known as the phase angle (θ). The velocity update in each iteration is determined as

$$v_i^{iter} = |cos\theta_i^{iter}|^{2*sin\theta_i^{iter}} \times (Pbest_i^{iter} - x_i^{iter}) + |sin\theta_i^{iter}|^{2*cos\theta_i^{iter}} \times (Gbest_i^{iter} - x_i^{iter}) \tag{10}$$

where the velocity of each particle is influenced by the values of the phase angle (θ) and is adjusted based on the differences between the personal best ($Pbest$), global best ($Gbest$), and the current position (x) of the particle.

3.7 General Structure

This paper proposes a neuroevolution approach for energy consumption prediction in mobile App development. The general structure of the proposed approach is shown in Figure 2. Our proposed approach is independent of the optimisation algorithm. As a result, we applied the approach to 13 search strategies, leading to 13 algorithms for energy prediction in mobile App development, namely, GA-AMLNN, DE-AMLNN, MA-AMLNN, PSO-AMLNN, CMA-ES-AMLNN, HPSO-AMLNN, CPSO-AMLNN, CLPSO-AMLNN, SAPE-DE-AMLNN, JADE-AMLNN, SHADE-AMLNN, LSHADE-AMLNN, and PPSO-AMLNN. From the figure, variable *i* controls the number of hidden layers. It can start with an initial value and reach a maximum number of layers. The value of this variable determines the size of each candidate solution. For example, if the value of this variable is 4, the size of the second part of the candidate solution is also 4. Since all the PBMH algorithms used are population-based, several MLNNs with different hyper-parameters and numbers of neurons are created randomly in the initialisation step, which are represented by the representation strategy introduced in Section 3.3.1. Based on the objective function, defined in Section 3.5, the search strategies are responsible for finding the best architecture of MLNN. Each search strategy has a different way of searching, leading to different results. Various search methods aim to evolve the generated population towards an evolved population using their operators. After the search strategy performs the search operation for *i*-th layer, one is added to the number of layers. If the number of layers does not exceed the maximum number of layers, the search strategy is repeated for the new layer. The final evaluation should be based on the final model found and the test set. 10-fold cross-validation (10CV) has been used for this purpose. In each iteration, nine folds are used for training the weights of the neural network and one remaining part is used for validating the neural network. It is worth mentioning that these training and validation parts used for training the neural network are part of the original training set. In other words, the original data set is divided into two parts: training and test, and the training set here is used for 10-fold cross validation.

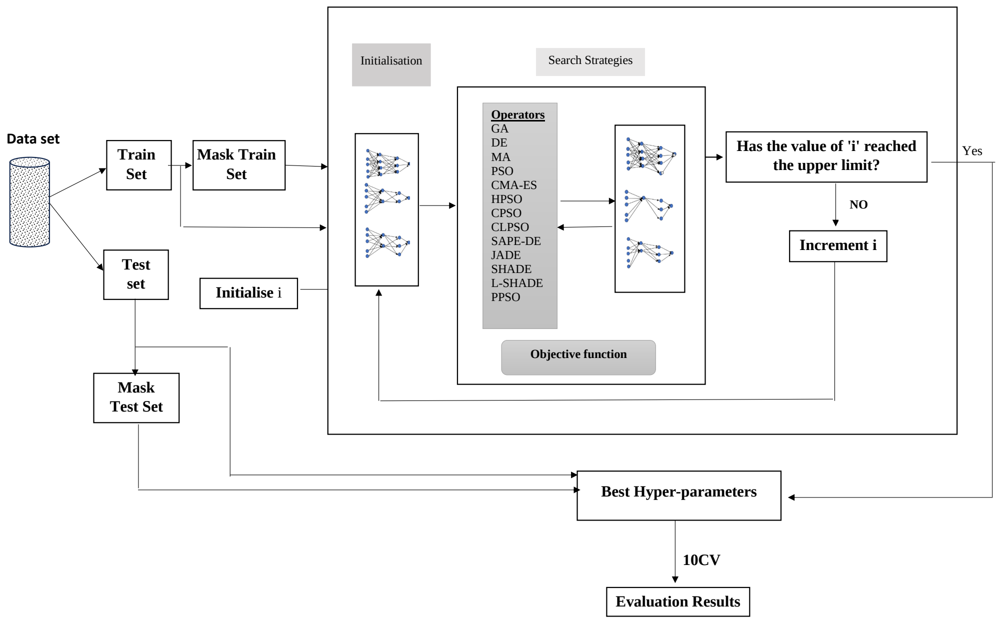

Figure 2: The general framwork of our proposed algorithm.

4 Results and Discussions

In this section, we conducted a comprehensive set of experiments to evaluate the effectiveness of our proposed approach. To achieve this, our evaluation focuses on two primary metrics: accuracy and F-measure. Also, to ensure robustness, the whole process is repeated 10 times and statistics such as mean and standard deviation are reported. A detailed description of the data set we used can be found in Section 3.1. For our evaluation, we have defined two scenarios: 1) evaluation in the case where there is no missing value, and 2) assessment in the case where there are missing values in the data set.

4.1 First Scenario

In this section, we have evaluated our proposed approach in the case where there are no missing values in our data set. To this end, the proposed approach is integrated with 13 PBMHs. While the representation and objective functions remained consistent across all algorithms, each algorithm employed a distinct search strategy. Also, the population size for all algorithms is set at 10. Since the objective function in the proposed algorithm is computationally expensive, we used a limited number of fitness evaluations for the proposed algorithm. In other words, the number of function evaluations for the optimisation process for each layer is selected as 30, and the maximum number of layers is 8. The remaining parameters, as listed in Table 2, are set to their default values.

Table 3 compares the performance of various algorithms using the accuracy and F-measure metrics. Among the algorithms evaluated, PSO-AMLNN, SAPE-DE-AMLNN, and DE-AMLNN emerge as the top three performers based on accuracy and F-measure metrics. PSO-AMLNN achieves the highest mean accuracy (87.63) and F-measure (87.96) scores, demonstrating its effectiveness in accurately classifying the test data and achieving a balance between precision and recall. SAPE-DE follows closely behind, with mean accuracy (87.38) and F-measure (87.26) scores, indicating its strong predictive capabilities. DE also shows notable performance, with mean accuracy (84.75) and F-measure (84.43) scores, suggesting its effectiveness in classification tasks.

On the other hand, GA-AMLNN, JADE-AMLNN, and SADE-AMLNN exhibit relatively weaker performance among the evaluated algorithms. GA-AMLNN presents the lowest mean accuracy (72.75) and F-measure (72.60) scores, indicating its limitations in accurately predicting the classes and capturing precision and recall. JADE-AMLNN also can not perform well compared to others, and it is the second-worst algorithm. SADE-AMLNN performs relatively better than GA-AMLNN and JADE-AMLNN but still falls behind the top-performing algorithms, with mean accuracy (80.25) and F-measure (80.38) scores.

To have a visual understanding, we show the generated neural network architectures found by using each algorithm in Table 4 in terms of the three most important parameters that are common in all solvers, which are the configuration of neurons in the hidden layers of the neural network, along with the specified

Table 2: Parameter settings for all algorithms.

Algorithms	Parameters	Value
Basic Algorithms		
GA-AMLNN	PC	0.95
	PM	0.05
	Selection	Tournament
DE-AMLNN	Weighting factor	0.8
	Crossover rate	0.9
	Strategy	DE/current-to-rand/1/bin
MA-AMLNN	Crossover probability	0.85
	Mutation probability	0.15
	Probability of local search	0.5
PSO-AMLNN	C_1	2.05
	C_2	2.05
	ω_{min}	0.4
	ω_{max}	0.9
CMA-ES-AMLNN	-	-
Advanced Algorithms		
HPSO-AMLNN	C_1	2.05
	C_2	2.05
	ω_{min}	0.4
	ω_{max}	0.9
	C_i	0.5
	C_f	0
CPSO-AMLNN	C_1	2.05
	C_2	2.05
	ω_{min}	0.4
	ω_{max}	0.9
CLPSO-AMLNN	C_1	2.05
	C_2	2.05
	ω_{min}	0.4
	ω_{max}	0.9
	$c - local$	1.2
SAPE-DE-AMLNN	Weighting factor	0.8
	Crossover rate	0.9
	Strategy	DE/current-to-rand/1/bin
JADE-AMLNN	Weighting factor	0.8
	Crossover rate	0.9
	Strategy	DE/current-to-rand/1/bin
	c_r	0.5
	p_t	0.1
	A_p	0.1
SHADE-AMLNN	Weighting factor	0.8
	Crossover rate	0.9
	Strategy	DE/current-to-rand/1/bin
	Memory size	50
LSHADE-AMLNN	Weighting factor	0.8
	Crossover rate	0.9
	Strategy	DE/current-to-rand/1/bin
	Memory size	50
PPSO-AMLNN	C_1	2.05
	C_2	2.05
	ω_{min}	0.4
	ω_{max}	0.9

Table 3: Experimental results for the first scenario on 10 independent iterations.

Algorithms	Accuracy		F-measure	
	Mean	Std.	Mean	Std.
GA-AMLNN	72.75	1.50	72.60	1.42
DE-AMLNN	84.75	2.08	84.43	2.15
MA-AMLNN	82.00	1.83	81.36	1.92
PSO-AMLNN	87.63	2.04	87.96	2.15
CMA-ES-AMLNN	81.63	1.59	81.35	1.63
HPSO-AMLNN	84.06	2.12	84.36	2.11
CPSO-AMLNN	85.69	2.41	85.65	2.30
CLPSO-AMLNN	84.56	2.16	84.65	2.11
SAPE-DE-AMLNN	87.38	1.80	87.26	1.42
JADE-AMLNN	76.94	1.88	76.95	1.47
SHADE-AMLNN	84.38	2.35	84.91	2.93
LSHADE-AMLNN	82.38	2.18	82.81	2.16
PPSO-AMLNN	84.81	1.24	84.99	1.30

learning rate and solver used during training.

From the table, the neural network architectures vary across the algorithms, demonstrating the diverse approaches employed by each algorithm to solve the problem. The number of neurons in the hidden layers varies significantly, ranging from 173 to 400. This disparity suggests that different algorithms adopt different levels of complexity in their neural network structures to address the task. Furthermore, each algorithm's learning rate, which determines the step size in weight adjustments during training, is also distinct. The specified learning rates range from 0.01 to 0.17, indicating varying sensitivity to weight updates during training. A lower learning rate implies more cautious adjustments, while a higher learning rate allows for more significant changes. In addition, the solver, Rprop, is found consistently across all algorithms in this study.

Comparing Table 3 (numerical results) and Table 4 (generated architectures), we can identify potential relationships between algorithm performance and neural network architectures. For instance, we can observe that PSO-AMLNN and SAPE-DE-AMLNN, which achieved high accuracy and F-measure scores in Table 3, have relatively complex structures with multiple hidden layers and a larger number of neurons. This complexity may enable them to capture complicated relationships within the data, resulting in improved performance. Conversely, GA-AMLNN, with a simpler architecture, has lower performance.

Additionally, the learning rate can also influence algorithm performance and convergence. Algorithms with lower learning rates, such as SAPE-DE-AMLNN and PSO-AMLNN, might exhibit a more cautious learning behaviour, gradually adjusting their weights to achieve optimal performance. On the other hand, algorithms with higher learning rates, like GA-AMLNN, may experience more abrupt weight updates, potentially affecting convergence and final performance.

Table 4: The architecture found by each algorithm in the first scenario.

Algorithm	Structure	Leaning rate	Solver
GA-AMLNN	[223]	0.17	Rprop
DE-AMLNN	[302,11]	0.05	Rprop
MA-AMLNN	[366,112]	0.04	Rprop
PSO-AMLNN	[173,262,294,12]	0.01	Rprop
CMA-ES-AMLNN	[343,18,367]	0.01	Rprop
HPSO-AMLNN	[217]	0.02	Rprop
CPSO-AMLNN	[202,226]	0.01	Rprop
CLPSO-AMLNN	[220,21,356]	0.01	Rprop
SAPE-DE-AMLNN	[278,39,186,295,324]	0.01	Rprop
JADE-AMLNN	[272]	0.1	Rprop
SHADE-AMLNN	[400]	0.05	Rprop
LSHADE-AMLNN	[294,392,221, 21,243,268]	0.02	Rprop
PPSO-AMLNN	[183,358,103,75,104,323]	0.01	Rprop

4.2 Second Scenario

This section provides our proposed algorithm's results when there are missing values in the data set. In the initial test, we introduced a scenario where 5% of the total elements are deliberately excluded. To illustrate, considering a data set with 8,000 samples and 23 available features, this implies that 920 elements out of a total of 18,400 elements are randomly omitted. For the experiments, we used the same parameters, available in Table 2.

Table 5 gives the results of our proposed algorithm on the data set with 5% missing values and 10 independent iterations. By analysing Table 5, we observe variations in the performance of algorithms when dealing with missing values. For instance, GA-AMLNN shows a decrease in both accuracy and F-measure, with mean scores of 64.75 and 64.81, respectively, compared to its performance in the first scenario without any missing values. This suggests that GA-AMLNN is more sensitive to the presence of missing values. In contrast, some algorithms demonstrate a more robust performance despite the missing values. CLPSO-AMLNN, SADE-AMLNN, SAPE-DE-AMLNN, and JADE-AMLNN show consistent mean accuracy and F-measure scores, indicating their ability to handle missing data and provide reliable predictions.

Comparing the results between the two tables highlights the importance of algorithm selection in the presence of missing values. Algorithms like CLPSO-AMLNN, SADE-AMLNN, SAPE-DE-AMLNN, and JADE-AMLNN maintain their effectiveness even when confronted with missing values. Conversely, GA-AMLNN and PSO-AMLNN demonstrate notable decreases in performance, indicating their vulnerability to the presence of missing data.

Table 6 presents the architectures found by each algorithm in the original data set with 5% missing values. Analysing Table 6 provides insights into how the presence of missing values affects the generated architectures. One interesting pattern is that the average number of neurons in the architectures obtained from Table 6 is slightly lower when compared to Table 4, indicating potential adaptations to account for the presence of missing data. For example, GA-AMLNN exhibits a structure of [128] in Table 6, whereas it had [223] neurons in Table 4. In addition, all algorithms, except HPSO-AMLNN and PPSO-AMLNN, suggest Rprop for their solvers, while these two proposes AdamW as their learning algorithms.

Table 5: Experimental results in the second scenario with 5% missing values on 10 independent iterations.

Algorithms	Accuracy		F-measure	
	Mean	Std.	Mean	Std.
GA-AMLNN	64.75	1.08	64.81	1.05
DE-AMLNN	78.25	2.44	78.27	2.41
MA-AMLNN	77.19	2.87	77.17	2.92
PSO-AMLNN	64.19	1.63	77.17	2.92
CMA-ES-AMLNN	83.25	2.31	83.30	2.30
HPSO-AMLNN	72.50	1.73	72.44	1.77
CPSO-AMLNN	72.75	1.13	72.79	1.10
CLPSO-AMLNN	85.06	2.03	85.05	2.03
SAPE-DE-AMLNN	83.75	2.85	83.74	2.87
JADE-AMLNN	85.25	2.45	85.24	2.45
SHADE-AMLNN	74.63	2.83	74.57	2.83
LSHADE-AMLNN	77.44	2.47	77.37	2.62
PPSO-AMLNN	73.13	1.19	73.08	1.22

Table 6: The architecture found by each algorithm in the second scenario with 5% missing values.

Algorithm	Structure	Leaning rate	Solver
GA-AMLNN	[128]	0.18	Rprop
DE-AMLNN	[287]	0.09	Rprop
MA-AMLNN	[187]	0.05	Rprop
PSO-AMLNN	[98]	0.14	Rprop
CMA-ES-AMLNN	[282,68]	0.01	Rprop
HPSO-AMLNN	[214]	002	AdamW
CPSO-AMLNN	[219]	0.13	Rprop
CLPSO-AMLNN	[320]	0.01	Rprop
SAPE-DE-AMLNN	[260]	0.01	Rprop
JADE-AMLNN	[356,84,133]	0.01	Rprop
SHADE-AMLNN	[282]	0.09	Rprop
LSHADE-AMLNN	[397]	0.1	Rprop
PPSO-AMLNN	[174]	0.02	AdamW

In the next experiment, we increased the missing values to 20%. Table 7 provides the results of experiments in the data set with 20% missing values. Analysing Table 7 allows for a discussion on the impact of increased missing value in the data set and a comparison with the previous two tables (Table 5: 5% missing values, and Table 3: original data set without missing values). With the inclusion of 20% missing values, we observe noticeable changes in algorithm performance. Some algorithms, such as GA-AMLNN, MA-AMLNN, PSO-AMLNN, SAPE-DE-AMLNN, and SHADE-AMLNN, experience decreased mean accuracy and F-measure scores compared to their performance in Table 5 (5% missing values). This suggests that these algorithms are more sensitive to increased missing values, leading to reduced prediction accuracy. In contrast, several algorithms, including DE-AMLNN, CMA-ES-AMLNN, CPSO-AMLNN, HPSO-AMLNN, SADE-AMLNN, JADE-AMLNN, and LSHADE-AMLNN, maintain relatively stable mean accuracy and F-measure scores in the presence of 20% missing values. This indicates their robustness to handle increased noise and their ability to generate reliable predictions. In addition, comparing Table 7 with Table 3 (original data set without missing values) reveals the overall impact of missing values on algorithm performance. Most algorithms in Table 7 exhibit lower mean accuracy and F-measure scores compared to their counterparts in Table 5. This demonstrates the challenges posed by missing values, as they can degrade the performance of the algorithms in predicting energy consumption accurately.

Table 7: Experimental Results in the second scenario with 20% missing values.

Algorithms	Accuracy		F-measure	
	Mean	Std.	Mean	Std.
GA-AMLNN	71.69	2.98	71.67	2.97
DE-AMLNN	78.94	2.77	78.86	2.87
MA-AMLNN	65.63	2.15	65.62	2.19
PSO-AMLNN	70.19	2.52	69.99	2.51
CMA-ES-AMLNN	84.50	2.38	84.54	2.34
HPSO-AMLNN	77.19	2.18	77.16	2.21
CPSO-AMLNN	79.13	2.28	79.13	2.33
CLPSO-AMLNN	77.75	2.93	77.73	2.92
SAPE-DE-AMLNN	68.88	2.12	68.83	2.20
JADE-AMLNN	80.88	2.52	80.77	2.65
SHADE-AMLNN	68.38	1.81	68.49	1.73
LSHADE-AMLNN	76.13	2.39	76.09	2.37
PPSO-AMLNN	79.94	2.71	79.83	2.91

Table 8 shows the architectures found by each algorithm in the second scenario with 20% missing values. The structures in Table 8 exhibit variations compared to Table 6, indicating algorithmic adaptations to handle the higher missing values. Certain algorithms, such as GA-AMLNN, DE-AMLNN, and MA-AMLNN, demonstrate changes in their neural network structures to adjust for the increased missing values. Such alterations also exit for the learning rate and solver, which reflects the algorithms' attempts to capture the underlying patterns in the presence of higher missing value levels.

Table 8: The architecture found by each algorithm in the second scenario with 2% missing values .

Algorithm	Structure	Leaning rate	Solver
GA-AMLNN	[110, 140]	0.07	Rprop
DE-AMLNN	[149]	0.01	Rprop
MA-AMLNN	[81]	0.11	Rprop
PSO-AMLNN	[187]	0.02	AdamW
CMA-ES-AMLNN	[342,21,137]	0.01	Rprop
HPSO-AMLNN	[350]	0.11	Rprop
CPSO-AMLNN	[126]	0.02	Rprop
CLPSO-AMLNN	[148,14]	0.07	Rprop
SAPE-DE-AMLNN	[55]	0.01	Rprop
JADE-AMLNN	[392,63,6,291,212,299,160]	0.05	Rprop
SHADE-AMLNN	[193]	0.20	Rprop
LSHADE-AMLNN	[199]	0.08	Rprop
PPSO-AMLNN	[313,89,201,344,376,365,344]	0.01	AdamW

In the next experiment, we conducted experiments under severe noise conditions, with 40% of the values being missed. The results in Table 9 indicate a considerable decrease in algorithm performance compared to the previous tables. The mean accuracy and F-measure scores for most algorithms, such as CPSO-AMLNN, SAPE-DE-AMLNN, and LSHADE-AMLNN, are noticeably lower, signifying the challenges introduced by severe noise in the data set. On the other hand, some algorithms, such as CMA-ES-AMLNN and CLPSO-AMLNN, demonstrate relatively stable mean accuracy and F-measure scores even in the presence of severe noise. This resilience indicates the algorithms' effectiveness in making reliable predictions.

Table 9: Experimental Results in the second scenario with 40% missing values.

Algorithms	Accuracy		F-measure	
	Mean	Std.	Mean	Std.
GA-AMLNN	78.38	2.92	78.38	2.92
DE-AMLNN	65.25	1.32	65.39	1.47
MA-AMLNN	74.81	2.19	74.76	2.17
PSO-AMLNN	71.50	2.92	71.39	2.01
CMA-ES-AMLNN	83.50	2.21	83.50	2.22
HPSO-AMLNN	62.94	2.38	62.90	2.43
CPSO-AMLNN	52.56	1.09	52.44	1.24
CLPSO-AMLNN	79.81	2.92	79.83	2.94
SAPE-DE-AMLNN	51.31	1.43	51.19	1.31
JADE-AMLNN	67.94	2.74	67.92	2.78
SHADE-AMLNN	62.69	2.00	62.76	2.08
LSHADE-AMLNN	48.75	1.25	48.70	1.20
PPSO-AMLNN	70.81	2.80	70.80	2.85

Table 10 indicates the architectural configurations obtained by each algorithm when applied to the original data set with a 40% rate of missing values. The quantitative analysis of the architectural configurations, learning rates, and solver choices provides insights into the strategies employed by the algorithms to handle severe noise. The observed variations reflect the algorithms' adaptive behaviours and attempts to optimise performance under challenging data conditions.

Table 10: The architecture found by each algorithm in the second scenario with 40% missing values.

Algorithm	Architecture	Leaning rate	Solver
GA-AMLNN	[318,55]	0.01	AdamW
DE-AMLNN	[124]	0.14	Rprop
MA-AMLNN	[94]	0.01	Rprop
PSO-AMLNN	[400,64,400,128,224,110]	0.01	AdamW
CMA-ES-AMLNN	[247]	0.12	Rprop
HPSO-AMLNN	[133]	0.14	Rprop
CPSO-AMLNN	[11]	0.09	Rprop
CLPSO-AMLNN	[224,287,307,391,124,294,292]	0.01	Rprop
SAPE-DE-AMLNN	]10]	0.11	Rprop
JADE-AMLNN	[173,250,95,185]	0.01	Rprop
SHADE-AMLNN	[105]	0.17	Rprop
LSHADE-AMLNN	[4]	0.05	Rprop
PPSO-AMLNN	[113]	0.07	Rprop

4.3 Overall Analysis

Statistical analysis is crucial in drawing meaningful conclusions from data and making informed decisions in PBMH algorithms. In this case, the alternative hypothesis, denoted as H_1, suggests the presence of a meaningful variation among the algorithms in question. On the other hand, the null hypothesis, referred to as H_0, states that there is no statistically significant difference among the algorithms. H_0 is the initial assumption that is tested, and only if the H_0 is shown to be false, the alternative hypothesis H_1 is accepted. Generally speaking, there are two types of statistical tests: multiple, for comparison of more than two samples, and pairwise tests, for comparison of two algorithms. To this end, the Wilcoxon signed-rank test, a pairwise test, and the Friedman test, a multiple test, are two important non-parametric statistical tests [3].

The Friedman test is a non-parametric alternative to the one-way ANOVA (Analysis of Variance) and is used to compare multiple related samples simultaneously. In the Friedman test, data is collected for multiple groups under different conditions. Each group is ranked independently across all states, and the ranks are used to calculate the test statistic. The test determines whether there are significant differences in the rankings among the groups. If the p-value obtained from the Friedman test is below a chosen significance level, α, it indicates significant differences among the groups. The results of the Friedman test, as shown in Table 11, provide valuable insights into the relative performance of various PBMH algorithms. The primary objective of this study was to compare and rank these algorithms based on their accuracy results. According to the ranking, the CMA-AMLNN, CLPSO-AMLNN, and JADE-AMLNN algorithms demonstrated the highest performance with a rank of 2. This indicates that these three algorithms outperformed all other algorithms, making them a promising choice for the hyper-parameter tuning of MLNN in the context of this study. Based on the chi-squared distribution table, with 0.05 degrees of freedom and a significance level of $\alpha = 0.05$, the critical value is 21.03. Since the calculated chi-squared value is more significant than this critical value, the alternative hypothesis is accepted. This indicates a statistically significant difference between the algorithms. Additionally, the p-value is extremely small, further supporting the rejection of the null hypothesis (H_0).

The Wilcoxon signed-rank test is another non-parametric test employed in this paper, used to determine if there is a significant difference between the two algorithms. The resulting p-value indicates whether the median difference is significantly different from zero, which allows us to infer whether there is a significant change between the two algorithms. The Wilcoxon signed-rank test is selected over the t-test because the former does not assume normal distributions, making it a safer choice. Moreover, the Wilcoxon test is less influenced by outliers than the t-test [3]. Table 12 shows the results of the Wilcoxon signed-rank test with a significance level of 5% based on accuracy. Based on the cumulative wins, we can see that CMA-AMLNN (7 wins, 4 ties, and 0 fail), JADE-AMLNN (8 wins, 4 ties, and 0 fail), and CLPSO-AMLNN (6 wins, 6 ties, and 0 fail) are the best-performing algorithms. On the other hand, GA-AMLNN (0 win, 1 ties, and 11 fails), SHADE-AMLNN (1 win, 6 ties, and 11 fails), and LSHADE-AMLNN (1 win, 0 ties, and 10 fails) are the worst-performing algorithms.

Despite providing valuable insights into the ranking and differences between the algorithms, both tests need more information regarding the algorithms' resistance to missing value. Consequently, a separate experiment was conducted to measure the algorithms' stability against the percentage of missing values, utilising the standard deviation criterion. In this context, a high standard deviation signifies that the algorithm is not resilient to missing value alterations, while a low standard deviation indicates increased resistance to missing value alterations. Table 13 shows the standard deviation, and it is clear that CMA-ES-AMLNN provides the lowest standard deviation, while JADE-AMLNN and CLPSO-AMLNN give higher standard deviations. These results are consistent with the previous tables. For instance, JADE-AMLNN's effectiveness is diminished when confronted with a substantial level of missing values (40% in this case). As such, when robustness against missing value alteration is a crucial consideration, it can be concluded that CMA-ES-AMLNN stands out as the most resilient PBMH algorithm.

Table 11: Results of Friedman test.

Algorithms	Average rank	Overall rank
GA-AMLNN	9.5	13
DE-AMLNN	5.75	4.5
MA-AMLNN	8.75	9.5
PSO-AMLNN	7.50	8
CMA-ES-AMLNN	4.25	2
HPSO-AMLNN	8.75	9.5
CPSO-AMLNN	7.00	6.5
CLPSO-AMLNN	4.25	2
SAPE-DE-AMLNN	7.00	6.5
JADE-AMLNN	4.25	2
SHADE-AMLNN	9.25	12
LSHADE-AMLNN	9.00	11
PPSO-AMLNN	5.75	4.5
Chi-square	61.48	
p-value	1.20869E-08	

5 Conclusion

Energy usage becomes a vital determinant in the consumer's decision-making process when contemplating a smartphone purchase. Moreover, the importance of sustainability necessitates exploring approaches to reduce mobile device energy consumption, given the substantial global consequences stemming from the widespread use of smartphones, which profoundly impact the environment. Despite the presence of energy-efficient programming practices within the dominant Android platform, the scarcity of documented machine learning-based energy prediction algorithms specifically tailored for mobile app development remains evident. To address this gap, our research introduces a novel neural network-based framework, augmented by a metaheuristic approach, to achieve robust energy prediction. Our proposed metaheuristic approach plays a pivotal role in identifying suitable learning algorithms and their optimal parameters and determining the most appropriate number of layers and neurons in each layer. Additionally, the limitations in accessing certain aspects of a mobile phone may result in missing data in the data set. To address this challenge, we employed an optimal algorithm selection strategy, incorporating 13 metaheuristic algorithms, to identify the best algorithm based on accuracy and resistance to missing values.

Notwithstanding the commendable performance demonstrated by the proposed approach, this work has opportunities for future extensions, which could be addressed by considering the following aspects:

- The proposed approach employs a basic approach for handling missing values, while in the future, the performance can be enhanced by using some sophisticated missing value handling approaches.
- Ensemble of solutions found by metaheuristic algorithms is another direction to enhance the results.
- The multi-objective variant of our proposed algorithm is also under investigation for future works.

6 Acknowledgement

This work was financed by FEDER (Fundo Europeu de Desenvolvimento Regional), from the European Union through CENTRO 2020 (Programa Operacional Regional do Centro), under project CENTRO-01-0247-FEDER-047256 – GreenStamp: Mobile Energy Efficiency Services.

This work is supported by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT.IP.

References

[1] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[2] Abdul Ali Bangash, Karim Ali, and Abram Hindle. A black box technique to reduce energy consumption of android apps. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 1–5, 2022.

[3] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[4] Timothy Dozat. Incorporating Nesterov Momentum into Adam. In *4th International Conference on Learning Representations*, pages 1–4, 2016.

[5] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1276–1284, 2013.

[6] Mojtaba Ghasemi, Ebrahim Akbari, Abolfazl Rahimnejad, Seyed Ehsan Razavi, Sahand Ghavidel, and Li Li. Phasor particle swarm optimization: a simple and efficient variant of PSO. *Soft Computing*, 23(19):9701–9718, 2019.

[7] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[8] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[9] J Harris. Our phones and gadgets are now endangering the planet. *The Guardian*, 17, 2018.

[10] Geoffrey Hecht, Naouel Moha, and Romain Rouvoy. An empirical study of the performance impacts of android code smells. In *Proceedings of the international conference on mobile software engineering and systems*, pages 59–69, 2016.

[11] Emanuele Iannone, Manuel De Stefano, Fabiano Pecorelli, and Andrea De Lucia. Predicting the energy consumption level of java classes in android apps: an exploratory analysis. In *Proceedings of the 9th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, pages 1–5, 2022.

[12] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E Hassan. What do mobile app users complain about? *IEEE software*, 32(3):70–77, 2014.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Jing J Liang, A Kai Qin, Ponnuthurai N Suganthan, and S Baskar. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation*, 10(3):281–295, 2006.

[15] Bo Liu, Ling Wang, Yi-Hui Jin, Fang Tang, and De-Xian Huang. Improved particle swarm optimization combined with chaos. *Chaos, Solitons & Fractals*, 25(5):1261–1271, 2005.

[16] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[18] Chu Luo, Aku Visuri, Simon Klakegg, Niels van Berkel, Zhanna Sarsenbayeva, Antti Möttönen, Jorge Goncalves, Theodoros Anagnostopoulos, Denzil Ferreira, Huber Flores, et al. Energy-efficient prediction of smartphone unlocking. *Personal and Ubiquitous Computing*, 23:159–177, 2019.

[19] Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, and James Clause. An empirical study of practitioners' perspectives on green software engineering. In *Proceedings of the 38th international conference on software engineering*, pages 237–248, 2016.

[20] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[21] Seyed Jalaleddin Mousavirad and Luís A Alexandre. Metaheuristic-based energy-aware image compression for mobile app development. *arXiv preprint arXiv:2212.06313*, 2022.

[22] Seyed Jalaleddin Mousavirad and Luís A Alexandre. Energy-aware JPEG image compression: A multi-objective approach. *Applied Soft Computing*, 141, 2023.

[23] Seyed Jalaleddin Mousavirad and Luís A Alexandre. A metaheuristic-based machine learning approach for energy prediction in mobile app development. *arXiv preprint arXiv:2306.09931*, 2023.

[24] Sona Mundody and K Sudarshan. Evaluating the impact of android best practices on energy consumption. In *IJCA Proceedings on International Conference on Information and Communication Technologies*, volume 8, pages 1–4, 2014.

[25] Moa Nyman. Estimating the energy consumption of a mobile music streaming application using proxy metrics. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2020.

[26] Earl Oliver and Srinivasan Keshav. Data driven smartphone energy level prediction. *University of Waterloo Technical Report*, 2010.

[27] Rui Pereira, Hugo Matalonga, Marco Couto, Fernando Castor, Bruno Cabral, Pedro Carvalho, Simão Melo de Sousa, and João Paulo Fernandes. Greenhub: a large-scale collaborative dataset to battery consumption analysis of android devices. *Empirical Software Engineering*, 26:1–55, 2021.

[28] Gustavo Pinto and Fernando Castor. Energy efficiency: a new concern for application software developers. *Communications of the ACM*, 60(12):68–75, 2017.

[29] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[30] Asanga Ratnaweera, Saman K Halgamuge, and Harry C Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on evolutionary computation*, 8(3):240–255, 2004.

[31] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE international conference on neural networks*, pages 586–591. IEEE, 1993.

[32] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *IEEE International Conference on Evolutionary Computation*, pages 69–73, 1998.

[33] Jagannath Singh and Arpan Maity. Energy consumption-based profiling of android apps. In *Mobile Application Development: Practice and Experience: 12th Industry Symposium in Conjunction with 18th ICDCIT 2022*, pages 21–32. Springer, 2023.

[34] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[35] Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In *IEEE Congress on Evolutionary Computation*, pages 71–78. IEEE, 2013.

[36] Ryoji Tanabe and Alex S Fukunaga. Improving the search performance of SHADE using linear population size reduction. In *IEEE Congress on Evolutionary Computation*, pages 1658–1665. IEEE, 2014.

[37] Jason Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing*, 10(8):673–686, 2006.

[38] Leonhard Wattenbach, Basel Aslan, Matteo Maria Fiore, Henley Ding, Roberto Verdecchia, and Ivano Malavolta. Do you have the energy for this meeting? an empirical study on the energy consumption of the google meet and zoom android apps. In *Proceedings of the 9th IEEE/ACM International Conference on Mobile Software Engineering and Systems*, pages 6–16, 2022.

[39] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.

[40] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[41] Jingqiao Zhang and Arthur C Sanderson. JADE: adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009.

Table 12: The results obtained from the Wilcoxon signed-rank test are based on the mean objective function value. Symbols +, −, and = are used to indicate whether an algorithm is statistically equivalent to, statistically superior to, or statistically inferior to another algorithm, respectively. Additionally, the last column summarises the cumulative wins (w), ties (t), and losses (l) of each algorithm.

Algorithms	GA-AMLNN	DE-AMLNN	MA-AMLNN	PSO-AMLNN	CMA-ES-AMLNN	HPSO-AMLNN	CPSO-AMLNN	CLPSO-AMLNN	SAPE-DE-AMLNN	JADE-AMLNN	SHADE-AMLNN	LSHADE-AMLNN	PPSO-AMLNN	w/t/l
GA-AMLNN		-	-	-	-	-	-	-	-	-	-	=	-	0/1/11
DE-AMLNN	+		+	+	-	+	+	-	+	-	+	+	=	8/1/3
MA-AMLNN	+	-		-	-	=	-	-	-	-	+	+	-	3/1/8
PSO-AMLNN	+	-	+		=	=	-	=	-	-	+	+	-	4/3/5
CMA-ES-AMLNN	+	+	+	=		=	+	=	=	=	+	+	+	7/5/0
HPSO-AMLNN	+	-	=	=	=		-	=	-	-	+	+	-	3/4/5
CPSO-AMLNN	+	-	+	+	-	+		=	=	=	+	+	-	6/3/3
CLPSO-AMLNN	+	+	+	=	=	=	=		=	=	+	+	+	6/6/0
SAPE-DE-AMLNN	+	-	+	+	=	+	=	=		=	+	+	-	6/4/2
JADE-AMLNN	+	+	+	+	=	+	=	=	=		+	+	+	8/4/0
SHADE-AMLNN	+	-	-	-	-	-	-	-	-	-		-	-	1/0/11
LSHADE-AMLNN	=	-	-	-	-	-	-	-	-	-	+		-	1/0/10
PPSO-AMLNN	+	=	+	+	-	+	+	-	+	-	+	+		8/1/3

Table 13: Stability results.

Algorithms	STD
GA-AMLNN	5.59
DE-AMLNN	8.26
MA-AMLNN	6.87
PSO-AMLNN	10.02
CMA-ES-AMLNN	1.19
HPSO-AMLNN	8.87
CPSO-AMLNN	14.32
CLPSO-AMLNN	3.59
SAPE-DE-AMLNN	16.43
JADE-AMLNN	3.61
SHADE-AMLNN	9.29
LSHADE-AMLNN	15.19
PPSO-AMLNN	6.40