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

Seyed Jalaleddin Mousavirad and Luís A. Alexandre

***Abstract*—Energy consumption is a fundamental concern in mobile application development, bearing substantial significance for both developers and end-users. 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 aims to achieve two goals: 1) identifying suitable learning algorithms and their corresponding hyperparameters, and 2) determining the optimal number of layers and neurons within each layer. 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 base and advanced metaheuristic algorithms, to identify the best algorithm based on accuracy and resistance to missing values. The representation in our proposed metaheuristic algorithm is variable-size, meaning that the length of the candidate solutions changes over time. We compared the algorithms based on the architecture found by each algorithm at different levels of missing values, accuracy, F-measure, and stability analysis. Additionally, we conducted a Wilcoxon signed-rank test for statistical comparison of the results. The extensive experiments show that our proposed approach significantly improves energy consumption prediction. Particularly, the JADE algorithm, a variant of differential evolution (DE), DE, and the covariance matrix adaptation evolution strategy deliver superior results under various conditions and across different missing value levels.**

***Index Terms*—Energy consumption prediction, evolutionary computation, hyperparameters, mobile application development, neural network, optimization.**

I. 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 [1]. Consequently, developers diligently acknowledge this energy issue and actively seek assistance to address it, albeit finding satisfactory guidance remains a rarity [2]. 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 [1]. 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 [3].

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]

Over the past few years, numerous research studies [4], [5], [6], [7], [8] 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, [9] 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 Spotify music and podcast App, to understand the correlation between the selected proxies and energy consumption. Reference [10] investigates how machine learning (ML) techniques perform in predicting the energy consumption of Java classes within Android applications. The study focuses solely on utilising structural properties derived from source code analysis. Their

Received 12 June 2024; revised 18 November 2024; accepted 4 February 2025. This work was supported in part by the Fundo Europeu de Desenvolvimento Regional (FEDER), from the European Union through CENTRO 2020 (Programa Operacional Regional do Centro) under Project CENTRO-01-0247-FEDER-047256 (GreenStamp: Mobile Energy Efficiency Services), and in part by the UID/04516/NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) with the financial support of FCT.IP. This article was recommended by Associate Editor E. Chen. *(Corresponding author: Seyed Jalaleddin Mousavirad.)*

Seyed Jalaleddin Mousavirad was with the Computer Engineering Department, Universidade da Beira Interior, 6201-001 Covilhã, Portugal. He is now with the Computer Engineering Department, Mid Sweden University, 85230 Sundsvall, Sweden (e-mail: jalalmoosavirad@gmail.com).

Luís A. Alexandre is with the NOVA LINCS, Universidade da Beira Interior, 6201-001 Covilhã, Portugal.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TSMC.2025.3543786, provided by the authors.

Digital Object Identifier 10.1109/TSMC.2025.3543786

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

initial findings indicate that learning models can not perform well in this application. Reference [11] suggested utilising byte-code transformations in conjunction with genetic search methods to decrease the energy usage of an application.

In another study [12], 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, [13] leveraged 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.

A recent study [14] identified image file size and quality as proxies for energy consumption, showing that higher values increase energy use. To mitigate this, a multiobjective optimization approach was proposed, balancing size and quality with minimal energy impact. Later refinements [15] incorporated user preferences, such as file size, by modifying the encoding representation. Another recent study introduces a machine-learning technique for predicting smartphone energy consumption [16]. Their approach utilises the histogram-based gradient boosting classification machine (HGBC) for the modeling 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 article 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 ML 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 generalization 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. A comprehensive survey on hyperparameter optimization is available in [17]. 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 article proposes a novel missing value-resilient metaheuristic-based ANN for finding the optimal hyperparameters.

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

1) We introduce a missing value-resilient ML-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.
2) We propose a general metaheuristic-based approach for hyperparameter tuning in ANNs. The utilization of metaheuristic techniques facilitates an efficient search for optimal hyperparameter configurations, contributing to the overall effectiveness of our energy prediction model.
3) 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. A pivotal goal of this article 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.
4) 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 article represents the first endeavour to investigate the effectiveness of these algorithms in addressing this specific problem.

The remainder of this article is organised as follows. Our proposed algorithm is described in Section III, while Section IV validates the proposed algorithm in the different conditions. Finally, we present the conclusions in Section V.

II. 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.

A. Base Algorithms

1) *Genetic Algorithm (GA) [18]:* 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 favoring the most successful ("fittest") solutions. There are several ways

for the selection process, while this article focuses on tournament selection due to some advantages like less bias toward the fittest.

2) *Differential Evolution (DE) [19]:* 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 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.
3) *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.
4) *Particle Swarm Optimization (PSO) [21]:* PSO is a PBMH inspired by swarm behavior, where the updating process relies on the best position (pbest) of each candidate solution and a global pbest. The velocity vector of a particle is updated as

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

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 pbest, and g_t represents the global pbest. Also, c_1 and c_2 are called cognitive coefficient and social coefficient, respectively, while ω signifies inertia weight.
5) *Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [22]:* CMA-ES algorithm is another PBMH employed in this article that belongs to the class of evolution strategies. It utilises a probabilistic model to adaptively update the search distribution and guide the search toward 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.

B. Advanced Algorithms

Advanced algorithms refer to enhanced versions of classical metaheuristic optimization methods that have been refined and extended to address specific limitations of their standard forms. These refinements often involve incorporating novel operators, adaptive mechanisms, or hybrid strategies to improve their performance.

1) *Self-Organising Hierarchical PSO With Jumping Time-varying Acceleration Coefficients (HPSO)* [23]: The key features of HPSO can be summarised as follows:
   a) Mutation is incorporated into the PSO algorithm.
   b) 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.
   c) The PSO algorithm includes a time-varying mutation step size.
2) *Chaos PSO (CPSO) [24]:* 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

$$\omega = \begin{cases} \omega_{\min} + \frac{(\omega_{\max} - \omega_{\min})(f - f_{\min})}{f_{\text{avg}} - f_{\min}}, & f \le f_{\text{avg}} \\ \omega_{\max}, & f \ge f_{\text{avg}} \end{cases} \tag{2}$$

where $\omega_{\max}$ and $\omega_{\min}$ 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 also employed as a local search mechanism around the pbest.
3) *Comprehensive Learning PSO (CLPSO) [25]:* To prevent premature convergence, a comprehensive learning (CL) strategy is proposed for particle learning. Instead of relying solely on their own pbests, all particles' pbest can be utilised to adjust the velocity of each particle. The updating scheme in CLPSO is defined as follows:

$$v^i_{t+1} = \omega v^i_t + c_1 r \left(\text{pbest}^f_{fi(d)} - x^i_t \right) \tag{3}$$

where $fi(d)$ determines which particles' *pbest* a specific particle i should follow. The decision to learn from nearby particles is based on the CL 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.
4) *DE With Self-Adaptation Populations (SAP-DE) [26]:* It introduces self-adaptive features for population size, crossover rates, and mutation rates. Two variants, $\text{SAP}-\text{DE}-\text{ABS}$ and $\text{SAP}-\text{DE}-\text{REL}$, are proposed to determine the population size parameter π.
In $\text{SAP}-\text{DE}-\text{ABS}$, 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, $\text{SAP}-\text{DE}-\text{REL}$ initialises the population size parameter by sampling a value from a uniform distribution ranging from -0.5 to $+0.5$.
5) *Adaptive DE With Optional External Archive (JADE) [27]:* 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.

6) *Success-History-Based Parameter Adaptation for DE (SHADE) [28]:* The main idea behind SHADE is to maintain a success-history archive that stores information about previous successful solutions. 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. 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.
7) *SHADE Using Linear Population Size Reduction (LSHADE) [29]:* LSHADE is an extension of the SHADE algorithm, which aims to further improve the performance of SHADE by introducing 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.
8) *Phasor PSO (PPSO) [30]:* 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

$$v_i^{\text{iter}} = |\cos\theta_i^{\text{iter}}|^{2\sin\theta_i^{\text{iter}}} \times \left(\text{Pbest}_i^{\text{iter}} - x_i^{\text{iter}}\right) + |\sin\theta_i^{\text{iter}}|^{2\cos\theta_i^{\text{iter}}} \times \left(\text{Gbest}_i^{\text{iter}} - x_i^{\text{iter}}\right) \quad (4)$$

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.

III. Proposed Algorithm

This research introduces a universal approach using metaheuristic techniques to discover optimal hyperparameters within MLNN for mobile app development. This methodology can be seamlessly combined with various optimization 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 optimization algorithms, we generate 13 new and distinct algorithms.

A. General Structure

In this work, we introduce a neuroevolution methodology aimed at predicting energy consumption within mobile application development (Fig. 1). Our ML-based energy prediction approach offers can be deployed within Android app development through various techniques as follows.

1) *Embedded-App Technique*: In this approach, the ML component is integrated into individual apps. However, this may lead to redundancy as multiple apps on the same device may have their own instance of the ML component.
2) *Embedded-Android Technique*: Alternatively, the ML component could be integrated into the Android operating system itself as a built-in feature. While this approach may offer optimal performance, its feasibility depends on the decision of major stakeholders, such as Google, which currently seems unlikely.
3) *Independent-App Technique*: In this method, the ML component is packaged as a standalone app. However, convincing users to install an additional app alongside their desired applications may pose a challenge.
4) *Web Service Technique*: Another option is to deploy the ML component as a Web service. In this scenario, when an application requires energy consumption prediction, it can send a request to the Web service and receive a recommendation. This approach remains effective until major Android stakeholders deem a built-in ML-based energy prediction component necessary.

The cornerstone of this methodology is its capability to dynamically tailor the architecture of MLNNs across a broad spectrum of optimization algorithms. This adaptability is facilitated through the employment of thirteen distinct search strategies, thereby generating a diverse suite of algorithms specifically designed for predicting energy consumption in mobile app development. The algorithms we developed for this prediction include 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, all of which are based on PBMH algorithms. For example, LSHADE-AMLNN utilises the LSHADE algorithm to optimise the selection of hyperparameters, as well as the number and configuration of neurons and layers in the neural networks.

A pivotal element of our proposed framework is the integration of a variable, denoted as i, which governs the number of hidden layers within the MLNNs. This variable not only specifies the initial configuration but also delineates the maximum allowable layers, directly influencing the dimensional complexity of each candidate solution. For illustrative purposes, an i value of 4 implies the construction of a candidate solution with four hidden layers.

At the inception of our methodology, a diverse array of MLNNs, each with varying hyperparameters and neuron counts, is generated. This initial population is derived using PBMH algorithms, signifying a wide exploratory scope from the outset. The search strategies are then tasked with refining these architectures, guided by a specifically defined objective function aimed at identifying the most effective network architecture for energy consumption prediction.

During the evolutionary phase, the search strategies increment the i variable following the evaluation of each layer's efficacy, thereby methodically enhancing the network's

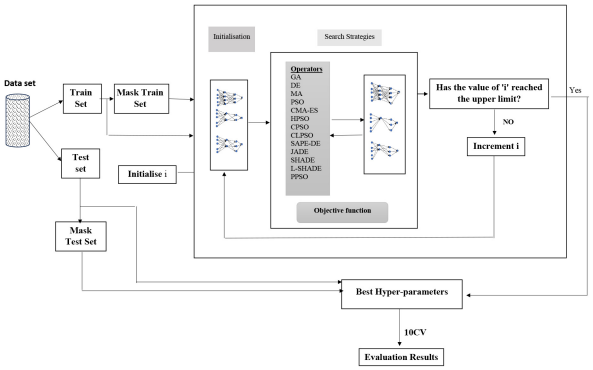

Fig. 1. General framework of our proposed algorithm.

complexity within predefined bounds. The evaluation of the evolved MLNN architectures is conducted through a rigorous 10-fold cross-validation process. This entails partitioning the original dataset into distinct training and testing subsets, with the training subset further subdivided to facilitate the validation process. This methodology not only underscores the robustness of our approach but also ensures its applicability and relevance to the dynamic field of mobile application development.

B. Preprocessing

This particular step in our study involves transforming and encoding the existing data set to ensure compatibility with ML 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

$$\text{ECPM} = \frac{BT_{\text{state}_1} - BT_{\text{state}_2}}{TS_{\text{state}_1} - TS_{\text{state}_2}} \times 60 \tag{5}$$

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 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

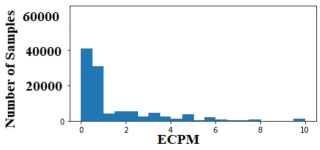

Fig. 2. Histogram of randomly-selected instances.

has been employed to determine the class assignments. Fig. 2 shows the histogram for ECPM for 120 000 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.

C. Representation of the Candidate Solutions

Our proposed algorithm aims to find all hyperparameters in a MLNN. To the best of our knowledge, in none of the previous research studies, this set of hyperparameters 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 hyperparameters (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 hyperparameters except the number of neurons, which is a string with a length of 9. The hyperparameters assigned to each decision variable, their types and values are given in Table I. We can see that all hyperparameters in the first segment are real values except the solver. A solver, here, refers to an optimization 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) [31], Adam with adaptive learning rate method (Adadelta) [32], Adam with decoupled weight decay regularization (AdamW) [33], infinity norm-based ADAM (Adamax) [31], accelerated stochastic gradient descent (ASGD) [34], Nesterov momentum into Adam (NAdam) [35], rectified Adam (RAdam) [36], root mean square propagation (RMSprop) [37], resilient backpropagation algorithm (Rprop) [38] and stochastic gradient descent method (SGD) [39], 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

TABLE I
HYPERPARAMETERS OF THE FIRST SEGMENT OF THE PROPOSED REPRESENTATION

Hyperparameter	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}

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. 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.

Therefore, we are tasked with determining appropriate values for 9 hyperparameters and the number of neurons in the hidden layers, encompassing a broad spectrum of possibilities within the search space. For example, consider the real-valued hyperparameters, which are discretised at intervals of 0.01. Additionally, assume that we have three layers and the maximum number of neurons in each layer is 50. This discretisation implies that, for the learning rate alone, there exist 100 potential values, though, in practice, the actual number of feasible values is greater. Consequently, the total number of potential combinations for the search space in this specific example is calculated as $100^4 \times 20^3 \times 10 \times 50^3$. This exponential increase in possibilities underscores the complexity of identifying the optimal set, rendering the optimization process notably challenging.

D. 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. However, while there are several approaches for handling missing values, this article does not focus on identifying the best one. Instead, our proposed method integrates a concealing (masking) strategy into the MLNN to handle 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 kth ($k = 1, 2, \ldots, s$) neuron in the first hidden layer of an MLNN is defined as

$$a_k^{(1)} = f\left(\sum_{i=1}^{q} w_{ik}^{(1)} x_i^o + \sum_{j=1}^{r} w_{jk}^{(1)} x_j^m + b_k^{(1)}\right) \tag{6}$$

where$a_k^{(1)}$ denotes the activation of the kth 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 ith observed feature, and $w_{ik}^{(1)}$ is the weight associated with the connection between the ith observed feature and the kth 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 jth missing feature, and $w_{jk}^{(1)}$ is the weight associated with the connection between the jth missing feature and the kth neuron in the first hidden layer. Also, $b_k^{(1)}$ represents the bias term for the kth 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. For each sample i and feature j, the mask value M_{ij} is set as follows:

$$M_{ij} = \begin{cases} 1, \text{ if feature } j \text{ in sample } i \text{ is observed} \\ 0, \text{ if feature } j \text{ in sample } i \text{ is missing.} \end{cases} \tag{7}$$

Hence, for each sample i, $M_i = (M_{i1}, M_{i2}, \ldots, M_{ip})$ represents the binary mask vector indicating missing (0) and observed (1) features.

To incorporate the concealing strategy into the MLNN, we compute the masked input x_i^m for each sample i by performing an element-wise multiplication between the original input vector x_i and its corresponding binary mask vector M_i, yielding

$$x_i^m = x_i \odot M_i \tag{8}$$

where $\odot$ denotes the element-wise multiplication operation, and $x_i^m \in \mathbb{R}^p$ represents the masked input vector with missing values concealed. 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.

E. 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 here is based on the classification accuracy, which is calculated based on the neural network's predictions, considering the concealing strategy for handling missing values. The classification error is computed as

$$\text{Classification error} = \frac{100}{P} \times \sum_{p=1}^{P} \xi\left(x_p\right) \tag{9}$$

with

$$\xi\left(x_p\right) = \begin{cases} 1, \text{ if } o_p \neq d_p \\ 0, \text{ otherwise} \end{cases} \tag{10}$$

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 by the neural network. To ensure reliable and robust results, the objective function is evaluated using k-fold cross-validation.

IV. 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 IV-A. The experiments were conducted on a computer running Windows 10, equipped with an AMD Quad Core FX-8800P processor and 16 GB of RAM. The implementation of the proposed methodology was carried out using the Mealpy framework for metaheuristic optimization and the PyTorch library for neural network tasks. 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.

A. Data Set

In this study, we utilised the GreenHub data set [40] 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 U.S. 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 [40] for further specifics on these features.

B. 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 optimization process for each layer is selected as 30, and the maximum number of layers is 8. The remaining parameters, as listed in Table I in the *Supplementary File*, are set to their default values.

Table II 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.

On the other hand, GA-AMLNN, JADE-AMLNN, and SADE-AMLNN exhibit relatively weaker performance among

TABLE II
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

the evaluated algorithms. GA-AMLNN presents the lowest mean accuracy (72.75) and F-measure (72.60) scores. 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 III 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 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 II (numerical results) and Table III (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 II, 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.

TABLE III
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

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 behavior, 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.

C. 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 with the first scenario.

Table IV gives the results of our proposed algorithm on the data set with 5% missing values and 10 independent iterations. By analysing Table IV, 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 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 V presents the architectures found by each algorithm in the original data set with 5% missing values. Analysing

TABLE IV
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 V
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

TABLE VI
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 VII
ARCHITECTURE FOUND BY EACH ALGORITHM IN THE SECOND SCENARIO WITH 20% 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

Table V 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 V is slightly lower when compared to Table III, indicating potential adaptations to account for the presence of missing data. For example, GA-AMLNN exhibits a structure of [128] in Table V, whereas it had [223] neurons in Table III. In addition, all algorithms, except HPSO-AMLNN and PPSO-AMLNN, suggest Rprop for their solvers, while these two proposes AdamW as their learning algorithms.

In the next experiment, we increased the missing values to 20%. Table VI provides the results of experiments in the data set with 20% missing values. Analysing Table VI allows for a discussion on the impact of increased missing value in the data set and a comparison with the previous two tables (Table IV: 5% missing values, and Table II: 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 IV (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 VI with Table II (original data set without missing values) reveals the overall impact of missing values on algorithm performance. Most algorithms in Table VI exhibit lower mean accuracy and F-measure scores compared to their counterparts in Table IV.

Table VII shows the architectures found by each algorithm in the second scenario with 20% missing values. The structures in Table VII exhibit variations compared to Table V, 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.

In the next experiment, we conducted experiments under severe noise conditions, with 40% of the values being missed. The results in Table VIII indicate a considerable decrease in

TABLE VIII
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 IX
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

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 IX 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 behaviors and attempts to optimise performance under challenging data conditions.

D. Overall Analysis

Statistical analysis is crucial in drawing meaningful conclusions from data and making informed decisions in PBMH algorithms. The Friedman test is a nonparametric alternative

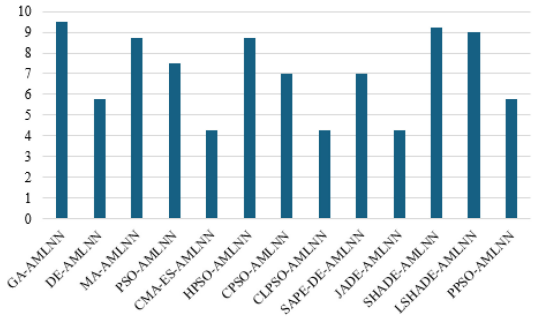

Fig. 3. Results of Friedman test.

to the one-way analysis of variance (ANOVA) and is used to compare multiple related samples simultaneously. 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 Fig. 3, 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 the same rank. 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 (1.2086E-08), further supporting the rejection of the null hypothesis (H_0).

The Wilcoxon signed-rank test is another nonparametric test employed in this article, used to determine if there is a significant difference 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 [41]. Table X 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

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

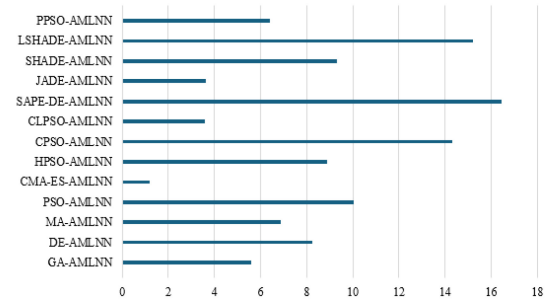

Fig. 4. Stability results.

standard deviation indicates increased resistance to missing value alterations. Fig. 4 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.

V. CONCLUSION

The decision-making process for smartphone purchases increasingly revolves around energy consumption considerations, driven by consumer awareness of sustainability issues. Despite the prevalence of energy-efficient practices in Android development, the lack of documented ML-based energy prediction algorithms tailored for mobile apps remains. To bridge this gap, our study introduces a novel neural network framework enhanced by metaheuristic techniques for robust energy prediction. Our metaheuristic approach proposed a novel approach to optimise learning algorithms and hyperparameters, as well as the architecture of neural networks. Furthermore, challenges related to incomplete data due to restricted access to certain device features are addressed through an algorithm selection strategy employing 13 metaheuristic algorithms. This strategy prioritises accuracy and resilience to missing values, ensuring the effectiveness of our predictive model.

Despite the strong performance of the proposed approach, there are opportunities for future enhancements: (1) employing more advanced techniques for handling missing values, and (2) incorporating continual learning to adapt to updates and variations in new operating systems.

REFERENCES

[1] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Min.*, 2013, pp. 1276–1284.

[2] I. Manotas et al., "An empirical study of practitioners' perspectives on green software engineering," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 237–248.

[3] G. Pinto and F. Castor, "Energy efficiency: A new concern for application software developers," *Commun. ACM*, vol. 60, no. 12, pp. 68–75, 2017.

[4] J. Singh and A. Maity, "Energy consumption-based profiling of android apps," in *Proc. Mobile Appl. Develop., Pract. Exp., 12th Ind. Symp. Conjunct. 18th (ICDCIT)*, 2023, pp. 21–32.

[5] G. Hecht, N. Moha, and R. Rouvoy, "An empirical study of the performance impacts of android code smells," in *Proc. Int. Conf. Mobile Softw. Eng. Syst.*, 2016, pp. 59–69.
[6] S. Mundody and K. Sudarshan, "Evaluating the impact of android best practices on energy consumption," in *Proc. IJCA Int. Conf. Inf. Commun. Technol.*, 2014, pp. 1–4.
[7] L. Wattenbach, B. Aslan, M. M. Fiore, H. Ding, R. Verdecchia, and I. 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 *Proc. 9th IEEE/ACM Int. Conf. Mobile Softw. Eng. Syst.*, 2022, pp. 6–16.
[8] Z. Dai, W. Wang, and Y. Wu, "Static energy consumption analysis for android applications," in *IOP Conf. Ser., Earth Environ. Sci.*, vol. 512, no. 1, 2020, Art. no. 12011.
[9] M. 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.
[10] E. Iannone, M. De Stefano, F. Pecorelli, and A. De Lucia, "Predicting the energy consumption level of java classes in android apps: An exploratory analysis," in *Proc. 9th IEEE/ACM Int. Conf. Mobile Softw. Eng. Syst.*, 2022, pp. 1–5.
[11] A. A. Bangash, K. Ali, and A. Hindle, "Black box technique to reduce energy consumption of android apps," in *Proc. ACM/IEEE 44th Int. Conf. Softw. Eng., New Ideas Emerg. Results*, 2022, pp. 1–5.
[12] E. Oliver and S. Keshav, "Data driven smartphone energy level prediction," Dept. Comput. Sci., Univ. Waterloo, Waterloo ON, Canada, Rep. CS-2010-06, 2010.
[13] C. Luo et al., "Energy-efficient prediction of smartphone unlocking," *Pers. Ubiquitous Comput.*, vol. 23, pp. 159–177, Feb. 2019.
[14] S. J. Mousavirad and L. A. Alexandre, "Energy-aware JPEG image compression: A multi-objective approach," *Appl. Soft Comput.*, vol. 141, Jul. 2023, Art. no. 110278.
[15] S. J. Mousavirad and L. A. Alexandre, "Metaheuristic-based energy-aware image compression for mobile app development," *Multimedia Tools Appl.*, 2024, to be published.
[16] S. J. Mousavirad and L. A. Alexandre, "A Metaheuristic-based machine learning approach for energy prediction in mobile app development," 2023, *arXiv:2306.09931*.
[17] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, Nov. 2020.
[18] D. Whitley, "A genetic algorithm tutorial," *Statist. Comput.*, vol. 4, no. 2, pp. 65–85, 1994.
[19] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.
[20] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," Dept. Comput. Sci., California Inst. Technol., Pasadena, CA, USA, Rep. C3P 826, 1989.
[21] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1998, pp. 69–73.
[22] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001.
[23] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 240–255, Jun. 2004.
[24] B. Liu, L. Wang, Y.-H. Jin, F. Tang, and D.-X. Huang, "Improved particle swarm optimization combined with chaos," *Chaos, Solitons Fractals*, vol. 25, no. 5, pp. 1261–1271, 2005.
[25] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281–295, Jun. 2006.
[26] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Comput.*, vol. 10, no. 8, pp. 673–686, 2006.
[27] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, Oct. 2009.
[28] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, 2013, pp. 71–78.
[29] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *Proc. IEEE Congr. Evol. Comput.*, 2014, pp. 1658–1665.
[30] M. Ghasemi, E. Akbari, A. Rahimnejad, S. E. Razavi, S. Ghavidel, and L. Li, "Phasor particle swarm optimization: A simple and efficient variant of PSO," *Soft Comput.*, vol. 23, no. 19, pp. 9701–9718, 2019.
[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[32] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*.
[33] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017, *arXiv:1711.05101*.
[34] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM J. Control Optim.*, vol. 30, no. 4, pp. 838–855, 1992.
[35] T. Dozat, "Incorporating Nesterov momentum into Adam," in *Proc. 4th Int. Conf. Learn. Represent.*, 2016, pp. 1–4.
[36] L. Liu et al., "On the variance of the adaptive learning rate and beyond," 2019, *arXiv:1908.03265*.
[37] A. Graves, "Generating sequences with recurrent neural networks," 2013, *arXiv:1308.0850*.
[38] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. IEEE Int. Conf. Neural Netw.*, 1993, pp. 586–591.
[39] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, nos. 4–5, pp. 185–196, 1993.
[40] R. Pereira et al., "GreenHub: A large-scale collaborative dataset to battery consumption analysis of android devices," *Empir. Softw. Eng.*, vol. 26, pp. 1–55, Mar. 2021.
[41] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 3–18, 2011.

Seyed Jalaleddin Mousavirad received the Ph.D. degree in computer engineering, specialising in artificial intelligence from the University of Kashan, Kashan, Iran, in 2018.

He is currently a Postdoctoral Researcher with Mid Sweden University, Sundsvall, Sweden. Previously, he served as a Postdoctoral Research Fellow with the University of Beira Interior, Portugal, where he was actively involved in the European project called GreenStamp. He served as an Assistant Professor with the Faculty of Engineering, Hakim Sabzevari University, Sabzevar, Iran. His international research experiences also include visiting a World-Class Research Group with Xi'an Jiaotong-Liverpool University, Suzhou, China. He has published six book chapters and over 100 papers in reputable academic journals and conferences.

Luís A. Alexandre received the B.Sc. degree in physics and applied mathematics, and the M.Sc. and Ph.D. degrees in electrical engineering and computers from the University of Porto, Porto, Portugal, in 1994, 1997, 2002, respectively, and the Habilitation degree in computer science and engineering from the Universidade da Beira Interior (UBI), Covilhã, Portugal, in 2013.

He is currently a Full Professor with UBI. His research interests are neural networks, computer vision and their applications, particularly to robotics.

Prof. Alexandre has been an Evaluator for the A3ES (Portuguese Agency for the Quality Assurance in Higher Education), and a member of the Portuguese National Science Foundation Team for Ph.D. Grant Proposal Assessment. He was also a member of the governing board of both the International Association for *Pattern Recognition* and of the European Neural Network Society, and the President of the Portuguese Association for *Pattern Recognition*.