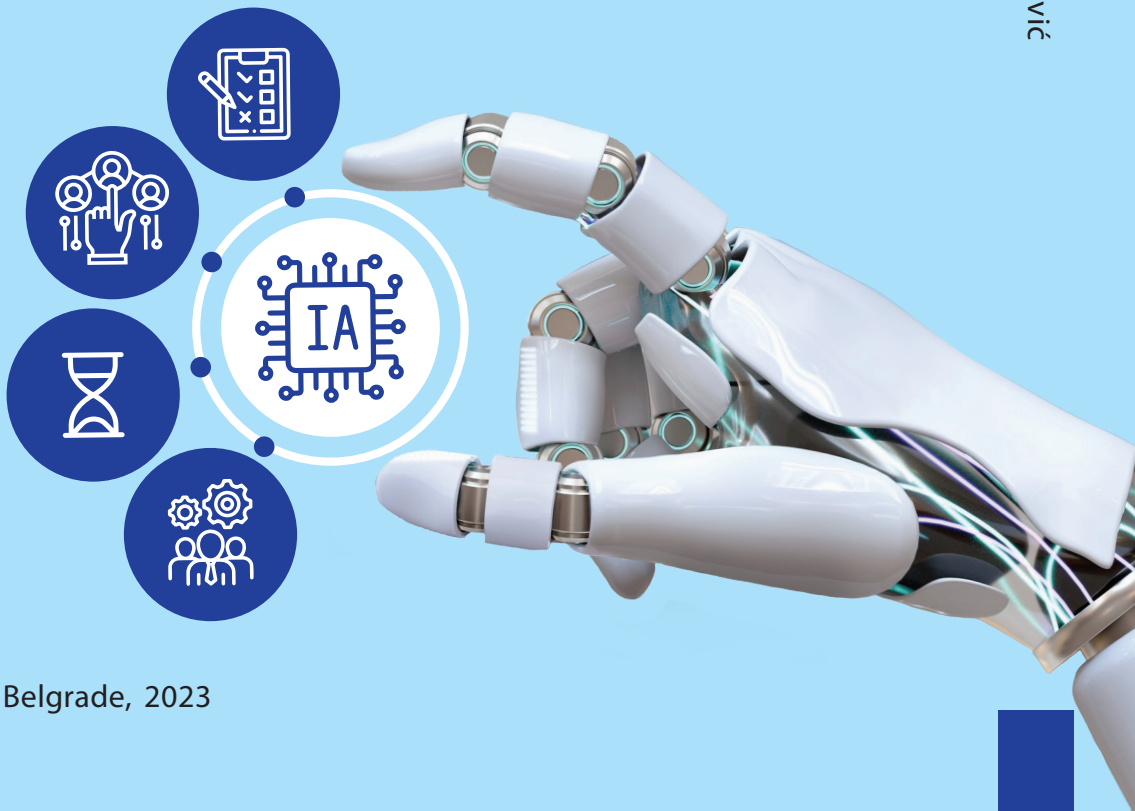


Artificial Intelligence Techniques in Software Engineering



Singidunum University

Nevena Ranković

Dragica Ranković

Jelena Kaljević

ARTIFICIAL INTELLIGENCE TECHNIQUES IN SOFTWARE ENGINEERING

1st edition

Belgrade, 2023

ARTIFICIAL INTELLIGENCE TECHNIQUES IN SOFTWARE ENGINEERING

Authors:

Nevena Ranković
Dragica Ranković
Jelena Kaljević

Reviewers:

Boško Nikolić
Laura Alethia de la Fuente
Nebojša Bačanin Džakula

Publisher:

SINGIDUNUM UNIVERSITY
32 Danijelova Street, Bgrade
www.singidunum.ac.rs

For the publisher:

Milovan Stanišić

Prepress:

Miloš Višnjic

Design:

Aleksandar Mihajlović

Year:

2023

Circulation:

300

Printed by:

Publikum, Belgrade

ISBN: 978-86-7912-809-6

Copyright © 2023 Singidunum University, Belgrade

All rights reserved. No part of this work covered by the copyright herein may be reproduced, transmitted, stored or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, without the prior written permission of the publisher.

"If the software project doesn't have a risk – don't do it!"

... And what if we can estimate risk and which factors influence the most,
and even successful completion of the software project using
Artificial Intelligence tools?

Authors

Preface

The swift advancement of computer technology leads to an even faster development and application of artificial intelligence, its various techniques and methods through the construction of various models for different purpose.

Today, software is applied in basically all spheres of life; people use it and do not think about its life cycle. It starts from the simplest mobile applications to complex software used to perform tasks that completely replace humans.

This book will show some of the fundamental techniques and methods of artificial intelligence used in everyday life and business. They mainly differ concerning the aspect of the application, but the most significant number of models combine various techniques and methods depending on the problem to be solved.

In addition to explaining and applying the essential aspects of division and their basic techniques, examples of different combined approaches will be given.

The central part of the book will be devoted to applying artificial intelligence tools and techniques in software engineering, more precisely, in software estimation.

In the context of estimating effort and costs associated with software project development and implementation, this book will introduce various approaches and methodologies, including artificial neural networks, Taguchi's method, software quality metrics, and other relevant criteria.

Furthermore, it can be concluded that three new, improved, specially constructed models can give much better results in estimation and thus significantly improve this area of software engineering.

Finally, this book is intended for students who want to expand their knowledge and skills in the field of artificial intelligence application in software engineering and software project management, as well as for all other experts in these areas.

We hope you will enjoy reading it as much as we did while writing it.

Contents

Preface	V
Introduction	IX

CHAPTER 1

THE MOST POPULAR ARTIFICIAL INTELLIGENCE TECHNIQUES

1. The Most Popular Artificial Intelligence Techniques	5
1.1. Machine Learning	6
1.2. Natural Language Processing (NLP)	8
1.2.1. Techniques and Methods of Natural Language Processing	10
1.3. Automation and Robotics (AR)	12
1.4. Machine Vision (MV)	17

CHAPTER 2

THE ROLE OF ARTIFICIAL INTELLIGENCE IN SOFTWARE DEVELOPMENT

2. The Role of Artificial Intelligence in Software Development	23
2.1. The Impact of AI in Different Phases of Software Development	24

CHAPTER 3

POWER OF ANNS FOR COST AND EFFORT ESTIMATION DURING SOFTWARE DEVELOPMENT

3. Power of Anns for Cost and Effort Estimation During Software Development	33
3.1. Utilizing Ann in Software Evaluation	34
3.2. Challenges in Software Evaluation	37
3.3. Hybrid Models Construction for Software Assessment	38
3.4. Novel Hybrid Models	40
3.5. Choosing Three Distinct Approaches	42

CHAPTER 4

TYPICAL METHODS AND METRICS USED IN SOFTWARE ENGINEERING

4. Typical Methods and Metrics Used in Software Engineering	47
4.1. Traditional Methods	47
4.2. Traditional Assessment	48
4.3. Cocomo2000 Software Estimation Through Source Code Analysis	50
4.3.1. Cocomo2000 Hybrid Model Main Characteristics	50
4.3.2. Prior Research Related to Cocomo2000 Methodology	54
4.4. Software Estimation Through Function Points	57
4.4.1. Cosmic FFP Hybrid Model Main Characteristics	57
4.4.2. Prior Research Related to Function Points Methodology	60
4.5. Software Estimation Through Use Case Points	61
4.5.1. Main Characteristics of the UCP Model	61
4.5.2. Prior Research Related to UCP Methodology	67
4.6. Advantages of Using Anns in Software Estimation	69
4.6.1. Anns Most Common Classification	70
4.6.2. Anns in Other Fields	70
4.7. Taguchi's Method vs. Full Factorial Design	71

CHAPTER 5

NOVEL, HYBRID MODELS FOR DISTINCT APPROACHES

5. Novel, Hybrid Models for Distinct Approaches	77
5.1. Novel, Hybrid Cocomo2000 Model	78
5.1.1. Datasets Utilized in Cocomo2000 Hybrid Model	86
5.1.2. Cocomo2000 Hybrid Model Methodology	87
5.2. Novel, Hybrid Cosmic FFP Model	101
5.2.1. Datasets Utilized in Cosmic FFP Hybrid Model	105
5.2.2. Cosmic FFP Hybrid Model Methodology	106
5.3. Novel, Hybrid UCP Model	111
5.3.1. Datasets Utilized in UCP Hybrid Model	115
5.3.2. UCP Hybrid Model Methodology	115

CHAPTER 6

RESULTS OF THREE NOVEL, HYBRID MODELS

6. Results of Three Novel, Hybrid Models	123
6.1. Cocomo2000 Hybrid Model and Ann	123
6.2. Cosmic FFP Hybrid Model and Ann	150
6.3. UCP Hybrid Model and Ann	172
6.4. Comparing Three Novel Hybrid Models	181

CHAPTER 7

GENERALIZATION PROBLEM IN AI

7. Generalization Problem in AI	187
7.1. What is The Required Number of Project in Dataset?	187
7.2. Hybrid Models vs. Svm Algorithm	190
7.3. Architectural Intricacy of Anns	197
7.4. Link Between Convergence Rate and Number of Iterations in Anns	198
7.5. Activation Function Choice and Encoding/Decoding Method	199
7.6. Potential Models Limitations	199

CHAPTER 8

MODELS CONTRIBUTION: SCIENCE vs. INDUSTRY

8. Models Contribution: Science vs. Industry	205
--	-----

CHAPTER 9

CONCLUDING REMARK

9. Concluding Remark	211
9.1. Who Will Benefit the Most From These Models?	213

Bibliography	216
---------------------	------------

Author Biographies	236
---------------------------	------------

Introduction

Why do we desperately need AI today? Or Why is AI such a big deal? Or Why is AI gaining so much attention?

These questions are the once frequently asked when the researches want to introduce Artificial Intelligence (AI) into their field of expertise. When we look back only a few years, we see that most of the daily jobs we do have become digitized. From daily shopping for groceries in supermarkets, scheduling doctor's appointments, to evaluating the success of billion-dollar software projects. In this age of Big data, humans cannot absorb, interpret, and make complex decisions based on the enormous amount of data generated. What to do in those situations when we are required to make significant decisions based on a large amount of information? Or how can we teach the machine to decide for us? Can we make it easier for ourselves and avoid the bias factor that humans have and machines don't? Is that even possible?

Artificial Intelligence is the foundation and the future for all computer learning. That's why researchers in different domains all over the world have been dealing with this problem for many years. It is now proven that AI can see the patterns in historical data and unlock tons of opportunities that human brain can overlook, with little to no errors and no coffee breaks. This is possible for machines, because they can make unbiased, unemotional decisions based only on data and statistics. To explain and understand how is that done, let's start with simple definitions.

What is Artificial Intelligence?

Different explanations and definitions can be used, but to opt for one, we will define it in this manner: *"The ability of the system to notice connections and similarities, to infer, to calculate, learn from experience based on reasoning, find the best solutions for given problems, comprehend complex ideas, use the natural language proficiently, classify, generalize, adapt on new scenario/changed circumstances"*. So, when it comes to solving the complex problems even the brightest data scientist cannot compete with the AI power to absorb huge volumes of data, analyze, and distill them in small bites of information.

Again, do we really need AI?

When we talk about software industry, no human being can be as data-driven as AI. Even if the company is using just a small add-in solution in their daily activities. AI can complement employees job tasks and be a real “game-changer” in today’s business world. This is possible, because machines don’t have natural limitations like human brains [1], [2], [3], [4].

Let’s observe a simple task that each employee is facing daily in almost every type of business, e.g. Enterprise Architecture: document classification. If a person is doing this manually, it is something that is prone to errors [5], [6], [7]. Therefore, a lot of companies want to avoid this. Furthermore they want to have digital transformation and as much as possible automation [8]. They need an AI technique to make this achievable. This leads us to our next question:

What is AI technique?

As for the example above, in the real world, the knowledge has some unwelcomed “belongings” [9], [10]:

- Its volume is huge, sometimes even unimaginable;
- It is not well-structured or well-formatted;
- It keeps changing regularly.

The AI Technique is a way to efficiently organize and use knowledge in such a way that [11], [12]:

- The people who provide it ought to be able to perceive it;
- Errors should be easily corrected;
- Even though it is insufficient or inaccurate, it should be useful in many situations.

On the example of business company, it can be concluded that the most time- consuming and the also the most significant step would be collecting, consolidating, cleaning, and organizing data. This is important, because proper data preparation produces accurate training set and results in more prices outcomes. After that, we need to feed our AI technique, e.g. Machine Learning algorithm, that performs mathematical computations. He needs to find an optimal function that will be capable of predicting the correct category of documents, it has never seen before [13], [14], [15].

This leads to the statement: in order to consider AI technologies, it is crucial to understand upfront the benefit between time invested and the business value gained [16]. Furthermore, an AI technique is a form of method applied to the existing reservoir of information, effectively arranging and utilizing it to the best extent [17], [18]:

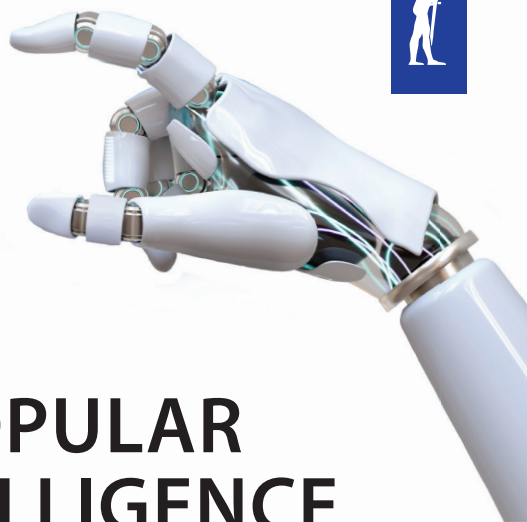
- Easy modifiable to correct errors;
- Useful in many ways even it is incomplete or to some degree inaccurate;
- Understood and clear by its provide;
- Having a clear purpose.

Before explaining the most popular AI techniques, it is worth to repeat the philosophy of AI: *"Creating almost identical intelligence in machines that we find and regard high in humans, to help us solve complex problems avoiding errors"* [19], [20]. With this, defining two main goals:

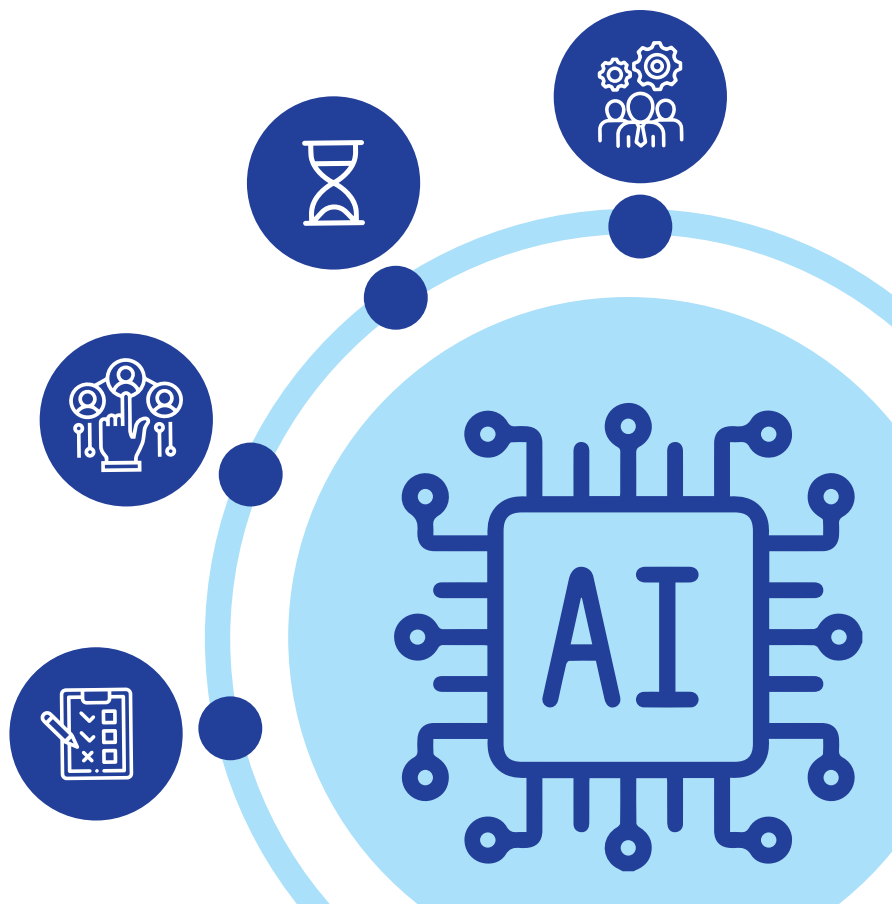
- To develop expert systems - computer programs that can learn, do demonstrations, provide explanations, and give people different types of suggestions;
- To give machines intelligent behavior by building programs that comprehend, reason, acquire knowledge, and act in human-like ways.

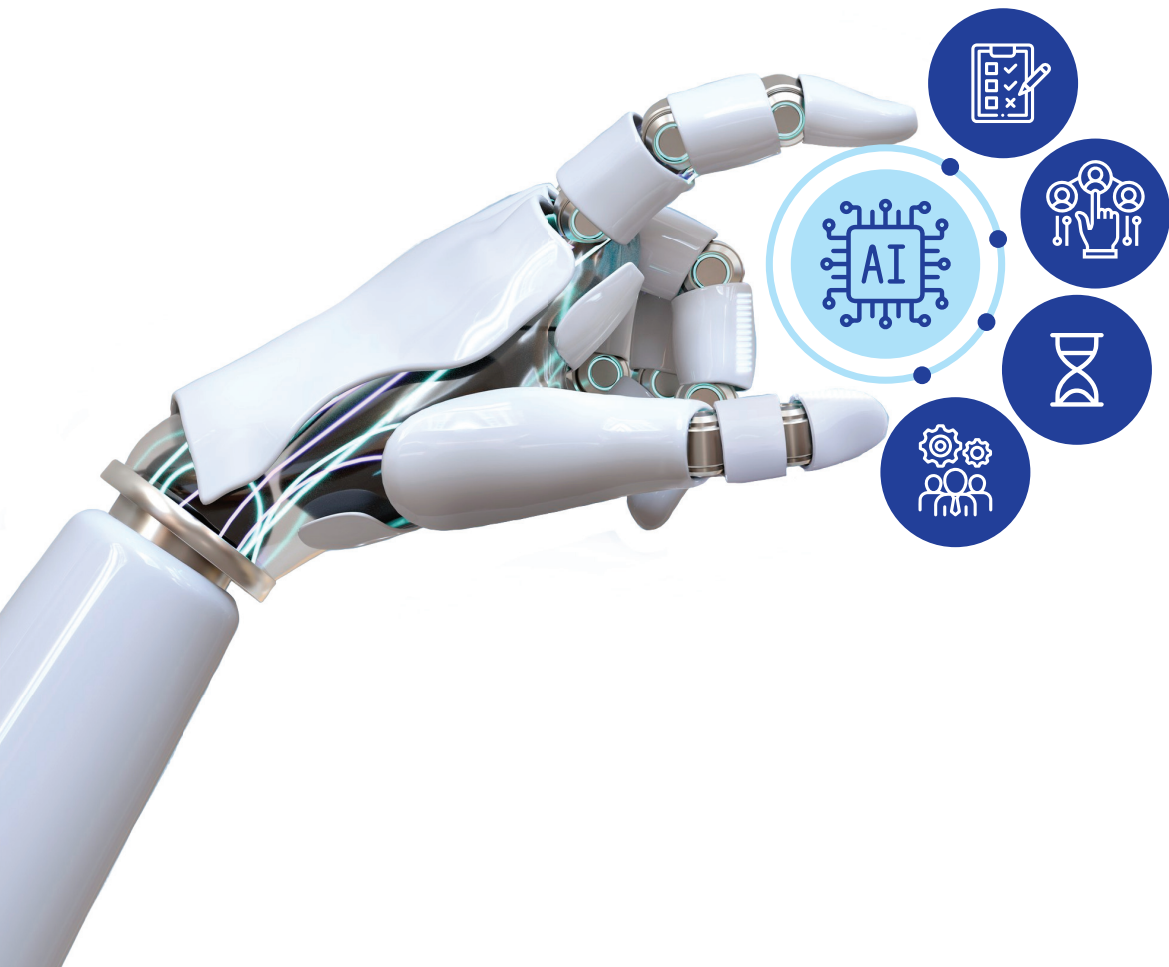


CHAPTER 1



THE MOST POPULAR ARTIFICIAL INTELLIGENCE TECHNIQUES



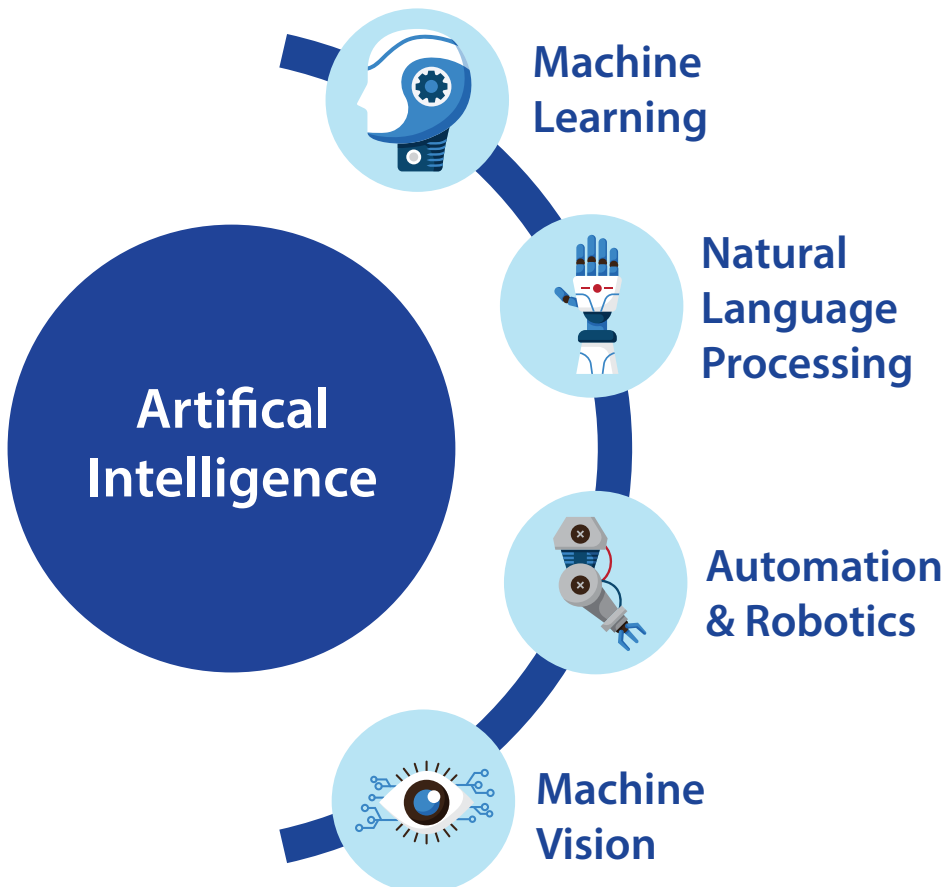


1. THE MOST POPULAR ARTIFICIAL INTELLIGENCE TECHNIQUES

The most commonly used artificial intelligence techniques in research and industry are, Figure 1:

1. Machine Learning (ML)
2. Natural Language Processing (NLP)
3. Automation and Robotics (AR)
4. Machine Vision (MV)

Figure 1. Most popular Artificial Intelligence techniques.



1.1. MACHINE LEARNING

Machine Learning (ML) is an Artificial Intelligence (AI) technique where machines are first trained to perform specific tasks and then automatically learn from current and historical data to identify patterns for estimation and prediction with minimal or no errors. Using the ML technique, the required information is obtained by applying specific algorithms to large sets of existing data, where the training and learning process is iterative. ML algorithms use different computational methods during training and learn directly from adequate data sets. The results are later used for verifying and validating new data sets.

ML algorithms are constructed to create an appropriate model on a given set of available data [21]. After that, by entering new input data, the ML algorithm based on the created model gives output values, i.e., a confident prediction. The resulting prediction must be checked for accuracy, i.e., the model's error. The model is applied or further trained on the comprehensive training data set based on the error value until a minimum satisfactory error is obtained [22], [23], [24]. Machine learning methods allow computers to work independently, to learn, perceive, evolve, and adapt [25].

Types of Machine learning:

1. Supervised Machine learning
 2. Unsupervised Machine learning
 3. Semi-supervised Machine learning
 4. Reinforcement Machine learning
1. **Supervised Machine learning** means learning by monitoring each step from the input to the known final output value. A model is created with many unknown input data for training with given function values over a training set with a known, corresponding output. Training data allows the creation of a model with a certain level of accuracy, which is further ready to receive new input data and predicted output values [26], [27], [28].

There are two main groups of algorithms for Supervised Machine learning:

1. *Classification algorithms* deal with classification problems; the output variable is categorical, for example, yes or no, true or false, and similar. These algorithms are used to search for spam and electronic messages. Some of the most used once in this category are: Random Forest Algorithm, Decision Tree Algorithm, Logistic Regression Algorithm, and Support Vector Machine Algorithm.

2. *Regression algorithms* deal with regression problems where the input and output variables are linearly dependent. They are used in weather forecasting and market trend analysis. Some of the most popular once in this category are: Simple Linear Regression Algorithm, Multivariate Regression Algorithm, Decision Tree Algorithm, and Lasso Regression [29].
2. **Unsupervised Machine learning** implies learning without tracking each step when no corresponding function is given in the training set and when the output value is unknown. It often comes down to the classification problem; a particular value is the class of belonging. In pattern discovery on the input data, the model adjusts its parameters, and clusters are formed based on similarities between the input data. This group of algorithms includes Clustering and Association algorithms [30], [31], [32].
 1. *Clustering algorithm* means similar grouping objects into clusters based on attributes of similarity or differences between objects. By supplying new data to the model, the result is a label of the class to which the output value belongs. The model generates a new class if a suitable membership class is not found. Some of the most popular clustering algorithms are: K-Means Clustering Algorithm, Mean-Shift Algorithm, DBSCAN Algorithm, Principal Component Analysis, and Independent Component Analysis [33].
 2. *Association algorithm* identifies the relationships between variables observed in a given data set. This algorithm determines the dependency of similar attributes and data and links similar variables. It is used in applications for data mining and market and business data analysis. The most frequently used ML association algorithms are: Apriori Algorithm, Eclat Algorithm, and FP-Growth Algorithm [34].
3. **Semi-supervised Machine learning algorithms** include features of both supervised and unsupervised learning. These algorithms combine different data sets and overcome the shortcomings of the first two types of algorithms [35], [36].

Example: A student studying at a university under the supervision of a teacher is an example of supervised learning, while a student preparing for an exam on his own and studying alone at home is an example of unsupervised learning. A student who uses both learning concepts is a semi-supervised form of learning.

4. **Reinforcement Machine learning algorithms** are based on the process of using feedback. After the training, the algorithm reviews the existing data set, learn from experience, and improves performance. Components of adequate activities are rewarded, and every wrong move is sanctioned. The goal is to maximize the performance of suitable activities. Unlike supervised algorithms, these algorithms lack labeled data, learning solely from experience [37], [38], [39], [40].

Example: Video game, the game itself represents a given environment, and every player's move defines its state. Suppose the player receives feedback through rewards and punishments, which affects the overall result. In that case, it affects the overall result of the game, the ultimate goal of achieving the maximum result.

Reinforcement learning algorithms can be applied in various fields, such as game theory, information theory, and multi-agent systems. There are two types of algorithms [41]:

1. *Positive reinforcement learning algorithms* imply the addition of a reinforcing stimulus, which would be repeated, resulting in a reward as the outcome.
2. *Negative reinforcement learning algorithms* imply the strengthening of particular avoidance behavior, which avoids a negative outcome.

1.2. NATURAL LANGUAGE PROCESSING (NLP)

Natural Language Processing (NLP) is an artificial intelligence technique where an algorithmic computer program is constructed to understand human language as it is spoken and written, otherwise known as natural language. NLP has actual application in everyday life and work in numerous fields, such as medical research, business intelligence, and numerous search engines. NLP algorithms allow computers to understand natural language as well as humans. Natural language processing uses artificial intelligence, whether spoken or written, to process data from real life and work so that a computer can understand it. Humans have different sensors, eyes to see, and ears to hear, so computers have programs to read and microphones to collect sound. The human brain processes the input data, while computers have a program to process their input, turning it into code that the computer can understand [42], [43], [44], [45].

There are two main stages of natural language processing:

1. Pre-processing of data, and
2. Algorithm development.

Data pre-processing involves preparing and "cleaning" the textual data and preparing it in a workable form for analysis by computers with which the algorithm can work. There are several ways to do this, including:

- *Tokenization*, which means breaking down the text into smaller units in order to make work easier;
- *"Stop word" removal*, which means removing common words from the text so that unique words remain, based on which we get the most information about the text;
- *Lemmatization and stemming*, which involves reducing words to their root words and their root forms so that the processing of the text is as good as possible;
- *Part-of-speech marking* means marking words based on part-of-speech, for example, nouns, verbs, and adjectives [46].

For data that has been previously processed, a suitable algorithm for their processing is constructed. There are several different natural language processing algorithms, but two main types are most commonly used [47], [48]:

- *System based on rules* implies the use and application of linguistic rules. It was used early in the development of natural language processing but is still in use today.
- *A system based on machine learning* implies the construction of a machine learning algorithm that uses different statistical methods and analyses.

These algorithms are trained on previous data sets to perform specific tasks. Based on the training data, they further adjust their methods to process as much data as possible. Different algorithms refine their own rules using iterative processing and learning and a combination of machine learning algorithms, deep learning, and natural language processing neural networks.

Usually, companies have a large amount of unstructured data with rich text, and in order to be usable, an efficient way of processing it is needed. Also, on social networks, a large amount of information and data is collected, stored in databases, and represents a natural language that can be analyzed effectively. Here, too, there is a need for an efficient way of processing them, that is, the creation of suitable algorithms capable of processing, analyzing, and translating them into a form that is understandable for use [49], [50].

Early machine learning algorithms were historically poor at interpreting them. By using improved methods of deep learning and machine learning, algorithms can quickly and efficiently interpret them, thus adequately analyzing and translating them into an understandable form.

1.2.1. Techniques and methods of natural language processing

The two main techniques used in natural language processing are syntax and semantic analysis [51], [52], [53]:

1. Syntax represents the arrangement of words in a sentence according to grammatical rules to make the correct sentence sense. NLP uses syntax to assess meaning from language based on grammatical rules, and the syntax techniques it includes are [54], [55]:
 - *Parsing* represents the analysis of a sentence in terms of its grammatical rules.
 - *Word segmentation* means taking a part of the text and deriving the word form from it. The program analyzes whether a "blank" space correctly separates the words.
 - *"Interrupting the sentence"* involves analyzing the boundaries of each sentence in significant texts, for example, by using the "period" symbol at the end of each sentence.
 - *"Morphological segmentation"* involves dividing words into smaller parts called morphemes. which is extremely important in machine translation and speech recognition.
 - *"Stemming"* implies the division of the word into its root forms, which is useful when analyzing the text for all examples of the exact analyzed word, and all its conjugations.

Such algorithms can recognize whether it is the same word, even though the letters are different.

2. Semantics implies the meaning and usage behind words, where algorithms are used to understand the meaning and structure of sentences. Semantic techniques are [56], [57]:
 - *Ambiguity* in the meaning of the word implies the meaning of the word based on the context. A suitable algorithm using this method can understand that the use of a specific word refers to a specific space belonging to the given the word.

- *Identification* of the named entity involves categorizing words into specific groups. A suitable algorithm using this method could analyze the text and identify all mentions of a given exact word. However, the algorithm could also recognize all instances of this word as multiple separate entities.
- *Natural language generation* involves using a database to determine the semantics behind words and generate new text.

Current methods and approaches to natural language processing use deep learning algorithms and certain types of artificial intelligence that can examine and exploit specific patterns in data sets to improve the understanding of algorithmic programs. Deep learning models require large amounts of relevant data to train a given natural language processing algorithm and determine relevant correlations. Compiling large datasets is a significant obstacle for natural language processing.

Previous methods, that is, earlier algorithms for natural language processing were based on rules. In contrast, simpler machine learning algorithms gave specific answers based on the exact words or phrases that were searched for in the text. Using deep learning algorithms, which is a significantly more flexible and intuitive way of searching, the algorithms learn to identify the speaker's intent based on many examples [58], [59], [60].

The three tools most in use for these purposes are:

- Natural Language Toolkit (NLTK) is an open-source module with assemblies and tutorials.
- Gensim is a Python library for topic modeling and document indexing.
- Intel Natural Language Processing Architect. NLTK is a Python library for topologies and deep learning techniques.

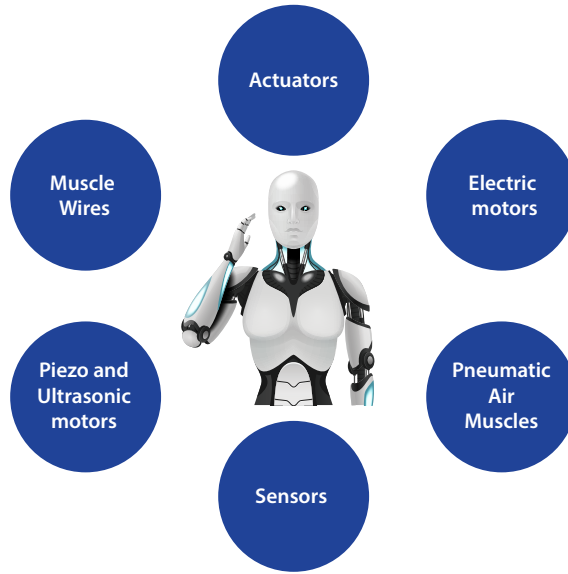
1.3. AUTOMATION AND ROBOTICS (AR)

Robotics is a field of artificial intelligence that involves using unique algorithms that can help study and create special machines to perform specific manual or other jobs, which we call intelligent robots. Robotics is a broad field that includes electrical, mechanical, and computer and computer engineering. Components from all the mentioned areas, with adequate algorithmic programs, form a whole intelligent system capable of performing tasks that can replace people. Although robotics and artificial intelligence have different goals and purposes, it is treated as part of artificial intelligence. With the help of artificial intelligence, robots are designed to look similar to humans. They can also work and perform tasks similar to humans. In the initial phase of the application of robotics, applications for these purposes were minimal. Later, combined with artificial intelligence algorithms, they become more intelligent and efficient. As a result, artificial intelligence has achieved an extremely significant role in the industrial sector by replacing humans in terms of efficiency, precision, and quality. A robot is a machine that resembles and imitates the behavior of humans, capable of performing certain actions and automatically replicating specific human movements, using commands it receives from algorithmic computer programs [61], [62], [63], [64].

Examples: a robot for mixing drugs in medicine, robots used in the automotive industry, floor cleaners, and other robots to perform various jobs that humans did until recently.

The components that are essential for robot machines are as follows [65], [66], [67], Figure 2:

Figure 2. Robot components.



-
1. *Actuators*: are devices that are responsible for the control and movement of the entire system or robotic machine. Actuators can create linear and rotary motion and thereby achieve physical machine movement using electrical, hydraulic, or air energy conversion.
 2. *Power supply*: This electrical device supplies each robot machine with electrical energy and electrical load. The primary function of a power supply is to convert electrical current to power a load.
 3. *Electric motors*: are devices that have the task of converting electrical energy into mechanical energy and are helpful in the rotational movement of robotic machines.
 4. *Pneumatic Air Muscles*: these are soft pneumatic devices, exceptionally well suited for robotics, that can contract, extend or operate under air pressure.
 5. *Muscle wire*: is made of a nickel-titanium alloy called Nitinol, which is thin in shape and can be stretched or contracted depending on the heat and electricity received. It can be bent and formed into various shapes in its martensitic form.

6. *Piezo motors and ultrasonic motors*: are electrical devices that have the task of receiving electrical signals and applying a directed force exported to a ceramic plate. These devices help the robot to move in the desired precisely defined programmed direction. These electric motors are mainly used for industrial robots.
7. *Sensors*: enable robot machines to see, hear, touch or move like humans. Sensors are devices or machines that can detect or change events or situations in the environment and, thus, can send data and information to the computer processor. These devices are usually equipped with the necessary electronic devices to react similarly to human organs. They are significant in artificial intelligence and robotics. Furthermore, they are based on artificial intelligence algorithms, and they control robot machines, sense the environment, and send information to computer processors in real-time.

Automation and Robotics has many areas of application, such as [68], [69]:

Robotics in the defense sector: this is one of the most important tasks of every country. Depending on its capabilities, each country wants its defense system to be as strong as possible.

Robotics in medical sectors: it has wide applications in various fields of medicine, such as laparoscopy, neurosurgery, orthopedic surgery, disinfection rooms, drug dispensing, and the like.

Robotics in the industrial sector: used in the production of almost all industrial products, such as cutting, assembly, disassembly, welding, packaging and labeling, product inspection and testing, painting, drilling, polishing, and many other operations. The development of robotic technology leads to efficiency, precision, productivity, and profitability; on the other hand, it reduces human activities, efforts, and physical strains and injuries [70], [71], [72]. Industrial robots have many advantages, as follows:

- Accuracy;
- Flexibility;
- Lower remuneration for work;
- Reduced noise level;
- Damage and scrap reduction;
- Increasing productivity [73].

Robotics in entertainment [74]: continuously increasing the use and use of robots. Most often in entertainment, movies, animations, games, and cartoons. Robots are beneficial in repetitive situations and actions. Nowadays, a robot holds a camera and helps to shoot a movie scene as often as needed without getting tired.

Robots in the mining industry are significant and use mining applications to perform various tasks such as automatic dosing, excavation and extraction, mapping and surveying, robotic drilling and handling of explosives, and the like. Robots can help dig and extract materials, thus protecting people from injury. Robots are also used for space exploration, and underwater drones are used for exploring large bodies of water [75], [76].

The most commonly used artificial intelligence techniques in robotics are:

1. **Computer Vision** is the most critical technology in robotics that uses convolutional neural networks.
2. **Natural language processing (NLP)** is used for voice commands to the robot. NLP and robotics are for natural language understanding and human-robot interaction.
3. **Edge computing** involves deploying analytical data from centralized nodes to the data source. Data processing happens close to the source, the data, which is essential if the device is not able to be permanently connected. These two techniques are used in complex robotics applications such as autonomous cars, drones, and similar.
4. **Complex Event Processing (CEP)** combines data from multiple streaming sources to arrive at conclusions for a more complex event, all with the goal of timely response to a given event as quickly as possible. The event itself is described as a state change combining one or more events into a complex event.
5. **Transfer learning** is a robotics technique that involves transferring knowledge gained from solving one problem and reapplying it to solve a similar related problem. It uses a pre-trained model for another similar concept or image recognition problem. In such cases, only the final finishing layers of the new model are trained, which gives a lower price and shortens the time. This technique is also applied to mobile devices for inference at the edge, meaning that models trained in the Cloud are then deployed at the edge. This idea is pertained in mobile device detection with TensorFlow Lite and transfer learning. The same principle is used in the field of robotics. A model is trained in the Cloud and then deployed

on the device. Transfer Learning is beneficial in cases with no access to the Cloud, for example, when the robot is offline. Robots can additionally use transfer learning to train other robots.

6. **Microprocessor-level hardware** acceleration for artificial intelligence algorithms generates nascent applications but is soon expected to be widely used in robotics.
7. **Reinforcement learning** is a technique that provides robotics with a basis for designing and simulating sophisticated behaviors that cannot be constructed with any other techniques. Different methods from different fields, especially physics and mathematics, are combined. This kind of research on reinforcement learning in robotics is gaining importance.
8. **Generative Adversarial Networks (GAN)** are used to get better data, especially images. This is particularly important when it is difficult to obtain the required data, or similar data are needed but not available for training. GAN thus has a growing application in the field of robotics training.
9. **Mixed reality** is an emerging technique used in programming by demonstration (PbD). PbD constructs an algorithm prototyping mechanism using a combination of physical and virtual objects.
10. **Emotion research**, affective computing is a technique that includes studying and developing systems and devices that can recognize, interpret, process and simulate human emotions. It is an interdisciplinary field that includes computer science, psychology, and cognitive science [64] [77], [78], [79], [80].

1.4. MACHINE VISION (MV)

The development of computer programs today is based on the application of artificial intelligence. Any form of human behavior, or some form of natural intelligence, involves using artificial intelligence algorithms to interpret or imitate human behavior. The most commonly used artificial intelligence techniques, which are used in machine form, are [81], [82], [83]:

- expert systems;
- fuzzy logic;
- inductive learning;
- artificial neural networks;
- genetic algorithms and
- swarm of intelligence.

Expert systems are by far the most commonly used technique of artificial intelligence. They consist of algorithmic computer programs that apply knowledge in a given domain to solve problems that are observed and occur in it [84], [85]. For example, the MICIN expert system is used and applied in medical diagnostics to solve and detect health problems, just as a doctor diagnoses each patient. The expert system is often based on fuzzy logic and uses knowledge bases and inference methods based on the knowledge it contains, most often using IF-THEN rules.

Fuzzy logic can interpret a fuzzy description of the nature of human reasoning but can enable systems to be more resilient. Unlike standard exclusive logic, in fuzzy logic, the precise value of a variable is replaced by a linguistic description. Knowledge can be interpreted as qualitative statements or fuzzy rules in systems using fuzzy logic [86], [87], [88], [89].

Inductive learning is an automatic technique for acquiring knowledge, which involves extracting knowledge based on the rules used or applying an adequate decision tree. This way of learning is part of machine learning. In order to generate knowledge, this learning uses mathematical analysis of a particular data set [90].

Artificial neural networks represent the branch of artificial intelligence with the broadest and most pervasive practical applications across various domains of life and work in contemporary times. They use different computational models that can learn and then applied to appropriate similar sets and databases. Similar to algorithmic inductive learning programs, artificial neural networks can generate knowledge from a specific domain based on previous examples.

They do not archive the acquired knowledge in an explicit form, such as rules or decision trees, but most often in the form of weighted links. However, artificial neural networks can provide better answers to various problems without explaining the appropriate answer [91], [92], [93]. Artificial neural network models distribute the calculation to several simple units, neurons, which are interconnected and work in parallel. So far, the most commonly used artificial neural network is the multi-layer perceptron. It consists of neurons arranged in a series of multiple layers [94].

A genetic algorithm is an artificial intelligence tool for optimization. This algorithm has the task of searching for a solution to a problem in an extensive defined data set; that is, it has an ample search space.

Example: the optimization problem is the traveling salesman problem, where the customer must visit several cities at the shortest distance. In this problem, all cities are known, but all possible routes are also known. Here, the problem boils down to searching all possible combinations of routes to find the optimal route. That is why a fast search technique is necessary, which can be solved the fastest with a suitable genetic algorithm. Genetic algorithms are computer programs based on the theory of natural selection, where the fittest animal species survive and reproduce. Similarly, a genetic algorithm combines solutions to create new ones and keeps only the best solutions in each generation [95], [96], [97], [98].

Swarm intelligence involves studying and simulating the exact behavior of swarms of insects.

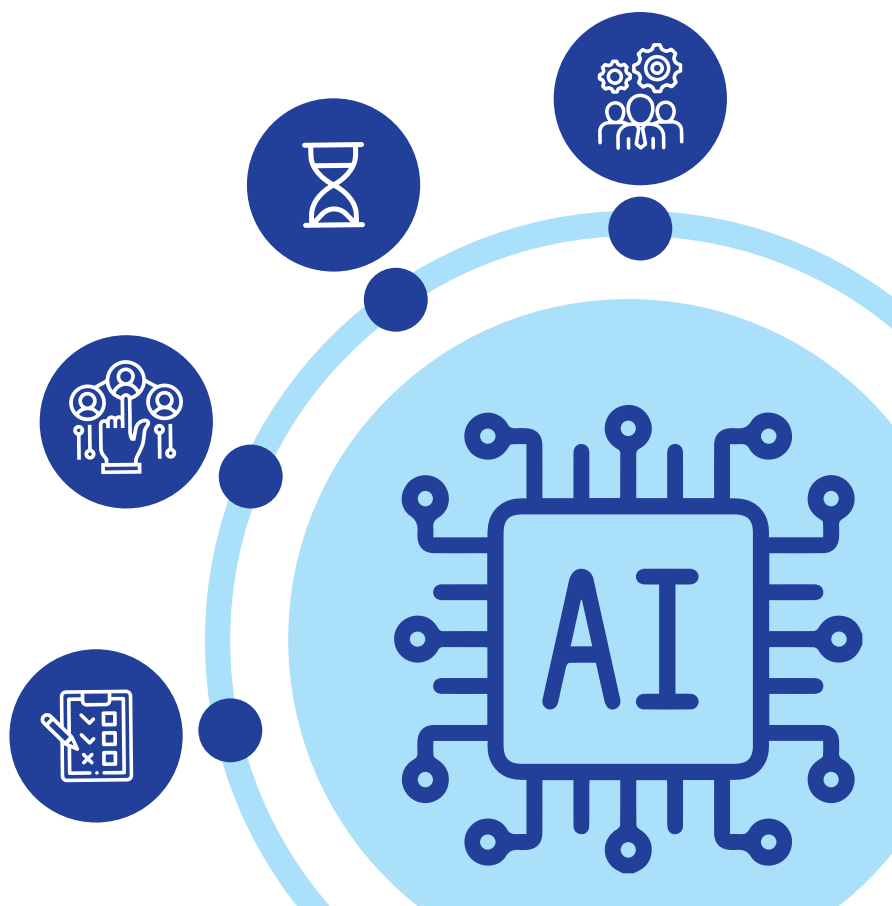
Example: ants or bees work together in a colony, creating a coherent system more significant than the sum of its parts. This technique aims to imitate the patterns of behavior of large groups of insects and find and construct specific patterns and solutions for certain problems [99], [100], [101].

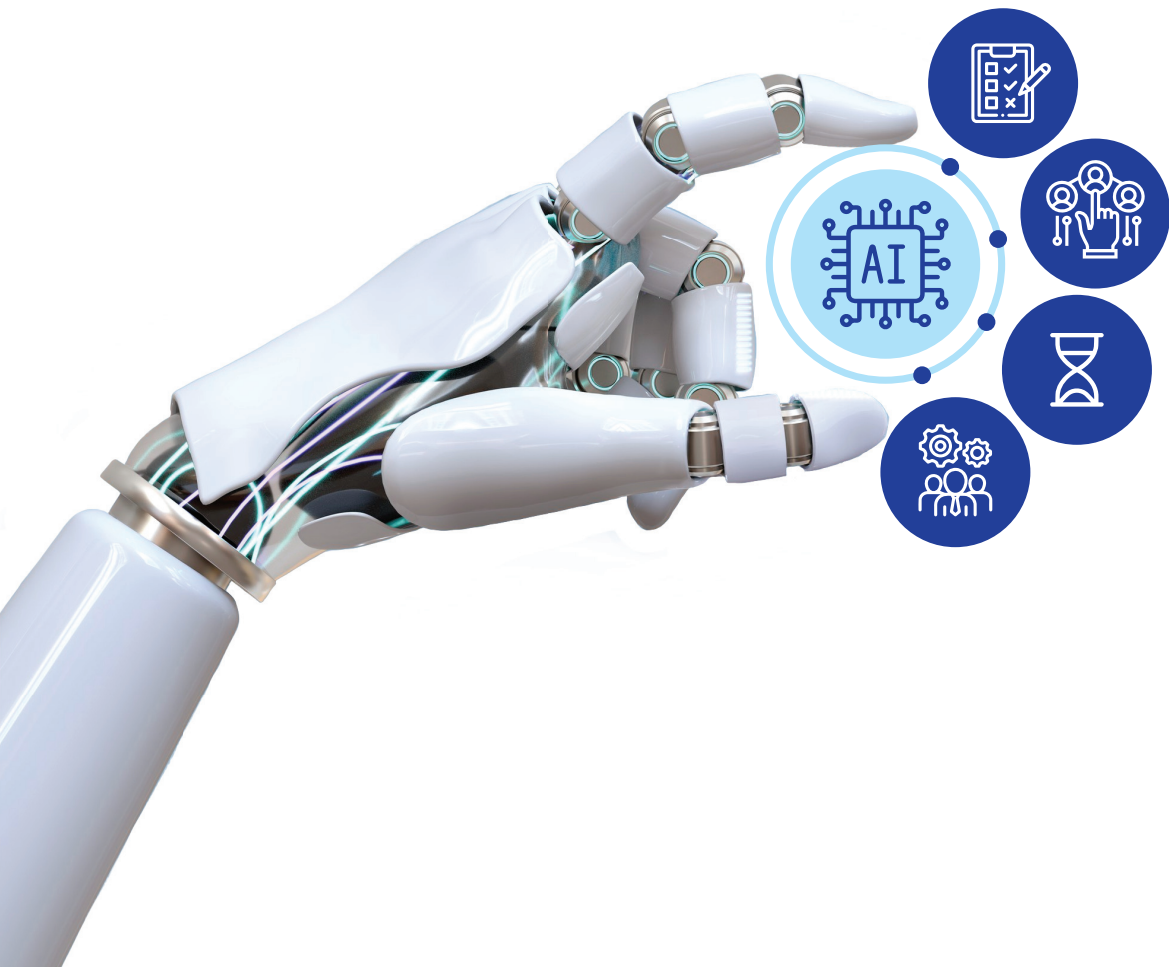


CHAPTER 2



THE ROLE OF ARTIFICIAL INTELLIGENCE IN SOFTWARE DEVELOPMENT





2. THE ROLE OF ARTIFICIAL INTELLIGENCE IN SOFTWARE DEVELOPMENT

Modern software development technologies differ in many ways from the initial technologies for creation and implementation. Companies are rapidly adopting almost all new technologies in their software development and, to the greatest extent, with the application of artificial intelligence tools, which is becoming almost unimaginable nowadays. The impact of artificial intelligence on the development and implementation of software solutions is changing how companies conduct business and make software smarter [102], [103].

Software development and implementation teams confirm the thesis that the application of artificial intelligence in software development has primarily led to improved development through various agile test automation, then automation test software, and the like. Various artificial intelligence tools significantly contribute to more reliable, efficient, and easier development of software solutions [104], [105].

Artificial intelligence significantly impacts how a specific application is developed and can be expected to improve its development in a given environment significantly. Understanding and using artificial intelligence is shaping future even more efficient and precise applications in software development. Nowadays, most companies, even about 80% of them, invest in developing and applying the necessary tools to perform their jobs. About 50% of companies have already created new artificial intelligence strategies, resulting in substantial business profits in the shortest possible time [106], [107], [108].

2.1. THE IMPACT OF AI IN DIFFERENT PHASES OF SOFTWARE DEVELOPMENT

In this subsection, we will explain how AI is involved in different phases of software development such as collection of requirements, software design, automatic code generation, software testing, and deployment control, Figure 3.

1. Collection of requirements

Collecting requirements is the initial phase of project development, which implies maximum human intervention. Artificial intelligence today offers a wide range of techniques and tools, such as Google ML Kit and Infosys Nia, for automating specific processes in the execution of such jobs in order to minimize human participation. This phase is of particular importance in the early detection of errors before proceeding to the next design phase. An artificial intelligence technique, natural language processing can significantly contribute to machines understanding user requests in natural language and automatically deriving high-level software models. Indeed, this approach cannot solve specific problems, which can be another challenge that future researchers can deal with [109], [110].

2. Software design

Determining the right design in a project's planning phase is a task where designers can quickly get it wrong. Pre-designed solutions impose dynamic changes until the user is satisfied, which requires a significant increase in time. By using artificial intelligence tools, automation can enable the most efficient methods for designing projects. Using the artificial intelligence tool AIDA (Artificial Intelligence Design Assistant), designers can understand user needs and use that knowledge to design a project. AIDA is a platform designed to create different combinations of web pages and can quickly and efficiently adapt the design to the user's needs [111], [112].

3. Automatic code generation

Writing code, especially for large projects, is still time-consuming and demanding, requiring some work experience. To solve time and money problems, experts design and write the code before starting the project development. This approach may not always be good, but it is certainly better than writing code from scratch. Programming with artificial intelligence techniques can help to some extent. It is also expected that when you explain or write the desired project to the system, it will be able to convert it into executable code. This will be possible by integrating natural language processing techniques and artificial intelligence tools [113], [114], [115].

4. Software testing

The most critical phase in software development is the testing phase, which confirms the quality of the product. If software testing is repeated several times, the source code is also changed, and repeating the same tests takes a lot of time and money. In this phase, applying artificial intelligence tools and techniques is very important.

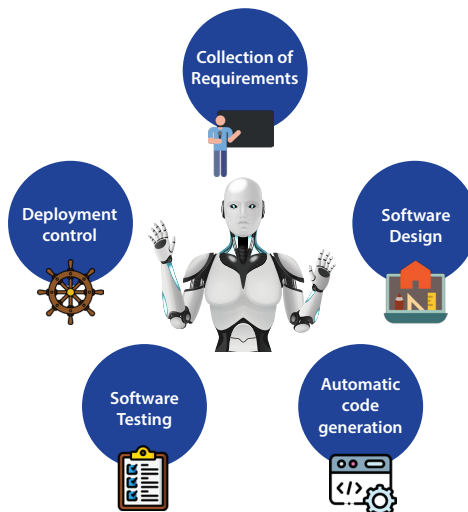
Several tools use algorithmic AI programs to construct use case tests and perform regression testing. These tools can automate testing and further ensure error-free testing. The most common machine learning-based artificial intelligence tools used to develop software projects are, e.g., Appvance, Functionize, and Testim.io [116], [117], [118].

5. Deployment control

Artificial intelligence technologies have had inevitable impacts on software deployment, such as increasing the efficiency of deployment control activities. In the software development paradigm, the deployment phase is the phase where developers often upgrade programs or applications to newer versions.

If the developers fail to execute the process correctly during the upgrade, there will be high risk in executing the software. AI can prevent developers from such vulnerabilities during upgrades and reduce the risk of deployment failures. Another impact of artificial intelligence is that it enables the analysis of application processes using machine learning algorithms [119], [120].

Figure 3. AI in different phases of software development.



How can AI improve software development even more?

1. AI-managed Code Compiler

To translate software code into machine language, developers use AI-managed code compilers. Glow or Compiler.ai are the most widely used compilers. They serve as a backend for high-level machine learning frameworks and make it possible to generate code and optimize neural network graphs. These AI tools improve the speed and quality of code compilation through machine learning. While these AI-managed compilers are still in the development stage, we can anticipate some fantastic features and functions from them in the future [121].

2. AI-Based Programming Assistants

Because they can assist a developer with code compilation, code debugging, and code-driven testing, these tools are pretty popular among software developers. Examples of AI-based programming assistants include Kite and Codota, which enable developers to write code in nearly every programming language. These assistants can use libraries, complete code lines, and correct syntax. These tools are made from learning models and trained on them. They use machine learning to write codes faster and with less typing while also increasing productivity [122].

3. AI-Aided Automated Software Testing

AI can make software testing more accessible, faster, and less time-consuming. Because it cannot make mistakes or lose focus while working with many data, artificial intelligence makes software testing more accurate and efficient. Bugs are found, and reports are generated by AI-aided automated testing without the need for human intervention. Eggplant and Test Sigma are two well-known AI-aided software testing tools that assist software testers in writing and carrying out automated tests to lessen the impact of bugs and boost code efficiency [123].

4. Software Security

Security is the most crucial aspect of any system because software processes sensitive data using business logic. Organizations are using AI worldwide to collect data and use Machine Learning to tell the difference between unusual behavior and expected behavior. For cybersecurity, AI systems can be used to detect malware, perform pattern recognition, and observe the behaviors of malware before it enters the system [124].

5. Automated DevOps

AI and machine learning (ML) have the potential to enhance the software deployment procedure, which requires programmers to upgrade applications and codes to more recent versions and complete several other software deployment control tasks. If developers are unable to deploy and upgrade the applications correctly, the software could be in danger. Developers can avoid such issues during upgrades and lessen the likelihood of deployment failure thanks to AI and ML. Algorithms based on machine learning were aided by AI in their examination of the deployment procedure, locating any potential roadblocks, and proposing solutions to developers [125].

6. Precise Planning and Cost Estimation

The software development process relies mainly on planning and cost estimation. AI can provide a roadmap for software development. We are able to input the company's initial state, the goal, and a list of possible actions. The AI tool can calculate potential outcomes and challenges based on this model, which improves decision-making and enables organizations to make necessary adjustments quickly, as follows:

- To estimate the price of a software product, we employ a similar model. The tool allows us to enter the required information, after which machine learning algorithms can calculate and determine the cost of the various variants;
- Because it enables businesses to manage their financial assets and create project plans, precise planning can influence the software development process. Tools based on AI and machine learning can be used to create precise solution planning [126].

This AI-aided strategy has the following advantages:

- Quicker and more intelligent data analysis;
- Gives the system the ability to figure out essential parameters and develop a plan;
- Enhances the system's capacity to handle incomplete data.

7. A better understanding of User behavior

By adjusting the screen size, font size, buttons, and many other on-page elements, machine learning algorithms can help us comprehend user behavior and then deliver variable content. Such dynamic and personalized responses have the potential to enhance the user experience and enable programmers to observe real-time user interaction data to make appropriate code modifications.

Online Marketplace portals use AI and ML to improve software functionality, collect customer feedback, reduce friction points, prevent abandoned carts, and boost conversion rates [127], [128].

8. AI-assisted Strategic Decision Making

Software development organizations benefit from the assistance of AI algorithms in adhering to the appropriate development methodology to produce the ideal solution. Major business key performance indicators (KPIs), a suitable framework, and other vital indexes are created by these algorithms using AI and machine learning. Software aided by AI and ML is able to evaluate and develop the most effective decision-making strategies, assisting programmers in making wise choices. The strategic decision-making process can be transformed by AI and ML, but these algorithms are not perfect and may not adapt to dynamic programming environments. As a result, management should evaluate the decisions and, if necessary, adjust the strategy accordingly [129].

We can conclude the following advantages of using AI technologies in software development:

Combining machine learning, deep learning, and natural language processing with AI results in unprecedented capabilities that can alter software development. In layman's terms, we can say that AI-aided tools perform routine tasks with maximum efficiency and mimic human behavior.

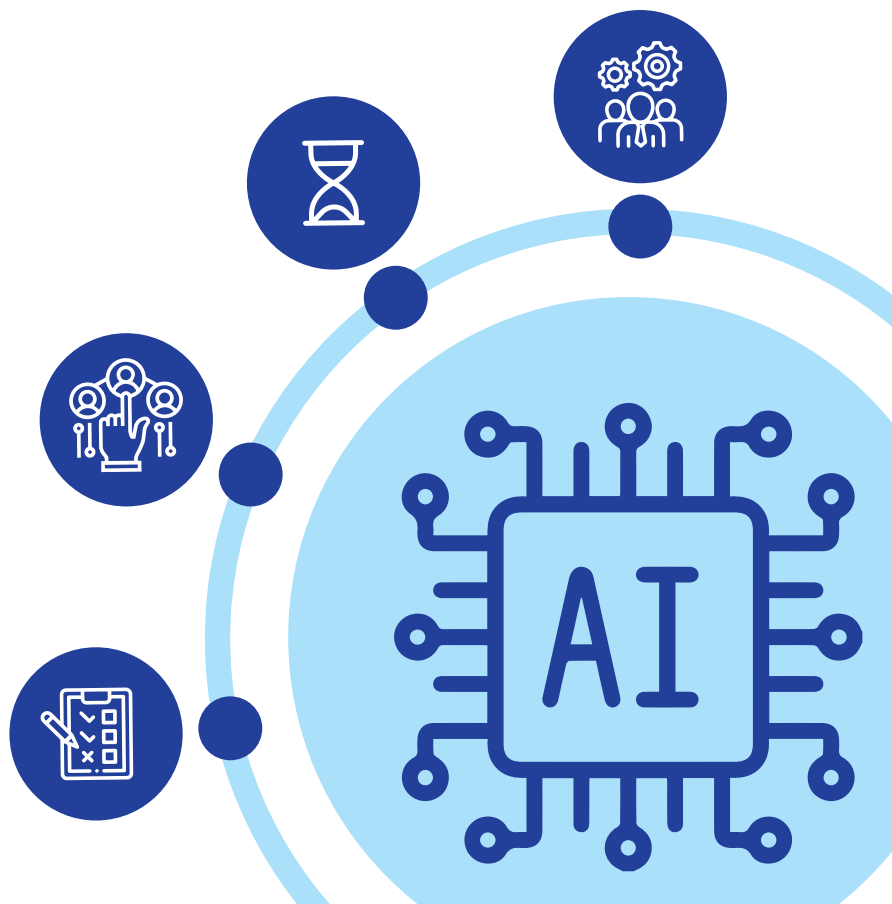
Software testers and developers benefit from AI and ML-powered tools that enable them to create high-quality software solutions. In addition to assisting businesses in creating software, AI and ML also assist businesses in comprehending customer behavior, testing the code, providing adequate security to the code, and making strategic decisions. In light of this, technology leaders must select AI-powered technologies and tools to transform their software development process.

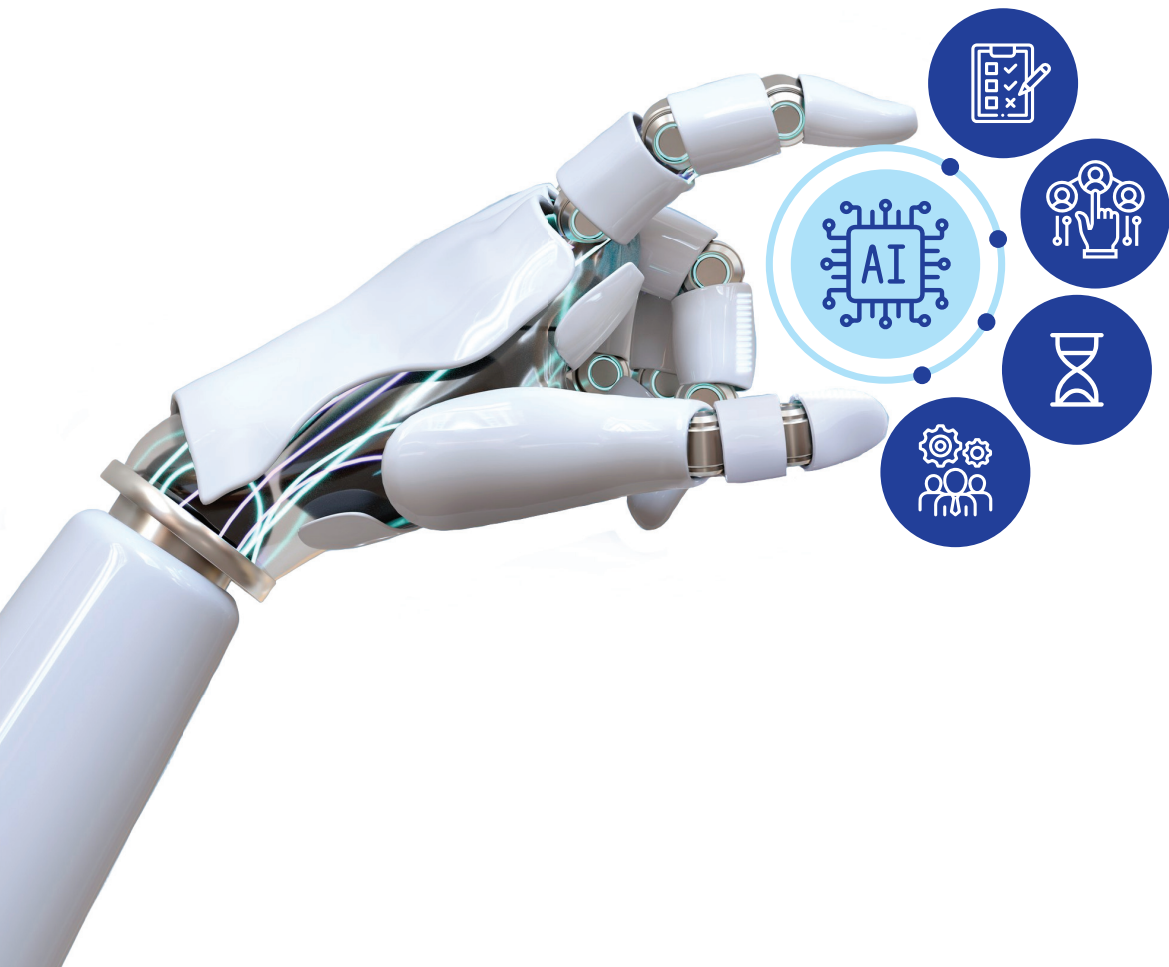


CHAPTER 3



POWER OF ANNS FOR COST AND EFFORT ESTIMATION DURING SOFTWARE DEVELOPMENT





3. POWER OF ANNS FOR COST AND EFFORT ESTIMATION DURING SOFTWARE DEVELOPMENT

Prior to commencing a software project's development, it is imperative to obtain a swift, precise, and high-quality estimate of the anticipated effort and associated costs. In today's software industry, the ability to meet user expectations is paramount. A key determinant of successful software project development and mitigating error risks hinges on accurately assessing the effort and costs dedicated to its implementation.

In this chapter, we introduce three novel models for estimating effort and costs in software project implementation. These models leverage artificial intelligence techniques, tools, and rigorously validated metrics. Each of these enhanced models builds upon different prevailing approaches commonly employed in practice, incorporating a fresh methodology and distinct artificial neural network (ANN) architectures rooted in Taguchi's orthogonal vector plans. We optimize these improved models through a combination of methods and ANNs, reducing the need for excessive experimentation, avoiding repetition, and shortening training time.

In practical experiments, we suggest applying actual industrial project datasets, clustering them based on various analytical criteria, depending on the chosen approach. The integration of fuzzy logic and clustering methods helps standardize input values across small, medium, or large projects, enhancing the overall reliability and precision of our results. Optimization using Taguchi's orthogonal vector plans plays a pivotal role in reducing the number of experiments while ensuring that all data pairs are equally scrutinized, without omitting crucial factors. These innovative techniques, tools, and combined methods yield significantly more accurate, faster, and efficient estimates of effort and costs during software project implementation, enhancing the likelihood of successful completion.

The aim of the new methodology is:

- creating and pinpointing the optimal model for estimating both effort and costs,
- Choosing the most efficient artificial neural network architecture that converges most rapidly,
- minimum magnitude relative error size,
- minimizing the number of experiments, and
- decreased software effort estimation duration attributable to the convergence speed.

After all performed experiments in different phases, the obtained results showed a significant reduction in the error of each model, significantly lower than the studies presented and published in the literature so far. In each experiment, the models were additionally validated and checked with the setting of appropriate criteria. Hence, they can be confidently utilized for estimating effort and costs in software development as well as across various domains in industry and science.

3.1. UTILIZING ANN IN SOFTWARE EVALUATION

The crucial key to successful software project implementation is accurately estimating time and financial resources. Both clients and software companies place immense importance on the development and completion of a project. The project's viability hinges on the time and financial investments required, impacting its initiation and successful conclusion. Typically, project costs are compared to those of similar completed projects in practice. While time and financial considerations are paramount, they do not stand alone in defining project realization. Alongside these pivotal factors, other equally significant aspects must also be taken into account, including quality, project complexity, team overtime, and more.

Inaccurate initial effort estimates can frequently result in project delays, necessitate extra funding requests, and lead to programmers and test engineers working overtime. Moreover, they have a substantial impact on software quality. Consequently, equally critical tasks like supplementary testing, documentation finalization, and user requirement clarification are frequently minimized, often resulting in delayed project completion or budget overruns. The consequences of projects with imprecise definitions and accurate estimates are a current reflection of this aspect of the software industry.

After extensive research conducted over several years by the Project Management Institute, including their analysis [130], and Professor Barry Boehm's research leadership [131], it was determined that effort and costs could be visualized using the concept of the '*cone of uncertainty*,' as illustrated in Figure 4. This approach demonstrates that estimated effort tends to converge closely to the final error value as a project progresses through its various phases.

Figure 4. "*Cone of uncertainty*".

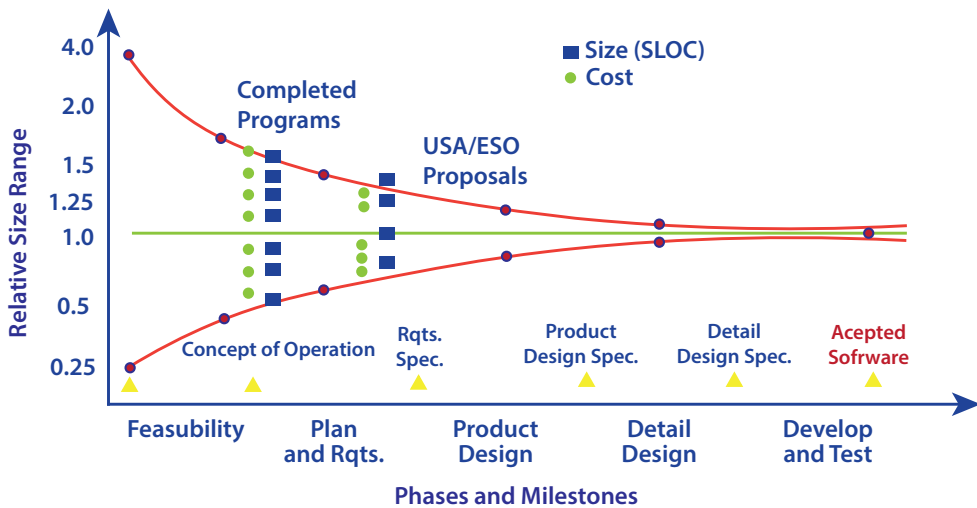
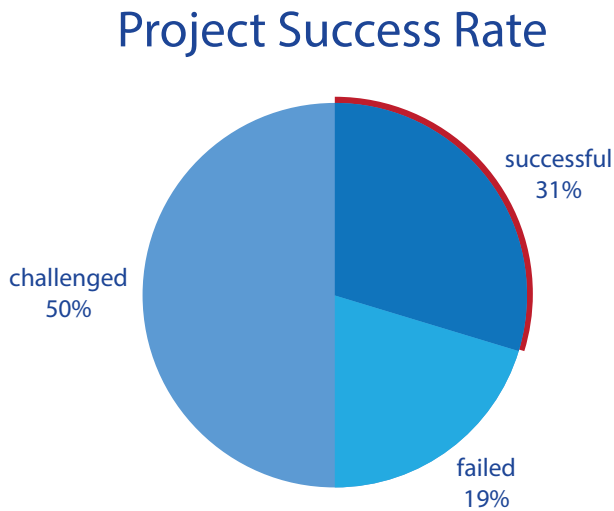


Table 1 [132], [133] reveals a declining trend in the successful implementation of projects, dropping by 6% over the decade from 2010 to 2020. In 2020, the rate of successful software projects stands at approximately 31%. Conversely, the number of unsuccessful projects decreased by 2%, reaching 19% in 2020. Notably, the number of pending projects saw a substantial 8% increase, currently standing at around 50%, as depicted in Figure 5 [134], [135].

Figure 5. Visual depiction of projects' outcomes in 2020, including successful, unsuccessful, and challenged cases.



The Standish Group reports [6], which curates a global database encompassing various project types, reveal that the primary issues plaguing unsuccessful and unresolved projects are missed deadlines and budgetary overruns. Approximately 80% of projects experience delays, with 50% encountering budgetary excesses. It becomes evident that time and budget represent the most critical risk factors in software implementation. Therefore, the most challenging and crucial task for software teams is to deliver a dependable, precise, and swift estimate of the effort and costs required for project implementation. Standish Group reports further demonstrate that the likelihood of failure significantly increases for larger projects [133], [134], [135], as compared to their smaller counterparts [136], [137].

In the present era, software firms employ a diverse array of software tools to address this challenge and fulfill customer demands. To develop dependable, top-tier software solutions, numerous researchers [138], [139], [140], [141] have put forward diverse amalgamations of parametric and non-parametric models for effort and cost estimation. To further enhance these models for superior accuracy and efficiency, it becomes imperative to scrutinize and empirically validate the most successful ones to date, enabling us to refine them effectively [142], [143], [144].

3.2. CHALLENGES IN SOFTWARE EVALUATION

Conventional software development methodologies depended on unreliable and imprecise practices, strategies, and models, resulting in a substantial number of unsuccessful and unrealized projects. Frequently utilized techniques encompassed the similarity approach, knowledge-driven analysis and synthesis methods harnessing expert insights, and assorted parametric (algorithmic) approaches [142].

1. **Similarity Method:** Effort and cost estimates draw from analogous projects. In reality, there are no projects that are entirely alike, and this approach helps identify the specific aspects of a similar project that can be applied. Identifying differences is crucial, and they must be evaluated through means such as similarity analysis or alternative methods.
 - The merit of the similarity method lies in its capacity to reduce effort and costs, as it is applied to real-world projects born out of practical experience.
 - The drawbacks become evident in the practical reality where truly identical projects are scarce. As a result, the method often resorts to comparing projects with some differences, leading to increased time requirements [142], [143], [144].
2. **Analysis/Synthesis:** Estimating software effort and cost with this approach involves breaking the project into smaller segments, each of which is assessed individually. Subsequently, an overall evaluation of the project's feasibility is derived from these individual assessments.
 - The benefits of employing this approach for evaluation include that it makes the assessment of project components more straightforward and less complex.
 - On the downside, this method demands additional time, yet the precision of the overall evaluation based on the summary of smaller project components is less dependable [142], [145].
3. **Expert knowledge-based:** This approach entails estimating effort and costs by drawing upon the expertise of one or more professionals who have previously been involved in comparable projects.
 - The merits include its speed and simplicity.
 - On the flip side, the drawbacks involve the subjectivity of expert knowledge, and when it is formed based on the input of multiple experts, arriving at an objective solution can be challenging [146], [147].

4. **Algorithmic (traditional) methods:** The project's size (measure) is used to estimate effort and costs, and an algorithm is built to figure out how long it will take to implement and how much it will take will cost.
- This approach offers the following benefits: it is impartial, speedy, and uncomplicated.
 - Drawbacks encompass the method's reliance on historical data, necessitating a strong understanding of and diligent tracking of such historical data [142], [148], and [149].

3.3. HYBRID MODELS CONSTRUCTION FOR SOFTWARE ASSESSMENT

Effectively executing the project necessitates a meticulous and precise evaluation of effort and costs, which heavily relies on numerous software functional, and non- functional conditions.

Figure 6 [142] illustrates the impact of the most prominent elements on effort and costs assessment throughout the software project development process.

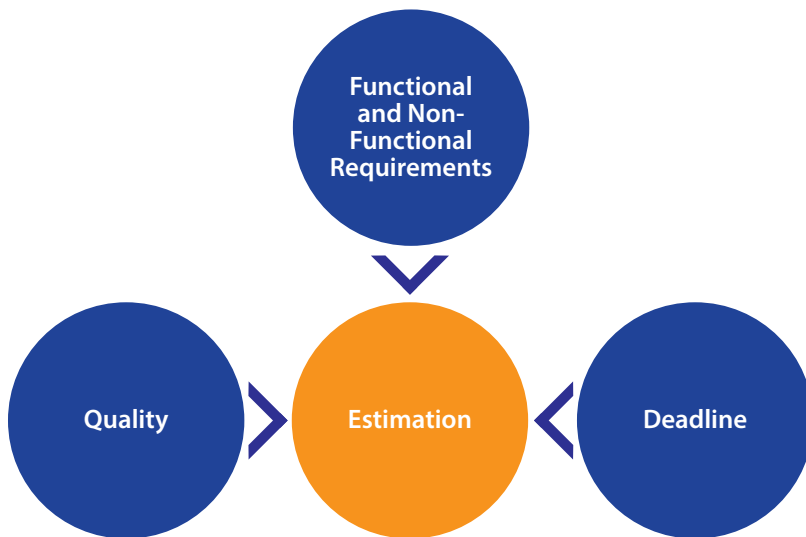
Functional requirements imply different activities, as follows:

- compliance with business rules,
- corrections and changes to transactions,
- adjustments and cancellations,
- various administrative functions,
- authentication,
- levels of authority,
- audit monitoring and control,
- external interfaces,
- requirements for certification,
- reporting requirements, and
- historical data.

Non-functional requirements include activities as follows:

- speed or response to commands in the operation of applications,
- security of sensitive data,
- portability,
- compatibility,
- capacity,
- reliability,
- business environment,
- localization, and more.

Figure 6. The effect of the foremost elements on the effort and costs assessment associated with software project implementation.



Software quality implies adhering to the organization's highest standards while meeting all requirements. This often includes tasks such as processing and analyzing extensive data, enhancing speed, and managing request and response transmissions, all of which may require additional time for implementation.

Project deadlines exert the most significant adverse impact on project failure. Consequently, software development should not be contingent on other critical factors, including team size, extra time for enhancing software quality, software implementation, initial release timing, and task completion for the specific software product in question [142], [150], [151], [152].

3.4. NOVEL HYBRID MODELS

In this chapter, we will introduce three novel models that leverage artificial intelligence tools and various artificial neural network architectures. The objective is to enhance and refine existing models, thereby providing a potent tool for improving software evaluation outcomes. These proposed models underwent optimization using diverse ANN architectures constructed based on Taguchi orthogonal vector plans [136], [137], [153], [154], [155], [156]. The pursuit of a streamlined, rapid, and accurate assessment holds paramount significance throughout the various phases of software project management and implementation, aiming to preempt cancellation, degradation, or exceeding the allocated time and budget. Employing innovative techniques and tools, coupled with the integration of different methodologies, augments the likelihood of successful project execution—a matter of utmost importance for software companies and customers anticipating a well-structured software product within predefined timeframes and budgets [157], [158].

We will provide comprehensive explanations for all the models introduced, commencing with the training phase, followed by testing and validation using additional real datasets. This approach aims to ensure efficient and precise allocation of effort and costs for the realization of software products. Each of these three novel models must satisfy specific criteria to surpass the estimation accuracy achieved by their predecessors.

1. Selection of the most straightforward ANN architecture;

In the initial training phase, encompassing the training of each ANN, we begin by selecting the most straightforward architecture based on the input values. The chosen ANN architecture that yields the most favorable outcome, i.e., the smallest relative error, is subsequently employed in both the second and third phases, albeit on distinct datasets. The process of selecting the architecture commences with the simplest configuration featuring the minimum number of nodes. It then progresses to a single hidden layer and so forth, until the error of the succeeding architecture becomes equal to or smaller than that of the preceding ANN architecture. The chosen architectural design is meticulously crafted in accordance with the corresponding Taguchi method.

2. Minimizing number of performed iterations (< 10);

Throughout the training phase, we closely monitor the Gradient Descent (GA) value for each ANN architecture as a stopping criterion. Training concludes when the GA value dips below 0.01% of the Mean Relative Error (MRE). With the application of the Taguchi orthogonal vector plan, each experiment for the proposed models will involve <10 iterations.

3. Choosing the architecture exhibiting the swiftest convergence towards the Mean Relative Error (MMRE);

Numerous experiments involving distinct models have revealed that certain architectures require a smaller number of iterations, often as few as five, to meet the 'stopping criterion' rapidly. This denotes their swift convergence to the final value. We will meticulously track the convergence speed in each of the proposed models, particularly focusing on the required iterations for each selected ANN architecture.

4. Choosing the suitable activation function within the ANN architecture;

During the training phase, we conducted experiments involving various architectures and explored different activation functions for both the hidden and output layers. This encompassed the examination of functions such as the sigmoid function, hyperbolic tangent, and Gaussian function.

5. Proper partitioning of the utilized datasets into a specific number of cluster;

Real projects exhibit diverse sizes, complexities, and distinct user requirements, resulting in heterogeneous data characteristics for each observed dataset. To achieve optimal estimates, it becomes imperative to partition each dataset into suitable subsets.

In the first of the novel, hybrid model (COCOMO2000), we employed a division into three clusters for each of the actual datasets used: small, medium, and large.

For the second new, hybrid Function Point Analysis (FPA) model, we leveraged an extensive dataset from the International Software Measurement Standards Group (ISBSG). This dataset was categorized into five clusters based on functional size, following a 70:30 split, with 70 projects used for the initial training phase and 30 projects reserved for the subsequent testing phase.

In the third novel, hybrid model, which uses Use Case Points (UCP), we employed a 70:30 data set division, with 70 projects designated for the initial training phase and 30 projects earmarked for the subsequent testing phase.

3.5. CHOOSING THREE DISTINCT APPROACHES

In three different approaches, three new, improved models were used (one from each approach) using the same methodology and different datasets depending on the input values of each observed approach. Furthermore, we incorporated the clustering approach [159] and fuzzification method [160], [161] in conjunction with different ANN architectures built upon Taguchi's method and varied function activation [162] [163]. For each of the proposed approaches, we assessed new and enhanced models using diverse real-world datasets. This evaluation involved the computation of various metrics and the establishment of criteria, including:

- mean absolute error (MAE),
- magnitude relative error (MRE),
- mean relative size error (MMRE) [164],
- prediction based on 25, 30, 50 criteria (PRED) [165], [166],
- Pearson, Spearman, R2 correlation coefficients [167].

The experimental part of the research consists of 3 phases:

1. The training phase of the suggested ANN architecture, during which a specific number of networks vie to be the designated 'Winner' network for a given dataset;
2. The testing phase is conducted using the same dataset but with different projects, employing the 'Winner' network selected during the initial training phase based on its superior performance, characterized by the lowest MMRE value;
3. The validation phase involves the same procedure as testing but on other sets of actual data using the "Winner" network.

In the experimental part, in all three approaches, for each selected model, the following methods and techniques are used:

- the method of dividing data sets into clusters concerning the real value of projects,
- the process of input value fuzzification for each dataset involves transforming actual input quantities into values within the range $[0, 1]$ - distinct ANN architectures are used in each of the suggested models,
- each of the employed ANN architectures is developed in accordance with the respective Taguchi method,
- the speed of convergence towards the MMRE value is monitored,
- different activation functions,
- various metrics are calculated,
- GA or "stop criterion" is used to know when to stop performing the experiment.

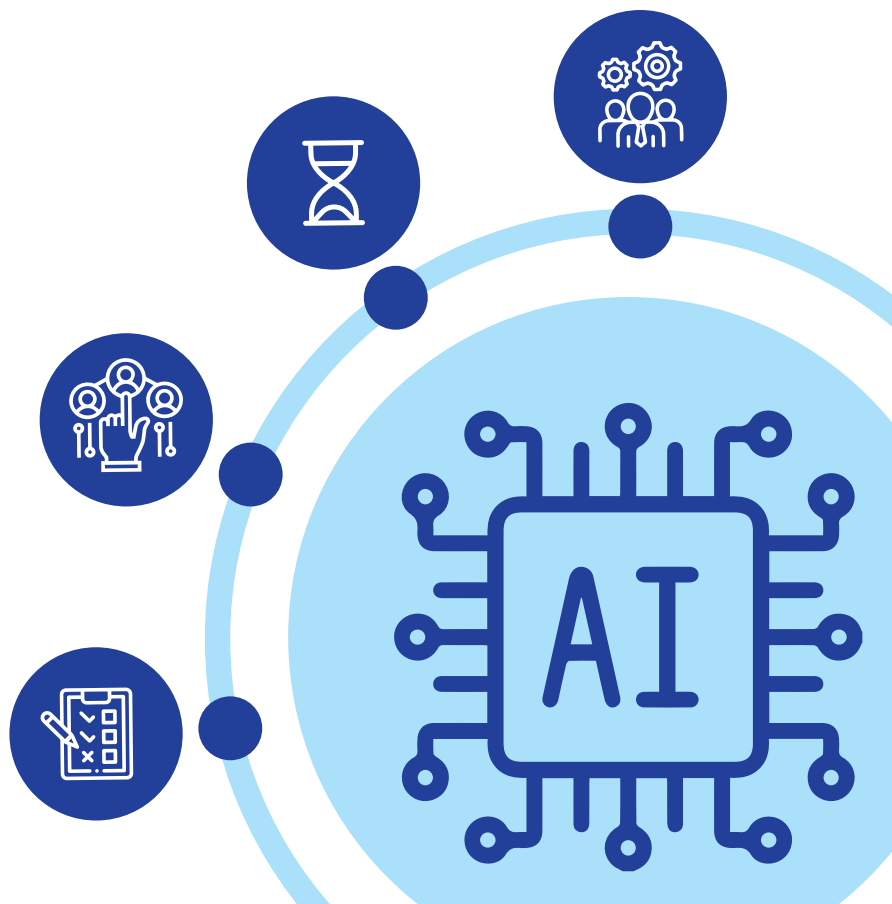
Numerous experiments have consistently demonstrated that, across all three proposed models, the error values are notably lower compared to those observed in prior research involving both parametric and non-parametric methods. This provides empirical evidence for identifying the optimal model, one that can be practically and dependably applied.

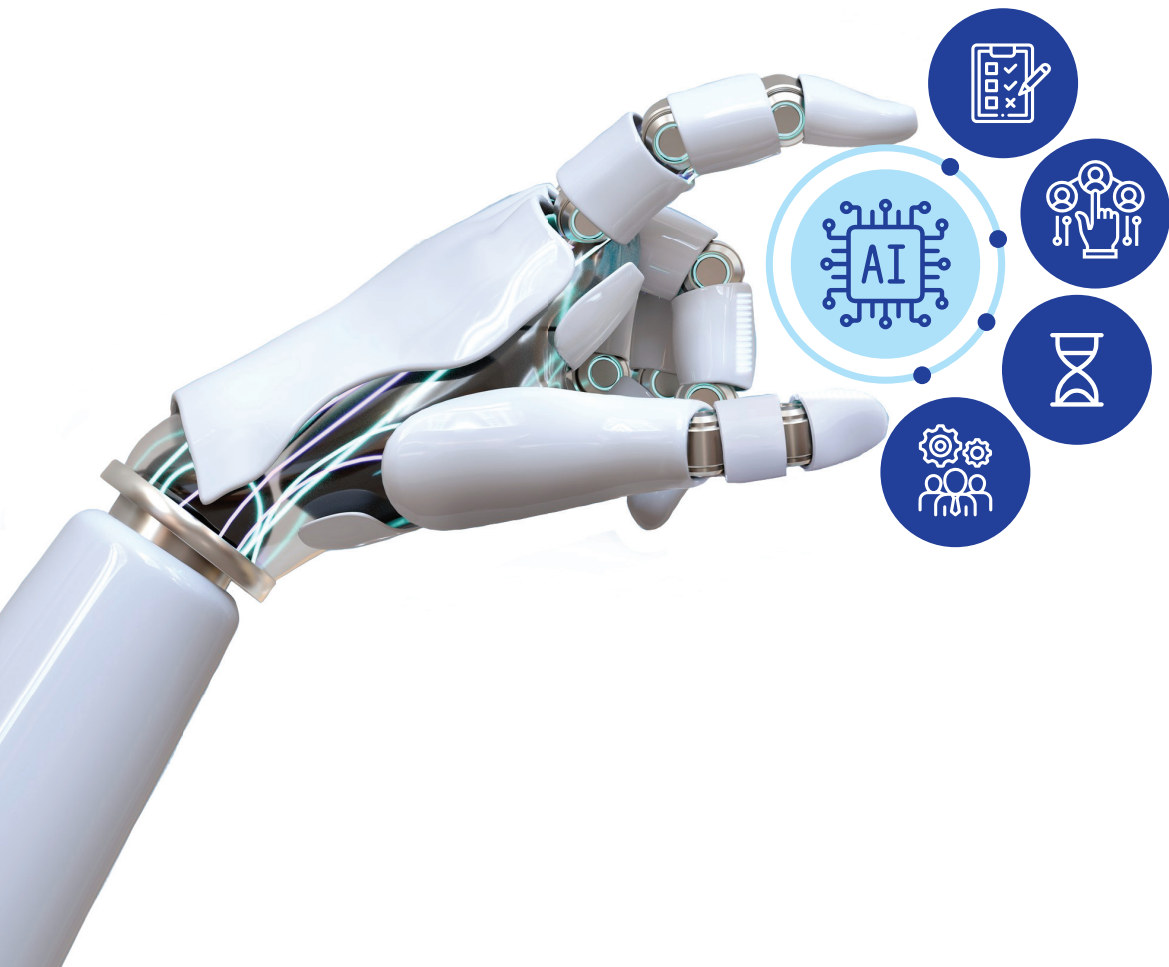


CHAPTER 4



TYPICAL METHODS AND METRICS USED IN SOFTWARE ENGINEERING





4. TYPICAL METHODS AND METRICS USED IN SOFTWARE ENGINEERING

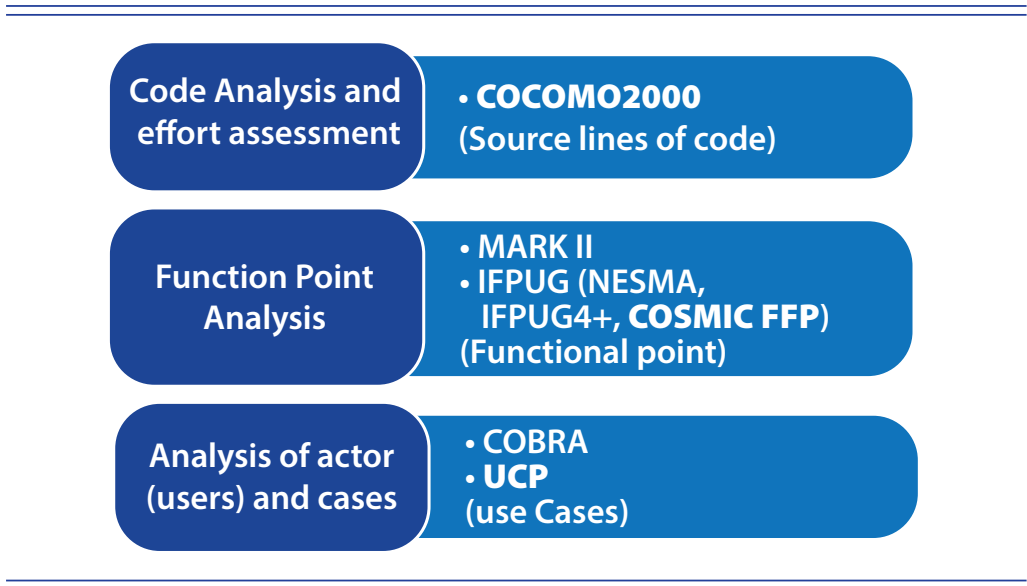
In the contemporary landscape, there exists a multitude of methods and metrics for estimating and quantifying software size, as well as the associated time and financial investments required for its development. These methods can be broadly categorized into two primary groups: parametric and non-parametric. They diverge in their approaches to evaluation, with three of the most prevalent approaches being the analysis of lines of code, functional points, and users and use cases, as depicted in Figure 7.

4.1. TRADITIONAL METHODS

The initial method centers on scrutinizing the lines of source code, yielding estimations for the effort and costs essential in the design and implementation of software projects. The preeminent model in this category is **COCOMO2000** (*Constructive Cost Model*) [168].

The second methodology hinges on functional points as a measure of software functionality, serving as an indicator of the project's overall value [169]. Initially, this approach featured two models: **IFPUG** (*International Function Point Users Group*) [170] and **Mark II** [171]. Subsequently, it diversified, with the IFPUG model giving rise to variations such as **NESMA** (*Netherlands Software Users Association for Metrics*) [172], IFPUG (version 4.1) [173], and **COSMIC FFP** (*Common Software Measurement International Consortium Full Function Point*) [174].

The third approach centers on user analysis and the utilization of use cases to achieve the most precise and efficient effort assessment. Prominent models within this category include **COBRA** (*Cost Estimation, Benchmarking, and Risk Assessment*) [175] and **UCP** (*Use Case Point Analysis*) [176].

Figure 7. Traditional methods.

4.2. TRADITIONAL ASSESSMENT

Recent studies frequently employ linear and multivariate regression techniques to predict the effort and costs associated with software project development. Linear regression is a method used to illustrate the connection between dependent and independent variables within a specific dataset [177]. With this method, it is possible to estimate efforts and costs with the minor error.

The relationship between the dependent variables Y_i , where $i=\overline{1,n}$; and the independent X_i , where $i=\overline{1,n}$ can be represented by the following formula (1):

$$Y_i = f(X_i) + r_i, \text{ where } i=\overline{1,n}; \quad (1)$$

The real value of Y_i is the sum of the estimated value of $f(X_i)$, where $i=\overline{1,n}$; and the error in estimating r_i where $i=\overline{1,n}$;

In regression models, whether it is linear regression or multivariate regression, the goal is to derive the appropriate function $f(X_i)$, where $i=\overline{1,n}$; and the error in estimating r_i where $i=\overline{1,n}$;

The duration necessary for the completion of a software project is an approximation of $f(X_i)$ value. These functions serve various purposes, primarily assessing the impact of input quantities, which are essentially model parameters. These parameters exert a substantial impact on the resulting output, which corresponds to the actual effort, functional size, and genuine effort, contingent on the chosen experimental approach.

The frequently used functions in various regression models include linear, polynomial, logarithmic, and exponential functions. Combinations of these functions can help identify the optimal model that minimizes the error value. An instance of a combined linear and exponential function approach can be observed in the COCOMO parametric method [178], [179], as demonstrated in formulas (4)-(10).

In regression models, a range of statistical metrics are commonly employed. Initially, these metrics describe the fundamental attributes of the dataset and subsequently compute the error. Key characteristics of the dataset, from which numerous insights can be gleaned, include minimum and maximum values, as well as the mean value and standard deviation. The mean value denotes the arithmetic average of the dataset and is determined using the following formula (2):

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i, \quad (2)$$

where N is the total number of the observed data.

The standard deviation measures the dispersion of all values within a given set from the mean value and is computed using the following formula (3):

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2} \quad (3)$$

Alongside basic dataset characteristics, various correlation coefficients, including Pearson's, Spearman's, and deterministic correlation coefficients (R^2), were employed in all three hybrid models.

Additionally, prediction was incorporated as another parameter to validate the efficacy of the proposed models. It was assessed using 3 criteria: on 25, 30, 50. These criteria measure the percentage of projects with a relative error value below a specified threshold.

In various studies [180], [181], [182], authors employ various statistical tests to juxtapose actual and estimated values and compute relative errors in accordance with the proposed model.

4.3. COCOMO2000 SOFTWARE ESTIMATION THROUGH SOURCE CODE ANALYSIS

This chapter offers a comprehensive exploration of the initial proposed approach, which centers around the analysis of the quantity of source lines of software code. The prevailing model utilized in this approach is COCOMO2000.

4.3.1. COCOMO2000 hybrid model main characteristics

Traditional (parametric) methods typically involve the computation of a system's size through a specially designed mathematical model. These models primarily rely on parameters derived from experimental measurements of actual values during the software project's development. Consequently, the assessment of a software project's size and complexity often hinges on the quantity of lines of source code. This approach, which quantifies effort and time requirements for project implementation, is most frequently employed. The COCOMO2000 parametric method stands out as a prominent technique within this category, utilizing the number of source lines of code as the unit of measurement for software size. With the COCOMO2000 method, it becomes feasible to estimate the necessary time and effort for project realization and execution. While this method offers a straightforward and uncomplicated means of estimating effort and costs, it exhibits several limitations over time due to advancements in technology and software tools. Discrepancies arise from variations in programming languages employed (e.g., Python, Java, R, C#) and the challenge of establishing equivalences for specific databases.

COCOMO2000 operates as an algorithmic model that establishes a connection among software metrics and project expenditures through mathematical functions. The actual effort in this context represents the project's true value, expressed in person months [PM], based on the number of lines of software code. Input values are derived from 22 parameters, categorized into two groups [183]:

The initial group encompasses five parameters, denoted as 'scale factors': **PREC, FLEX, RESL, TEAM, PMAT**;

The second group comprises seventeen parameters, designated as 'effort multipliers': **RELY, CPLX, DATA, RUSE, TIME, STOR, PVOL, ACAP, PCAP, PCON, APEX, PLEX, LTEX, TOOL, SCED, SITE, DOCU**.

Scale factors are described as follows:

1. **PREC** (*Precedentedness*) - signifies the project's goals and the necessary technology.
2. **FLEX** (*Development Flexibility*) - signifies the adjustment of development with regard to requirements, standards, and constraints.
3. **RESL** (*Architecture and Risk Resolution*) - quantifies software components, quality, and potential risks.
4. **TEAM** (*Team Cohesion*) - indicates the complexity and team members' adaptability to collaborative work.
5. **PMAT** (*Process Maturity*) - reflects the process's maturity level and oversees the advancement of software-based processes, including Capability Maturity Model Integration (CMMI).

Effort multipliers are described as follows:

1. **RELY** (*Required Software Reliability*) - signifies the reliability of executing particular software functions within a defined timeframe.
2. **DATA** (*Data Base Size*) - defines the volume of the database.
3. **CPLX** (*Product Complexity*) - characterizes the actions executed on the product, categorized into five domains: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations.
4. **RUSE** (*Required Reusability*) - describes the reusability of the chosen software product.
5. **DOCU** (*Documentation*) - showcases the documentation mandatory to accompany each part software life cycle.
6. **TIME** (*Time Constraint*) - depicts the temporal significance of software execution.
7. **STORE** (*Storage Constraint*) - illustrates data storage and its constraints.

8. **PVOL** (*Platform Volatility*) - conveys the influence of modifications in software and hardware architecture on the system.
9. **ACAP** (*Analyst Capability*) - quantifies the capability of analysts.
10. **PCAP** (*Programmer Capability*) - assesses the programmer's competence.
11. **AEXP** (*Applications Experience*) - evaluates the programmer's experience with a specific application.
12. **PEXP** (*Platform Experience*) - quantifies experience with a specific platform.
13. **LTEX** (*Language and Tool Experience*) - evaluates experience with diverse programming languages and tools.
14. **PCON** (*Personnel Continuity*) - represents the complexity and efficiency of the team.
15. **TOOL** (*Use of Software Tools*) - describes a distinct measure of the employed software tools.
16. **SITE** (*Multisite development*) - represents the geographical distance of team members.
17. **SCED** (*Required Development Schedule*) - showcases the required schedule of tasks on the project.

The COCOMO2000 equation, from which input values and project size are derived [183], is depicted in Figure 8 and can be expressed through the following equations, as shown in (4)-(10):

$$Effort = A \times [SIZE]^E \times \prod_{i=1}^{17} EM_i \quad (4)$$

$$E = B + 0.01 \times \sum_{j=1}^5 SF_j, A = 2.94, B = 0.91 \quad (5)$$

$$Effort [PM] = 2.94 \times [SIZE]^E \times PEM_i \quad (6)$$

$$PEM_i = \prod_{i=1}^{17} EM_i \quad (7)$$

$$Time = C \times (Effort)^F \quad (8)$$

$$F = D + 0.2 \times 0.01 \times \sum_{j=1}^5 SF_j, C = 3.67, D = 0.28 \quad (9)$$

$$People = Effort / Time \quad (10)$$

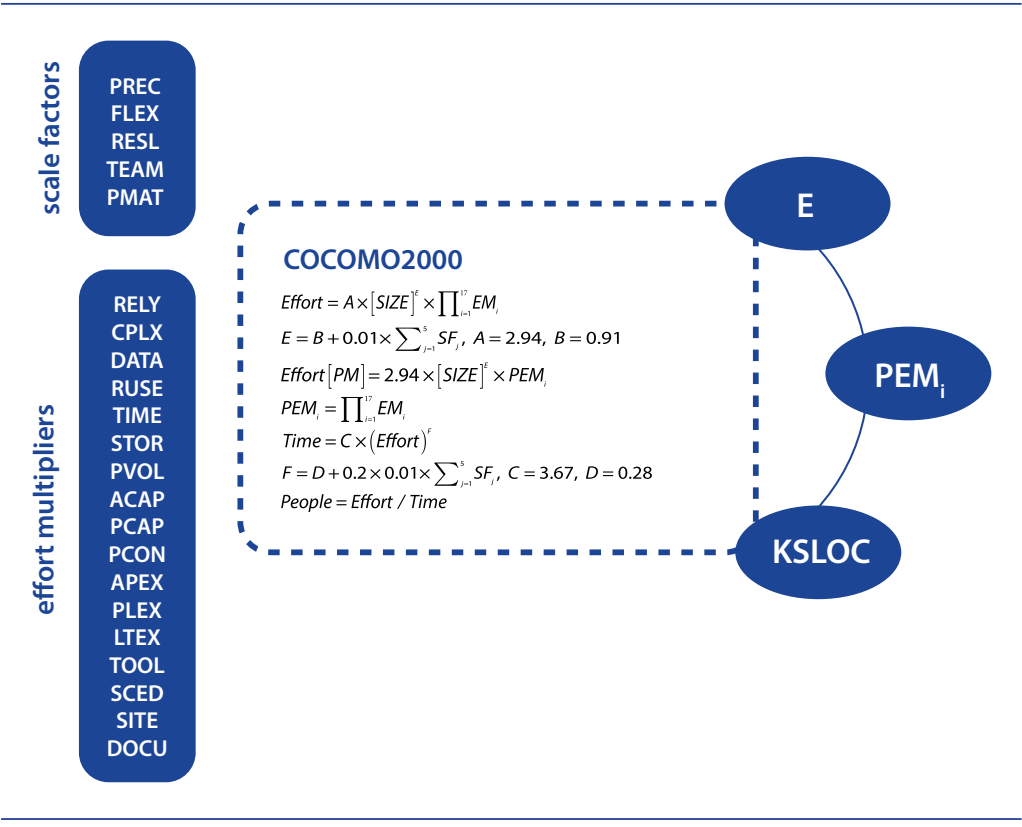
A and B serve as foundational constants for adjustment; KSLOC (1000 of source lines of software code) represents the software project's size; SF_i stands for 5 scale factors, while EM_i represents 17 effort multipliers. As previously noted, the conducted experiments utilize the COCOMO2000 Post Architecture model. This mathematical model, as mentioned earlier, amalgamates factors and effort multipliers to compute the required person-months $[PM]$ for executing a specific software project.

All included factors are rated on a scale that ranges from 'very low' to 'extremely high.' A sample illustrating various parameters assigned within the COCOMO2000 model can be found in Table 2.

Table 2. Example of weighting coefficients assigned to COCOMO2000 parameters.

Parameter	Symbol	Wery Low	Low	Neutral High	High	Wery High	Extreme High
PREC	SF_1	0.05	0.04	0.03	0.02	0.01	0
FLEX	SF_2	0.05	0.04	0.03	0.02	0.01	0
RESL	SF_3	0.05	0.04	0.03	0.02	0.01	0
DOCU	EM_5	0.85	0.93	1	1.08	1.17	
PCAP	EM_{10}	1.37	1.16	1	0.87	0.74	
PCON	EM_{11}	1.26	1.11	1	0.91	0.83	
AEXP	EM_{12}	1.23	1.1	1	0.88	0.8	

Figure 8. COCOMO2000 formula, based on input values and project’s size.



4.3.2. Prior research related to COCOMO2000 methodology

Many software companies that offer a wide range of industrial tools, services, and products consistently grapple with meeting the stringent demands of their software clients and customers. To adhere to these high standards and develop top-quality software, various types of research [138, 139], [140], and [141] employ diverse software models and methods for estimating efforts and costs. Previous effort models such as SLIM (Software Lifecycle Management) [184], SEER-SEM (System Evaluation and Resource Estimation Software Model) [63], COCOMO81 (Constructive Cost Model) [186], and COCOMO2000 [187, 188] have been employed in various software development projects. The COCOMO2000 model assesses effort and cost based on the number of source lines of code, incorporating scale factors, cost factors, and software size. Notably, the most promising results have been achieved through testing the Early Design and Post Architecture versions of the COCOMO2000 model. In the first approach

outlined in this book (3.1), a more detailed iteration of the Post Architecture model was utilized for training and testing in the conducted experiments. The COCOMO2000 dataset, encompassing 161 projects, was also employed.

The researchers in the study [66] employed neural network techniques with backpropagation on the COCOMO2000 dataset, consisting of 60 and 93 software cost estimation projects. They also utilized the Dolphin algorithm for the software evaluation approach. As a result, they achieved a reduced MRE value in their findings. However, this book demonstrates that it is possible to further reduce the MRE value and achieve superior results by using a simplified ANN architecture, resulting in fewer iterations and shorter evaluation times. Recent research [189], introduces the use of a combined technique that merges the COCOMO2000 mathematical approach with the neural network (distribution-free method) within a unique structure or model for effort and cost estimation. This approach has been shown to yield estimates of software costs that are more closely aligned with actual costs, resulting in a minimized MRE value. In a separate investigation [68], the authors empirically demonstrated the improvements gained by applying ANN to the COCOMO2000 model, although some relatively high relative error values were observed in the results. For a more precise and reliable software cost estimation, researchers in [191] present various algorithms and neural networks, comparing their performance. Additionally, they found that the hidden neural network activation functions used in their study performed less effectively than the showcased sigmoid function. In another study [192], a two-hidden-layer network model was employed to further minimize the MRE value. Multi-layer neural network-based non-parametric models for estimating software project costs and efforts [193, 194] achieved slightly improved results, resulting in a lower MRE value.

Over 33,000 experiments in which various ANNs processed data using actual project values are presented in the paper [195]. Different metrics were used to evaluate the data from these experiments that were gathered from various sources. The authors of the studies [196, 197, 198, and 199] process the dataset using a variety of ANN architectures. It has been demonstrated that the heterogeneity of the data sets varies the outcomes in comparison to earlier studies. It can be deduced that no single method would guarantee adequate and dependable results.

This book will employ distinct data sources from prior research and compare the results using the same novel methodology. The stability, effectiveness, and dependability of demonstrated method are all facilitated by this approach. In the experimental section, a range of data sets, including COCOMO81, COCOMO2000, NASA60, NASA93, Kemerer15, and Desharnais datasets, were utilized to validate the application of the proposed methodology across various project values.

Four distinct artificial neural network architectures were developed using Taguchi Latin Square extracted plan, alongside diverse datasets. To mitigate the diverse dataset structures, a clustering technique was employed to categorize the data into 3 groups, thereby improving the chosen model within the framework of COCOMO2000. The fuzzification technique was utilized for further homogenization of the input values.

In contrast to other studies, this book adopts a combination of traditional and distribution-free models and methods, resulting in the most favorable experimental outcomes for cost and effort estimation. A novel approach involves employing 'stopping criteria' that leverage the machine learning rate as a basis for halting iterations. This method allows for ongoing monitoring of the convergence speed for each proposed architecture, and the experiment concludes once predetermined criteria are satisfied.

The introduction of Gradient Descent as a new machine learning speed criterion allows for precise tracking of the experiment's progress and velocity. This method enables the identification of the best model with the fewest iterations, leading to reduced execution times and lower MMRE values. Ultimately, this criterion enables the selection of the most accurate, consistent, and dependable model among the proposed ANN architectures:

The pivotal determinations shaping the novel, hybrid COCOMO2000 model encompass the following:

- Evaluating the convergence rate of four distinct proposed ANN architectures.
- Assessing the achieved MMRE values.
- Segregating all utilized data sets into clusters.
- Investigating various activation functions for the hidden ANN layer.
- Identifying the most effective techniques for encoding and decoding input values, such as the fuzzification method.
- Determining the minimum number of necessary experiments.
- Conducting tests and validations on various alternate data sets.

4.4. SOFTWARE ESTIMATION THROUGH FUNCTION POINTS

In this chapter, we will delve into the function points analysis approach. The model predominantly applied in recent times within this approach is COSMIC FFP.

4.4.1. COSMIC FFP hybrid model main characteristics

To address the limitations of the prior method, which relied on counting lines of source software code - SLOC to gauge system size, a fresh approach rooted in function point analysis (FPA) has surfaced.

Function point analysis quantifies system functionality using values represented as functional points. While different real projects and systems can possess similar functionalities, they can still vary, utilizing diverse technologies and programming languages, resulting in different lines of source code. This analysis has prompted the creation of numerous models based on this method to achieve efficient and precise estimates of functional size.

This book will introduce the latest and most frequently employed methods within the function points family, specifically COSMIC FFP (*Common Software Measurement International Consortium Full Function Point*). This approach estimates effort and cost based on the functional size of software projects, relying on 14 parameters that are condensed into 4 input values through suitable transformations: 1. *Entry*, 2. *Exit*, 3. *Read*, and 4. *Write* (refer to Figure 9). Evaluating these 14 system parameters ensures the reliable measurement of functional size:

- **Data communication** - information transfer and exchange data between users and the system;
- **Distributed data processing** - evaluating the distribution of data processing;
- **Performance** - assessing the desired performance of the system;
- **Heavily used configuration** - software and hardware platform;
- **Transaction rate** - the rate at which transactions are executed within the system;
- **On-Line data entry** - the proportion of data entered manually;
- **End-user efficiency** - effective utilization of software by users;
- **On-Line update** - the quantity of data subject to updates;
- **Complex processing** - the intricacy of data processing within the system;
- **Reusability** - reuse of code;

- **Installation ease;**
- **Operational ease** - user-friendliness;
- **Multiple sites** - geographical dispersion of the team;
- **Facilitate change** [142].

In contrast to many other approaches that utilize 5 input quantities, the COSMIC FFP method relies on just 4 input variables. The 4 input parameters featured in our proposed model, which adopts the COSMIC FFP method as its foundation, serve as the input values for 2 distinct ANN architectures. Each of these ANN architectures is developed using the Taguchi method, and the input quantities are as follows:

1. **Entry** – a variable that represents messages that users send to the system or messages that one system sends to another to transfer the necessary data and information. Messages do not have to be system entries.
2. **Exit** – a variable that represents messages the system or module returns as a response to the user in the form of data that can be read from files. Messages can be in the form of arithmetic and logical operations.
3. **Read** – a variable that represents messages that update data in the system. These are usually different files, tables, and other data.
4. **Write** – a variable that represents messages that send data from the system, which can be in the form of tables, and files and do not represent formulas or operations.

In this hybrid model, the functional size of the system is characterized by the cumulative count of all transferred messages used. The system can be conceptualized as a 4-dimensional vector space, where each dimension represents the total count of transferred messages falling into four categories: data input, data output, data file writes, and data file reads. This approach is particularly noteworthy because it allows for the detection and assessment on the effect of even minor alterations on the functional size. One of its key advantages lies in its technology independence and the absence of an upper limit on the functional size's value. In other words, regardless of the volume of messages in the system, there is no saturation threshold and the complexity of functionality can grow indefinitely [200], [201].

Functional size is calculated using a four-dimensional vector denoted as FFP, as outlined in formula (11):

$$FFP=(E,X,R,W) \quad (11)$$

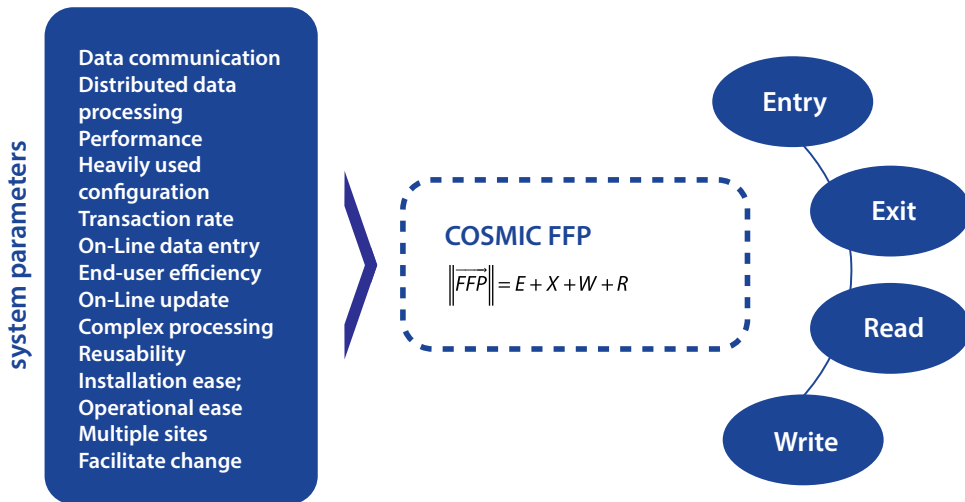
FFP represents the total number of messages in the entire observed system and is calculated as the norm of the vector in the formula Figure 9, (12):

$$\|\overrightarrow{FFP}\| = E + X + W + R \quad (12)$$

where: E - Entry, X - Exit, W - Write, and R - Read.

Since even little modifications can have an immense effect on the size of a system, the COSMIC FFP approach is modular and can be applied to many different kinds of systems, and there are no boundaries on the functionality size within the option chosen dataset.

Figure 9. Determining the overall count of messages within the system, represented as the vector's norm.



4.4.2. Prior research related to Function Points methodology

Experts Kemerer (1987) [202] and Albrecht and Gafni (1983) [169] emitted groundbreaking studies on the Functional Point Analysis (FPA) method. Primarily in the context of IBM projects, Albrecht's research concentrated on the relationship between project size and the time needed for its development. Albrecht developed the idea of "functional points" - a word that later came to be associated with the FPA strategy - to gauge system functionality. This strategy was endorsed by the International Function Point User Group (IFPUG) in 1994, setting the groundwork for future development of the FPA strategy. The Netherlands Software Metrics Association's NESMA model [172] and the United Kingdom's MARK II model [204] are two further iterations of FPA models that continued to develop. Particularly noteworthy are the COSMIC Full Function Point (COSMIC FFP) technique and IFPUG version 4.1. Key functional points in this approach include Entry, Exit, Read, and Write.

Numerous experiments conducted by authors like [206], [207], and [208] have consistently shown that COSMIC FFP outperforms other methods in terms of effectiveness and accuracy, even when efforts were made to incorporate new metrics like user stories within agile methodologies.

The paper [209] conducted a comparative analysis of the effects of using COSMIC FFP and IFPUG for effort prediction, with COSMIC FFP yielding more accurate effort predictions for software development projects.

An approximation approach using a convolutional neural network and a word embedding model trained on Wikipedia + Gigaword, together with functional point measurements, was developed in a different study [210] to provide precise predictions. A different study [211] used COSMIC functional size along with a number of algorithms based on machine learning to evaluate software product development.

Authors in [212] and [213] proposed the Neuro-Fuzzy Function Point Calibration Model (NFFPCM) and a method to calibrate functional point complexity (FP) weights. Their research demonstrated improved accuracy in mean (MRE) values when estimating software effort, particularly when using the ISBSG repository [144]. This book, on the other hand, employs the entire ISBSG dataset without any calibration or adjustment. Additionally, it presents methods involving fuzzification, a reduction in the number of required iterations for various data set combinations from the ISBSG repository, and a technique for clustering input values across the entire dataset. Two additional data sets are used for validation purposes.

The pivotal determinations shaping the novel, hybrid COSMIC FFP model encompass the following:

- Investigating how four input values affect the MMRE value.
- Conducting a comparative analysis of two distinct artificial neural network architectures and the resulting outcomes.
- Segmenting the utilized dataset into clusters based on functional size and the unique characteristics of each project within the ISBSG repository.
- Identifying the most effective techniques for encoding and decoding input values, including the fuzzification method.
- Determining the minimum number of experiments necessary.
- Evaluating and validating the results on alternative datasets.

4.5. SOFTWARE ESTIMATION THROUGH USE CASE POINTS

This chapter introduces a method centered around use case point analysis, specifically highlighting the UCP model as an exemplary choice due to its exceptional performance, boasting a minimal error rate of less than 10%.

4.5.1. Main characteristics of the UCP model

The UCP (*Use Case Point Analysis*) approach employs an estimation of the software project's actual size by examining user cases and use cases. This method delves into various aspects, including system actors and various scenarios, to gauge the effort needed for a specific software project. It relies on 21 parameters, categorized into two groups: 13 relate to technical characteristics of the system, while the remaining 8 pertain to environmental factors.

The technical characteristics of the system under evaluation encompass:

- System distribution
- System response time
- System efficiency
- Complexity of internal processes
- Potential for code reuse
- Ease of final product installation

- User-friendliness of the final product
- Compatibility with other platforms
- System maintenance
- Parallel processing capabilities
- System security requirements
- Integration with external systems
- End-user training

The environmental factors considered for evaluation include:

- Adherence to the chosen development process
- Familiarity with the applications to be used
- Proficiency in the object-oriented technologies to be employed
- Competence of the chief analyst
- Team motivation
- Stability requirements of the system
- Working hours of team members
- Complexity of the programming language to be utilized

The UCP method utilizes both system users and use cases to determine the actual size. System users are categorized into three groups: *simple* (assigned a weight factor of 1 based on their interaction with the system), *medium* (assigned a weight factor of 2 considering internal/external communication), and *complex* (assigned a weight factor of 3 based on the complexity of interactions).

Similarly, there are 3 categories of use cases determined by the number of executed transactions (involving both users and system messaging): *simple* (less than 3 transactions with a weighting factor of 5), *medium* (4 to 7 transactions with a weighting factor of 10), and *complex* (more than 8 transactions with a weighting factor of 15).

The system size is then defined using a six-dimensional vector, with each element representing the complexity of the previously mentioned user categories and use case categories within the system.

Based on G. Karner equations the following evaluated valued could be calculated [214]:

UAW (*Unadjusted Actors Weight*) - this input parameter serves as a functional point that assesses the complexity level of system users, which may include simple system operators or other external systems. Each user is categorized based on their complexity level, which can be classified as: Simple, Intermediate, or Complex, as shown in formulas (13)-(16).

$$\text{SimpleA} = \sum(\text{SimpleActor}) * \text{SimpleWeight}, \text{ where } \text{SimpleWeight} = 1; \quad (13)$$

$$\text{AverageA} = \sum(\text{AverageActor}) * \text{AverageWeight}, \text{ where } \text{AverageWeight} = 2; \quad (14)$$

$$\text{ComplexA} = \sum(\text{ComplexActor}) * \text{ComplexWeight}, \text{ where } \text{ComplexWeight} = 3; \quad (15)$$

$$\text{UAW} = \text{SimpleA} + \text{AverageA} + \text{ComplexA} \quad (16)$$

UUCW (*Unadjusted Use Case Weight*) - this input value is a functional point that can determine the level of complexity of use cases. Each use case is ranked according to its level of complexity and can be: *Simple*, *Average* and *Complex*, see formulas (17)-(20).

$$\text{SimpleUUCW} = \sum(\text{SimpleUCW}) * \text{SimpleWeight}, \text{ where } \text{SimpleWeight} = 5, \\ (\text{transactions} \leq 3, \text{ analysis classes} \leq 5) \quad (17)$$

$$\text{AverageUUCW} = \sum(\text{AverageUCW}) * \text{AverageWeight}, \text{ where } \text{AverageWeight} = 10, \\ (4 \leq \text{transactions} \leq 7, 5 \leq \text{analysis classes} \leq 10) \quad (18)$$

$$\text{ComplexUUCW} = \sum(\text{ComplexUCW}) * \text{ComplexWeight}, \text{ where } \text{ComplexWeight} = 15; \\ (\text{transactions} > 7, \text{ analysis classes} > 10) \quad (19)$$

$$\text{UUCW} = \text{SimpleUUCW} + \text{AverageUUCW} + \text{ComplexUUCW} \quad (20)$$

By using the two sizes listed above, the size of the system to be developed is obtained.

UUCP (*Unadjusted Use Case Points*) is determined by following equation (21):

$$\text{UUCP} = \text{UUCW} + \text{UAW} \quad (21)$$

TCF (*Technical Complexity Factor*) is an estimate of the technical complexity of the system and can be described by the following formulas (22), (23):

$$\text{TCF} = 0.6 + (0.01 * \text{FactorT}) \quad (22)$$

$$\text{FactorT} = \sum \text{Weight} * \text{AssignedValue}, \text{ where } \text{AssignedValue} \text{ is from } 0 \text{ to } 5 \text{ and} \\ \text{represents a technical factor of the estimated process, see Table 3.} \quad (23)$$

ECF (*Environmental Complexity Factor*) is one of the factors affecting the size of the project expressed in Use Case points. It is calculated according to the following formulas (24), (25):

$$\mathbf{ECF} = 1.4 + (-0.03 * \mathbf{FactorE}) \quad (24)$$

FactorE = $\sum \mathbf{Weight} * \mathbf{AssignedValue}$, where *AssignedValue* is from 0 to 5 and represents a environmental factor of the estimated process, see Table 4. (25)

AUCP (*Adjusted Use Case Point*) is the final size of the system expressed in Use Case points and is calculated as follows (26):

$$\mathbf{AUCP} = \mathbf{UUCP} * \mathbf{TCF} * \mathbf{ECF} \quad (26)$$

Representation of real effort by UCP approach as a six-dimensional vector, where its value is calculated as the norm of the vector as follows, Figure 10, see formulas (27), (28):

$$\mathbf{UCP} = (\mathbf{UAW}, \mathbf{UUCW}, \mathbf{UUCP}, \mathbf{TCF}, \mathbf{ECF}, \mathbf{AUCP}) \quad (27)$$

$$\|\overrightarrow{UCP}\| = \mathbf{UAW} + \mathbf{UUCW} + \mathbf{UUCP} + \mathbf{TCF} + \mathbf{ECF} + \mathbf{AUCP} \quad (28)$$

Representation of real effort by UCP approach as a four-dimensional vector, where its value is calculated as the norm of the vector as follows, Figure 10, see formulas (29) (30):

$$\mathbf{UCP} = (\mathbf{UAW}, \mathbf{UUCW}, \mathbf{TCF}, \mathbf{ECF}) \quad (29)$$

$$\|\overrightarrow{UCP}\| = \mathbf{UAW} + \mathbf{UUCW} + \mathbf{TCF} + \mathbf{ECF} \quad (30)$$

where **UUCP** = $\mathbf{UAW} + \mathbf{UUCW}$, and **AUCP** = $\mathbf{UUCP} * \mathbf{TCF} * \mathbf{ECF}$.

Table 3. Thirteen technical complexity factors.

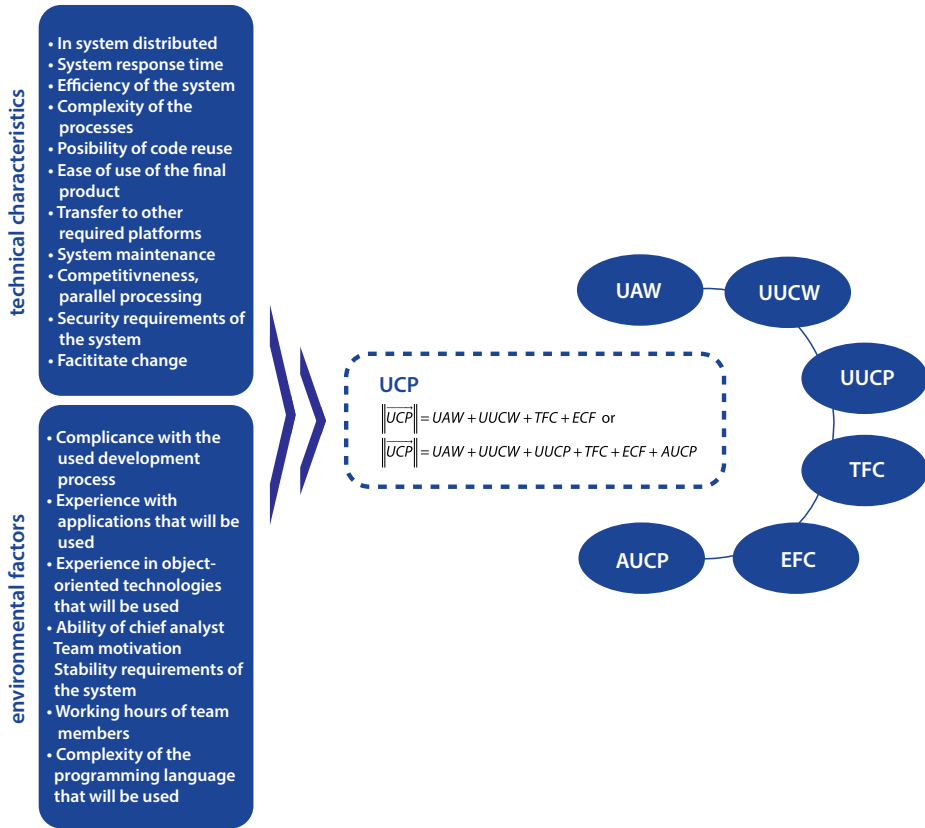
Factor	Description	Weight	Assigned Value	Weight & Assigned Value
T ₁	Distributed system	2.0	5	10
T ₂	Response time/performance objectives	1.0	5	5
T ₃	End-user efficiency	1.0	3	3
T ₄	Internal processing complexity	1.0	2	2
T ₅	Code reusability	1.0	3	3
T ₆	Easy to install	0.5	1	0.5
T ₇	Easy to use	0.5	5	2.5
T ₈	Portability to other platforms	2.0	2	4
T ₉	System maintenance	1.0	2	2
T ₁₀	Concurrent/parallel processing	1.0	3	3
T ₁₁	Security festures	1.0	5	5
T ₁₂	Access for third parties	1.0	1	1
T ₁₃	End-user traning	1.0	1	1
Total (TF):				42

Table 4. Eight environmental factors.

Factor	Description	Weight	Assigned Value	Weight & Assigned Value
E_1	Familiarity with development process used	1.5	3	4.5
E_2	Application experience	0.5	3	1.5
E_3	Object-oriented experience of team	1.0	3	2
E_4	Lead analyst capability	0.5	5	2.5
E_5	Motivation of the team	1.0	2	2
E_6	Stability of requirements	2.0	1	2
E_7	Part-time staff	-1.0	0	0
E_8	Difficult programming language	-1.0	4	-4
Total (EF):				10.5

Real Effort, which is calculated as the UCP vector's norm in either scenario, stands for the actual functional size or number of use case points. Although it is not standardized inside ISO standards like the previous two are, this approach is currently the most used one for evaluating effort [178].

Figure 10. Computing the Real Effort Utilizing a Six-Dimensional or Four-Dimensional Vector.



4.5.2. Prior research related to UCP methodology

The most recent and widely adopted method for evaluating the time and financial resources mandatory for software product development is the UCP method. Its notable advantage lies in achieving relative error values ranging from 20% to 35%. However, this approach can yield even better results, with an error rate as low as about 10%. Many researchers have embraced this method, incorporating it into their studies alongside other parametric and artificial intelligence models.

One study used the UCP approach to determine the size and work required for mobile applications, concentrating in particular on Android-based mobile apps. A modified version of the UCP method was also introduced in this context.

In addition, researchers suggested a framework to encourage reuse in software application development using UCP-based methodologies. The framework demonstrated its ability to fulfill five quality requirements and proved suitable for application in the early stages of software development.

A comprehensive systematic review investigated the use of Use Case Points (UCP) and expert judgment-based effort estimation techniques. This review presented the results of 4 distinct models, including the UCP method and Neuro-Fuzzy logic. Notably, the Neuro-fuzzy logic model exhibited superior early fitting accuracy when utilizing revised use case points and a modified environment. In another study, efforts were made to estimate project effort for 7 real-world software development projects, incorporating statistical analysis. Additionally, this study compared an enhanced Human-centric (HCI) model with the conventional Use Case Points approach. The authors also proposed improvements to the original iUCP effort estimation formula.

The pivotal determinations shaping the novel, hybrid UCP model encompass the following:

- Assessing the impact of two linearly dependent input values, namely UUCP and AUCP, on the MMRE value.
- Conducting a comparative analysis of two distinct artificial neural network architectures and analyzing the results obtained.
- Dividing the utilized dataset into a 70:30 scale, with 70% of the selected projects in the dataset allocated for the training process and the remaining 30% for the testing process.
- Identifying the most efficient methods for encoding and decoding input values, including the fuzzification method.
- Determining the minimum number of experiments required for the study.
- Evaluating the proposed models through testing and validation using diverse datasets.

4.6. ADVANTAGES OF USING ANNS IN SOFTWARE ESTIMATION

With the rapid advancement of computer technologies, particularly in the 1990s, Artificial Neural Networks (ANNs) have gained significant prominence. Each ANN architecture embodies a system comprising interconnected components known as nodes or artificial neurons. These neurons are linked through specific connections called synapses, which facilitate the transmission of data. The arrangement of neurons and connections within a network results in a distinctive ANN architecture, characterized by its number of nodes and layers.

The foremost layer is invariably referred to as the input layer, while the ultimate layer is designated as the output layer. Any intermediary layers are denoted as hidden layers. The input layer can accommodate various input sizes, and pertinent data is introduced via input variables. This information is then transmitted to the hidden layer, where it is processed in line with predetermined rules before moving through synapses and arriving at the output layer. The weight factor or coefficient [227], [228] examines the intensity of the connections between neurons.

An artificial intelligence tool offers the capability to explore:

- various combinations of input sizes,
- diverse quantities of nodes within the network,
- a range of hidden layer configurations, and different output sizes, all tailored to the specific problem at hand.

Initially, the artificial neural network must undergo a training process to prepare it for subsequent utilization. This training procedure entails several key steps, including the determination of input values, the formulation of rules or functions employed during training, and the manner in which output values are derived. Each ANN architecture is structured in a specific configuration, defined by the number of input variables, the quantity of weighting coefficients applied to connections between nodes, and the count of output values. This configuration governs the interconnections between nodes and typically consists of an input layer, 1-3 hidden layers, and 1 output layer.

The neuron's activation function acts a vital part within any neural network and together with the weight coefficients serves as the basis for the first training phase. During the initial training phase, errors tend to be more pronounced. However, with each subsequent iteration, adjustments are made to the weighting coefficients, progressively diminishing errors and converging toward accurate estimated values [229], [230].

4.6.1. ANNs most common classification

The various subsections and topologies of artificial neural networks appear when a broad spectrum of conceivable variations of input quantities, nodes, connections, and output values are taken into account. They can be divided into layered, fully linked, and cellular neural networks depending on how they are connected. Additionally, we can categorize them as feedforward networks - where signal transmission occurs in a single direction - and feedback networks - which are recurrent or return networks—based on the direction of data processing and signal transmission.

Artificial neural networks can be trained in either a supervised or unsupervised manner [231]. Unsupervised networks have no need for labeled input and output data, but supervised neural networks depend on it.

4.6.2. ANNs in other fields

In the foreseeable future, the utilization of this artificial intelligence tool is poised to increase significantly. Its suitability for modeling complex systems characterized by numerous conditional, undetermined, or unknown factors is particularly noteworthy. An illustrative example is the estimation of time and costs for software project development, a topic explored in this book. In this context, traditional analytical methods can be supplanted by models employing artificial neural networks.

The advent of various artificial intelligence tools, particularly artificial neural networks (ANNs), has paved the way for solving a diverse array of problems that were previously impervious to classical computational techniques. Today, ANNs find applications across myriad domains, including image processing, speech recognition, object recognition, and sensory signal analysis. They are also instrumental in natural language processing, text recognition (both printed and handwritten), and numerous other areas. ANNs have made significant contributions to fields like medical diagnostics, military and law enforcement applications, and telecommunications [232], making them pivotal in the realms of informatics and information and communication technologies.

In the models presented across the three distinct approaches, diverse Taguchi orthogonal arrays were employed to streamline optimization challenges. These models belong to the Multilayer Feed Forward Neural Network (MFFN) class, which plays a pivotal role in addressing various scientific problems. When

constructing high-performance MFFNs, there exists no universally accepted theory for calculating ideal parameter configurations. Consequently, even minor adjustments to parameters can yield substantial variations in network behavior. In a study [137], an analysis of neural network design factors and objective functions is presented, with recommendations favoring architectures featuring one or two hidden layers. The Kolmogorov-Smirnov test was used to imply that doubling the amount of neurons in the hidden layer would be equivalent to an increase of one input neuron, i.e., $2 \times N_{\text{Input}} + 1$. The outcomes of 240 experiments conducted in [137] underline the need for specific neural network configurations to achieve convergence and accuracy when subjected to a set of test data. Although the number of hidden layers (one or two) has minimal impact on the observed error accuracy, it significantly influences convergence speed. Given these results, a trial-and-error approach is commonly adopted since most existing theoretical generalizations fall short in elucidating neural network performance in practical scenarios.

The framework used in the suggested models adheres to the most intuitive ANN topologies, including networks without hidden layers, more intricate structures with one hidden layer, and those with two hidden layers. The size of the observation set used for ANN training must not be greater than the number of projects in the dataset, it is crucial to remember.

4.7. TAGUCHI'S METHOD VS. FULL FACTORIAL DESIGN

A reliable design process was introduced by Dr. Genichi Taguchi in 1986. The main goal of this method is to identify the interactions between the control variables and noise variables so that the control factors can be changed to lessen the impact of the noise variables. It stands out as one of the best methods for accelerating experimentation, improving quality, and lowering expenses. Using orthogonal vector plans to obtain accurate data regarding project parameters even with a constrained number of experiments is a crucial part of a robust design strategy [233, 234].

In the realm of software planning and implementation, a robust design approach entails adhering to predefined criteria. Taguchi's methodology is capable of achieving this goal, irrespective of the diverse influences stemming from external or supplementary requirements. Taguchi harnessed orthogonal arrays to optimize product design robustness. His vector plans are founded on a unique set of Latin Squares. By uncovering and applying orthogonal vector

plans, the number of critical parameters essential for a project's successful development is reduced. This streamlined approach allows for quicker estimation of a project's efforts and costs since impact parameters are not duplicated. The cornerstone of this design methodology is the "factorial experiment," which examines all possible combinations of parameter values in a systematic manner. Crucially, the implementation of Taguchi's vector plans is instrumental in conducting these experiment.

The resilient experiment design of Taguchi in each orthogonal vector plan is affected by the number of parameters, the weighting factors (coefficients), and the number of levels for each parameter. That is, the number of times each level needs to be tested for each parameter. $N = L^P$ is the number of experiments necessary for a comprehensive factorial analysis (for instance: $N = 3^{13} = 1\,594\,323$ experiments are required when the full factorial plan (FFP) calls for the use of three levels with 13 parameters. Only $3^3 = 27$ experiments are required when employing the Taguchi vector plan, which has 13 parameters (weight coefficients) on three levels. The number of experiments is reduced by 99.99830649% using the Taguchi's optimization method of robust design ($0.9999830649 = 1 - (27/1\,594\,323)$)).

Taguchi's method considers a chosen cluster of combinations, ensuring that no parameters are repeated. Furthermore, these parameters can be evaluated in isolation from one another. The Taguchi orthogonal vector plan examines each level of a specific parameter, as depicted in Figure 11, where each of the other $(P-1)$ parameters at their respective L levels is tested at least once.

Figure 11. Taguchi's Orthogonal Vector plan vs. Full Factorial Design (FFD).

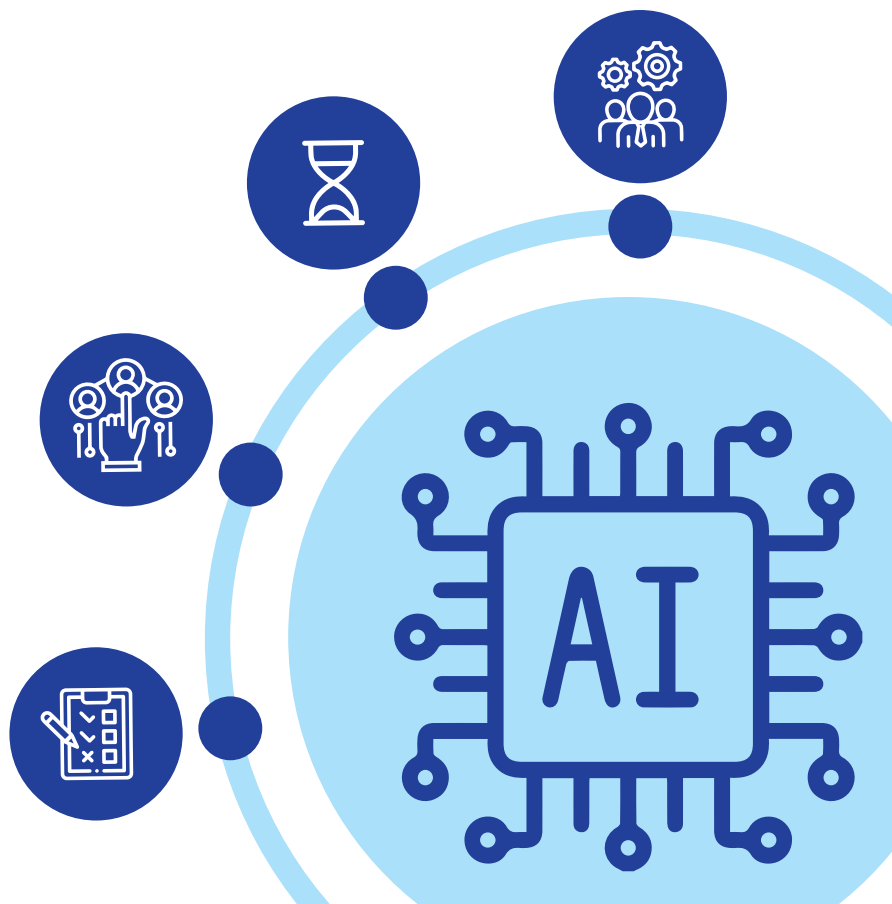
Taguchi Design	Experiment number	Full Factorial Design	Experiment number
$L_4(2^3)$	4	2^3	8
$L_8(2^7)$	8	2^7	128
$L_{12}(2^{11})$	12	2^{11}	2048
$L_{16}(2^{15})$	16	2^{15}	32768
$L_9(3^4)$	9	3^4	81
$L_{18}(3^7)$	18	3^7	2187

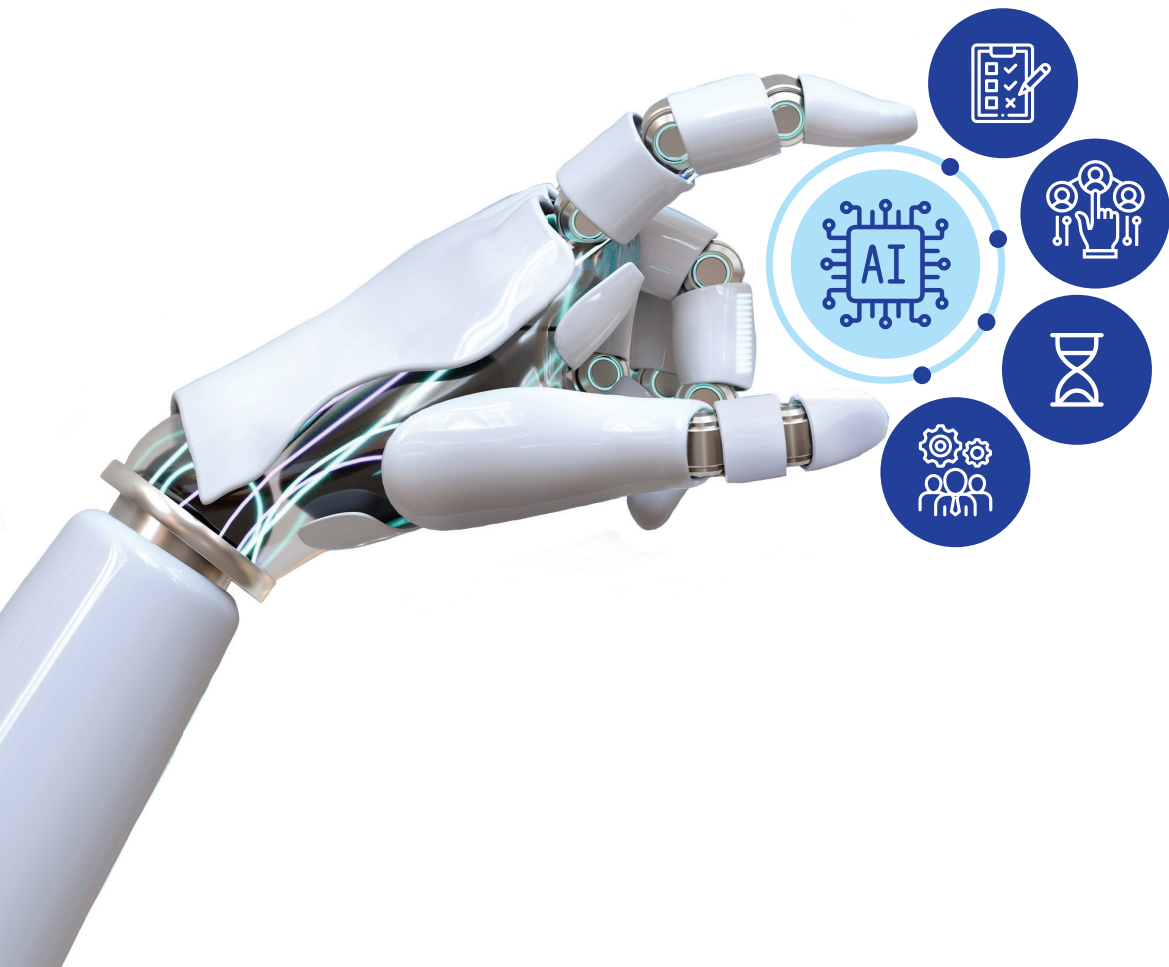


CHAPTER 5



NOVEL, HYBRID MODELS FOR DISTINCT APPROACHES





5. NOVEL, HYBRID MODELS FOR DISTINCT APPROACHES

During the software creation, development, and implementation processes, artificial intelligence tools, particularly artificial neural networks (ANN), are frequently employed in numerous research endeavors for effort and cost assessment. ANN is a powerful technique for information processing and can make substantial contributions to the development of new software evaluation models [236], [237]. ANNs can learn from diverse datasets, producing precise and dependable results while also providing the capability to assess the risk of potential errors.

Three layers make up each ANN structure: input, hidden, and output. Various designs integrating three, four, and six input values will be used in conjunction with increasing amounts of weight factors (coefficients) to create each of these models. A Taguchi orthogonal vector plan is used to create each architecture in each of the newly proposed models, which incorporate input values and weight parameters [238], [239]. Each model that is put forth aims to find the simplest network or scheme that also meets other requirements. A different amount of nodes that can be combined in the hidden layer according to the orthogonal plan are present. Each ANN in the suggested architecture has its error measured by MRE. MRE calculates the difference between an actual project's actual effort and its estimated actual effort. This value is sensitive to individual forecasts and takes into account the numerical value of each observation in the data distribution. The absolute difference between the actual value and the estimated value is represented by the MAE metric, which evaluates accuracy.

In addition to MRE and MAE values, MMRE (Mean Magnitude Relative Error), which measures the mean value of MRE, is also calculated. To find the ANN that produces the lowest MMRE value after the initial phase of the experiment (after ANN), there are 27 different combinations ("ANN candidates") within the ANN-L27 architecture.

What is the rationale for evaluating the selected error criteria?

Each model employs distinct ANN architectures that are designed based on various Taguchi orthogonal vector plans. These plans enhance the efficiency, reliability, and precision of estimating the effort and cost involved in software project development. Taguchi's orthogonal vector plan is instrumental in selecting combinations without repetition, ensuring that all factors are independently and equally evaluated. By employing this method, the full

factorial plan is reduced, which means that instead of testing all conceivable ANN combinations, Taguchi's approach only considers pairs of combinations, relying on a unique set of Latin squares. This approach to data analysis provides precise insights into the factors that have the most significant impact on the quality of the developed product. Furthermore, it minimizes the number of experiments required, thereby reducing time and conserving resources. The ultimate goal of Taguchi's approach is to provide an excellent product while possibly cutting expenditures on development.

Three novel, hybrid models, each corresponding to one of the three aforementioned approaches, are designed by constructing distinct ANN architectures utilizing Taguchi orthogonal vector plans. In this chapter, these three innovative models will be introduced and elaborated upon:

- Novel, hybrid COCOMO2000 model,
- Novel, hybrid COSMIC FFP model, and
- Novel, hybrid UCP model.

5.1. NOVEL, HYBRID COCOMO2000 MODEL

In this chapter, an enhanced version of the COCOMO2000 model within the COCOMO approach is presented. The enhanced COCOMO2000 model utilizes various distinct architectures and orthogonal vector plans, including L9, L18, L27, and L36:

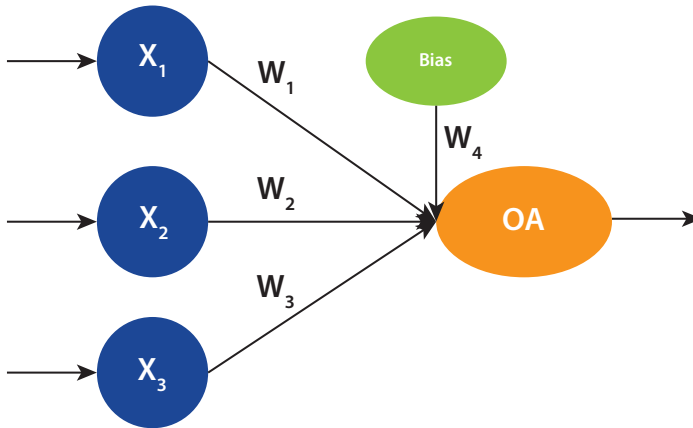
1. COCOMO2000 and ANN-L9

Because there are no hidden layers, the initial ANN architecture, ANN-L9, is the simplest. It is based on the orthogonal vector plan of Taguchi (L9) with four parameters ($W_i, i = \overline{1,4}$) and three different levels [136], [137], [241]: L1, L2 and L3, see Figure 12, Table 5. It is experimented with nine ANN candidates denoted as ANN1, ANN2,...,ANN9 who compete to become a "Winner" network. Bias represents an additional weighting factor to complete the selected orthogonal plan and has a value of 1.

Table 5. Taguchi orthogonal vector plan ($L9=3^4$).

ANN-L9	W_1	W_2	W_3	W_4
ANN1	L1	L1	L1	L1
ANN2	L1	L2	L2	L2
ANN3	L1	L3	L3	L3
ANN4	L2	L1	L2	L3
ANN5	L2	L2	L3	L1
ANN6	L2	L3	L1	L2
ANN7	L3	L1	L3	L2
ANN8	L3	L2	L1	L3
ANN9	L3	L3	L2	L1

Figure 12. ANN-L9.



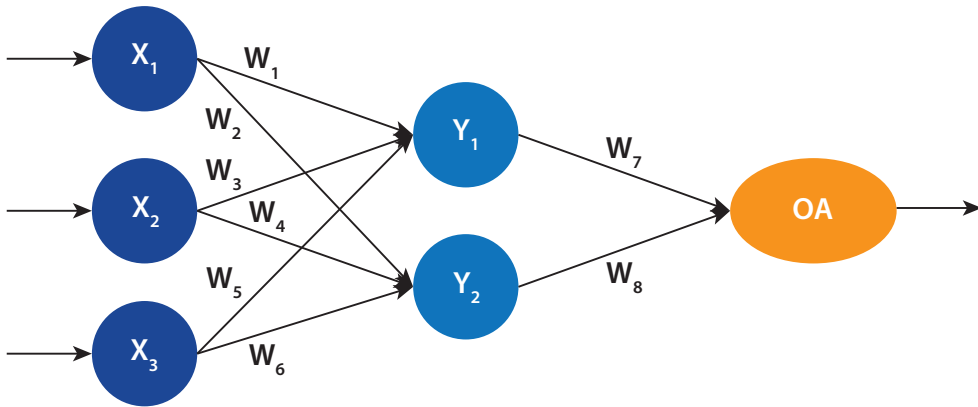
2. COCOMO2000 and ANN-L18

A single hidden layer is used in the design of the second proposed ANN architecture, which is denoted as ANN-L18. It is based on the Taguchi orthogonal vector plan (L18) with eight parameters ($W_i, i = \overline{1,8}$) and three different levels [136], [137], [241]: L1, L2 and L3, see Figure 13, Table 6. Compared to the previous architecture, this is a combined version, because the first parameter has only two levels, and the remaining seven have all three levels. It is experimented with eighteen ANN candidates denoted as ANN1, ANN2,...,ANN18 who compete to become a "Winner" network.

Table 6. Taguchi orthogonal vector plan (L18=2¹3⁷).

ANN-L18	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8
ANN1	L1	L1	L1	L1	L1	L1	L1	L1
ANN2	L1	L1	L2	L2	L2	L2	L2	L2
ANN3	L1	L1	L3	L3	L3	L3	L3	L3
ANN4	L1	L2	L1	L1	L2	L2	L3	L3
ANN5	L1	L2	L2	L2	L3	L3	L1	L1
ANN6	L1	L2	L3	L3	L1	L1	L2	L2
ANN7	L1	L3	L1	L2	L1	L3	L2	L3
ANN8	L1	L3	L2	L3	L2	L1	L3	L1
ANN9	L1	L3	L3	L1	L3	L2	L1	L2
ANN10	L2	L1	L1	L3	L3	L2	L2	L1
ANN11	L2	L1	L2	L1	L1	L3	L3	L2
ANN12	L2	L1	L3	L2	L2	L1	L1	L3
ANN13	L2	L2	L1	L2	L3	L1	L3	L2
ANN14	L2	L2	L2	L3	L1	L2	L1	L3
ANN15	L2	L2	L3	L1	L2	L3	L2	L1
ANN16	L2	L3	L1	L3	L2	L3	L1	L2
ANN17	L2	L3	L2	L1	L3	L1	L2	L3
ANN18	L2	L3	L3	L2	L1	L2	L3	L1

Figure 13. ANN-L18.



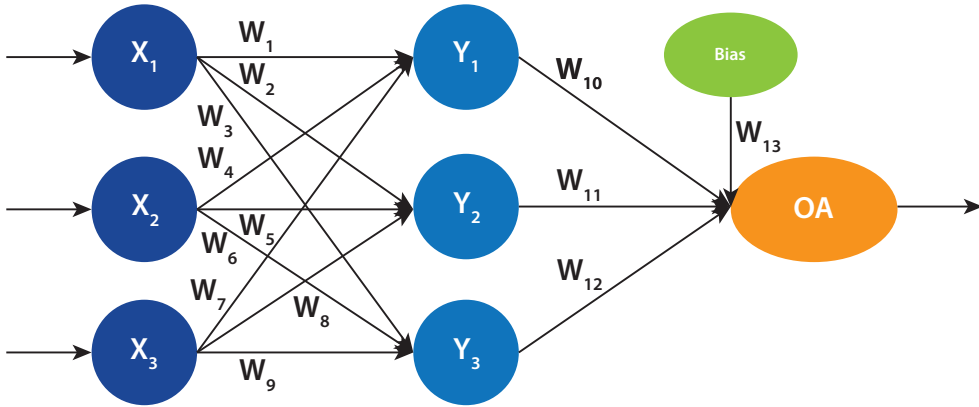
3. COCOMO2000 and ANN-L27

One hidden layer is also used in the design of the third proposed ANN architecture, which is denoted as ANN-L27. It is based on the Taguchi orthogonal vector plan (L27) with thirteen parameters W_i , $i = 1, 13$ and three different levels [136], [137], [241]: L1, L2 and L3, see Figure 14, Table 7. It has experimented with twenty-seven ANN candidates denoted as ANN1, ANN2,...,ANN27 who compete to become a "Winner" network. Bias represents an additional weighting factor to complete the selected orthogonal plan and has a value of 1.

Table 7. Taguchi orthogonal vector plan ($L_{27}=3^{13}$)

ANN-L27	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
ANN1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1
ANN2	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2	L2	L2	L2
ANN3	L1	L1	L1	L1	L3	L3	L3	L3	L3	L3	L3	L3	L3
ANN4	L1	L2	L2	L2	L1	L1	L1	L2	L2	L2	L3	L3	L3
ANN5	L1	L2	L2	L2	L2	L2	L2	L3	L3	L3	L1	L1	L1
ANN6	L1	L2	L2	L2	L3	L3	L3	L1	L1	L1	L2	L2	L2
ANN7	L1	L3	L3	L3	L1	L1	L1	L3	L3	L3	L2	L2	L2
ANN8	L1	L3	L3	L3	L2	L2	L2	L1	L1	L1	L3	L3	L3
ANN9	L1	L3	L3	L3	L3	L3	L3	L2	L2	L2	L1	L1	L1
ANN10	L2	L1	L2	L3	L1	L2	L3	L1	L2	L3	L1	L2	L3
ANN11	L2	L1	L2	L3	L2	L3	L1	L2	L3	L1	L2	L3	L1
ANN12	L2	L1	L2	L3	L3	L1	L2	L3	L1	L2	L3	L1	L2
ANN13	L2	L2	L3	L1	L1	L2	L3	L2	L3	L1	L3	L1	L2
ANN14	L2	L2	L3	L1	L2	L3	L1	L3	L1	L2	L1	L2	L3
ANN15	L2	L2	L3	L1	L3	L1	L2	L1	L2	L3	L2	L3	L1
ANN16	L2	L3	L1	L2	L1	L2	L3	L3	L1	L2	L2	L3	L1
ANN17	L2	L3	L1	L2	L2	L3	L1	L1	L2	L3	L3	L1	L2
ANN18	L2	L3	L1	L2	L3	L1	L2	L2	L3	L1	L1	L2	L3
ANN19	L3	L1	L3	L2	L1	L3	L2	L1	L3	L2	L1	L3	L2
ANN20	L3	L1	L3	L2	L2	L1	L3	L2	L1	L3	L2	L1	L3
ANN21	L3	L1	L3	L2	L3	L2	L1	L3	L2	L1	L3	L2	L1
ANN22	L3	L2	L1	L3	L1	L3	L2	L2	L1	L3	L3	L2	L1
ANN23	L3	L2	L1	L3	L2	L1	L3	L3	L2	L1	L1	L3	L2
ANN24	L3	L2	L1	L3	L3	L2	L1	L1	L3	L2	L2	L1	L3
ANN25	L3	L3	L2	L1	L1	L3	L2	L3	L2	L1	L2	L1	L3
ANN26	L3	L3	L2	L1	L2	L1	L3	L1	L3	L2	L3	L2	L1
ANN27	L3	L3	L2	L1	L3	L2	L1	L2	L1	L3	L1	L3	L2

Figure 14. ANN-L27.



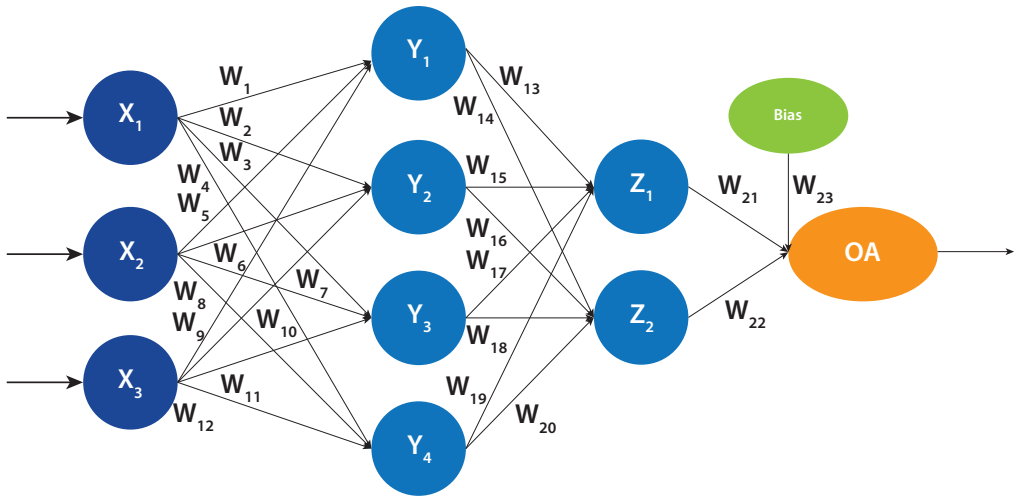
4. COCOMO2000 and ANN-L36

Two hidden layers create the fourth proposed ANN architecture, denoted as ANN-L36. It is based on the Taguchi orthogonal vector plan (L36) with twenty-three parameters ($W_i, i = \overline{1,23}$) and three different levels [136], [137], [241]: L1, L2 and L3, see Figure 15, Table 8. This architecture is a combined network, because the first eleven parameters have only two levels, and the remaining twelve have all three levels. It is experimented with thirty-six ANN candidates denoted as ANN1, ANN2,...,ANN36 who compete to become a "Winner" network. Bias represents an additional weighting factor to complete the selected orthogonal plan and has a value of 1.

Table 8. Taguchi orthogonal vector plan (L36=2¹¹3¹²).

ANN-L36	W ₁	W ₂	W ₃	W ₄	W ₅	W ₆	W ₇	W ₈	W ₉	W ₁₀	W ₁₁	W ₁₂	W ₁₃	W ₁₄	W ₁₅	W ₁₆	W ₁₇	W ₁₈	W ₁₉	W ₂₀	W ₂₁	W ₂₂	W ₂₃
ANN1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1
ANN2	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2
ANN3	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3
ANN4	L1	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2	L1	L1	L1	L1	L2	L2	L2	L2	L3	L3	L3	L3
ANN5	L1	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L3	L3	L3	L3	L1	L1	L1	L1
ANN6	L1	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2	L3	L3	L3	L3	L1	L1	L1	L1	L2	L2	L2	L2
ANN7	L1	L1	L2	L2	L2	L1	L1	L1	L2	L2	L2	L1	L1	L2	L3	L1	L2	L3	L3	L1	L2	L2	L3
ANN8	L1	L1	L2	L2	L2	L1	L1	L1	L2	L2	L2	L2	L2	L3	L1	L2	L3	L1	L1	L2	L3	L3	L1
ANN9	L1	L1	L2	L2	L2	L1	L1	L1	L2	L2	L2	L3	L3	L1	L2	L3	L1	L2	L2	L3	L1	L1	L2
ANN10	L1	L2	L1	L2	L2	L1	L2	L2	L1	L1	L2	L1	L1	L3	L2	L1	L3	L2	L3	L2	L1	L3	L2
ANN11	L1	L2	L1	L2	L2	L1	L2	L2	L1	L1	L2	L2	L2	L1	L3	L2	L1	L3	L1	L3	L2	L1	L3
ANN12	L1	L2	L1	L2	L2	L1	L2	L2	L1	L1	L2	L3	L3	L2	L1	L3	L2	L1	L2	L1	L3	L2	L1
ANN13	L1	L2	L2	L1	L2	L2	L1	L2	L1	L2	L1	L1	L2	L3	L1	L3	L2	L1	L3	L3	L2	L1	L2
ANN14	L1	L2	L2	L1	L2	L2	L1	L2	L1	L2	L1	L2	L3	L1	L2	L1	L3	L2	L1	L1	L3	L2	L3
ANN15	L1	L2	L2	L1	L2	L2	L1	L2	L1	L2	L1	L3	L1	L2	L3	L2	L1	L3	L2	L2	L1	L3	L1
ANN16	L1	L2	L2	L2	L1	L2	L2	L1	L2	L1	L1	L1	L2	L3	L2	L1	L1	L3	L2	L3	L3	L2	L1
ANN17	L1	L2	L2	L2	L1	L2	L2	L1	L2	L1	L1	L2	L3	L1	L3	L2	L2	L1	L3	L1	L1	L3	L2
ANN18	L1	L2	L2	L2	L1	L2	L2	L1	L2	L1	L1	L3	L1	L2	L1	L3	L3	L2	L1	L2	L2	L1	L3
ANN19	L2	L1	L2	L2	L1	L1	L2	L2	L1	L2	L1	L1	L2	L1	L3	L3	L3	L1	L2	L2	L1	L2	L3
ANN20	L2	L1	L2	L2	L1	L1	L2	L2	L1	L2	L1	L2	L3	L2	L1	L1	L1	L2	L3	L3	L2	L3	L1
ANN21	L2	L1	L2	L2	L1	L1	L2	L2	L1	L2	L1	L3	L1	L3	L2	L2	L2	L3	L1	L1	L3	L1	L2
ANN22	L2	L1	L2	L1	L2	L2	L1	L1	L1	L2	L1	L2	L2	L2	L3	L3	L1	L2	L1	L1	L3	L3	L2
ANN23	L2	L1	L2	L1	L2	L2	L1	L1	L1	L1	L2	L2	L3	L3	L1	L1	L2	L3	L2	L2	L1	L1	L3
ANN24	L2	L1	L2	L1	L2	L2	L2	L1	L1	L1	L2	L3	L1	L1	L2	L2	L3	L1	L3	L3	L2	L2	L1
ANN25	L2	L1	L1	L2	L2	L2	L1	L2	L2	L1	L1	L1	L3	L2	L1	L2	L3	L3	L1	L3	L1	L2	L2
ANN26	L2	L1	L1	L2	L2	L2	L1	L2	L2	L1	L1	L2	L1	L3	L2	L3	L1	L1	L2	L1	L2	L3	L3
ANN27	L2	L1	L1	L2	L2	L2	L1	L2	L2	L1	L1	L3	L2	L1	L3	L1	L2	L2	L3	L2	L3	L1	L1
ANN28	L2	L2	L2	L1	L1	L1	L1	L2	L2	L1	L2	L1	L3	L2	L2	L2	L1	L1	L3	L2	L3	L1	L3
ANN29	L2	L2	L2	L1	L1	L1	L1	L2	L2	L1	L2	L2	L1	L3	L3	L3	L2	L2	L1	L3	L1	L2	L1
ANN30	L2	L2	L2	L1	L1	L1	L1	L2	L2	L1	L2	L3	L2	L1	L1	L1	L3	L3	L2	L1	L2	L3	L2
ANN31	L2	L2	L1	L2	L1	L2	L1	L1	L1	L2	L2	L1	L3	L3	L3	L2	L3	L2	L2	L1	L2	L1	L1
ANN32	L2	L2	L1	L2	L1	L2	L1	L1	L1	L2	L2	L2	L1	L1	L1	L3	L1	L3	L2	L3	L2	L2	L2
ANN33	L2	L2	L1	L2	L1	L2	L1	L1	L1	L2	L2	L3	L2	L2	L2	L1	L2	L1	L1	L3	L1	L3	L3
ANN34	L2	L2	L1	L1	L2	L1	L2	L1	L2	L2	L1	L1	L3	L1	L2	L3	L2	L3	L1	L2	L2	L3	L1
ANN35	L2	L2	L1	L1	L2	L1	L2	L1	L2	L2	L1	L2	L1	L2	L3	L1	L3	L1	L2	L3	L3	L1	L2
ANN36	L2	L2	L1	L1	L2	L1	L2	L1	L2	L2	L1	L3	L2	L3	L1	L2	L1	L2	L3	L1	L1	L2	L3

Figure 15. ANN-L36.



Each experiment, based on the selected approach and datasets, comprises three distinct phases:

1. *Training*: This phase involves training the proposed ANN architectures to compete and determine a "Winner" network.
2. *Testing*: The ANN identified as the "Winner" in the first part of the experiment, with the lowest MMRE value on the same dataset, undergoes testing.
3. *Validation*: In the validation phase, the ANN with the lowest MMRE value from the initial experiment is tested on different datasets to assess its performance under varied conditions.

5.1.1. Datasets utilized in COCOMO2000 hybrid model

The experiments demanded validation across plenty data sets since they used various types of information from different sources. The COCOMO2000 formula (4–10) is used to calculate the predicted effort for software development projects and convert it into person-months [PM] since the experiment uses artificial neural network topologies with 3 input values and 1 output.

For the initial training phase of the experiment, a COCOMO2000 dataset consisting of 100 projects was employed to train models for all 4 proposed ANN architectures. Subsequently, in the second stage of the experiment, which involves testing the developed models, a COCOMO2000 dataset comprising 20 projects was utilized for each of the 4 proposed ANN architectures.

To enhance the robustness of the proposed approach, multiple datasets were employed for the final stage of the experiment, which focuses on validating the developed models. These datasets include COCOMO81 (with 51 randomly selected projects), a NASA dataset containing 60 projects, and a Kemerer dataset comprising 15 projects (refer to Table 9).

It is worth noting that the datasets used [242] exhibit a wide range of actual effort values, ranging from a minimum of 6.0 person-months (PM) in Dataset_1 to a maximum of 11,399.9 PM in Dataset_3. This variation underscores the considerable standard deviation in the data range (as depicted in Table 10).

Table 9. Information on used datasets (COCOMO2000).

	Dataset	Number of projects	Experiment
Dataset_1	COCOMO2000 dataset	100	Training
Dataset_2	COCOMO2000 dataset	20	Testing
Dataset_3	COCOMO81 dataset	Random 51	Validation1
Dataset_4	NASA dataset	60	Validation2
Dataset_5	Kemerer dataset	15	Validation3

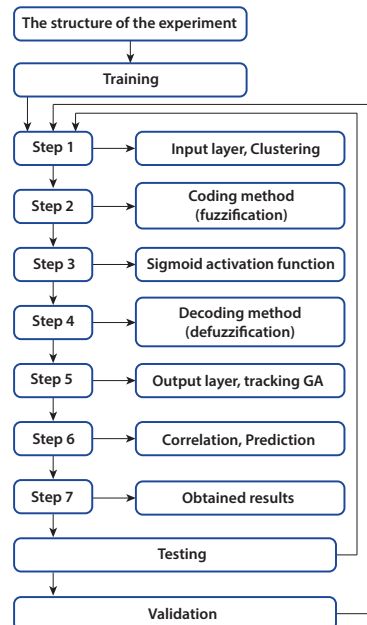
Table 10. Basic statistics about datasets (COCOMO2000).

Dataset	No. of projects	Min [PM]	Max [PM]	Mean [PM]	Std. deviation [PM]
Dataset_1	100	6.0	8211.0	616.0	1131.5
Dataset_2	20	28.1	606.8	277.0	206.3
Dataset_3	51	33.0	11399.9	841.8	1994.9
Dataset_4	60	8.4	3240.0	406.4	656.9
Dataset_5	15	23.2	1780.0	316.7	456.7

5.1.2. COCOMO2000 hybrid model methodology

A multitude of trial experiments were conducted to determine the most suitable methodology for the experimental phase of the COCOMO2000 approach. Figure 16 illustrates the robust design algorithm employed to establish the sequence of steps for the experiments.

Figure 16. Algorithm for conducting the experiment using the robust design approach for COCOMO2000.



Step 1: Input layer, Clustering

The input layer consists of data from the first 100 projects of the COCOMO2000 dataset, divided into three clusters according to the value of actual effort expressed in person-months [PM]: less than 90PM (small projects), between 90PM and 500PM (medium projects), and more than 500PM (large projects). The three input values for the proposed neural network architectures ANN-L9, ANN-L18, ANN-L27, and ANN-L36 are: $X_1 = E$, $X_2 = PEM_i$, and $X_3 = KLOC$ (see formulas 5-7).

Step 2: Coding method (fuzzification)

In all trial experiments, the method that gave the best results was fuzzification which includes: mapping of all input values X_1 , X_2 and X_3 into real values given in the interval $[0, 1]$: The function $\mu_D(X): R \rightarrow [0, 1]$, translates the real values of input signals into coded values from the interval $[0, 1]$, in the following way: $\mu_D(X_i) = (X_i - X_{min}) / (X_{max} - X_{min})$ (min-max normalization) [243], where D is the set of data on which the experiment is performed.

Step 3: Activation function choices - hidden layer

The functions of the hidden layer and the output layer use two different activation functions for proposed ANN architectures. *EstEffANN* represents the output value of proposed model (31):

$$1. y_i = \frac{1}{1 + e^{-x_i}}, i = \overline{1, n} \quad (31)$$

a) None hidden and one output layer function for ANN-L9 architecture, see Figure 12, Table 5 (32):

$$EstEffANN-L9 = 1 / (1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_2 + X_3 \cdot W_3 + 1 \cdot W_4)}) \quad (32)$$

where *EstEffANN-L9* is the output value.

b) One hidden and one output layer functions for ANN-L18 architecture, see Figure 13, Table 6 (33)-(35):

$$Y_1 = 1 / (1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_3 + X_3 \cdot W_5)}) \quad (33)$$

$$Y_2 = 1 / (1 + e^{-(X_1 \cdot W_2 + X_2 \cdot W_4 + X_3 \cdot W_6)}) \quad (34)$$

$$EstEffANN-L18 = 1 / (1 + e^{-(Y_1 \cdot W_7 + Y_2 \cdot W_8)}) \quad (35)$$

where Y_1 and Y_2 are calculated values from hidden layer and *EstEffANN-L18* is the output value.

c) One hidden and one output layer functions for ANN-L27 architecture, see Figure 14, Table 7 (36)-(39):

$$Y_1 = \frac{1}{1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_2 + X_3 \cdot W_3)}} \quad (36)$$

$$Y_2 = \frac{1}{1 + e^{-(X_1 \cdot W_4 + X_2 \cdot W_5 + X_3 \cdot W_6)}} \quad (37)$$

$$Y_3 = \frac{1}{1 + e^{-(X_1 \cdot W_7 + X_2 \cdot W_8 + X_3 \cdot W_9)}} \quad (38)$$

$$EstEffANN - L27 = \frac{1}{1 + e^{-(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + W_{13})}} \quad (39)$$

where Y_1 , Y_2 , and Y_3 are calculated values from hidden layer and *EstEffANN-L27* is the output value.

d) Two hidden and one output layer functions for ANN-L36 architecture, see Figure 15, Table 7 (40)-(46):

$$Y_1 = \frac{1}{1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_2 + X_3 \cdot W_3)}} \quad (40)$$

$$Y_2 = \frac{1}{1 + e^{-(X_1 \cdot W_4 + X_2 \cdot W_5 + X_3 \cdot W_6)}} \quad (41)$$

$$Y_3 = \frac{1}{1 + e^{-(X_1 \cdot W_7 + X_2 \cdot W_8 + X_3 \cdot W_9)}} \quad (42)$$

$$Y_4 = \frac{1}{1 + e^{-(X_1 \cdot W_{10} + X_2 \cdot W_{11} + X_3 \cdot W_{12})}} \quad (43)$$

$$Z_1 = \frac{1}{1 + e^{-(Y_1 \cdot W_{13} + Y_2 \cdot W_{14} + Y_3 \cdot W_{15} + Y_4 \cdot W_{16})}} \quad (44)$$

$$Z_2 = \frac{1}{1 + e^{-(Y_1 \cdot W_{17} + Y_2 \cdot W_{18} + Y_3 \cdot W_{19} + Y_4 \cdot W_{20})}} \quad (45)$$

$$EstEffANN - L36 = \frac{1}{1 + e^{-(Z_1 \cdot W_{21} + Z_2 \cdot W_{22} + W_{23})}} \quad (46)$$

where Y_1 , Y_2 , Y_3 , and Y_4 are calculated values from the first hidden layer, Z_1 and Z_2 are calculated values from the second hidden layer and *EstEffANN-L36* is the output value.

Example of hyperbolic tangent for ANN-L27 and ANN-L36 according to formula (47):

$$y_i = \frac{e^{x_i} - e^{-x_i}}{e^{x_i} + e^{-x_i}}, i = \overline{1, n} \quad (47)$$

a) Hidden and output layer functions for ANN-L27 architecture, according to formulas (48)-(51):

$$Y_1 = \frac{e^{(X_1 \cdot W_1 + X_2 \cdot W_4 + X_5 \cdot W_6)} - e^{-(X_1 \cdot W_1 + X_2 \cdot W_4 + X_5 \cdot W_6)}}{e^{(X_1 \cdot W_1 + X_2 \cdot W_4 + X_5 \cdot W_6)} + e^{-(X_1 \cdot W_1 + X_2 \cdot W_4 + X_5 \cdot W_6)}} \quad (48)$$

$$Y_2 = \frac{e^{(X_1 \cdot W_2 + X_3 \cdot W_5 + X_6 \cdot W_8)} - e^{-(X_1 \cdot W_2 + X_3 \cdot W_5 + X_6 \cdot W_8)}}{e^{(X_1 \cdot W_2 + X_3 \cdot W_5 + X_6 \cdot W_8)} + e^{-(X_1 \cdot W_2 + X_3 \cdot W_5 + X_6 \cdot W_8)}} \quad (49)$$

$$Y_3 = \frac{e^{(X_1 \cdot W_3 + X_4 \cdot W_7 + X_5 \cdot W_9)} - e^{-(X_1 \cdot W_3 + X_4 \cdot W_7 + X_5 \cdot W_9)}}{e^{(X_1 \cdot W_3 + X_4 \cdot W_7 + X_5 \cdot W_9)} + e^{-(X_1 \cdot W_3 + X_4 \cdot W_7 + X_5 \cdot W_9)}} \quad (50)$$

$$EstEffANN - L27 = \frac{e^{(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + 1 \cdot W_{13})} - e^{-(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + 1 \cdot W_{13})}}{e^{(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + 1 \cdot W_{13})} + e^{-(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + 1 \cdot W_{13})}} \quad (51)$$

where Y_1, Y_2, Y_3 are calculated values from hidden layer and $EstEffANN-L27$ is the output value.

b) Hidden and output layer functions for ANN-L36 architecture, according to formulas (52)-(58):

$$Y_1 = \frac{e^{(X_1 \cdot W_1 + X_2 \cdot W_4 + X_5 \cdot W_6)} - e^{-(X_1 \cdot W_1 + X_2 \cdot W_4 + X_5 \cdot W_6)}}{e^{(X_1 \cdot W_1 + X_2 \cdot W_4 + X_5 \cdot W_6)} + e^{-(X_1 \cdot W_1 + X_2 \cdot W_4 + X_5 \cdot W_6)}} \quad (52)$$

$$Y_2 = \frac{e^{(X_1 \cdot W_2 + X_3 \cdot W_5 + X_6 \cdot W_8)} - e^{-(X_1 \cdot W_2 + X_3 \cdot W_5 + X_6 \cdot W_8)}}{e^{(X_1 \cdot W_2 + X_3 \cdot W_5 + X_6 \cdot W_8)} + e^{-(X_1 \cdot W_2 + X_3 \cdot W_5 + X_6 \cdot W_8)}} \quad (53)$$

$$Y_3 = \frac{e^{(X_1 \cdot W_3 + X_4 \cdot W_7 + X_5 \cdot W_9)} - e^{-(X_1 \cdot W_3 + X_4 \cdot W_7 + X_5 \cdot W_9)}}{e^{(X_1 \cdot W_3 + X_4 \cdot W_7 + X_5 \cdot W_9)} + e^{-(X_1 \cdot W_3 + X_4 \cdot W_7 + X_5 \cdot W_9)}} \quad (54)$$

$$Y_4 = \frac{e^{(X_1 \cdot W_4 + X_2 \cdot W_5 + X_3 \cdot W_{10})} - e^{-(X_1 \cdot W_4 + X_2 \cdot W_5 + X_3 \cdot W_{10})}}{e^{(X_1 \cdot W_4 + X_2 \cdot W_5 + X_3 \cdot W_{10})} + e^{-(X_1 \cdot W_4 + X_2 \cdot W_5 + X_3 \cdot W_{10})}} \quad (55)$$

$$Z_1 = \frac{e^{(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + Y_4 \cdot W_{13})} - e^{-(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + Y_4 \cdot W_{13})}}{e^{(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + Y_4 \cdot W_{13})} + e^{-(Y_1 \cdot W_{10} + Y_2 \cdot W_{11} + Y_3 \cdot W_{12} + Y_4 \cdot W_{13})}} \quad (56)$$

$$Z_2 = \frac{e^{(Y_1 \cdot W_{14} + Y_2 \cdot W_{15} + Y_3 \cdot W_{16} + Y_4 \cdot W_{17})} - e^{-(Y_1 \cdot W_{14} + Y_2 \cdot W_{15} + Y_3 \cdot W_{16} + Y_4 \cdot W_{17})}}{e^{(Y_1 \cdot W_{14} + Y_2 \cdot W_{15} + Y_3 \cdot W_{16} + Y_4 \cdot W_{17})} + e^{-(Y_1 \cdot W_{14} + Y_2 \cdot W_{15} + Y_3 \cdot W_{16} + Y_4 \cdot W_{17})}} \quad (57)$$

$$EstEffANN - L36 = \frac{e^{(Z_1 \cdot W_{21} + Z_2 \cdot W_{22} + Y_5 \cdot W_{23} + 1 \cdot W_{24})} - e^{-(Z_1 \cdot W_{21} + Z_2 \cdot W_{22} + Y_5 \cdot W_{23} + 1 \cdot W_{24})}}{e^{(Z_1 \cdot W_{21} + Z_2 \cdot W_{22} + Y_5 \cdot W_{23} + 1 \cdot W_{24})} + e^{-(Z_1 \cdot W_{21} + Z_2 \cdot W_{22} + Y_5 \cdot W_{23} + 1 \cdot W_{24})}} \quad (58)$$

where Y_1 , Y_2 , Y_3 , and Y_4 are calculated values from the first hidden layer, Z_1 and Z_2 are calculated values from the second hidden layer and *EstEffANN-L36* is the output value.

Initial values in all proposed ANN architectures for weighting factors are taken from set [-1, 0, 1]. After the first iteration, the value of the cost effect function is calculated and arranged based on Taguchi Orthogonal Array and according to the following formula - for example, ANN-L27 and ANN-L36 are arranged as follows [136], [119] (59):

$$\begin{aligned}
 L1W_1 &= \text{cost1} + \text{cost2} + \dots + \text{cost9} \\
 L2W_1 &= \text{cost10} + \text{cost11} + \dots + \text{cost18} \\
 L3W_1 &= \text{cost19} + \text{cost20} + \dots + \text{cost27} \\
 &\dots \\
 L1W_{13} &= \text{cost1} + \text{cost5} + \dots + \text{cost26} \\
 L1W_{13} &= \text{cost2} + \text{cost6} + \dots + \text{cost27} \\
 L1W_{13} &= \text{cost3} + \text{cost4} + \dots + \text{cost25} \\
 \text{where cost}(i) &= \sum \text{MRE}(\text{ANN-L27}(i))
 \end{aligned} \tag{59}$$

Calculating the levels for ANN-L36 architecture [136], [119] according to formula (60):

$$\begin{aligned}
 L1W_1 &= \text{cost1} + \text{cost2} + \dots + \text{cost18} \\
 L2W_1 &= \text{cost19} + \text{cost20} + \dots + \text{cost36} \\
 &\dots \\
 L1W_{23} &= \text{cost1} + \text{cost5} + \dots + \text{cost34} \\
 L2W_{23} &= \text{cost2} + \text{cost6} + \dots + \text{cost35} \\
 L3W_{23} &= \text{cost3} + \text{cost4} + \dots + \text{cost36} \\
 \text{where cost}(i) &= \sum \text{MRE}(\text{ANN-L36}(i))
 \end{aligned} \tag{60}$$

For each subsequent iteration, the interval is divided as follows [136], [119] according to formula (61):

$$\begin{aligned}
 L1W_{1,new} &= L2W_{1,old} \\
 L2W_{1,new} &= L2W_{1,old} + (L3W_{1,old} - L2W_{1,old})/2 \\
 L3W_{1,new} &= L3W_{1,old}
 \end{aligned} \tag{61}$$

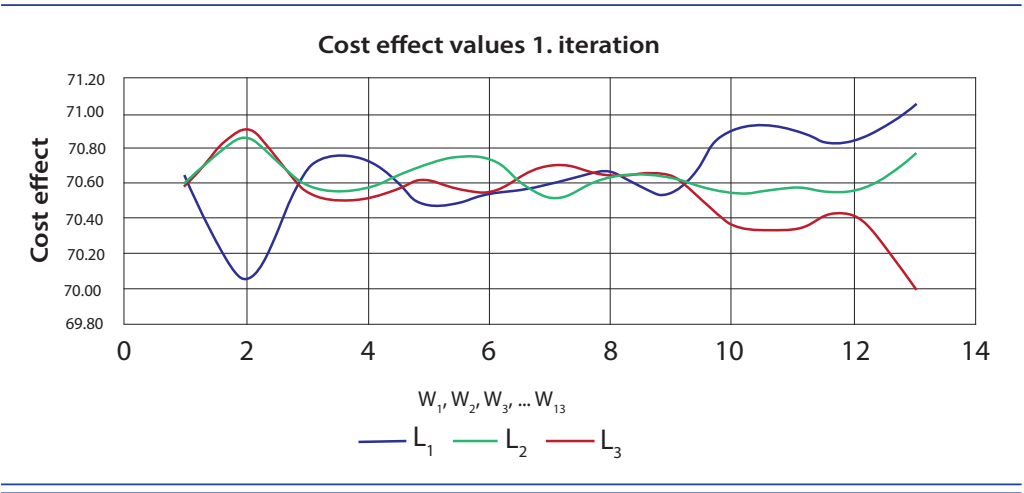
where suffix “old” means values from the interval of the previous iteration, and “new” means the value calculated based on the division of the previous intervals.

Each dataset's input values converge according to the cost-effect function. Figures 17-22 and Tables 11-20 provide an illustration of the interval division and cost-effect function computations over 6 iterations.

Table 11. Cost effect function values - 1st iteration.

Cost effect function values for the 1 st iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	70.64	70.04	70.68	70.72	70.48	70.53	70.59	70.66	70.54	70.91	70.89	70.84	71.05
L2	70.59	70.86	70.59	70.57	70.71	70.74	70.52	70.63	70.63	70.54	70.57	70.55	70.77
L3	70.58	70.91	70.55	70.52	70.62	70.54	70.70	70.64	70.64	70.36	70.34	70.42	69.99

Figure 17. Visual representation of the cost effect function values - 1st iteration.



We may infer from Figure 17 and Table 11 that the weighting factors $W_2, W_{10},$ and W_{13} have the most impact on the change in MRE values in the subsequent iteration, whereas the weighting component W_8 has the least impact in the current iteration.

Figure 18. Segmentation of the interval based on the cost effect function value.

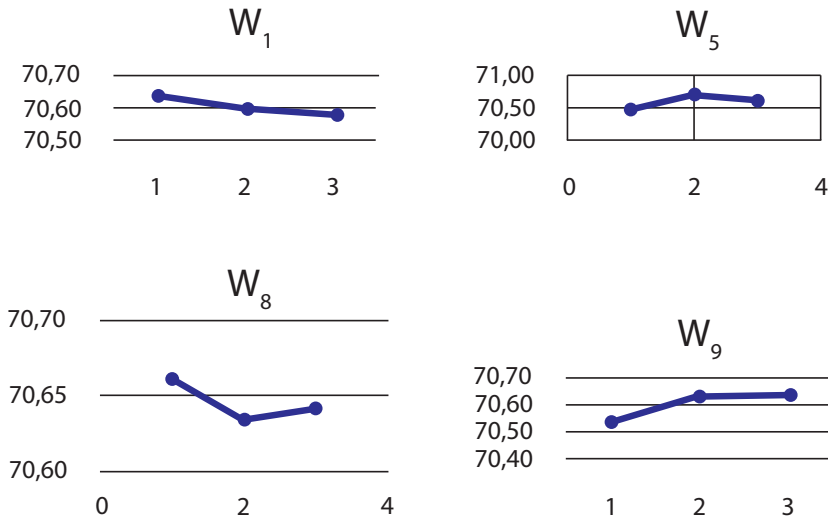


Table 12. Intervals for the 2nd iteration.

Intervals for the 2 nd iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	0.00	-1.00	0.00	0.00	-1.00	-1.00	-1.00	0.00	-1.00	0.00	0.00	0.00	0.00
L2	0.50	-0.50	0.50	0.50	-0.50	-0.50	-0.50	0.50	-0.50	0.50	0.50	0.50	0.50
L3	1.00	0.00	1.00	1.00	0.00	0.00	0.00	1.00	0.00	1.00	1.00	1.00	1.00

Table 13. Cost effect function values - 2nd iteration.

Cost effect function values for the 2 nd iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	69.88	69.68	69.85	69.87	69.74	69.83	69.86	69.91	69.84	69.88	69.99	69.87	70.06
L2	69.78	69.90	69.80	69.79	69.85	69.83	69.81	69.78	69.84	69.77	69.79	69.77	69.83
L3	69.76	69.84	69.78	69.76	69.83	69.76	69.75	69.78	69.75	69.77	69.65	69.78	69.53

According to Figure 19 and Table 13, weighting factors W_{11} and W_{13} have the most impact on the change in MRE values in the succeeding iteration, whereas weighting factors W_3 and W_5 have the least impact in this iteration.

Figure 19. Visual representation of the cost effect function values - 2nd iteration.

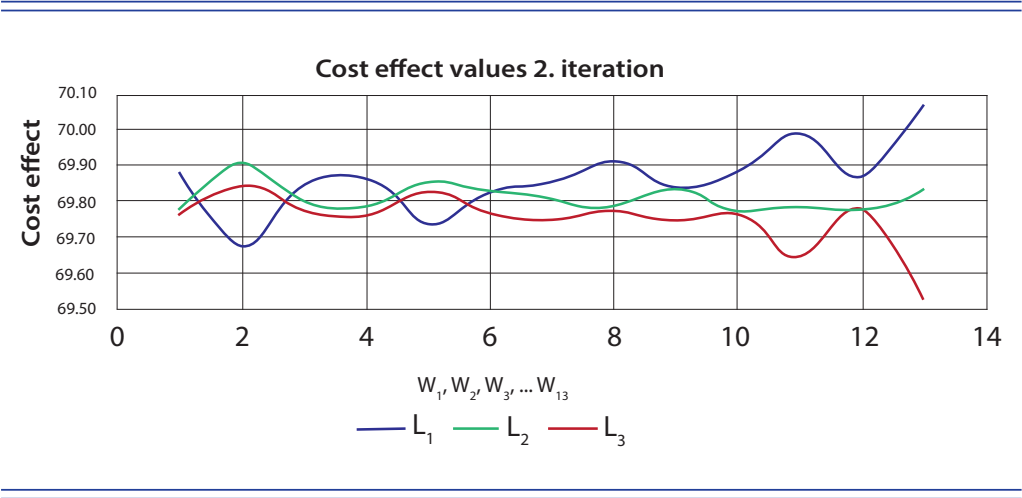


Table 14. Intervals for the - 3rd iteration.

Intervals for the 3 rd iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	0.50	-1.00	0.50	0.50	-1.00	-0.50	-0.50	0.50	-0.50	0.50	0.50	0.50	0.50
L2	0.75	-0.75	0.75	0.75	-0.75	-0.25	-0.25	0.75	-0.25	0.75	0.75	0.75	0.75
L3	1.00	-0.50	1.00	1.00	-0.50	0.00	0.00	1.00	0.00	1.00	1.00	1.00	1.00

Table 15. Cost effect function values - 3rd iteration.

Cost effect function values for the 3 rd iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	69.41	69.38	69.41	69.42	69.39	69.40	69.44	69.42	69.43	69.43	69.46	69.41	69.46
L2	69.37	69.39	69.37	69.37	69.38	69.39	69.38	69.38	69.39	69.37	69.37	69.37	69.39
L3	69.37	69.38	69.36	69.36	69.38	69.36	69.33	69.37	69.33	69.35	69.32	69.36	69.31

Figure 20 and Table 15 analysis show that weighting factors W_{11} and W_{13} have the most impact on the change in MRE values in the future iteration, whereas weighting factors W_2 and W_5 have the least impact in this iteration.

Figure 20. Visual representation of the cost effect function values - 3rd iteration.

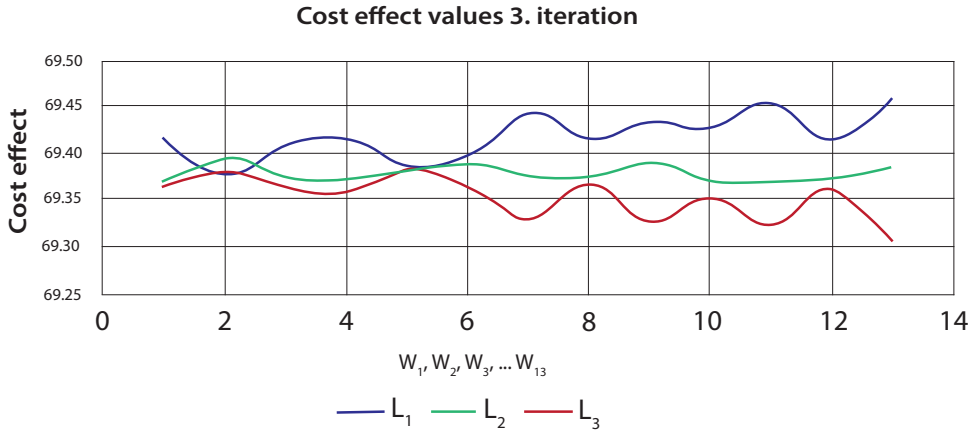


Table 16. Intervals for the 4th iteration.

Intervals for the 4 th iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	0.75	-1.00	0.75	0.75	-0.75	-0.25	-0.25	0.75	-0.25	0.75	0.75	0.75	0.75
L2	0.88	-0.88	0.88	0.88	-0.63	-0.13	-0.13	0.88	-0.13	0.88	0.88	0.88	0.88
L3	1.00	-0.75	1.00	1.00	-0.50	0.00	0.00	1.00	0.00	1.00	1.00	1.00	1.00

Table 17. Cost effect function values - 4th iteration.

Cost effect function values for the 4 th iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	69.22	69.22	69.22	69.23	69.22	69.22	69.24	69.22	69.24	69.23	69.24	69.23	69.24
L2	69.21	69.22	69.21	69.21	69.21	69.21	69.21	69.21	69.22	69.21	69.21	69.21	69.21
L3	69.21	69.21	69.21	69.21	69.21	69.21	69.19	69.21	69.19	69.20	69.20	69.20	69.19

The change in MRE values in the accomplishing iteration has the greatest effect by weighting factors W_7 and W_9 , while weighting factors W_2 and W_5 have the least impact during this iteration, according to Figure 21 and Table 17.

Figure 21. Visual representatioan of the cost effect function values - 4th iteration.

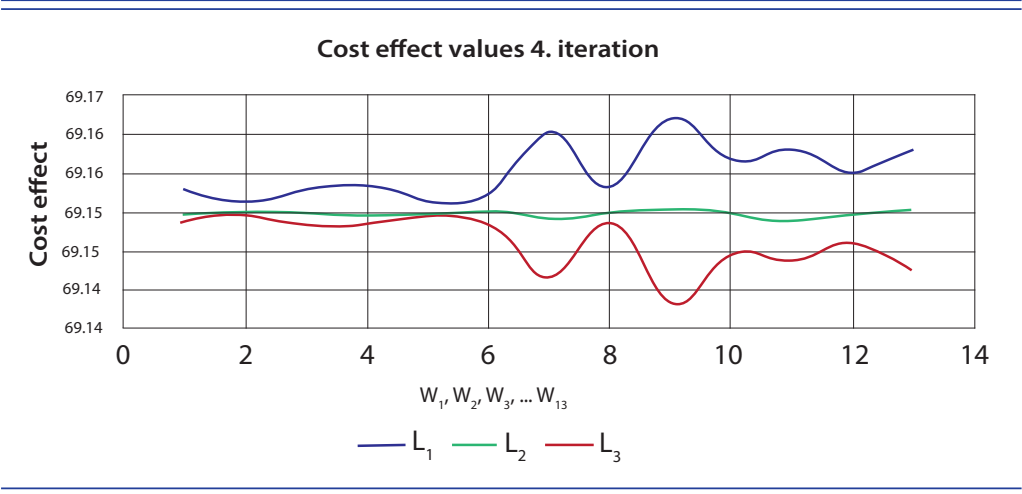


Table 18. Intervals for the 5th iteration.

Intervals for the 5 th iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	0.88	-0.88	0.88	0.88	-0.63	-0.13	-0.13	0.88	-0.13	0.88	0.88	0.88	0.88
L2	0.94	-0.81	0.94	0.94	-0.56	-0.06	-0.06	0.94	-0.06	0.94	0.94	0.94	0.94
L3	1.00	-0.75	1.00	1.00	-0.50	0.00	0.00	1.00	0.00	1.00	1.00	1.00	1.00

Table 19. Cost effect function values - 5th iteration.

Cost effect function values for the 5 th iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	69.15	69.15	69.15	69.15	69.15	69.15	69.16	69.15	69.16	69.16	69.16	69.16	69.16
L2	69.15	69.15	69.15	69.15	69.15	69.15	69.15	69.15	69.15	69.15	69.15	69.15	69.15
L3	69.15	69.15	69.15	69.15	69.15	69.15	69.14	69.15	69.14	69.14	69.14	69.15	69.14

Figure 22 and Table 19 analysis reveals that weighting elements W_9 and W_{10} have the most impact on how MRE values change in the subsequent iteration, whereas weighting factors W_2 and W_5 have the least impact in this iteration.

Figure 22. Visual representation of the cost effect function values - 5th iteration.

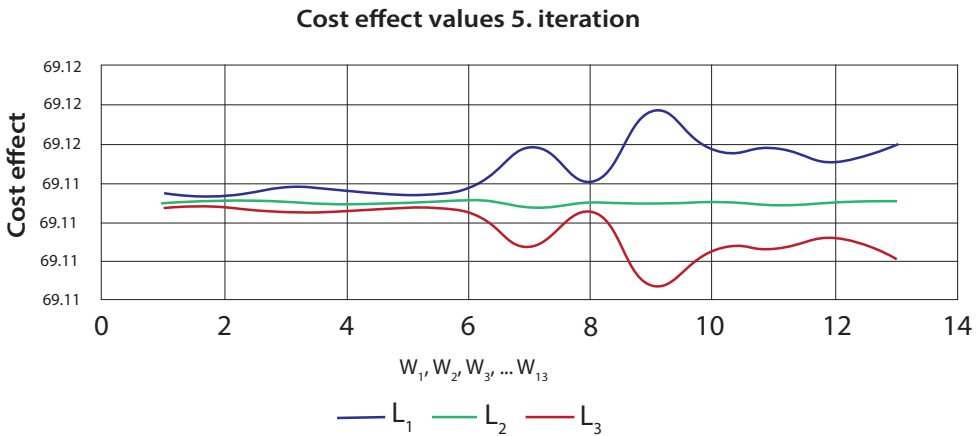


Table 20. Intervals for the 6th iteration.

Intervals for the 6 th iteration													
W_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}
L1	0.88	-0.88	0.88	0.88	-0.63	-0.13	-0.13	0.88	-0.13	0.88	0.88	0.88	0.88
L2	0.94	-0.81	0.94	0.94	-0.56	-0.06	-0.06	0.94	-0.06	0.94	0.94	0.94	0.94
L3	1.00	-0.75	1.00	1.00	-0.50	0.00	0.00	1.00	0.00	1.00	1.00	1.00	1.00

When using the interval division method, the MRE value shows a noticeable decline with each additional repetition. The number of iterations required to arrive at the least relative error is determined by the cost-effect function. Each of the suggested ANNs has an actual speed of convergence that indicates how quickly it approaches the minimal magnitude relative error. For instance, the cost-effect values for the ANN-L27 network function are calculated for each level (L1, L2, and L3) after each iteration. The cost-effect function for each of the ANN architectures (ANN-L9, ANN-L18, ANN-L27, and ANN-L36) must be calculated using Taguchi Orthogonal Arrays. This process keeps on until the GA threshold of 0.01 is reached.

The cost-effect function value for the specified ANN-L27 design is shown in the figures above for each of the three levels over all five iterations. These numbers also shed light on the degree to which each weighting component affects the rate at which the suggested ANN design converges. There are 13 weighting elements identified in the ANN-L27 architecture as $W_1, W_2, \dots, W_{13}$. As can be observed from W_1 to W_5 in Figure 22, when the curves are close together, it means that the weighting variables have a small impact. On the other hand, if the curves are further distant from one another, it means that the weighting variables have a greater influence on the convergence rate, as seen in Figure 22 from W_8 to W_{10} . This suggests that not all weight factors have an equal impact on the speed of convergence.

During each iteration, the same process is repeated for all suggested architectures. Each ANN architecture's rate of convergence is dependent on the interval's proper division, which is determined by the cost-effect function's values. The cost-effect function values for each of the three levels vary because the input signal values (projects) are heterogeneous. Figures 17–22 show how the first values of these cost-effect function values are unequal, indicating weak initial convergence. So that the first and third levels are symmetric, each iteration must divide the intervals correctly. The ANN architecture converges to a particular value (MRE) when the second level reaches equal values.

Step 4: Decoding method (defuzzification)

In all parts of the performed experiment, the appropriate method of defuzzification (decoding) was used according to the following formulas [244] (62)-(64):

$$X_i = (X_{min} + \mu_D(X_i)) \cdot (X_{max} - X_{min}) \quad (62)$$

$$OA(ANN - L27 = X_i, i = \overline{(1, 27)}) \quad (63)$$

$$OA(ANN - L36 = X_i, i = \overline{(1, 36)}) \quad (64)$$

where $i=9, i=18, i=27$ and $i=36$.

Step 5: Output layer, tracking GA

The performance of estimation strategies should be evaluated. There are a range of quantitative relation measurements of accuracy metrics (etc.) that could be used for assessing the quality of a particular estimation strategy. For each iteration in this experiment, the output values are obtained according to the following formulas/measures (65)-(69) [245]:

$$Deviation = |ActEffort - EstEffort| \quad (65)$$

$$MAE_i = \frac{1}{n} \sum_{i=1}^n |ActEffort - EstEffort| \quad (66)$$

$$MRE = Deviation / ActEffort \quad (67)$$

$$MRE = \frac{1}{n} \sum_{i=1}^n MRE_i \quad (68)$$

$$MMRE = mean(MRE) \quad (69)$$

For each of the experimental parts in every iteration, the Gradient Descent is monitored with the condition $GA < 0.01$, calculated as (70):

$$GA = MRE_{i1} - MRE_{i2} < 0.01, \\ \text{where } i=1, \dots, n \text{ } n \text{ is a number of ANN a Winner candidates.} \quad (70)$$

The difference of the minimum values for each iteration in each ANN architecture is denoted by delta, where $\delta(i) = \delta_i$, and is calculated as follows (71), (72):

$$\delta_i = MMRE_{ik} - MMRE_{(i+1)k} \text{ } i\text{-number of ANN, } k\text{-number of iteration.} \quad (71)$$

$$\delta_i > \delta_{i+1}, \text{ then } ANN_i \text{ converges to } MMRE_i \text{ for each of the proposed ANN architectures.} \quad (72)$$

It is possible to determine, in this manner, which of the ANN architectures meets the specified rapidity criterion - that is, which one converges at the quickest rate. By experimenting with different ANN architectures, a GA criterion was set to meet the required number of iterations or stopping convergence at $GA < 0.01$.

In the training part of the experiment of the each "ANN candidate" of the selected ANN architecture according to Taguchi's Orthogonal Array, in each subsequent iteration, a reduction of MRE of less than 1% is achieved, which in this experiment represents a "stop criterion" [114], [115].

Step 6: Correlation, Prediction

Correlation represents the interrelationship between real and estimated value. The higher the correlation coefficient, the more stable the correlation. In this experiment, Pearson's [246], Spearman's [247] and R^2 [248] test were followed. Correlation ranges from +1 to -1 and a correlation of +1 means that there is a perfect positive relationship between variables. The general formula for calculating the correlation coefficients between two variable is as follows (73):

$$\text{Correl}(X, Y) = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^N (x_i - \bar{x})^2 \sum_{i=1}^N (y_i - \bar{y})^2}} \quad (73)$$

Prediction at three criteria: PRED(25), PRED(30), and PRED(50) to calculate the percentage of the total number of ANNs that meet the GA criterion, see formula (74), [249].

$$\text{PRED}(x) = \frac{1}{n} \cdot \sum_{i=1}^N \begin{cases} 1, & \text{if } \text{MRE} \leq x \\ 0, & \text{otherwise} \end{cases}$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 25\%$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 30\%$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 50\%, \text{ where } k = 25, 30 \text{ and } 50. \quad (74)$$

Step 7: Obtained results

In the same dataset, trained models are tested using the same methodology and algorithm as other projects (these are not used in the experiment's training phase). The proposed and trained models are subjected to validation, a second quality check, using a different set of data from different sources. Additionally, the "Winner" network that delivers the best outcomes for each of the proposed ANN architectures is subjected to testing and validation.

5.2. NOVEL, HYBRID COSMIC FFP MODEL

In this chapter, we introduce an enhanced COSMIC FFP model as an integral part of the COSMIC FFP approach. We utilize the architectures and adequate orthogonal vector plans: L12 and L36prim, in the development of the improved COSMIC FFP model:

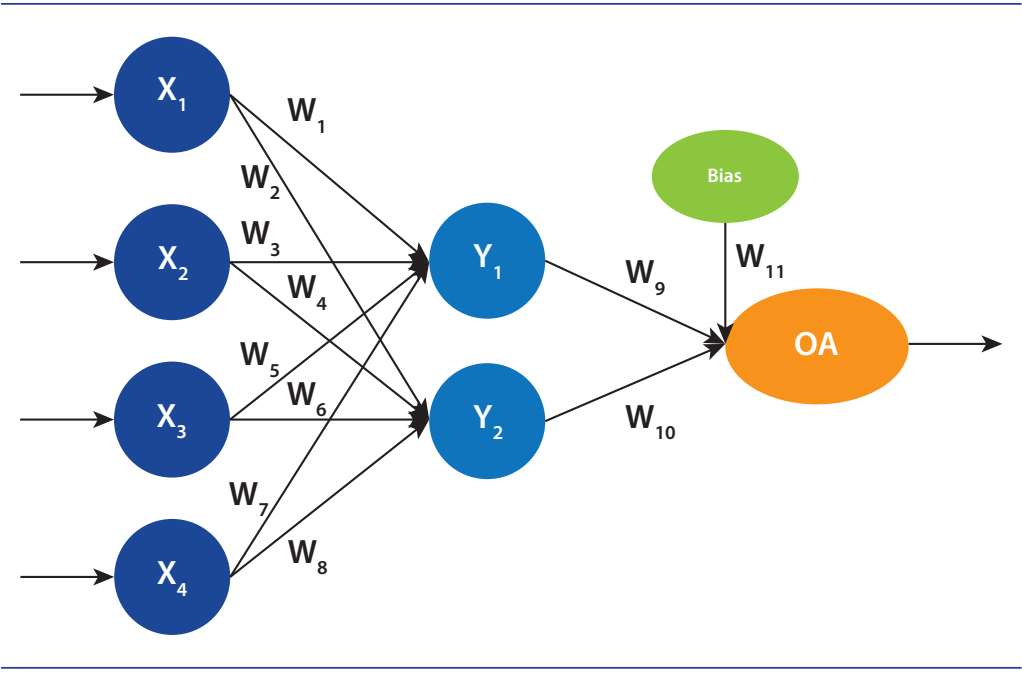
1. COSMIC FFP and ANN-L12

The first proposed architecture is ANN-L12. It consists of four input values, one hidden layer with two nodes, one output, and the total number of eleven weighting factors ($W_i, i = \overline{1,11}$) and whose initial values are from the interval $[-1, 1]$. The Taguchi Orthogonal Array used in the construction of this proposed architecture contains two levels L1 and L2, see Figure 23, Table 21 [136], [137], [114]. Bias represents an additional weighting factor to complete the selected orthogonal plan and has a value of 1.

Table 21. Taguchi orthogonal vector plan (L12= 2^{11}).

ANN-L12	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}
ANN1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1
ANN2	L1	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2
ANN3	L1	L1	L2	L2	L2	L1	L1	L1	L2	L2	L2
ANN4	L1	L2	L1	L2	L2	L1	L2	L2	L1	L1	L2
ANN5	L1	L2	L2	L1	L2	L2	L1	L2	L1	L2	L1
ANN6	L1	L2	L2	L2	L1	L2	L2	L1	L2	L1	L1
ANN7	L2	L1	L2	L2	L1	L1	L2	L2	L1	L2	L1
ANN8	L2	L1	L2	L1	L2	L2	L2	L1	L1	L1	L2
ANN9	L2	L1	L1	L2	L2	L2	L1	L2	L2	L1	L1
ANN10	L2	L2	L2	L1	L1	L1	L1	L2	L2	L1	L2
ANN11	L2	L2	L1	L2	L1	L2	L1	L1	L1	L2	L2
ANN12	L2	L2	L1	L1	L2	L1	L2	L1	L2	L2	L1

Figure 23. ANN-L12.

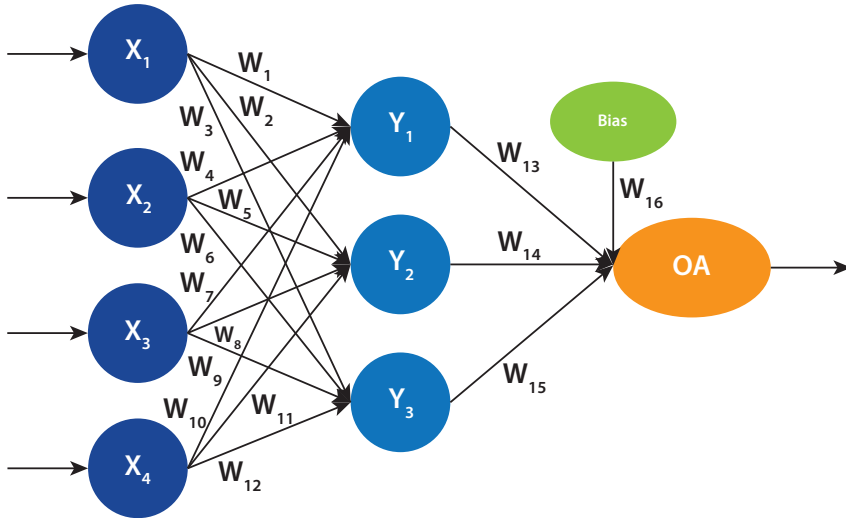


2. COSMIC FFP and ANN-L36prim

The second proposed architecture is ANN-L36prim. It consists of four input values, one hidden layer with three nodes, one output, and the total number of sixteen weighting factors ($W_i, i = \overline{1,16}$) and whose initial values are from the interval $[-1, 1]$. The Taguchi Orthogonal Array used in the construction of this proposed architecture contains two levels L1 and L2, see Figure 24, Table 22 [136], [137], [114]. Bias represents an additional weighting factor to complete the selected orthogonal plan and has a value of 1.

Table 22. Taguchi orthogonal vector plan (L36prim=3¹¹2⁴3¹).

ANN-L36 prim	W ₁	W ₂	W ₃	W ₄	W ₅	W ₆	W ₇	W ₈	W ₉	W ₁₀	W ₁₁	W ₁₂	W ₁₃	W ₁₄	W ₁₅	W ₁₆
ANN1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1
ANN2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L1	L1	L1	L1
ANN3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L1	L1	L1	L1
ANN4	L1	L1	L1	L1	L2	L2	L2	L2	L3	L3	L3	L3	L1	L2	L2	L1
ANN5	L1	L1	L1	L1	L3	L3	L3	L3	L2	L2	L2	L2	L1	L2	L2	L1
ANN6	L3	L3	L3	L3	L1	L1	L1	L1	L2	L2	L2	L2	L1	L2	L2	L1
ANN7	L1	L1	L2	L3	L1	L2	L3	L3	L1	L1	L1	L3	L2	L1	L2	L1
ANN8	L2	L2	L3	L1	L2	L3	L1	L1	L2	L3	L3	L1	L2	L1	L2	L1
ANN9	L3	L3	L1	L2	L3	L1	L2	L2	L3	L1	L1	L2	L2	L1	L2	L1
ANN10	L1	L1	L3	L2	L1	L3	L2	L3	L2	L1	L3	L2	L2	L2	L1	L1
ANN11	L2	L2	L1	L3	L2	L1	L3	L1	L3	L2	L1	L3	L2	L2	L1	L1
ANN12	L3	L3	L2	L1	L3	L2	L1	L2	L1	L3	L2	L1	L2	L2	L1	L1
ANN13	L1	L2	L3	L1	L3	L2	L1	L3	L3	L2	L1	L2	L1	L1	L1	L2
ANN14	L2	L3	L1	L2	L1	L3	L2	L1	L1	L3	L2	L3	L1	L1	L1	L2
ANN15	L3	L1	L2	L3	L2	L1	L3	L2	L2	L1	L3	L1	L1	L1	L1	L2
ANN16	L1	L2	L3	L2	L1	L1	L3	L2	L3	L3	L2	L1	L1	L2	L2	L2
ANN17	L2	L3	L1	L3	L2	L2	L1	L3	L1	L1	L3	L2	L1	L2	L2	L2
ANN18	L3	L1	L2	L1	L3	L3	L2	L1	L2	L2	L1	L3	L1	L2	L2	L2
ANN19	L1	L2	L1	L3	L3	L3	L1	L2	L2	L1	L2	L3	L2	L1	L2	L2
ANN20	L2	L3	L2	L1	L1	L1	L2	L3	L3	L2	L3	L1	L2	L1	L2	L2
ANN21	L3	L1	L3	L2	L2	L2	L3	L1	L1	L3	L1	L2	L2	L1	L2	L2
ANN22	L1	L2	L2	L3	L3	L1	L2	L1	L1	L3	L3	L2	L2	L2	L1	L2
ANN23	L2	L3	L3	L1	L1	L2	L3	L2	L2	L1	L1	L3	L2	L2	L1	L2
ANN24	L3	L1	L1	L2	L2	L3	L1	L3	L3	L2	L2	L1	L2	L2	L1	L2
ANN25	L1	L3	L2	L1	L2	L3	L3	L1	L3	L1	L2	L2	L1	L1	L1	L3
ANN26	L2	L1	L3	L2	L3	L1	L1	L2	L1	L2	L3	L3	L1	L1	L1	L3
ANN27	L3	L2	L1	L3	L1	L2	L2	L3	L2	L3	L1	L1	L1	L1	L1	L3
ANN28	L1	L3	L2	L2	L2	L1	L1	L3	L2	L3	L1	L3	L1	L2	L2	L3
ANN29	L2	L1	L3	L3	L3	L2	L2	L1	L3	L1	L2	L1	L1	L2	L2	L3
ANN30	L3	L2	L1	L1	L1	L3	L3	L2	L1	L2	L3	L2	L1	L2	L2	L3
ANN31	L1	L3	L3	L3	L2	L3	L2	L2	L1	L2	L1	L1	L2	L1	L2	L3
ANN32	L2	L1	L1	L1	L3	L1	L3	L3	L3	L3	L2	L2	L2	L1	L2	L3
ANN33	L3	L2	L2	L2	L1	L2	L1	L1	L3	L1	L3	L3	L2	L1	L2	L3
ANN34	L1	L3	L1	L2	L3	L2	L3	L1	L2	L2	L3	L1	L2	L2	L1	L3
ANN35	L2	L1	L2	L3	L1	L3	L1	L2	L3	L3	L1	L2	L2	L2	L1	L3
ANN36	L3	L2	L3	L1	L2	L1	L2	L3	L1	L1	L2	L3	L2	L2	L1	L3

Figure 24. ANN-L36prim.

Each experiment, based on the selected approach and datasets, comprises three distinct phase:

1. *Training* of two different ANN architectures constructed according to the corresponding Taguchi orthogonal vector plans (ANN-L12 and ANN36prim);
2. *Testing* on ANN "Winner", which gave the best results (the lowest MMRE value) in the first part of the experiment, for two proposed architectures on the same dataset;
3. *Validation* on ANN "Winner" that gave the best results (the lowest MMRE value) in the first part of the experiment, for each selected architecture, but using some other i.e. different datasets.

5.2.1. Datasets utilized in COSMIC FFP hybrid model

The experimental process involved the use of different datasets for various stages:

1. For the first and second parts of the experiment, the ISBSG (International Software Benchmarking Standards Group) repository was employed. ISBSG offers a wide range of information derived from diverse organizations, applications, and development scenarios. It emphasizes the significance of functional values, the type of development (new, improved, or re-developed), the primary programming language or language generation (e.g., 3GL, 4GL), and the development platform (mainframe, midrange, or PC) as crucial estimation criteria.
2. In the third part of the experiment, different datasets were utilized, including the Desharnais dataset (available at <http://promise.site.uottawa.ca>) and a composite dataset comprised of projects from various companies. These datasets provided additional diversity to assess the proposed COSMIC FFP model's performance.

Results in Table 23 cover all three portions of the experiment and highlight the varied character of the designs within each dataset studied. The programming languages used, the length of time it took to construct the application, and the wide range of functional values are all examples of this heterogeneity, and they all contribute to significant standard deviations, as shown in Table 24.

Table 23. Information on used datasets (COSMIC FFP).

	Dataset	Number of projects	Experiment
Dataset_1	ISBSG (Functional Size<10)	37	Training
	ISBSG (Functional Size<10)	15	Testing
Dataset_2	ISBSG (10<Functional Size<50)	45	Training
	ISBSG (10<Functional Size<50)	17	Testing
Dataset_3	ISBSG (50<Functional Size<100)	30	Training
	ISBSG (10<Functional Size<100)	13	Testing
Dataset_4	ISBSG (100<Functional Size<500)	60	Training
	ISBSG (10<Functional Size<500)	17	Testing
Dataset_5	ISBSG (Functional Size>500)	14	Training
	ISBSG (Functional Size>500)	7	Testing
Dataset_6	Desharnais	14	Validation1
Dataset_7	Combined	33	Validation2

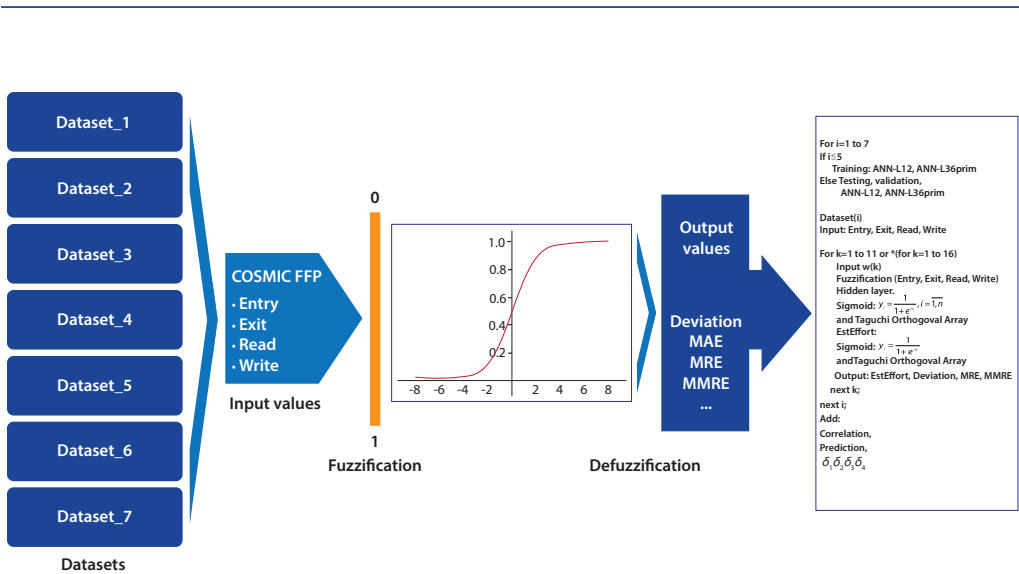
Table 24. Basic statistics about datasets (COSMIC FFP).

Dataset	No. of projects	Min [PM]	Max [PM]	Mean [PM]	Std. deviation [PM]
Dataset_1	52	2.0	9.0	5.404	2.4031
Dataset_2	62	10.0	48.0	24.823	11.3203
Dataset_3	43	50.0	99.0	77.116	15.1441
Dataset_4	77	104.0	492.0	234.130	113.8159
Dataset_5	21	561.0	2090.0	1016.048	458.4442
Dataset_6	14	140.0	3860.0	1011.429	920.4251
Dataset_7	33	493.0	2589.0	1193.424	419.4201

5.2.2. COSMIC FFP hybrid model methodology

The experimental methodology for the COSMIC FFP approach was carefully chosen through a series of trial experiments. Figure 25 indicates that the experiment's steps have been arranged based on a robust design algorithm.

Figure 25. Robust design algorithm for performing the experiment (COSMIC FFP).



Step 1: Input values are the data about particular project from the selected ISBSG dataset and are represented by four parameters (Entry, Exit, Read, Write), which describe the functional size.

Step 2: All input values are transformed according to the following formula:

The function $\mu_D(X): R \rightarrow [0, 1]$, translates the real values of input signals into coded values from the interval $[0, 1]$, in the following way: $\mu_D(X_i) = (X_i - X_{min}) / (X_{max} - X_{min})$ (often called min-max normalization) [121], where D is the set of data on which the experiment is performed, X_i is the input value, X_{min} is the smallest input value, and X_{max} the greatest input value on the observed dataset.

Step 3: The sigmoid function, as the activation function of the hidden layer was used (31):

$$y_i = \frac{1}{1 + e^{-x_i}}, i = 1, n \quad (31)$$

The construction of the activation function is based on a combination of input values and corresponding weight coefficients W_i for each of the proposed ANN architectures.

a) Hidden and output layer functions for ANN-L12 architecture, see Figure 23, Table 21 (75)-(77):

$$Y1 = \frac{1}{1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_2 + X_3 \cdot W_3 + X_4 \cdot W_4)}} \quad (75)$$

$$Y2 = \frac{1}{1 + e^{-(X_1 \cdot W_5 + X_2 \cdot W_6 + X_3 \cdot W_7 + X_4 \cdot W_8)}} \quad (76)$$

$$EstEffortANN - L12 = \frac{1}{1 + e^{-(Y1 \cdot W_9 + Y2 \cdot W_{10} + 1 \cdot W_{11})}} \quad (77)$$

where $Y1$ and $Y2$ are the hidden layer functions and $EstEffortANN-L12$ represents output function.

b) Hidden and output layer functions for ANN-L36prim architecture, see Figure 24, Table 22 (78)-(81):

$$Y1 = \frac{1}{1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_2 + X_3 \cdot W_3 + X_4 \cdot W_{10})}} \quad (78)$$

$$Y2 = \frac{1}{1 + e^{-(X_1 \cdot W_4 + X_2 \cdot W_5 + X_3 \cdot W_6 + X_4 \cdot W_{11})}} \quad (79)$$

$$Y3 = \frac{1}{1 + e^{-(X_1 \cdot W_7 + X_2 \cdot W_8 + X_3 \cdot W_9 + X_4 \cdot W_{12})}} \quad (80)$$

$$EstEffortANN-L36prim = \frac{1}{1 + e^{-(Y_1 \cdot W_{13} + Y_2 \cdot W_{14} + Y_3 \cdot W_{15} + 1 \cdot W_{16})}} \quad (81)$$

where Y_1 , Y_2 , and Y_3 are the hidden layer functions and $EstEffortANN-L36prim$ represents output function.

In the first proposed ANN-L12 architecture, an orthogonal vector plan of two levels L1 and L2, and the initial values of the weighting factors W_i that take the values from the interval $[-1, 1]$, were used. The second proposed architecture has an orthogonal vector plan of three levels L1, L2, and L3, and the initial values of the weighting factors W_i that takes the values from the interval $[-1, 0, 1]$. For each subsequent iteration, new weight factor values must be calculated as follows (e.g., for ANN-L12 architecture) [136], [119] (82):

$$W_{1 \cdot L1} = cost1 + cost2 + \dots + cost6$$

$$W_{1 \cdot L2} = cost7 + cost8 + \dots + cost12$$

....

$$W_{12 \cdot L1} = cost1 + cost5 + \dots + cost12$$

$$W_{12 \cdot L2} = cost2 + cost3 + \dots + cost11$$

$$\text{where } cost(i) = \sum MRE(ANN(i)) \quad (82)$$

For each subsequent iteration, the interval $[-1, 1]$ is divided depending on the cost effect function as follows (83) [7], [119]:

$$W_{1 \cdot L1new} = W_{1 \cdot L1old}$$

$$W_{1 \cdot L2new} = W_{1 \cdot L2old} + (W_{1 \cdot L2old} - W_{1 \cdot L1old})/2 \quad (83)$$

where $W_{1 \cdot L1old}$ and $W_{1 \cdot L2old}$ are values from the previous iteration. The set of input values of each dataset converges depending on the value of the cost effect function.

Step 4: Method of defuzzification was used according to the following formula (84), (85) [244]:

$$X_i = (X_{min} + \mu D(X_i)) \cdot (X_{max} - X_{min}) \quad (84)$$

$$OA(ANN_i) = X_i, \text{ where } i = 12, i = 36. \quad (85)$$

where OA represents actual effort of the particular project, that is calculated based on ANN-L12 and ANN-L36prim.

Step 5: For each iteration in our experiment, the output values are obtained according to the following formulas/measures [245] (65)-(70):

$$Deviation = |ActEffort - EstEffort| \quad (65)$$

$$MAE_i = \frac{1}{n} \sum_{i=1}^n |ActEffort - EstEffort| \quad (66)$$

$$MRE = Deviation / ActEffort \quad (67)$$

$$MRE = \frac{1}{n} \sum_{i=1}^n MRE_i \quad (68)$$

$$MMRE = mean(MRE) \quad (69)$$

For each experimental part in every iteration, the Gradient Descent with the condition $GA < 0.01$ is monitored and calculated as [114], [115] (70):

$$GA = MRE_{i1} - MRE_{i2} < 0.01,$$

$$\text{where } i=1, \dots, n \text{ } n \text{ is a number of ANN a Winner candidates.} \quad (70)$$

Step 6: Examination of the influence of input values on the change of MMRE value (86)-(89):

1. The influence of the first input parameter (Entry) and its value on the change of MMRE value is calculated as:

$$\delta 1 = mean(MMRE) - mean(MMRE_1)$$

$$\text{where } MMRE_1 \text{ is } mean(MMRE) \text{ when } X_1=0; \quad (86)$$

2. The influence of the second input parameter (Exit) and its value on the change of MMRE value is calculated as:

$$\delta 2 = mean(MMRE) - mean(MMRE_2)$$

$$\text{where } MMRE_2 \text{ is } mean(MMRE) \text{ when } X_2=0; \quad (87)$$

3. The influence of the third input parameter (Read) and its value on the change of MMRE value is calculated as:

$$\delta 3 = mean(MMRE) - mean(MMRE_3)$$

$$\text{where } MMRE_3 \text{ is } mean(MMRE)$$

$$\text{when } X_3=0; \quad (88)$$

4. The influence of the fourth input parameter (Write) and its value on the change of MMRE value is calculated as:

$$\delta 4 = \text{mean}(\text{MMRE}) - \text{mean}(\text{MMRE}_4)$$

where MMRE_4 is $\text{mean}(\text{MMRE})$

when $X_4=0$; (89)

Step 7: Correlation, Prediction

Pearson's [246], Spearman's [247] and R^2 [248] coefficients are monitored during the experiment (73).

$$\text{Correl}(X, Y) = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^N (x_i - \bar{x})^2 \sum_{i=1}^N (y_i - \bar{y})^2}} \quad (73)$$

Additionally, Prediction at 25%, 30%, and 50% is the percentage of the total number of ANNs that meet the GA criterion (74) [249].

$$\text{PRED}(x) = \frac{1}{n} \cdot \sum_{i=1}^n \begin{cases} 1, & \text{if } \text{MRE} \leq x \\ 0, & \text{otherwise} \end{cases}$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 25\%$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 30\%$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 50\%, \text{ where } k = 25, 30, 50. \quad (74)$$

The second and third parts of the experiment are conducted following the same procedures as the first part, but they involve different projects and datasets. The second part utilizes the ISBSG dataset, but it includes projects that were not included in the first part. In the third part, both the Desharnais dataset and a combined dataset are employed.

5.3. NOVEL, HYBRID UCP MODEL

In this chapter, an enhanced UCP model is introduced as an integral component of the UCP approach. The enhanced UCP model utilizes the following architectures and adequated orthogonal vector plans: L16 and L36prim:

1. UCP and ANN-L16

The first proposed architecture is ANN-L16. It consists of six input values, one hidden layer with two nodes, one output, and the total number of fifteen weighting factors ($W_i, i = 1, 15$) and their initial values are from the interval $[-1, 1]$. The Taguchi Orthogonal Array used in the construction of this proposed architecture contains two levels L1 and L2, see Figure 26, Table 25 [7], [8], [114]. Bias represents an additional weighting factor to complete the selected orthogonal plan and has a value of 1.

Figure 26. ANN-L16.

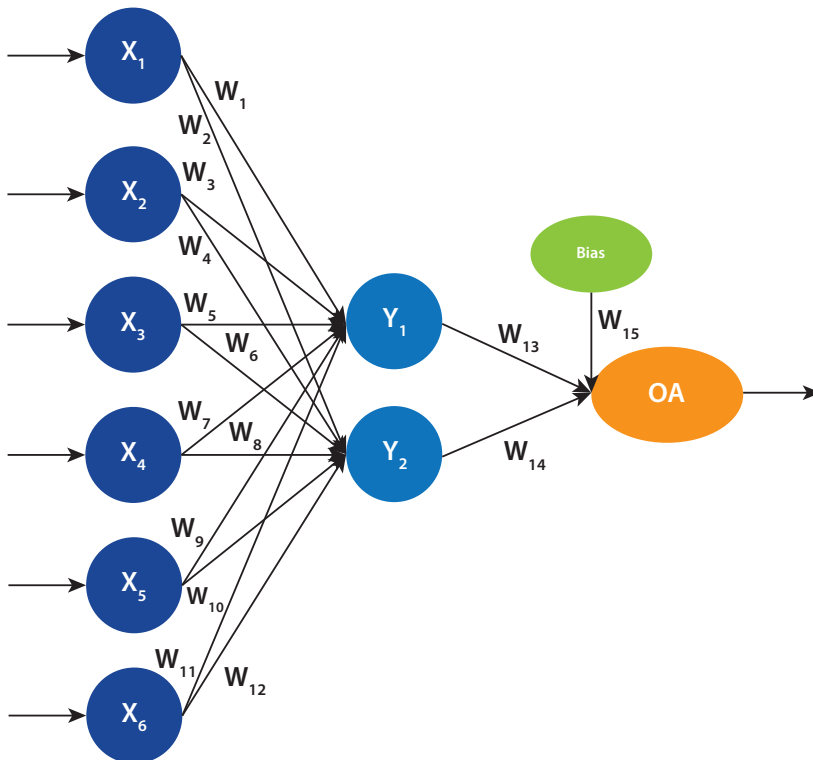


Table 25. Taguchi orthogonal vector plan (L16=2¹⁵).

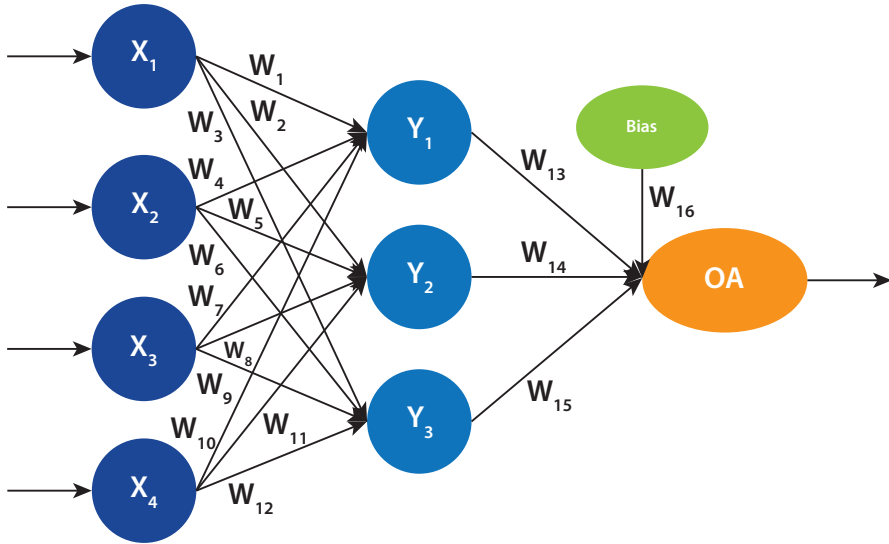
ANN-L16	W ₁	W ₂	W ₃	W ₄	W ₅	W ₆	W ₇	W ₈	W ₉	W ₁₀	W ₁₁	W ₁₂	W ₁₃	W ₁₄	W ₁₅
ANN1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1
ANN2	L1	L1	L1	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2	L2	L2
ANN3	L1	L1	L1	L2	L2	L2	L2	L1	L1	L1	L1	L2	L2	L2	L2
ANN4	L1	L1	L1	L1	L2	L2	L2	L2	L2	L2	L2	L1	L1	L1	L1
ANN5	L1	L2	L2	L1	L1	L2	L2	L1	L1	L2	L2	L1	L1	L2	L2
ANN6	L1	L2	L2	L1	L1	L2	L2	L2	L2	L1	L1	L2	L2	L1	L1
ANN7	L1	L2	L2	L2	L2	L1	L1	L1	L1	L2	L2	L2	L2	L1	L1
ANN8	L1	L2	L2	L2	L2	L1	L1	L2	L2	L1	L1	L1	L1	L2	L2
ANN9	L2	L1	L2	L1	L2	L1	L2	L1	L2	L1	L2	L1	L2	L1	L2
ANN10	L2	L1	L2	L1	L2	L1	L2	L2	L1	L2	L1	L2	L1	L2	L1
ANN11	L2	L1	L2	L2	L1	L2	L1	L1	L2	L1	L2	L2	L1	L2	L1
ANN12	L2	L1	L2	L2	L1	L2	L1	L2	L1	L2	L1	L1	L2	L1	L2
ANN13	L2	L2	L1	L1	L2	L2	L1	L1	L2	L2	L1	L1	L2	L2	L1
ANN14	L2	L2	L1	L1	L2	L2	L1	L2	L1	L1	L2	L2	L1	L1	L2
ANN15	L2	L2	L1	L2	L1	L1	L2	L1	L2	L2	L1	L2	L1	L1	L2
ANN16	L2	L2	L1	L2	L1	L1	L2	L2	L1	L1	L2	L1	L2	L2	L1

2. UCP and ANN-L36prim

The second proposed architecture is ANN-L36prim. It consists of four input values, one hidden layer with three nodes, one output, and the total number of sixteen weighting factors ($W_i, i = \overline{1,16}$), their initial values are from the interval $[-1, 0, 1]$. The Taguchi Orthogonal Array used in the construction of this proposed architecture is combined, where the first eleven parameters and the last sixteenth parameter are with three levels L1, L2, and L3, while the remaining four parameters are with two levels L1 and L2, see Figure 27, Table 26 [7], [8], [114]. Bias represents an additional weighting factor to complete the selected orthogonal plan and has a value of 1.

Table 26. Taguchi orthogonal vector plan (L36prim=3¹¹2⁴3¹).

ANN-L36 prim	W ₁	W ₂	W ₃	W ₄	W ₅	W ₆	W ₇	W ₈	W ₉	W ₁₀	W ₁₁	W ₁₂	W ₁₃	W ₁₄	W ₁₅	W ₁₆
ANN1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1	L1
ANN2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L2	L1	L1	L1	L1
ANN3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L3	L1	L1	L1	L1
ANN4	L1	L1	L1	L1	L2	L2	L2	L2	L3	L3	L3	L3	L1	L2	L2	L1
ANN5	L1	L1	L1	L1	L3	L3	L3	L3	L2	L2	L2	L2	L1	L2	L2	L1
ANN6	L3	L3	L3	L3	L1	L1	L1	L1	L2	L2	L2	L2	L1	L2	L2	L1
ANN7	L1	L1	L2	L3	L1	L2	L3	L3	L1	L1	L1	L3	L2	L1	L2	L1
ANN8	L2	L2	L3	L1	L2	L3	L1	L1	L2	L3	L3	L1	L2	L1	L2	L1
ANN9	L3	L3	L1	L2	L3	L1	L2	L2	L3	L1	L1	L2	L2	L1	L2	L1
ANN10	L1	L1	L3	L2	L1	L3	L2	L3	L2	L1	L3	L2	L2	L2	L1	L1
ANN11	L2	L2	L1	L3	L2	L1	L3	L1	L3	L2	L1	L3	L2	L2	L1	L1
ANN12	L3	L3	L2	L1	L3	L2	L1	L2	L1	L3	L2	L1	L2	L2	L1	L1
ANN13	L1	L2	L3	L1	L3	L2	L1	L3	L3	L2	L1	L2	L1	L1	L1	L2
ANN14	L2	L3	L1	L2	L1	L3	L2	L1	L1	L3	L2	L3	L1	L1	L1	L2
ANN15	L3	L1	L2	L3	L2	L1	L3	L2	L2	L1	L3	L1	L1	L1	L1	L2
ANN16	L1	L2	L3	L2	L1	L1	L3	L2	L3	L3	L2	L1	L1	L2	L2	L2
ANN17	L2	L3	L1	L3	L2	L2	L1	L3	L1	L1	L3	L2	L1	L2	L2	L2
ANN18	L3	L1	L2	L1	L3	L3	L2	L1	L2	L2	L1	L3	L1	L2	L2	L2
ANN19	L1	L2	L1	L3	L3	L3	L1	L2	L2	L1	L2	L3	L2	L1	L2	L2
ANN20	L2	L3	L2	L1	L1	L1	L2	L3	L3	L2	L3	L1	L2	L1	L2	L2
ANN21	L3	L1	L3	L2	L2	L2	L3	L1	L1	L3	L1	L2	L2	L1	L2	L2
ANN22	L1	L2	L2	L3	L3	L1	L2	L1	L1	L3	L3	L2	L2	L2	L1	L2
ANN23	L2	L3	L3	L1	L1	L2	L3	L2	L2	L1	L1	L3	L2	L2	L1	L2
ANN24	L3	L1	L1	L2	L2	L3	L1	L3	L3	L2	L2	L1	L2	L2	L1	L2
ANN25	L1	L3	L2	L1	L2	L3	L3	L1	L3	L1	L2	L2	L1	L1	L1	L3
ANN26	L2	L1	L3	L2	L3	L1	L1	L2	L1	L2	L3	L3	L1	L1	L1	L3
ANN27	L3	L2	L1	L3	L1	L2	L2	L3	L2	L3	L1	L1	L1	L1	L1	L3
ANN28	L1	L3	L2	L2	L2	L1	L1	L3	L2	L3	L1	L3	L1	L2	L2	L3
ANN29	L2	L1	L3	L3	L3	L2	L2	L1	L3	L1	L2	L1	L1	L2	L2	L3
ANN30	L3	L2	L1	L1	L1	L3	L3	L2	L1	L2	L3	L2	L1	L2	L2	L3
ANN31	L1	L3	L3	L3	L2	L3	L2	L2	L1	L2	L1	L1	L2	L1	L2	L3
ANN32	L2	L1	L1	L1	L3	L1	L3	L3	L3	L3	L2	L2	L2	L1	L2	L3
ANN33	L3	L2	L2	L2	L1	L2	L1	L1	L3	L1	L3	L3	L2	L1	L2	L3
ANN34	L1	L3	L1	L2	L3	L2	L3	L1	L2	L2	L3	L1	L2	L2	L1	L3
ANN35	L2	L1	L2	L3	L1	L3	L1	L2	L3	L3	L1	L2	L2	L2	L1	L3
ANN36	L3	L2	L3	L1	L2	L1	L2	L3	L1	L1	L2	L3	L2	L2	L1	L3

Figure 27. ANN-L36prim.

Each experiment, based on the selected approach and datasets, comprises three distinct phase:

1. *Training* of two different ANN architectures constructed according to the corresponding Taguchi orthogonal vector plans (ANN-L16 and ANN36prim);
2. *Testing* on ANN "Winner", which gave the best results (the lowest MMRE value) in the first part of the experiment, for two proposed architectures on the same dataset;
3. *Validation* on ANN "Winner" that gave the best results (the lowest MMRE value) in the first part of the experiment, for each selected architecture, but using some other i.e. different datasets.

5.3.1. Datasets utilized in UCP hybrid model

The Use Case Point Benchmark Dataset by Radek Silhavy (UCP Benchmark Dataset) [252] was used in both the first and second phases of the experiment for the enhanced UCP model. However, different datasets - specifically consolidated datasets made up of initiatives from multiple industrial companies—were used in the third stage of the experiment. The findings shown in Table 27 point to a more regular structure among the projects employed in all three of the experiment's components. This observation is based on the standard deviation values provided in Table 28.

Table 27. Information on used datasets (UCP).

	Dataset	Number of projects	Experiment
Dataset_1	UCP Benchmark Dataset	50	Training
Dataset_2	UCP Benchmark Dataset	21	Testing
Dataset_3	Combined	18	Validation1
Dataset_4	Combined Industrial projects	17	Validation2

Table 28. Basic statistics about dataset (UCP).

Dataset	No. of projects	Min [PM]	Max [PM]	Mean [PM]	Std. deviation [PM]
Dataset_1	50	5775.0	7970.0	6506.940	653.0308
Dataset_2	21	6162.6	6525.3	6393.993	118.1858
Dataset_3	18	2692.1	3246.6	2988.392	233.2270
Dataset_4	17	2176.0	3216.0	2589.400	352.0859

5.3.2. UCP hybrid model methodology

Through a series of test tests, the technique for the experimental component of the hybrid UCP approach was established. A robust design algorithm was used to determine the experiment's step-by-step flow, as shown in Figure 28.

Step 1: Input layer

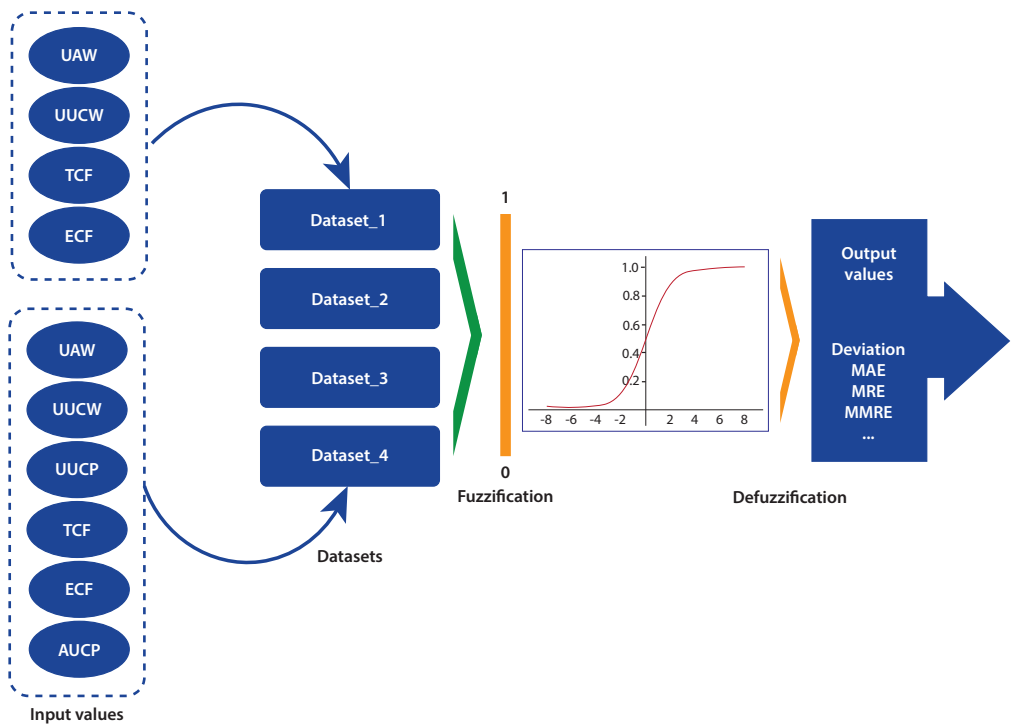
The input values of the first proposed architecture ANN-L16 are six input values, four of which are independent: UAW, UUCW, TCF, and ECF and two dependent: UUCP i AUCP.

The input values of the second proposed architecture ANN-L36prim are four independent input values: UAW, UUCW, TCF, and ECF.

Step 2: All input values are transformed according to the following formula:

The function $\mu_D(X): R \rightarrow [0, 1]$, translates the real values of input signals into coded values from the interval $[0, 1]$, in the following way: $\mu_D(X_i) = (X_i - X_{min}) / (X_{max} - X_{min})$ (min-max normalization) [121], where D is the set of data on which the experiment is performed, X_i is the input value, X_{min} is the smallest input value, and X_{max} the greatest input value on the observed dataset.

Figure 28. Robust design algorithm for performing the experiment (UCP).



Step 3: The sigmoid function, as the activation function of the hidden layer was used (31):

$$y_i = \frac{1}{1 + e^{-x_i}}, i = \overline{1, n} \quad (31)$$

The construction of the activation function is based on a combination of input values and corresponding weight coefficients W_i for each of the proposed ANN architectures.

a) Hidden and output layer functions for ANN-L16 architecture, see Figure 26, Table 25 (90)-(92):

$$Y_1 = 1 / (1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_3 + X_3 \cdot W_5 + X_4 \cdot W_7 + X_5 \cdot W_9 + X_6 \cdot W_{11})}) \quad (90)$$

$$Y_2 = 1 / (1 + e^{-(X_1 \cdot W_2 + X_2 \cdot W_4 + X_3 \cdot W_6 + X_4 \cdot W_8 + X_5 \cdot W_{10} + X_6 \cdot W_{12})}) \quad (91)$$

$$EstEffANN-L16 = 1 / (1 + e^{-(Y_1 \cdot W_{13} + Y_2 \cdot W_{14} + 1 \cdot W_{15})}) \quad (92)$$

where Y_1 , Y_2 , and Y_3 are the hidden layer functions and *EstEffortANN-L16* represents output function.

b) Hidden and output layer functions for ANN-L36prim architecture, see Figure 27, Table 26 (78)-(81):

$$Y_1 = \frac{1}{1 + e^{-(X_1 \cdot W_1 + X_2 \cdot W_4 + X_3 \cdot W_7 + X_4 \cdot W_{10})}} \quad (78)$$

$$Y_2 = \frac{1}{1 + e^{-(X_1 \cdot W_2 + X_2 \cdot W_5 + X_3 \cdot W_8 + X_4 \cdot W_{11})}} \quad (79)$$

$$Y_3 = \frac{1}{1 + e^{-(X_1 \cdot W_3 + X_2 \cdot W_6 + X_3 \cdot W_9 + X_4 \cdot W_{12})}} \quad (80)$$

$$EstEffortANN - L36prim = \frac{1}{1 + e^{-(Y_1 \cdot W_{13} + Y_2 \cdot W_{14} + Y_3 \cdot W_{15} + 1 \cdot W_{16})}} \quad (81)$$

where Y_1 , Y_2 , and Y_3 are the hidden layer functions and *EstEffortANN-L36prim* represents output function.

In the first proposed ANN-L16 architecture, an orthogonal vector plan of two levels L1 and L2, and the initial values of the weighting factors W_i that take the values from the interval $[-1, 1]$, were used.

The second proposed architecture has an orthogonal vector plan of three levels L1, L2, and L3, and the initial values of the weighting factors W_i that take the values from the interval $[-1, 0, 1]$. For each subsequent iteration, new weight factor values must be calculated as follows (e.g., for ANN-L16 architecture) [7], [119] (93):

$$W_{1L1} = \text{cost1} + \text{cost2} + \dots + \text{cost8}$$

$$W_{1L2} = \text{cost9} + \text{cost10} + \dots + \text{cost16}$$

....

$$W_{15L1} = \text{cost1} + \text{cost6} + \dots + \text{cost16}$$

$$W_{15L2} = \text{cost2} + \text{cost3} + \dots + \text{cost15}$$

$$\text{where cost}(i) = \sum MRE(ANN(i)) \quad (93)$$

For each subsequent iteration, the interval $[-1, 1]$ is divided depending on the cost effect function as follows [7], [119] (94):

$$W_{1L1\text{new}} = W_{1L1\text{old}}$$

$$W_{1L2\text{new}} = W_{1L2\text{old}} + (W_{1L3\text{old}} - W_{1L2\text{old}})/2$$

$$W_{1L3\text{new}} = W_{1L3\text{old}} \quad (94)$$

where $W_{1L1\text{old}}$, $W_{1L2\text{old}}$, and $W_{1L3\text{old}}$ are values from the previous iteration. The set of input values of each dataset converges depending on the value of the cost effect function.

Step 4: Method of defuzzification was used according to the following formula (84), (85) [244]:

$$X_i = (X_{\min} + \mu D(X_i)) \cdot (X_{\max} - X_{\min}) \quad (84)$$

$$OA(ANN_i) = X_i, \text{ where } i = 16, i = 36. \quad (85)$$

where OA represents actual effort of the particular project, that is calculated based on ANN-L12 and ANN-L36prim.

Step 5: For each iteration in our experiment, the output values are obtained according to the following formulas/measures [245] (65)-(70):

$$\text{Deviation} = |\text{ActEffort} - \text{EstEffort}| \quad (65)$$

$$MAE_i = \frac{1}{n} \sum_{i=1}^n |\text{ActEffort} - \text{EstEffort}| \quad (66)$$

$$MRE = \text{Deviation} / \text{ActEffort} \quad (67)$$

$$MRE = \frac{1}{n} \sum_{i=1}^n MRE_i \quad (68)$$

$$MMRE = \text{mean}(MRE) \quad (69)$$

For each of the experimental part in every iteration, the Gradient Descent is monitored with the condition $GA < 0.01$, calculated as [114], [115] (70):

$$GA = MRE_{i1} - MRE_{i2} < 0.01,$$

where $i=1, \dots, n$ n is a number of ANN. (70)

Step 6: Influence of dependent variables UUCP and AUCP on the change of MMRE value.

1. The influence of the input parameter UUCP and its value is calculated as (95):

$$\delta 1 = \text{mean}(\text{MMRE}) - \text{mean}(\text{MMRE}_1) \quad (95)$$

where MMRE_1 is $\text{mean}(\text{MMRE})$ when $\text{UUCP}=0$;

2. The influence of the input parameter AUCP and its value is calculated as (96):

$$\delta 2 = \text{mean}(\text{MMRE}) - \text{mean}(\text{MMRE}_2) \text{ when } \text{AUCP}=0; \quad (96)$$

Step 7: Correlation, Prediction

Pearson's [246], Spearman's [247] and R^2 [248] coefficients are monitored during the experiment (73).

$$\text{Correl}(X, Y) = \frac{\sum_{i=1}^N (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^N (x_i - \bar{x})^2 \sum_{i=1}^N (y_i - \bar{y})^2}} \quad (73)$$

Additionally, Prediction at 25%, 30%, and 50% is the percentage of the total number of ANNs that meet the GA criterion (74) [249].

$$\text{PRED}(x) = \frac{1}{n} \cdot \sum_{i=1}^n \begin{cases} 1, & \text{if } \text{MRE} \leq x \\ 0, & \text{otherwise} \end{cases}$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 25\%$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 30\%$$

$$\text{PRED}(k) = \text{count}(\text{MRE}) < 50\%, \text{ where } k = 25, 30, 50. \quad (74)$$

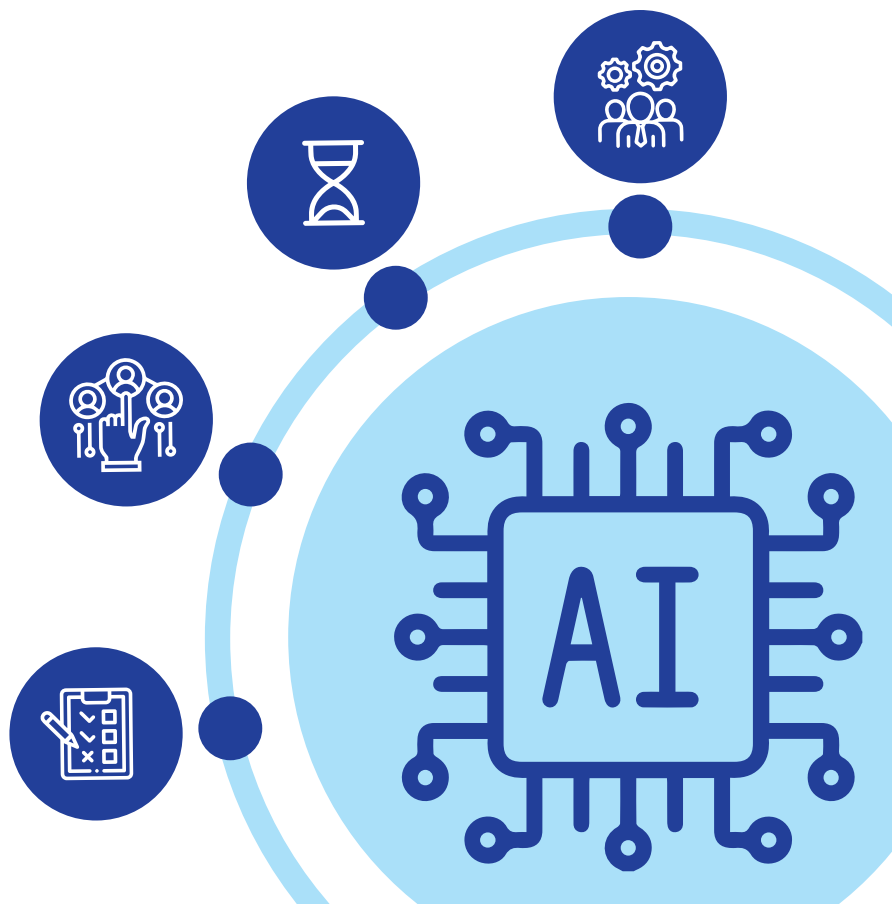
Although the second and third phases of the experiment use different projects and datasets, they operate in the same way as the first phase. The UCP Benchmark (*Mendeley*) dataset is used in the second section, however it includes projects that weren't in the first. Combination industrial datasets were used in the third section.

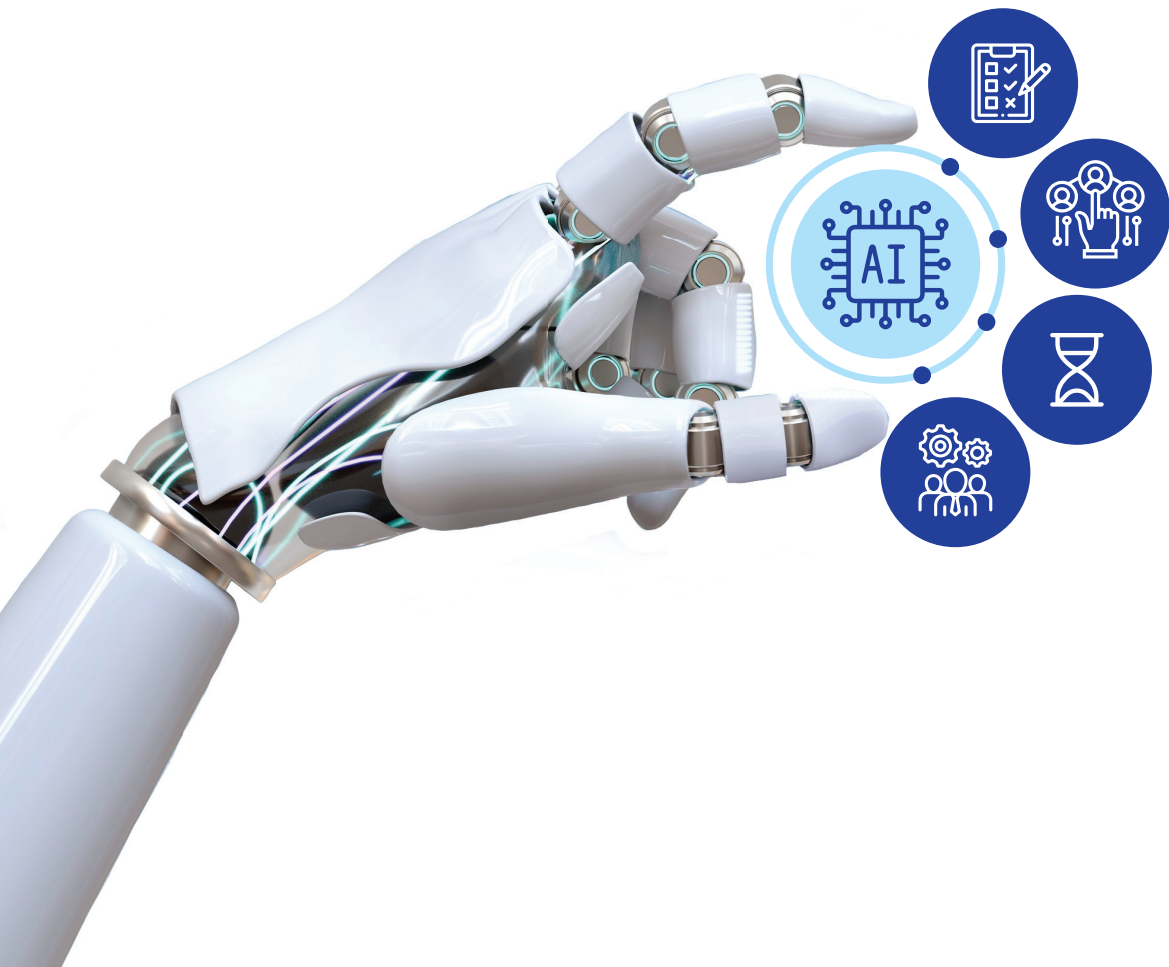


CHAPTER 6



RESULTS OF THREE NOVEL, HYBRID MODELS





6. RESULTS OF THREE NOVEL, HYBRID MODELS

6.1. COCOMO2000 HYBRID MODEL AND ANN

Within the first proposed, improved COCOMO2000 approach, four different ANN architectures based on Taguchi's orthogonal vector plans were used, designated as ANN-L9, ANN-L18, ANN-L27, and ANN-L36. In the first part of the experiment - training these ANN architectures, the used COCOMO2000 data set was divided into three clusters, according to the value of actual effort: small, medium, and large.

- Small cluster (Actual Effort <50PM);
- Medium cluster (50PM<Actual Effort<500PM);
- Large cluster (Actual Effort>500PM).

Table 29. shows the results obtained by training the first proposed ANN-L9 architecture on a small cluster. In addition to examining the MRE values for each of the nine ANN architecture candidates, the GA criterion was also monitored. Based on all MRE values in each performed iteration, the "Winner" network was determined, i.e., the ANN network with the lowest MRE value (the ANN with the best performances). Additionally, the MMRE value was calculated for each iteration. The obtained value of the "Winner" network (ANN9) is 80.7%, and the value of MMRE is 81.1%. The required GA criterion was met after seven iterations.

These results showcase the performance of the ANN-L9 architecture when trained on both small and medium clusters, with different levels of actual effort. The "Winner" network is determined based on the minimized MRE value, and the MMRE value provides an overview of the model's overall accuracy. Additionally, the GA criterion ensures that the training process is effectively optimized.

Table 29. ANN-L9 results of training part on small cluster.

ANN-L9	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6	Iteration7
ANN1	0.9026	0.8533	0.8272	0.8163	0.8115	0.8093	0.8082
ANN2	1.0384	0.9318	0.8705	0.8389	0.8230	0.8150	0.8111
ANN3	1.2779	1.0384	0.9176	0.8624	0.8347	0.8209	0.8140
ANN4	1.2576	1.0384	0.9189	0.8629	0.8350	0.8210	0.8141
ANN5	0.8573	0.8293	0.8173	0.8120	0.8095	0.8083	0.8077
ANN6	1.0384	0.9170	0.8621	0.8345	0.8208	0.8139	0.8105
ANN7	1.0384	0.9215	0.8643	0.8358	0.8214	0.8143	0.8107
ANN8	1.3158	1.0384	0.9126	0.8592	0.8329	0.8200	0.8135
ANN9	0.8093	0.8082	0.8076	0.8074	0.8073	0.8072	0.8072
GA	9	8	8	7	6	3	0
"Winner" ANN	0.8093	0.8082	0.8076	0.8074	0.8073	0.8072	0.8072
MMRE	1.0595	0.9307	0.8665	0.8366	0.8218	0.8144	0.8108

Table 30. ANN-L9 results of training part on medium cluster.

ANN-L9	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6	Iteration7
ANN1	0.5440	0.5147	0.5038	0.4990	0.4966	0.4954	0.8082
ANN2	0.6121	0.5578	0.5238	0.5081	0.5012	0.4978	0.8111
ANN3	0.7289	0.6121	0.5496	0.5187	0.5060	0.5055	0.8140
ANN4	0.7179	0.6121	0.5507	0.5195	0.5064	0.5003	0.8141
ANN5	0.5137	0.5035	0.4987	0.4965	0.4954	0.4948	0.8077
ANN6	0.6121	0.5511	0.5193	0.5062	0.5003	0.4973	0.8105
ANN7	0.6121	0.5513	0.5199	0.5067	0.5005	0.4974	0.8107
ANN8	0.7510	0.6121	0.5478	0.5174	0.5055	0.4996	0.8135
ANN9	0.4953	0.4948	0.4945	0.4944	0.4943	0.4943	0.8072
GA	9	8	7	6	3	0	0
"Winner" ANN	0.4953	0.4948	0.4945	0.4944	0.4943	0.4943	0.8072
MMRE	0.6208	0.5566	0.5231	0.5074	0.5007	0.498	0.8108

In the first novel, hybrid COCOMO2000 model, the ANN-L9 architecture was also trained on the large cluster. Here are the results from Table 31:

Table 31: Results for ANN-L9 on the Large Cluster:

- The "Winner" network (ANN9) achieved an MRE value of 206.9%, with an MMRE value of 207.6%.
- After 8 iterations, the GA criteria was accomplished. These findings show how the ANN-L9 architecture performs when trained on a large cluster, when software projects require a significant amount of actual work. The "Winner" network is chosen based on the network with the lowest MRE value, and the model's overall accuracy is shown by the MMRE value. The GA criterion ensures that training is carried out to its full potential.

Table 31. ANN-L9 results of training part on large cluster.

ANN-L9	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6	Iteration7	Iteration8
ANN1	2.6835	2.3669	2.2043	2.1321	2.0995	2.0840	2.0764	2.0708
ANN2	3.2240	2.7535	2.4259	2.2484	2.1580	2.1133	2.0911	2.0745
ANN3	4.0305	3.2240	2.6704	2.3710	2.2190	2.1430	2.1059	2.0982
ANN4	4.0277	3.2240	2.6696	2.3705	2.2188	2.1429	2.1059	2.0781
ANN5	2.3114	2.1785	2.1069	2.0938	2.0812	2.0750	2.0719	2.0697
ANN6	3.2240	2.6843	2.3905	2.2298	2.1487	2.1086	2.0888	2.0739
ANN7	3.2240	2.6560	2.3931	2.2133	2.1401	2.1044	2.0867	2.0733
ANN8	4.2484	3.2240	2.6462	2.3547	2.2101	2.1384	2.1036	2.0776
ANN9	2.1156	2.0912	2.0800	2.0744	2.0717	2.0703	2.0696	2.0691
GA	9	9	9	8	8	7	6	0
"Winner" ANN	2.1156	2.0912	2.0800	2.0744	2.0717	2.0703	2.0696	2.0691
MMRE	3.2321	2.7114	2.3985	2.2320	2.1497	2.1089	2.0889	2.0761

Figure 29 displays the minimized MMRE results for all 3 clusters used in the ANN-L9 training procedure with the COCOMO2000 dataset. Here are the key observations:

- The MMRE value is the lowest for the small cluster, standing at 49.8%.
- The MMRE value increases for the medium cluster to 78.2% and further for the large cluster, reaching 207.6%.

This demonstrates that the ANN-L9 architecture performs most accurately when trained on the small cluster, which contains software projects with lower actual efforts.

Figure 30 shows the minimized MMRE results for the "Winner" ANN obtained in all 3 clusters. Key points to note:

- The minimized MMRE "Winner" ANN value is the lowest for the small cluster after seven iterations, at 49.4%.
- The minimized MMRE "Winner" ANN value increases for the medium cluster to 77.2% and further for the large cluster, reaching 206.9%.

These results indicate that the ANN-L9 architecture's "Winner" network consistently performs best when trained on the small cluster, providing more accurate effort estimations for software projects with lower actual efforts.

Figure 29. MMRE results in three parts of the experiment for all clusters (ANN-L9).

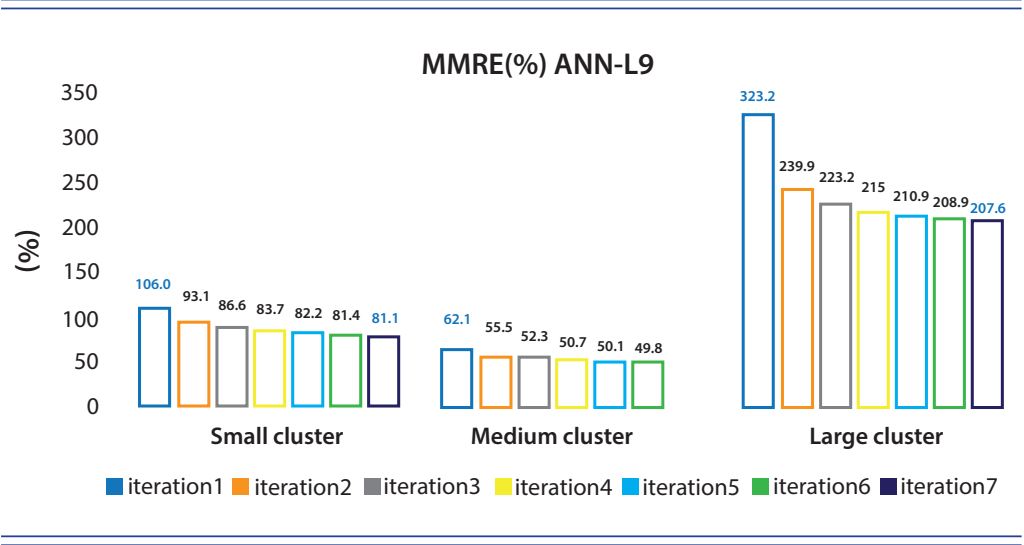


Figure 30. MMRE for "Winner" ANN9 (ANN-L9).

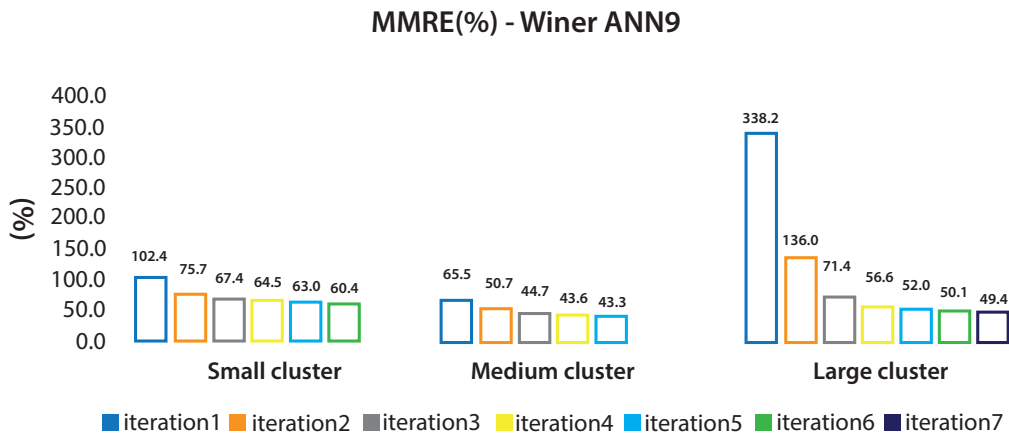


Table 32 presents the results of training the second proposed ANN-L18 architecture on a small cluster using the COCOMO2000 dataset. Here are the key findings:

- MRE values were monitored for all 18 ANNs.
- The "Winner" ANN network, identified as ANN5, achieved the lowest MRE value of 63.3%.
- The minimized MMRE value for the "Winner" network was 63.7%.
- After 6 iterations the GA criterion was accomplished.

This indicates that the ANN-L18 architecture, specifically ANN5, performed well in estimating the effort for software projects in the small cluster, with projects having lower actual efforts.

Table 32. ANN-L18 results of training part on small cluster.

ANN-L18	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6
ANN1	0.6927	0.6600	0.6432	0.6422	0.6347	0.6337
ANN2	0.9060	0.7421	0.6767	0.6521	0.6415	0.6371
ANN3	1.3997	0.9060	0.7304	0.6720	0.6487	0.6405
ANN4	1.1008	0.9060	0.7367	0.6787	0.6504	0.6412
ANN5	0.6696	0.6478	0.6391	0.6355	0.6340	0.6334
ANN6	0.9060	0.7304	0.6701	0.6480	0.6403	0.6365
ANN7	1.1462	0.8225	0.7031	0.6653	0.6418	0.6392
ANN8	0.8565	0.7219	0.6697	0.6483	0.6403	0.6365
ANN9	0.7440	0.6809	0.6537	0.6439	0.6374	0.6351
ANN10	0.7424	0.6798	0.6531	0.6452	0.6409	0.6350
ANN11	1.1189	0.8186	0.7007	0.6648	0.6452	0.6389
ANN12	0.8670	0.7318	0.6735	0.6507	0.6410	0.6368
ANN13	1.1056	0.8085	0.6985	0.6635	0.6452	0.6389
ANN14	0.9340	0.7390	0.6745	0.6497	0.6411	0.6368
ANN15	0.7678	0.6824	0.6525	0.6432	0.6370	0.6345
ANN16	0.7659	0.6849	0.6546	0.6454	0.6376	0.6352
ANN17	1.1132	0.8138	0.6997	0.6633	0.6453	0.6389
ANN18	0.9126	0.7287	0.6696	0.6474	0.6399	0.6363
GA	18	18	17	12	7	0
"Winner" ANN	0.6696	0.6478	0.6391	0.6355	0.6340	0.6334
MMRE	0.9305	0.7503	0.6777	0.6533	0.6413	0.6369

Table 33 presents the results of training the second proposed ANN-L18 architecture on the medium cluster using the COCOMO2000 dataset. Here are the key findings:

- MRE values were monitored for all 18 ANNs.
- The "Winner" ANN network, identified as ANN5, achieved the lowest MRE value of 44.6%.
- The minimized MMRE value for the "Winner" ANN network was 44.8%.
- After 6 iterations the GA criterion was satisfied.

This indicates that the ANN-L18 architecture, specifically ANN5, performed well in estimating the effort for software projects in the medium cluster, with projects having moderate actual efforts.

Table 33. ANN-L18 results of training part on medium cluster.

ANN-L18	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5
ANN1	0.4828	0.4516	0.4421	0.4395	0.4385
ANN2	0.6031	0.5035	0.4618	0.4471	0.4419
ANN3	0.8619	0.6031	0.4935	0.4572	0.4455
ANN4	0.7139	0.6031	0.4973	0.4593	0.4463
ANN5	0.4596	0.4459	0.4406	0.4389	0.4382
ANN6	0.6031	0.4963	0.4580	0.4454	0.4413
ANN7	0.7686	0.5561	0.4785	0.4529	0.4420
ANN8	0.5583	0.4874	0.4563	0.4451	0.4412
ANN9	0.5069	0.4641	0.4484	0.4425	0.4399
ANN10	0.5093	0.4640	0.4481	0.4421	0.4416
ANN11	0.7199	0.5516	0.4779	0.4525	0.4438
ANN12	0.5648	0.4943	0.4594	0.4462	0.4417
ANN13	0.7246	0.5463	0.4758	0.4520	0.4437
ANN14	0.6232	0.5001	0.4596	0.4462	0.4417
ANN15	0.5145	0.4639	0.4477	0.4420	0.4397
ANN16	0.5174	0.4643	0.4482	0.4425	0.4399
ANN17	0.7323	0.5459	0.4756	0.4521	0.4438
ANN18	0.5930	0.4907	0.4563	0.4449	0.4411
GA	18	16	12	7	0
"Winner" ANN	0.4596	0.4459	0.4406	0.4406	0.4382
MMRE	0.6143	0.5073	0.4625	0.4471	0.4418

Table 34 displays the results of training the second proposed ANN-L18 architecture on the large cluster using the COCOMO2000 dataset. Here are the key findings:

- MRE values were tracked for all 18 ANNs.
- The "Winner" ANN network, identified as ANN5, achieved the lowest MRE value of 137.0%.
- The minimized MMRE value for the "Winner" ANN network was 137.3%.
- After 9 iterations the GA criterion was achieved.

This indicates that the ANN-L18 architecture, specifically ANN5, performed well in estimating the effort for software projects in the large cluster, with projects having substantial actual efforts.

Table 34. ANN-L18 results of training part on large cluster.

ANN-L18	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6	Iteration7	Iteration8
ANN1	2.2856	1.7571	1.5579	1.5332	1.3971	1.3722	1.3474	1.2977
ANN2	3.2217	2.3971	1.8901	1.6456	1.4775	1.4123	1.3471	1.2168
ANN3	4.6984	3.2217	2.2804	1.8274	1.5625	1.4535	1.3445	1.1266
ANN4	4.0394	3.2217	2.2956	1.8558	1.5755	1.4601	1.3447	1.1138
ANN5	1.7055	1.5347	1.4575	1.3962	1.3718	1.3600	1.3483	1.3404
ANN6	3.2217	2.3458	2.3085	1.6000	1.4670	1.4071	1.3473	1.2277
ANN7	4.3513	2.8428	2.0895	1.7318	1.4675	1.4321	1.3967	1.3259
ANN8	2.8017	2.1803	1.7853	1.5675	1.4572	1.4025	1.3478	1.2384
ANN9	2.4713	1.8972	1.6139	1.4969	1.4168	1.4168	1.4168	1.4168
ANN10	2.5235	1.9305	1.6313	1.5409	1.4638	1.4072	1.3506	1.2374
ANN11	4.0333	2.7887	2.0845	1.7657	1.5195	1.4327	1.3459	1.1723
ANN12	2.8972	2.2917	1.8397	1.6219	1.4697	1.4087	1.3477	1.2257
ANN13	4.1009	2.7487	2.0528	1.7043	1.5125	1.4541	1.3957	1.2788
ANN14	3.3436	2.3450	1.8372	1.5963	1.4644	1.4058	1.3472	1.2300
ANN15	2.3735	1.8501	1.5985	1.4911	1.4101	1.3789	1.3477	1.2853
ANN16	2.4273	1.8440	1.5911	1.5019	1.4078	1.3778	1.3477	1.2876
ANN17	4.1881	2.7463	2.0388	1.6937	1.5084	1.4273	1.3461	1.1839
ANN18	3.0075	2.1786	1.7659	1.5612	1.4491	1.3982	1.3473	1.2456
GA	18	17	16	12	8	6	3	0
"Winner" ANN	1.7055	1.5347	1.4575	1.3962	1.3718	1.3600	1.3565	1.3404
MMRE	3.2051	2.3401	1.8732	1.6184	1.4666	1.4115	1.4062	1.3732

The MMRE results for all three clusters for which the ANN-L18 training procedure had been carried out using the COCOMO2000 dataset are depicted in Figure 31. The results demonstrate that the large cluster has the greatest MMRE value at 137.0% while the tiny cluster has the lowest value at 44.6%.

The "Winner" ANN network's MMRE results for each of the 3 clusters are shown in Figure 32, which reflects the ANN-L18 training procedure using the COCOMO2000 dataset. It is clear that the MMRE "Winner" score for the small cluster is lowest after the sixth iteration at 44.8% and greatest after the ninth iteration for the large cluster at 137.3%.

Figure 31. MMRE results in three parts of the experiment for all clusters (ANN-L18).

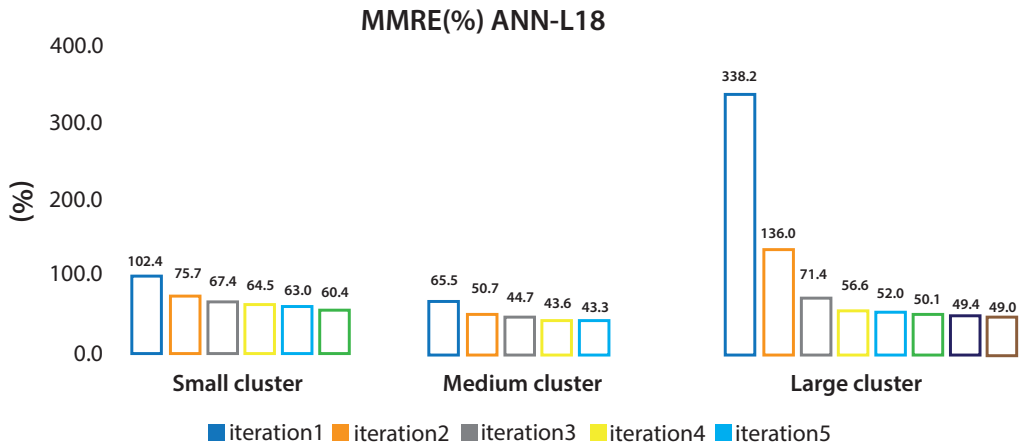
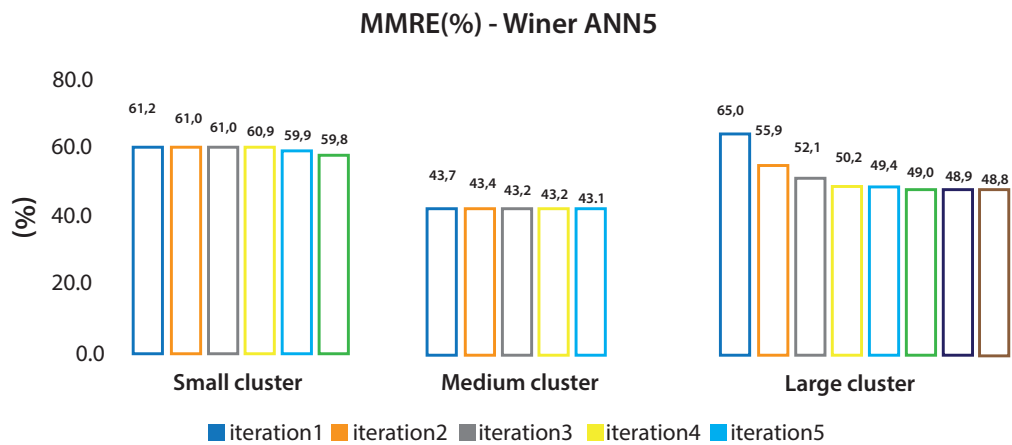


Figure 32. MMRE for "Winner" ANN5 (ANN-L18).



The results of training the third suggested ANN-L27 design on a small cluster are shown in Table 35. The monitoring of the GA criterion is shown in the table along with the MRE values for each of the 27 ANNs. The "Winner" ANN network, or ANN network with the lowest MRE value, was selected based on all MRE values in each iteration. Each iteration also included a calculation of the MMRE value. The MRE value for the "Winner" network (ANN5) was 59.8%, and the MMRE value was 60.4%. After six repetitions, the required GA condition was satisfied.

Table 35. ANN-L27 results of training part on small cluster.

ANN-L18	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6
ANN1	0.6310	0.6211	0.6145	0.6114	0.6050	0.6035
ANN2	0.9378	0.6991	0.6390	0.6185	0.6012	0.6012
ANN3	1.9290	1.6275	0.7495	0.6539	0.6203	0.6071
ANN4	1.7768	1.0431	0.7249	0.6491	0.6244	0.6086
ANN5	0.6122	0.6103	0.6100	0.6094	0.6074	0.5980
ANN6	0.7755	0.6568	0.6226	0.6129	0.6007	0.6016
ANN7	1.1557	0.7661	0.6570	0.6266	0.6112	0.6048
ANN8	1.6486	0.9378	0.6973	0.6399	0.6207	0.6061
ANN9	0.6213	0.6169	0.6140	0.6114	0.6024	0.6048
ANN10	1.5089	0.8575	0.6776	0.6336	0.6208	0.6054
ANN11	0.6728	0.6335	0.6165	0.6121	0.6032	0.6028
ANN12	0.9806	0.7219	0.6472	0.6219	0.6124	0.6051
ANN13	0.7474	0.6611	0.6259	0.6142	0.6057	0.6018
ANN14	1.1506	0.7363	0.6457	0.6208	0.5990	0.6009
ANN15	0.9078	0.6901	0.6394	0.6190	0.6014	0.6022
ANN16	0.7512	0.6614	0.6295	0.6165	0.6064	0.6026
ANN17	1.1578	0.7615	0.6568	0.6265	0.6096	0.6044
ANN18	0.8996	0.6912	0.6363	0.6174	0.6042	0.6031
ANN19	1.0255	0.7098	0.6408	0.6184	0.6014	0.6021
ANN20	1.4649	0.8537	0.7674	0.6340	0.6170	0.6063
ANN21	0.6680	0.6349	0.6168	0.6124	0.6063	0.6026
ANN22	0.9519	0.7199	0.6504	0.6236	0.6110	0.6050
ANN23	0.7114	0.6543	0.6274	0.6159	0.6144	0.6038
ANN24	1.1687	0.7669	0.6570	0.6262	0.6055	0.6055
ANN25	0.9192	0.6945	0.6354	0.6167	0.6010	0.6013
ANN26	0.7755	0.6690	0.6310	0.6170	0.6040	0.6027
ANN27	1.1016	0.7327	0.6498	0.6234	0.6063	0.6017
GA	27	24	23	19	12	0
"Winner" ANN	0.6122	0.6103	0.6100	0.6094	0.5990	0.5980
MMRE	1.0241	0.7566	0.6737	0.6449	0.6304	0.6037

The third suggested ANN-L27 architecture's training results on the medium cluster are shown in Table 36. The monitoring of the GA criterion is shown in the table along with the MRE values for each of the 27 ANNs. The "Winner" network, or ANN network with the lowest MRE value, was selected based on all MRE values in each iteration. Each iteration also included a calculation of the MMRE value. The MRE value for the "Winner" network (ANN5) was 43.2%, and the MMRE value was 43.3%. After 5 repetitions, the required GA condition was satisfied.

Table 36. ANN-L27 results of training part on medium cluster.

ANN-L27	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5
ANN1	0.4384	0.4372	0.4339	0.4333	0.4315
ANN2	0.6031	0.4727	0.4417	0.4356	0.4326
ANN3	1.1374	0.9861	0.5023	0.4440	0.4343
ANN4	1.0667	0.6592	0.4875	0.4445	0.4358
ANN5	0.4370	0.4352	0.4338	0.4333	0.4319
ANN6	0.5181	0.4465	0.4348	0.4328	0.4317
ANN7	0.7090	0.5020	0.4495	0.4388	0.4334
ANN8	1.0269	0.6031	0.4682	0.4405	0.4341
ANN9	0.4434	0.4344	0.4324	0.4320	0.4310
ANN10	0.9410	0.5722	0.4605	0.4398	0.4336
ANN11	0.4487	0.4370	0.4345	0.4331	0.4322
ANN12	0.6236	0.4900	0.4451	0.4371	0.4329
ANN13	0.4899	0.4539	0.4386	0.4346	0.4327
ANN14	0.7199	0.4906	0.4445	0.4366	0.4329
ANN15	0.5942	0.4630	0.4382	0.4341	0.4320
ANN16	0.4917	0.4466	0.4365	0.4333	0.4318
ANN17	0.7495	0.5213	0.4499	0.4382	0.4332
ANN18	0.5713	0.4639	0.4422	0.4359	0.4330
ANN19	0.6980	0.4762	0.4412	0.4354	0.4326
ANN20	0.9032	0.5770	0.4621	0.4398	0.4338
ANN21	0.4457	0.4388	0.4353	0.4339	0.4323
ANN22	0.6218	0.4893	0.4444	0.4366	0.4325
ANN23	0.4670	0.4458	0.4385	0.4345	0.4324
ANN24	0.7483	0.5218	0.4519	0.4395	0.4338

ANN25	0.5832	0.4712	0.4432	0.4362	0.4331
ANN26	0.5181	0.4536	0.4383	0.4340	0.4323
ANN27	0.6804	0.4860	0.4430	0.4361	0.4326
GA	27	23	21	8	0
"Winner" ANN	0.4370	0.4344	0.4324	0.4320	0.4310
MMRE	0.6546	0.5065	0.4471	0.4364	0.4328

The results of training the third suggested ANN-L27 architecture on the big cluster are shown in Table 37. The monitoring of the GA criterion is shown in the table along with the MRE values for each of the 27 ANN architectures. The "Winner" network, or ANN network with the lowest MRE value, was selected based on all MRE values in each iteration. Each iteration also included a calculation of the MMRE value. The MRE value for the "Winner" network (ANN5) was 48.8%, and the MMRE value was 49.0%. After 8 iterations, the required GA condition was satisfied.

Table 37. ANN-L27 results of training part on large cluster.

ANN-L27	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6	Iteration7	Iteration8
ANN1	0.8118	0.5817	0.5246	0.5024	0.4937	0.4902	0.4885	0.4877
ANN2	3.3434	1.3232	0.6881	0.5716	0.5237	0.5029	0.4944	0.4906
ANN3	6.3782	3.3434	1.1184	0.6868	0.5600	0.5187	0.5006	0.4936
ANN4	6.1151	2.8706	1.0432	0.6536	0.5496	0.5143	0.4989	0.4928
ANN5	0.9728	0.6767	0.5674	0.5220	0.5023	0.4942	0.4905	0.4887
ANN6	2.6104	0.9960	0.6420	0.5473	0.5135	0.4985	0.4925	0.4910
ANN7	4.0522	1.4850	0.7570	0.5770	0.5257	0.5038	0.4948	0.4908
ANN8	5.9577	2.3527	0.9094	0.6159	0.5382	0.5093	0.4998	0.4918
ANN9	0.6504	0.5594	0.5208	0.5015	0.4941	0.4904	0.4886	0.4877
ANN10	5.4448	1.9726	0.8363	0.5973	0.5327	0.5069	0.4961	0.4914
ANN11	1.7757	0.8519	0.6073	0.5379	0.5094	0.4969	0.4918	0.4893
ANN12	3.4136	1.2364	0.7039	0.5641	0.5206	0.5015	0.4939	0.4903
ANN13	2.1496	0.9207	0.6229	0.5415	0.5109	0.4976	0.4921	0.4896
ANN14	4.1912	1.3973	0.7341	0.5687	0.5222	0.5022	0.4942	0.4905
ANN15	3.4102	1.3117	0.7213	0.5679	0.5222	0.5022	0.4941	0.4905
ANN16	2.2011	1.0095	0.6531	0.5520	0.5156	0.4993	0.4930	0.4899
ANN17	4.4730	1.6015	0.7776	0.5817	0.5159	0.5045	0.4951	0.4909
ANN18	3.0551	1.0895	0.5822	0.5530	0.5157	0.4995	0.4931	0.4899
ANN19	4.0497	1.2850	0.6981	0.5599	0.5184	0.5004	0.4935	0.4901
ANN20	5.2322	1.8373	0.8142	0.5906	0.5300	0.5057	0.4956	0.4912
ANN21	1.5027	0.7971	0.5922	0.5322	0.5069	0.4960	0.4914	0.4891
ANN22	3.5553	1.3456	0.7283	0.5705	0.5232	0.5027	0.4943	0.4905
ANN23	1.8601	0.9019	0.6243	0.5431	0.5117	0.4979	0.4923	0.4896
ANN24	4.4058	1.4439	0.7444	0.5715	0.5233	0.5027	0.4944	0.4906
ANN25	3.1148	1.0546	0.6540	0.5477	0.5134	0.4987	0.4926	0.4897
ANN26	2.6104	1.0176	0.6444	0.5477	0.5134	0.4985	0.4926	0.4897
ANN27	3.9873	1.4593	0.7541	0.5748	0.5247	0.5033	0.4946	0.4907
GA	27	27	27	27	25	24	4	0
"Winner" ANN	0.6504	0.5594	0.5208	0.5015	0.4937	0.4902	0.4885	0.4877
MMRE	3.3824	1.3601	0.7135	0.5659	0.5197	0.5014	0.4938	0.4903

The MMRE results for all three clusters in which the ANN-L27 training procedure was executed out using the COCOMO2000 dataset are demonstrated in Figure 33. The medium cluster has the lowest MMRE value (43.3%) and the tiny cluster has the greatest MMRE value (60.4%). The MMRE outcomes for the "Winner" network across all three clusters for which the ANN-L27 training method was carried out using the COCOMO2000 dataset are shown in Figure 34. The MMRE "Winner" value is lowest after the fifth iteration in the medium cluster (43.2%) and greatest after the sixth iteration in the small cluster (59.8%).

Figure 33. MMRE results in three parts of the experiment for all clusters (ANN-L27).

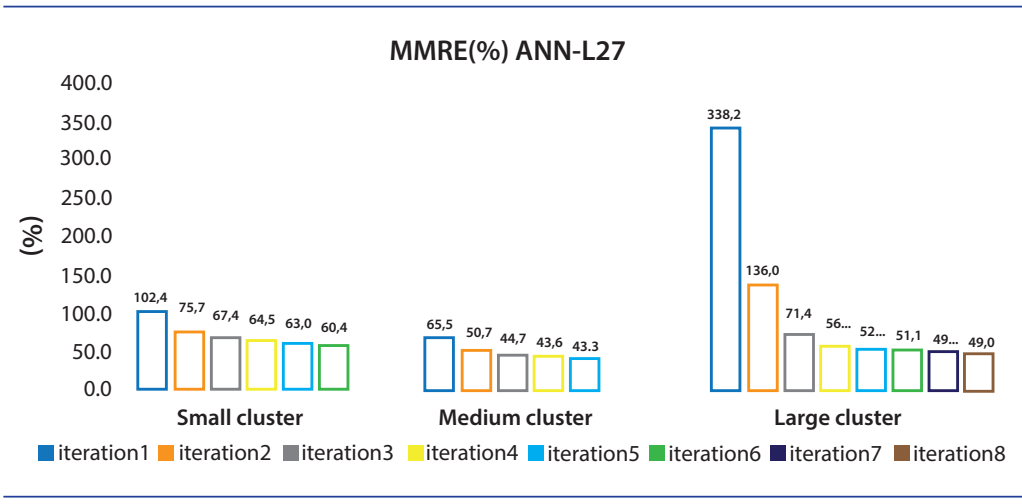
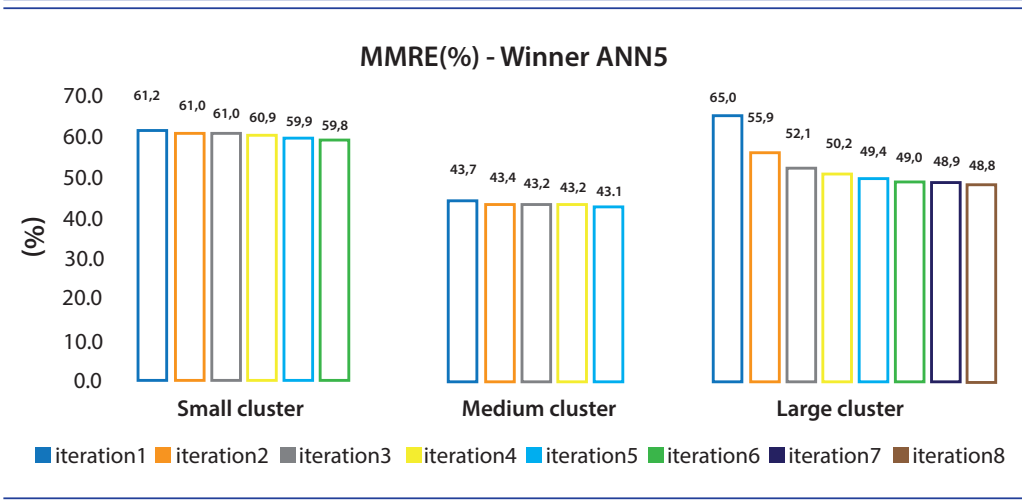


Figure 34. MMRE for "Winner" ANN5 (ANN-L27).



The results of training the fourth suggested ANN-L36 architecture on a small cluster are shown in Table 38. It includes the GA criterion monitoring as well as the MRE values for each of the 36 ANNs. Based on all MRE values in each iteration, the "Winner" network, which is the ANN with the lowest MRE value, was selected. Each iteration also included a calculation of the MMRE value. The MRE value of the "Winner" network in this example (ANN23) is 52.4%, and the MMRE value is 52.6%. After 5 iterations, the GA criteria was satisfied.

Table 38. ANN-L36 results of training part on small cluster.

ANN-L36	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5
ANN1	1.0750	0.6596	0.5535	0.5295	0.5261
ANN2	0.8307	0.5933	0.5370	0.5295	0.5263
ANN3	1.0778	0.5919	0.5377	0.5299	0.5265
ANN4	0.7791	0.5919	0.5431	0.5317	0.5272
ANN5	0.7294	0.5811	0.5373	0.5290	0.5260
ANN6	0.8307	0.5954	0.5390	0.5289	0.5259
ANN7	0.5919	0.5453	0.5337	0.5285	0.5259
ANN8	1.5664	0.8307	0.5933	0.5355	0.5287
ANN9	0.5824	0.5437	0.5317	0.5273	0.5252
ANN10	0.7778	0.5868	0.5422	0.5303	0.5266
ANN11	0.5377	0.5373	0.5319	0.5273	0.5253
ANN12	1.4382	0.7747	0.5815	0.5335	0.5278
ANN13	0.6346	0.5500	0.5336	0.5283	0.5257
ANN14	0.7186	0.5743	0.5368	0.5295	0.5263
ANN15	1.3281	0.6930	0.5681	0.5328	0.5277
ANN16	1.3844	0.7512	0.5771	0.5343	0.5283
ANN17	0.6506	0.5679	0.5406	0.5306	0.5268
ANN18	0.5369	0.5354	0.5322	0.5278	0.5256
ANN19	0.5371	0.5336	0.5314	0.5275	0.5254
ANN20	1.4595	0.7612	0.5785	0.5335	0.5272
ANN21	0.7103	0.5911	0.5383	0.5302	0.5266
ANN22	0.7808	0.6843	0.5580	0.5328	0.5277
ANN23	0.5338	0.5262	0.5262	0.5241	0.5237
ANN24	1.2434	0.7121	0.5646	0.5327	0.5276

ANN25	0.6925	0.5612	0.5334	0.5278	0.5257
ANN26	0.6913	0.5699	0.5363	0.5294	0.5262
ANN27	1.4412	0.7345	0.5647	0.5323	0.5274
ANN28	0.6305	0.5475	0.5318	0.5275	0.5254
ANN29	1.0769	0.6485	0.5465	0.5305	0.5267
ANN30	0.9373	0.6384	0.5516	0.5317	0.5273
ANN31	1.0277	0.6313	0.5364	0.5291	0.5260
ANN32	0.9366	0.6364	0.5497	0.5321	0.5275
ANN33	0.5965	0.5474	0.5331	0.5277	0.5254
ANN34	1.4717	0.7564	0.5747	0.5332	0.5278
ANN35	0.9262	0.6067	0.5373	0.5297	0.5264
ANN36	0.5558	0.5344	0.5282	0.5257	0.5245
GA	36	32	31	16	0
"Winner" ANN23	0.5338	0.5262	0.5262	0.5241	0.5237
MMRE	0.8978	0.6201	0.5464	0.5300	0.5264

The fourth suggested ANN-L36 architecture's training results are shown in Table 39 for the medium cluster. It includes the GA criterion monitoring as well as the MRE values for each of the 36 ANNs. Based on all MRE values in each iteration, the "Winner" network, which is the ANN with the lowest MRE value, was selected. Each iteration also included a calculation of the MMRE value. The MRE value of the "Winner" network in this example (ANN18) is 42.9%, and the MMRE value is 43.0%. After 5 iterations, the GA criteria was satisfied.

The fourth suggested ANN-L36 architecture's training outcomes on a big cluster are shown in Table 40. It includes the GA criterion monitoring as well as the MRE values for each of the 36 ANNs. Based on all MRE values in each iteration, the "Winner" network, which is the ANN with the lowest MRE value, was selected. Each iteration also included a calculation of the MMRE value. The MRE value of the "Winner" network in this example (ANN23) is 49.5%, and the MMRE value is 49.9%. After 8 repetitions, the GA criteria was satisfied.

The MMRE results for each of the three clusters on which the ANN-L36 training method was carried out using the COCOMO2000 dataset are shown in Figure 35. It demonstrates that the small cluster (52.6%) has the greatest MMRE score and the medium cluster (43.0%) has the lowest.

The "Winner" network's MMRE results for all three clusters where the ANN-L36 training procedure was carried out using the COCOMO2000 dataset are shown in Figure 36. It suggests that after five iterations, the value of MMRE "Winner" is lowest in the medium cluster (42.9%), and greatest in the small cluster (52.4%).

Table 39. ANN-L36 results of training part on medium cluster.

ANN-L36	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5
ANN1	0.7478	0.5171	0.4503	0.4335	0.4304
ANN2	0.6121	0.4634	0.4374	0.4318	0.4300
ANN3	0.7528	0.4570	0.4348	0.4313	0.4297
ANN4	0.5759	0.4570	0.4388	0.4328	0.4306
ANN5	0.5425	0.4605	0.4374	0.4312	0.4296
ANN6	0.6121	0.4665	0.4378	0.4312	0.4296
ANN7	0.4570	0.4381	0.4331	0.4307	0.4295
ANN8	1.0545	0.6121	0.4668	0.4377	0.4310
ANN9	0.4512	0.4364	0.4314	0.4297	0.4289
ANN10	0.5770	0.4618	0.4381	0.4304	0.4300
ANN11	0.4375	0.4345	0.4318	0.4301	0.4292
ANN12	0.9752	0.5777	0.4600	0.4360	0.4306
ANN13	0.4809	0.4423	0.4333	0.4307	0.4295
ANN14	0.5364	0.4483	0.4352	0.4316	0.4299
ANN15	0.9110	0.5432	0.4534	0.4340	0.4305
ANN16	0.9484	0.5638	0.4588	0.4360	0.4312
ANN17	0.4953	0.4523	0.4381	0.4324	0.4303
ANN18	0.4345	0.4345	0.4326	0.4307	0.4295
ANN19	0.4364	0.4354	0.4319	0.4301	0.4293
ANN20	0.9951	0.5710	0.4589	0.4355	0.4307
ANN21	0.5358	0.4593	0.4372	0.4314	0.4298
ANN22	0.8326	0.5147	0.4374	0.4331	0.4306
ANN23	0.4493	0.4363	0.4310	0.4295	0.4290
ANN24	0.8640	0.5363	0.4528	0.4338	0.4306
ANN25	0.5193	0.4464	0.4343	0.4314	0.4298
ANN26	0.5204	0.4466	0.4354	0.4319	0.4301
ANN27	0.9806	0.5555	0.4540	0.4343	0.4308

ANN28	0.4805	0.4392	0.4332	0.4308	0.4295
ANN29	0.7657	0.4952	0.4381	0.4323	0.4302
ANN30	0.6735	0.4879	0.4443	0.4331	0.4306
ANN31	0.7330	0.4859	0.4408	0.4312	0.4296
ANN32	0.6723	0.4845	0.4429	0.4328	0.4304
ANN33	0.4567	0.4379	0.4321	0.4303	0.4293
ANN34	0.9970	0.5663	0.4566	0.4351	0.4310
ANN35	0.6656	0.4694	0.4385	0.4322	0.4302
ANN36	0.4417	0.4343	0.4314	0.4298	0.4291
GA	36	32	26	11	0
"Winner" ANN23	0.4345	0.4343	0.4310	0.4295	0.4289
MMRE	0.6562	0.4825	0.4411	0.4322	0.4300

Table 40. ANN-L36 results of training part on large cluster.

ANN-L36	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6	Iteration7	Iteration8
ANN1	4.1635	1.6235	0.7459	0.5744	0.5296	0.5112	0.5028	0.4987
ANN2	3.2240	1.3560	0.7411	0.5811	0.5340	0.5135	0.5040	0.4993
ANN3	4.2005	1.6235	0.7987	0.5957	0.5399	0.5162	0.5053	0.5000
ANN4	2.8892	1.6235	0.8403	0.6103	0.5453	0.5189	0.5064	0.5005
ANN5	2.6633	0.9957	0.6390	0.5531	0.5218	0.5078	0.5012	0.4980
ANN6	3.2240	1.3712	0.7422	0.5819	0.5343	0.5084	0.5040	0.4994
ANN7	1.6235	0.8878	0.6240	0.5493	0.5204	0.5073	0.5010	0.4979
ANN8	5.8725	3.2240	1.2190	0.7031	0.5718	0.5303	0.5120	0.5032
ANN9	1.5068	0.7249	0.5699	0.5280	0.5105	0.5025	0.4986	0.4967
ANN10	2.8629	0.7113	0.6100	0.5754	0.5318	0.5125	0.5036	0.4991
ANN11	0.9447	0.8942	0.5640	0.5265	0.5099	0.5022	0.5012	0.4991
ANN12	5.4711	2.6383	1.0347	0.6566	0.5590	0.5247	0.5092	0.5032
ANN13	2.0327	0.8981	0.6253	0.5490	0.5202	0.5071	0.5009	0.4978
ANN14	2.5764	1.2812	0.7285	0.5778	0.5327	0.5129	0.5037	0.4992
ANN15	5.1480	2.0595	0.8763	0.6155	0.5462	0.5190	0.5066	0.5006
ANN16	5.3438	2.4835	1.0037	0.6496	0.5420	0.5238	0.5088	0.5017
ANN17	2.1142	1.1889	0.7202	0.5770	0.5325	0.5129	0.5037	0.4988
ANN18	0.9569	0.6796	0.5644	0.5265	0.5099	0.5022	0.4985	0.4966

ANN19	0.8937	0.6611	0.5606	0.5250	0.5087	0.5019	0.4983	0.4966
ANN20	5.5777	2.5461	1.0109	0.6511	0.5572	0.5239	0.5089	0.5018
ANN21	2.5450	2.0720	0.7279	0.5782	0.5330	0.5131	0.5038	0.4993
ANN22	4.7003	2.3669	1.0066	0.6528	0.5568	0.5244	0.5091	0.5019
ANN23	0.6252	0.5375	0.5122	0.5026	0.4985	0.4966	0.4957	0.4953
ANN24	4.8896	1.9939	0.8702	0.6054	0.5459	0.5189	0.5065	0.5006
ANN25	2.3724	0.9136	0.6262	0.5516	0.5201	0.5070	0.5008	0.4978
ANN26	2.4106	1.2552	0.7283	0.5788	0.5333	0.5132	0.5038	0.4993
ANN27	5.5051	2.1358	0.8818	0.6157	0.5206	0.5189	0.5065	0.5006
ANN28	1.9674	0.9270	0.6263	0.5489	0.5201	0.5071	0.5009	0.4978
ANN29	4.3198	1.4610	0.7399	0.5789	0.5329	0.5130	0.5037	0.4992
ANN30	3.7039	1.7743	0.8511	0.6109	0.5449	0.5185	0.5063	0.5005
ANN31	4.1055	1.3604	0.7268	0.5749	0.5319	0.5231	0.5035	0.4991
ANN32	3.6928	1.7478	0.8479	0.6103	0.5448	0.5184	0.5063	0.5005
ANN33	1.6759	0.9248	0.6310	0.5508	0.5210	0.5075	0.5011	0.4979
ANN34	5.5939	2.5583	1.0151	0.6523	0.5578	0.5242	0.5090	0.5018
ANN35	3.6434	1.3732	0.7334	0.5783	0.5328	0.5130	0.5037	0.4992
ANN36	1.2241	0.7011	0.5666	0.5270	0.5101	0.5023	0.4985	0.4967
GA	36	36	36	35	35	28	14	0
"Winner" ANN23	0.6252	0.5375	0.5122	0.5026	0.4985	0.4966	0.4960	0.4967
MMRE	3.2296	1.4882	0.7586	0.5840	0.5323	0.5133	0.5038	0.4993

Figure 35. MMRE results in three parts of the experiment for all clusters (ANN-L36).

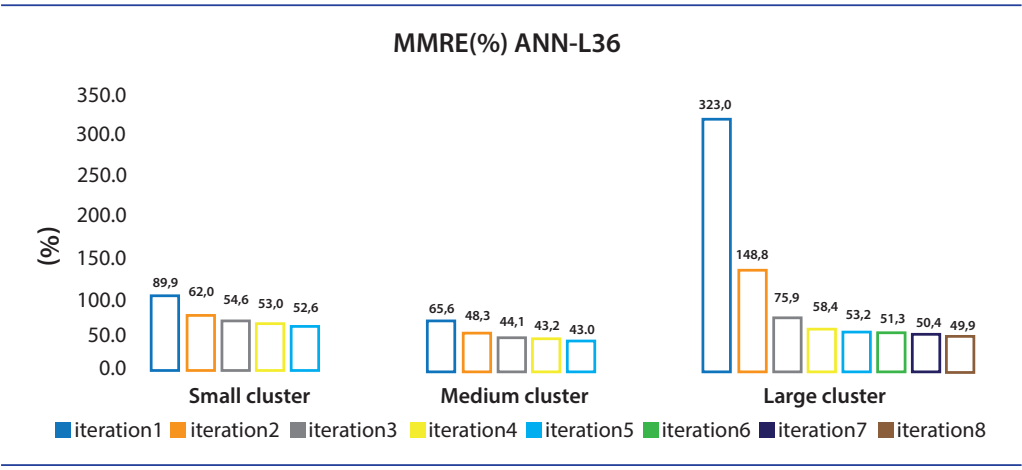
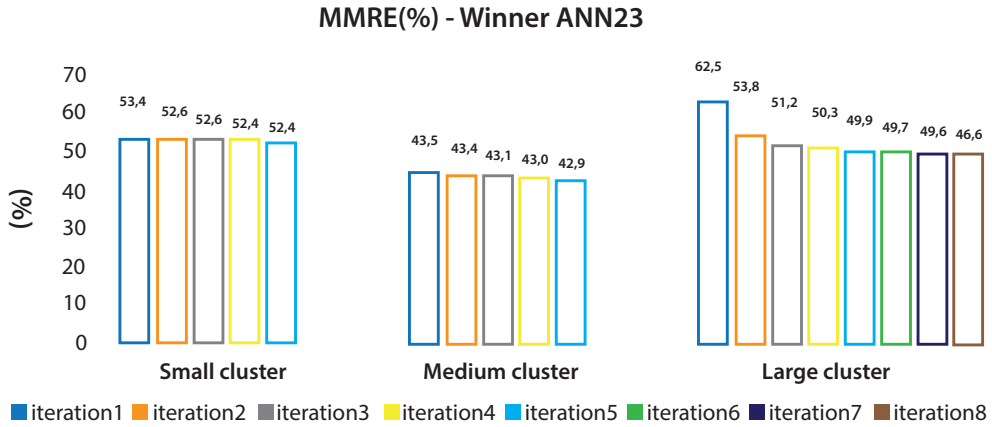


Figure 36. MMRE for “Winner” ANN23 (ANN-L36).

The convergence speed pertains to how quickly the proposed algorithm reaches the minimum Mean Relative Error (MRE) value in an experiment, which, from a project perspective, is determined by determining the relative error difference with the least value between successive output iterations. This convergence rate varies depending on the specific Artificial Neural Network (ANN) architecture employed. For small and medium clusters using ANN-L27, ANN-L36, the condition $GA < 0.01$ is met after 5 iterations, while for the large cluster, this condition is achieved after 8 iterations with ANN-L9, ANN-L27, ANN-L36 architectures. This approach's convergence rate significantly reduces the number of iterations, resulting in reduced time for effort estimation during software project development.

From Figure 37, we can conclude that:

$\delta i(ANN-L36) > \delta i(ANN-L18) > \delta i(ANN-L9) > \delta i(ANN-L27)$, that is, the ANN-L36 architecture converges the fastest to the minimum MMRE on a small cluster.

From Figure 38, it can be concluded that:

$\delta i(ANN-L36) > \delta i(ANN-L27) > \delta i(ANN-L18) > \delta i(ANN-L9)$, that is, the ANN-L36 architecture again converges the fastest for medium cluster.

From Figure 39, it can be concluded that:

$\delta i(ANN-L36) > \delta i(ANN-L27) > \delta i(ANN-L18) > \delta i(ANN-L9)$, that is, again, the ANN-L36 architecture converges the fastest to the minimum MMRE observed for a large cluster.

Figure 37. Convergence rate of the four proposed ANNs on small cluster.

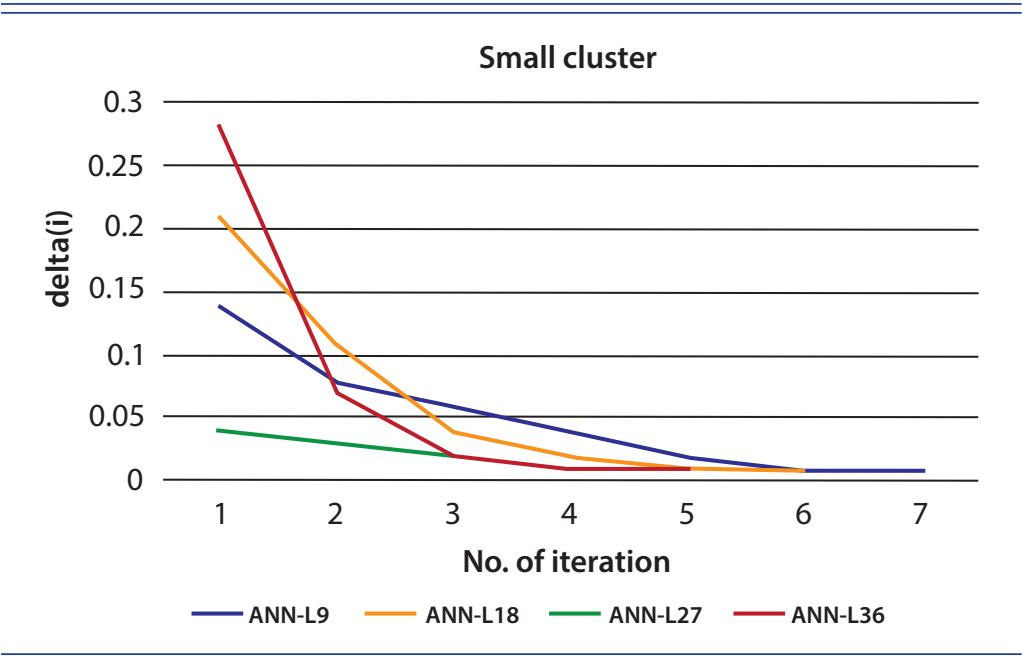


Figure 38. Convergence rate of the four proposed ANNs on medium cluster.

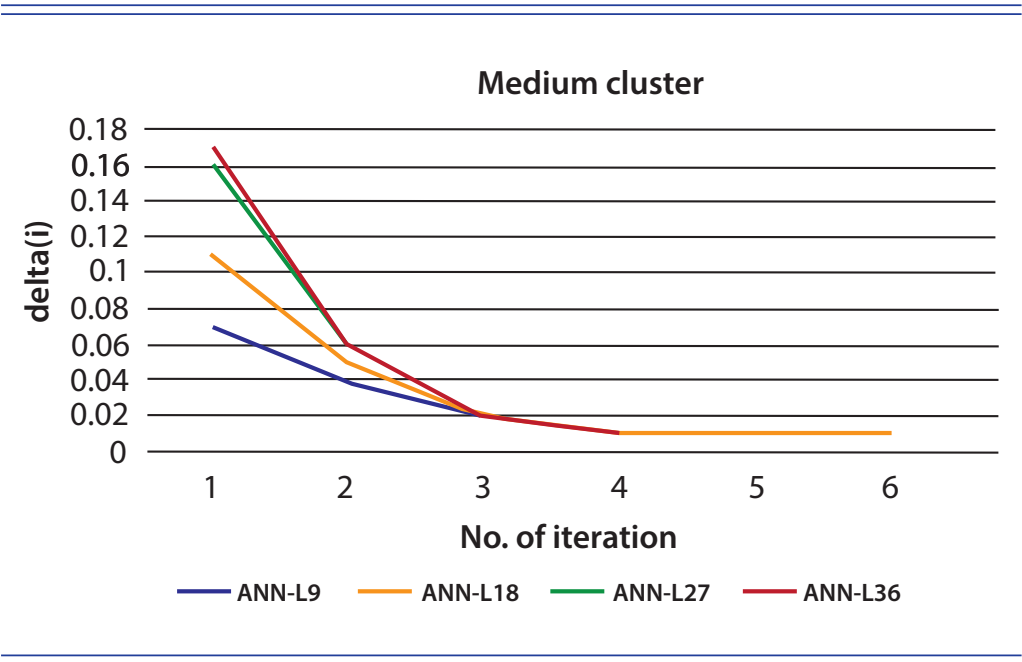
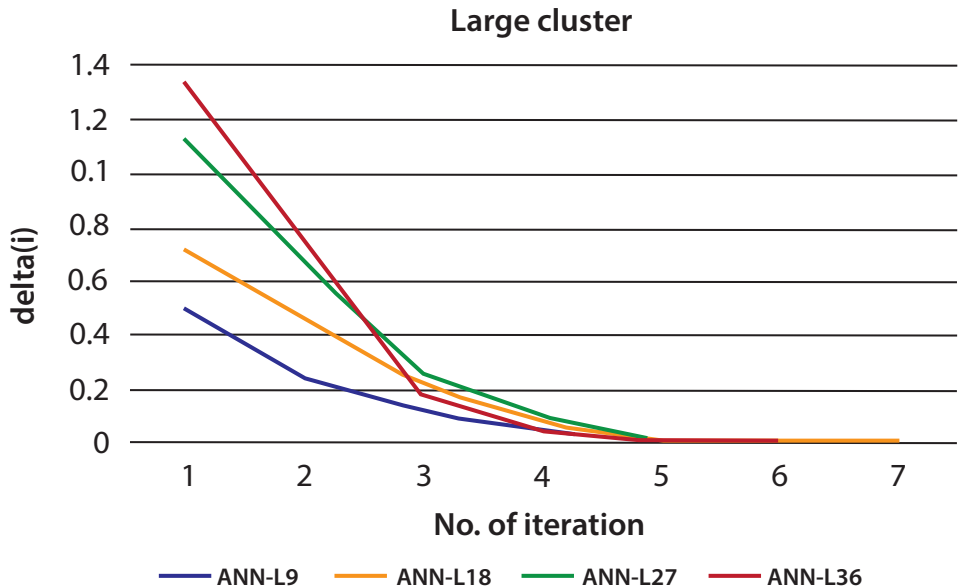


Figure 39. Convergence rate of the four proposed ANNs on large cluster.



A COCOMO2000 dataset made up of 100 projects was used in the first stage of this experiment, which focuses on training particular architectures (ANN-L9, ANN-L18, ANN-L27, and ANN-L36). These initiatives were divided into the following clusters according to their actual effort, which was quantified in person-months (PM): Projects with a PM of less than 90 made up a small cluster of 32 projects, those with a PM of between 90 and 500 made up a medium cluster of 40 projects, and those with a PM of more than 500 made up a large cluster of 28 projects.

The average Mean Magnitude of Relative Error (MMRE) was assessed throughout this phase. The ANN-L36 network recorded an amazing MMRE value of 43.3%, making it the greatest outcome out of all those that were obtained. Comparatively, the MMRE on the ANN-L27 network was 45.3%, which was 2% less than the top performance. However, the MMRE values of the ANN-L18 and ANN-L9 networks, which were respectively 59.7% and 72.0%, were less favorable, indicating greater MMRE values and thus fewer accurate estimations.

According to the information in Table 8, the ANN-L36 network not only yields the best results but also converges to the minimum MMRE value more quickly. Furthermore, the Mean Absolute Error (MAE) was determined to increase the effectiveness of the studies. This number indicates the average absolute disparities between the total number of projects' actual effort and estimated effort.

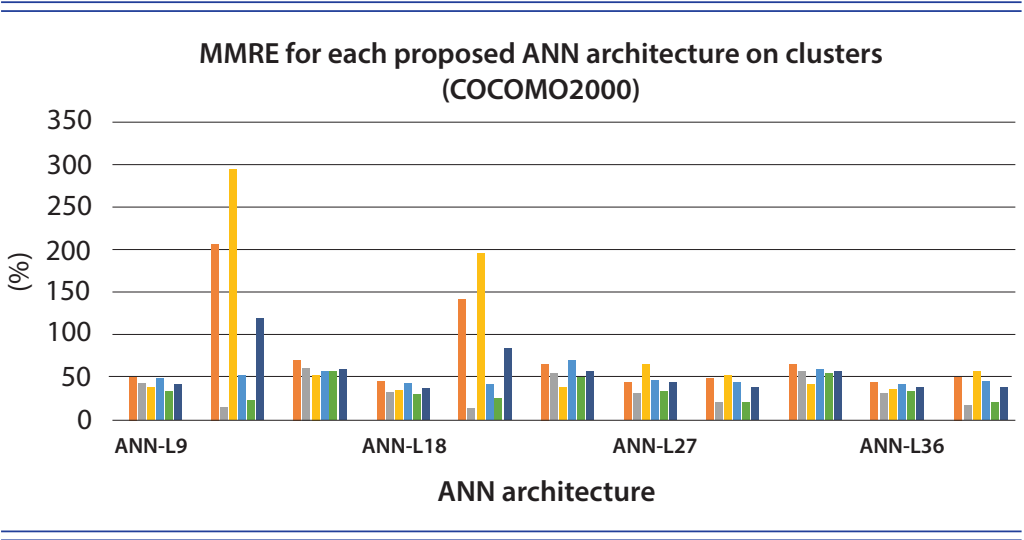
The findings indicate that the ANN-L36 network consistently performs better, with an average MMRE value of 43.3% across the experiment's three phases. With the ANN-L27 design (45.3%), somewhat greater MMRE values, or about 2% higher, are seen. As seen in Table 41 and Figure 40, ANN-L18 performs less well with a 59.7% MMRE, and ANN-L9 achieves the worst outcome with a 72% MMRE.

In summary, increasing the number of hidden layers in the network appears to lead to more reliable MMRE value estimations, with ANN-L36 demonstrating the most consistent and accurate performance.

Table 41. MMRE for each proposed ANN architecture on clusters (COCOMO2000).

Experiment	ANN-L9			ANN-L18			ANN-L27			ANN-L36		
	Small cluster	Medium cluster	Large cluster	Small cluster	Medium cluster	Large cluster	Small cluster	Medium cluster	Large cluster	Small cluster	Medium cluster	Large cluster
Training	80.7	49.6	206.9	69.6	44.8	141.1	65.4	43.3	47.3	65.6	43.1	49.7
Testing	63.3	41.6	13.6	59.9	32.3	13.6	54.9	30.8	20.3	57.4	31.3	16.1
Validation1	30.8	38.4	293.7	52.8	35.3	195.4	37.5	64.5	52.3	42.4	35.3	55.9
Validation2	53.2	48.0	53.8	56.4	42.7	41.0	68.4	45.7	45.1	59.5	41.2	44.1
Validation3	50.8	32.8	23.2	56.1	31.0	24.0	50.3	33.7	19.6	55.4	32.8	19.5
MMRE(%)	55.8	42.1	118.2	59.0	37.2	83.0	55.3	43.6	36.9	56.1	36.7	37.1

Figure 40. MMRE values of proposed ANN architectures for each part of the experiment (COCOMO2000).



It is clear from the experiment's training phase that the medium cluster, with an outstanding 43.1%, produces the best results for the ANN-L36 architecture. During the experiment's testing phase, it was found that the big cluster produced the greatest results for the ANN-L9 and ANN-L18 architectures, both of which produced a 13.6% MMRE. The medium cluster consistently yields the best results across all suggested topologies during the experiment's validation phase. The prediction metric (PRED%), which measures the proportion of projects that fulfill the predetermined GA criterion, was carefully watched during all three phases of the tests. It is important to note that for all suggested architectures, the PRED values are fairly close to one another.

The ANN-L9 network comes out with the best prediction performance throughout the training phase. However, depending on the particular dataset employed, the ANN-L27 and ANN-L36 networks excel in achieving the highest PRED values throughout the testing and validation phases. As demonstrated in Table 42, the ANN-L27 architecture performs best in the third validation phase whereas the ANN-L36 network excels in the first and second validation phases.

The results shown in Table 10 demonstrate how the variety of the datasets utilized has a direct impact on the outcomes of the predictions made throughout each of the three experiment phases. In particular, datasets 1 to 5 show notable variability in terms of the programming languages used, the length of development, and a wide range of actual effort estimates, with a considerable standard deviation.

Table 42. Monitoring the prediction of each part of the experiment (COCOMO2000).

Training				
PRED(%)	ANN-L9(%)	ANN-L18(%)	ANN-L27(%)	ANN-L36(%)
PRED(25)	20.0	21.0	16.0	16.0
PRED(30)	22.0	23.0	22.0	21.0
PRED(50)	51.0	46.0	42.0	43.0
Testing				
PRED(25)	35.1	45.5	60.4	45.2
PRED(30)	50.4	50.2	65.3	50.3
PRED(50)	80.7	75.2	80.0	70.3
Validation1				
PRED(25)	25.5	17.6	17.6	25.5
PRED(30)	31.4	21.6	23.5	33.3
PRED(50)	62.7	43.1	41.2	56.9

Validation2				
PRED(25)	25.0	18.3	16.7	16.7
PRED(30)	33.3	21.7	20.0	25.0
PRED(50)	68.3	56.7	35.0	43.3
Validation3				
PRED(25)	25.1	19.0	25.1	25.1
PRED(30)	38.2	25.1	25.1	41.6
PRED(50)	71.4	69.9	75.9	73.4

The nature of the projects has an impact on the minimized MRE value as well. Therefore, it is crucial to test and validate these models utilizing a variety of data sources, or many other datasets, in addition to the training results achieved on three clusters. The correlation coefficient in this method denotes how well the estimated value and the actual effort agree. The dependability of the procedure used to produce the estimated values for each given design is confirmed by the statistical validation of correlation coefficient values across all four proposed architectures. All three identified clusters were included in these correlations, which were calculated using the whole dataset.

Lower correlation values were seen in ANN-L9 and ANN-L18, which are more basic architectures, whereas greater correlation values were seen in ANN-L27 and ANN-L36, which are more complicated structures. This leads to the conclusion that the ANN-L36 design produced the best correlation results, as shown by Pearson's and Spearman's coefficients. On the other hand, the ANN-L27 design was favored by the R-squared (R^2) coefficient. For a detailed explanation of these correlation coefficients, which together show the relationship between the estimated and real values of the actual effort for each of the chosen architectures (ANN-L9, ANN-L18, ANN-L27, and ANN-L36), refer to Table 43.

Table 43. Correlation for each ANN architecture (COCOMO2000).

Correlation	ANN-L9	ANN-L18	ANN-L27	ANN-L36
Pearson's	0.617	0.615	0.714	0.866
Spearman's rho	0.869	0.864	0.862	0.904
R^2	0.827	0.861	0.862	0.869

It is obvious from the trial as a whole that the ANN-L36 architecture consistently produces the best outcomes. As seen in Table 42, it gets a remarkable MMRE score of 43.3% across all clusters. Additionally, compared to other architectures, it requires the fewest iterations—only 5 iterations for small and larger clusters. Figures 37, 38, 39 show that the ANN-L36 architecture converges at a quicker rate than any other proposed architectures.

For all of the datasets utilized in the experiment, ANN-L36 constantly performs exceptionally well in matching actual effort values when the three correlation coefficients are taken into account. Additionally, across all experiment phases, ANN-L36 consistently yields the greatest results in terms of prediction monitoring.

In conclusion, the ANN-L36 architecture consistently produces the best and most dependable outcomes across all suggested structures, regardless of the project's nature.

The results shown in the previous tables and figures of this approach (COCOMO2000) were processed in the R programming language (available at: <https://www.r-project.org/>) and checked in the Python programming language (available at: <https://www.python.org/>) within the RStudio environment (available at: <https://www.rstudio.com/products/rstudio/>). The data required for statistical analysis were processed in the IBM SPSS Statistical 25 software tool (available at: <https://www.ibm.com/support/pages/downloading-ibm-spss-statistics-25>).

6.2. COSMIC FFP HYBRID MODEL AND ANN

Two unique ANN designs, referred to as ANN-L12 and ANN-L36prim, were used in the second suggested COSMIC FFP technique. The ISBSG dataset, which was separated into five clusters depending on functional size, was utilized to train various ANN designs in the experiment's first phase. Through a number of tests, it was found that using a 70:30 split produced the most dependable outcomes in both the training and testing stages. This indicates that 70% of the projects within each cluster were used for training, while the remaining 30% were used for testing.

The results of training the first suggested ANN-L12 architecture on the first cluster are shown in Table 44. The GA criterion was monitored in addition to measuring the Mean Relative Error (MRE) values for each of the 12 ANN design candidates. The "Winner" network, or the ANN network with the lowest MRE (the one exhibiting the best performance), was chosen based on the MRE values in each iteration. Additionally, for each iteration, the Mean Magnitude of Relative Error (MMRE) value was calculated. The MRE for the "Winner" network, known as ANN1, was 36.1%, and the MMRE score was 36.4%. Surprisingly, the GA criteria was met after just five rounds.

Table 44. ANN-L12 results of training part on Dataset_1 (COSMIC FFP).

ANN-L12 Dataset_1	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5
ANN1	0.3615	0.3615	0.3612	0.3612	0.3612
ANN2	1.0439	0.5517	0.3919	0.3678	0.3638
ANN3	1.0532	0.5517	0.3969	0.3682	0.3641
ANN4	0.5519	0.3901	0.3657	0.3621	0.3614
ANN5	0.4077	0.3846	0.3754	0.3685	0.3650
ANN6	0.4247	0.3976	0.3791	0.3698	0.3655
ANN7	0.4356	0.4006	0.3803	0.3704	0.3657
ANN8	0.5197	0.3918	0.3649	0.3616	0.3613
ANN9	0.4118	0.3885	0.3761	0.3687	0.3651
ANN10	0.8539	0.4473	0.3775	0.3677	0.3640
ANN11	0.9055	0.4476	0.3757	0.3670	0.3638
ANN12	0.5747	0.4071	0.3860	0.3736	0.3674
GA	12	11	10	3	0
"Winner" ANN	0.3615	0.3615	0.3612	0.3612	0.3612
MMRE	0.6287	0.4266	0.3776	0.3672	0.3640

The results of training the first suggested ANN-L12 architecture on the second cluster of data are shown in Table 45. Similar to the prior cluster, the evaluation included monitoring the GA criterion and evaluating the Mean Relative Error (MRE) values for each of the 12 ANN design choices. Based on the lowest MRE value throughout all iterations, or the ANN network that performed the best, the "Winner" network was chosen.

The "Winner" network, known as ANN1, in this instance, obtained an MRE of 35.4%, while the associated Mean Magnitude of Relative Error (MMRE) result was 35.7%. Fascinatingly, the required GA condition was met after 6 iterations.

Table 45. ANN-L12 results of training part on Dataset_2 (COSMIC FFP).

ANN-L12 Dataset_2	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6
ANN1	0.3564	0.3564	0.3564	0.3564	0.3541	0.3541
ANN2	1.3445	0.7097	0.4489	0.3782	0.3615	0.3575
ANN3	1.3515	0.7097	0.4497	0.3787	0.3619	0.3578
ANN4	0.7038	0.4498	0.3763	0.3594	0.3564	0.3560
ANN5	0.4517	0.3910	0.3734	0.3645	0.3605	0.3585
ANN6	0.4504	0.3977	0.3754	0.3652	0.3608	0.3586
ANN7	0.4618	0.4004	0.3755	0.3653	0.3608	0.3586
ANN8	0.6826	0.4437	0.3744	0.3595	0.3564	0.3560
ANN9	0.4620	0.3957	0.3743	0.3648	0.3606	0.3585
ANN10	1.0738	0.5676	0.4078	0.3686	0.3592	0.3568
ANN11	1.1302	0.5688	0.4058	0.3678	0.3591	0.3568
ANN12	0.7125	0.4536	0.3888	0.3693	0.3618	0.3591
GA	12	11	11	9	2	0
"Winner" ANN	0.3564	0.3564	0.3564	0.3564	0.3541	0.3541
MMRE	0.7651	0.4870	0.3922	0.3665	0.3594	0.3573

The results from training the first suggested ANN-L12 architecture on the third cluster of data are shown in Table 46. Analyzing the Mean Relative Error (MRE) values for each of the 12 ANN design options while also keeping an eye on the GA criterion was part of the evaluation procedure, just like with the prior clusters. Based on all MRE values across each iteration, the "Winner" network, or ANN with the lowest MRE value and the highest performance, was selected.

The "Winner" network in this case, designated as ANN12, attained an MRE of 16.6% and a corresponding Mean Magnitude of Relative Error (MMRE) score of 16.8%. It's impressive that after only 5 iterations, the necessary GA condition was met.

Table 46. ANN-L12 results of training part on Dataset_3 (COSMIC FFP).

ANN-L12 Dataset_3	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5
ANN1	0.2139	0.2037	0.1816	0.1743	0.1700
ANN2	0.2744	0.2303	0.1875	0.1732	0.1714
ANN3	0.2716	0.2126	0.1768	0.1728	0.1712
ANN4	0.1750	0.1708	0.1708	0.1686	0.1683
ANN5	0.1870	0.1697	0.1683	0.1680	0.1664
ANN6	0.2116	0.1836	0.1714	0.1694	0.1679
ANN7	0.2153	0.1754	0.1694	0.1685	0.1665
ANN8	0.1749	0.1708	0.1708	0.1687	0.1683
ANN9	0.2132	0.1838	0.1711	0.1692	0.1678
ANN10	0.2156	0.1856	0.1766	0.1699	0.1692
ANN11	0.2358	0.1909	0.1776	0.1698	0.1697
ANN12	0.1733	0.1693	0.1682	0.1673	0.1666
GA	12	9	6	1	0
"Winner" ANN	0.1733	0.1693	0.1682	0.1673	0.1664
MMRE	0.2135	0.1872	0.1742	0.1700	0.1686

The results of training the first suggested ANN-L12 architecture on the fourth cluster of data are shown in Table 47. The assessment included monitoring the GA criterion and closely examining the Mean Relative Error (MRE) values for each of the 12 ANN design options. Based on all MRE values across each iteration, the "Winner" network, or ANN with the lowest MRE value and the highest performance, was chosen.

The "Winner" network in this instance, denoted as ANN1, attained an MRE of 29.4%, while the matching Mean Magnitude of Relative Error (MMRE) score was 29.6%. It is significant that after 6 iterations, the necessary GA condition was satisfied.

Results of training the first suggested ANN-L12 architecture on the fifth cluster of data are shown in Table 48. The evaluation includes monitoring the GA criterion and analyzing the Mean Relative Error (MRE) values for each of the 12 ANN design choices, similar to previous clusters. Based on all MRE values across each iteration, the "Winner" network, or ANN with the lowest MRE value and the highest performance, was selected.

The "Winner" network in this case, known as ANN1, attained an MRE of 24.1% and a Mean Magnitude of Relative Error (MMRE) value of 24.2%. Notably, after six cycles, the necessary GA condition was satisfied.

Table 47. ANN-L12 results of training part on Dataset_4 (COSMIC FFP).

ANN-L12 Dataset_4	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6
ANN1	0.2969	0.2969	0.2969	0.2969	0.2969	0.2969
ANN2	1.3032	0.6647	0.3807	0.3074	0.2962	0.2959
ANN3	1.2854	0.6647	0.3827	0.3080	0.2963	0.2958
ANN4	0.6385	0.3778	0.3055	0.2954	0.2948	0.2945
ANN5	0.3869	0.3160	0.3073	0.3022	0.2996	0.2983
ANN6	0.3654	0.3203	0.3080	0.3023	0.2996	0.2982
ANN7	0.3639	0.3195	0.3080	0.3023	0.2996	0.2982
ANN8	0.6399	0.3803	0.3063	0.2967	0.2958	0.2956
ANN9	0.3727	0.3202	0.3085	0.3025	0.2997	0.2983
ANN10	1.0339	0.5043	0.3359	0.2978	0.2957	0.2955
ANN11	1.0942	0.5153	0.3369	0.2980	0.2956	0.2955
ANN12	0.6845	0.3689	0.3118	0.3026	0.2995	0.2981
GA	12	11	10	5	2	0
"Winner" ANN	0.2969	0.2969	0.2969	0.2954	0.2948	0.2945
MMRE	0.7055	0.4207	0.3240	0.3010	0.2974	0.2967

Table 48. ANN-L12 results of training part on Dataset_5 (COSMIC FFP).

ANN-L12 Dataset_5	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6
ANN1	0.2483	0.2408	0.2408	0.2408	0.2408	0.2408
ANN2	1.0844	0.5713	0.3295	0.2660	0.2459	0.2428
ANN3	1.1081	0.5713	0.3312	0.2668	0.2472	0.2434
ANN4	0.5504	0.3156	0.2629	0.2462	0.2429	0.2414
ANN5	0.3452	0.2638	0.2502	0.2460	0.2435	0.2422
ANN6	0.3217	0.2669	0.2553	0.2478	0.2444	0.2426
ANN7	0.3307	0.2701	0.2542	0.2471	0.2435	0.2420
ANN8	0.5143	0.3110	0.2613	0.2454	0.2425	0.2412
ANN9	0.3203	0.2617	0.2528	0.2472	0.2441	0.2425
ANN10	0.8970	0.4293	0.2956	0.2558	0.2457	0.2428
ANN11	0.9698	0.4362	0.2953	0.2548	0.2452	0.2425
ANN12	0.5569	0.3338	0.2672	0.2496	0.2445	0.2423
GA	12	11	10	7	3	0
"Winner" ANN	0.2483	0.2408	0.2408	0.2408	0.2408	0.2408
MMRE	0.6039	0.3560	0.2747	0.2511	0.2442	0.2422

Figure 41 shows how the newly added GA criterion behaves for each of the ISBSG dataset's five clusters. The graphic suggests that the GA requirement was satisfied in each cluster after five or six iterations, depending on the cluster. These findings indicate that the least Mean Magnitude of Relative Error (MMRE) value within each cluster can be reached with a reasonably modest number of iterations.

Figure 41. GA for all five clusters (ANN-L12).

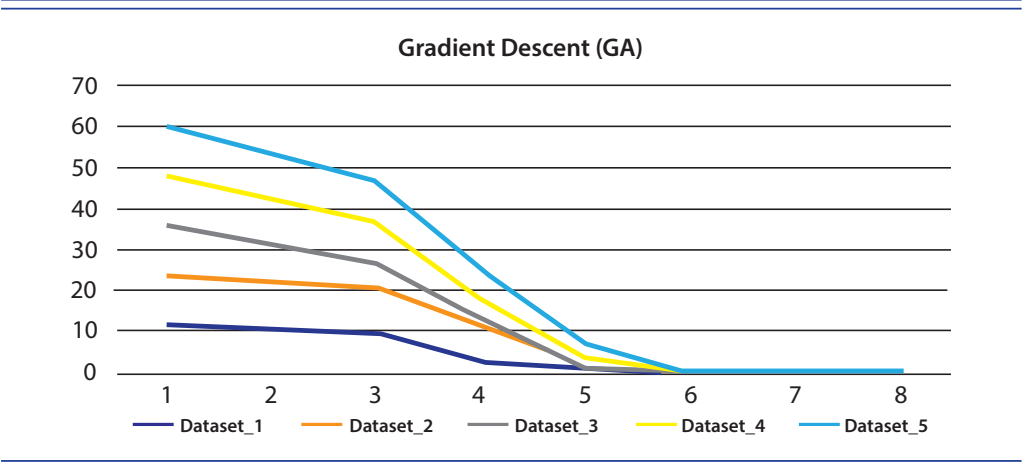


Figure 42 graphically illustrates the minimized MMRE values for the "Winner" network in each of the clusters used. The figure suggests that the difference in MMRE values between the first and last iteration is minimal and appears to remain relatively constant. This observation indicates that the MMRE values stabilize and do not vary significantly over the course of the iterations, reinforcing the reliability and consistency of the "Winner" network's performance in each cluster.

Figure 42. "Winner" MMRE for all five clusters (ANN-L12).

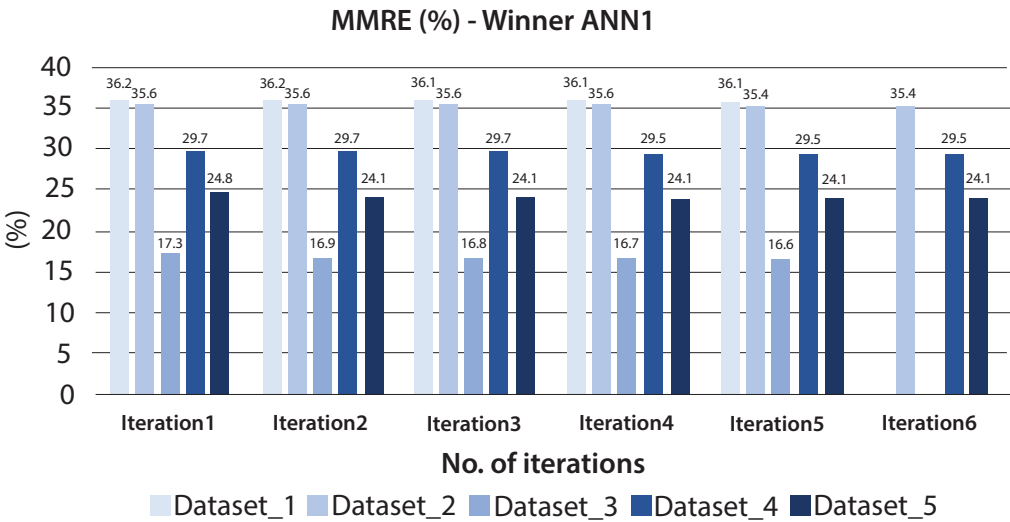
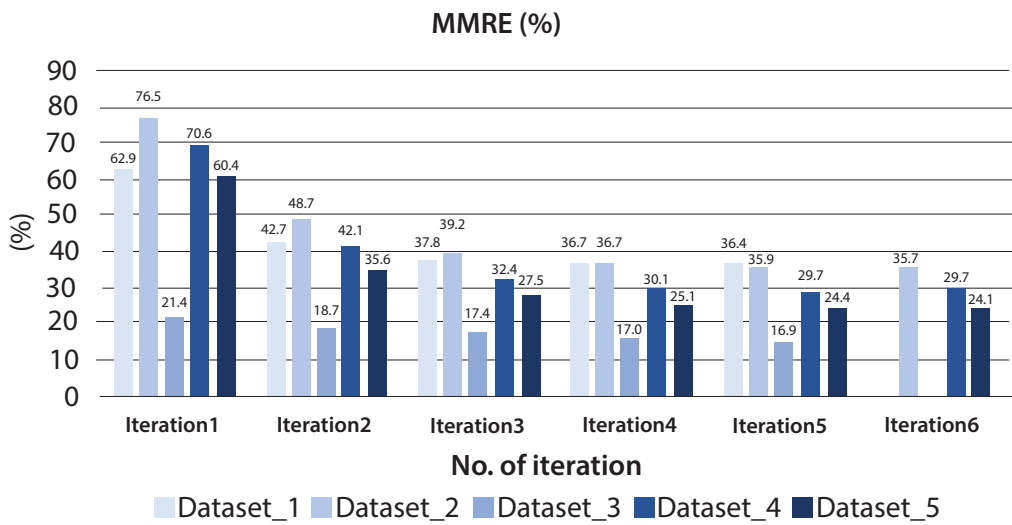


Figure 43 visually represents the minimized MMRE values for each cluster utilized. Unlike the constant and minimal difference observed in the MMRE values for the "Winner" ANN network mentioned earlier, Figure 43 shows that the MMRE values for each cluster follow an exponential-like trend. They appear to gradually decrease and tend towards the minimum MMRE as the iterations progress.

This pattern suggests that, as the iterations continue, the MMRE values improve and approach their lowest possible values, indicating an ongoing refinement of the network's performance and accuracy in estimating the desired outcomes.

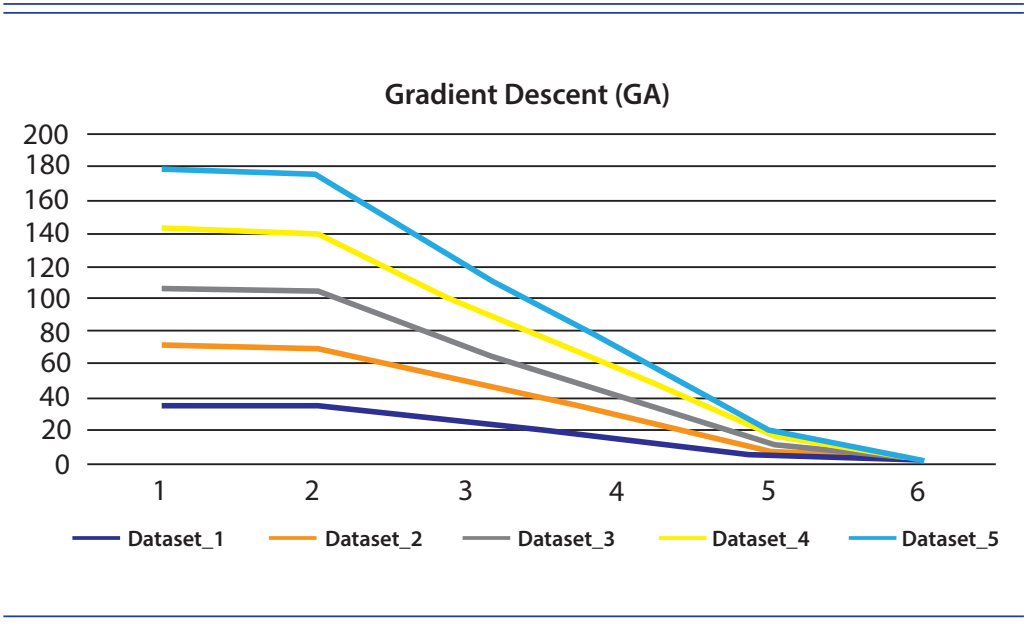
Figure 43. MMRE value for all five clusters (ANN-L12).



The results of training the second suggested ANN-L36prim architecture on each of the ISBSG dataset's five clusters are shown in Table 50. Each cluster's needed number of iterations was rigorously watched, and it is reported that the set GA requirement was satisfied for each of the 5 clusters after 6 iterations.

All five clusters' GA requirements are shown graphically in Figure 44, further demonstrating that after six iterations, the GA criterion was consistently met in each cluster. This means that the ANN-L36prim design will exhibit robust convergence behavior across these various data clusters.

Figure 44. GA for all five clusters (ANN-L36prim).



The results of training the second suggested ANN-L36prim architecture on the first cluster of data are shown in Table 49. Similar to the cluster results from earlier evaluations, the "Winner" network—representing the ANN with the lowest MRE value and the highest performance—was chosen by evaluating the Mean Relative Error (MRE) values for each iteration.

Achieving an MRE of 32.1% in this particular instance, the "Winner" network, known as ANN1, also received a comparable MMRE value of 32.1%. Notably, after 6 repetitions, the necessary GA criteria was satisfied.

Table 49. "Winner" MMRE vs. MMRE Dataset_1 (ANN-L36prim).

No. of Iteration	MMRE (%)					
	1.	2.	3.	4.	5.	6.
Dataset_1	34.8	33.5	32.8	32.5	32.1	32.1
Winner						
Dataset_1	47.7	37.6	34.1	33.1	32.2	32.1

Figure 45 provides a graphical representation of the Mean Magnitude of Relative Error (MMRE) values for the "Winner" network relative to the MMRE value on Dataset_1 over 6 iterations. This visualization allows for a clear comparison of how the MMRE values evolve and change with each iteration, particularly in relation to Dataset_1. It provides insights into the improvement and convergence of the network's performance during the training process.

Figure 45. "Winner" MMRE vs. MMRE Dataset_1 (ANN-L36prim).

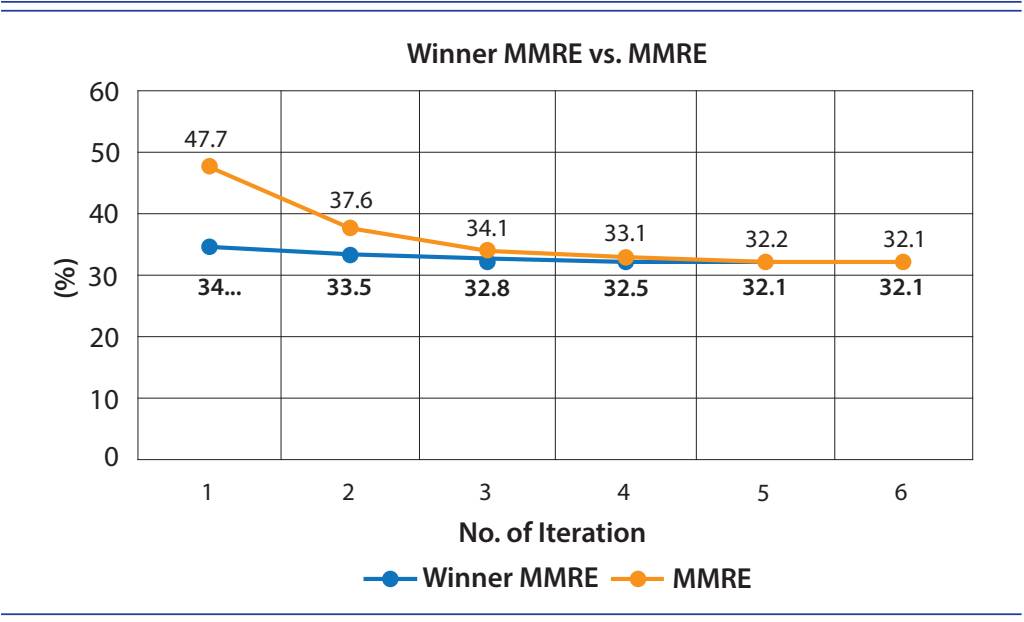


Table 50 displays the outcomes of training the second suggested ANN-L36prim architecture on the second cluster of data. As in previous clusters, the evaluation involved assessing the Mean Relative Error (MRE) values for each iteration and selecting the "Winner" network based on the lowest MRE value and best performance.

In this specific cluster, the "Winner" network, labeled as ANN36, achieved an MRE of 32.0%. Additionally, the corresponding MMRE value was also 32.0%. It's noteworthy that the GA criterion was met after 6 iterations, indicating that the network's performance reached the desired level.

Table 50. "Winner" MMRE vs. MMRE Dataset_2 (ANN-L36prim).

	MMRE (%)					
No. of Iteration	1.	2.	3.	4.	5.	6.
Dataset_2	35.2	34.1	33.8	32.7	32.2	32.0
Winner	49.7	38.6	36.9	35.0	32.3	32.0

Figure 46 provides a graphical representation of the minimized MMRE values for the "Winner" network relative to the MMRE value on Dataset_2 across 6 iterations. This graphical representation offers a visual comparison of how the MMRE values change and evolve in relation to Dataset_2 throughout the training process. It helps illustrate the improvement and convergence of the network's performance during these iterations, particularly concerning Dataset_2.

Figure 46. "Winner" MMRE vs. MMRE Dataset_2 (ANN-L36prim).

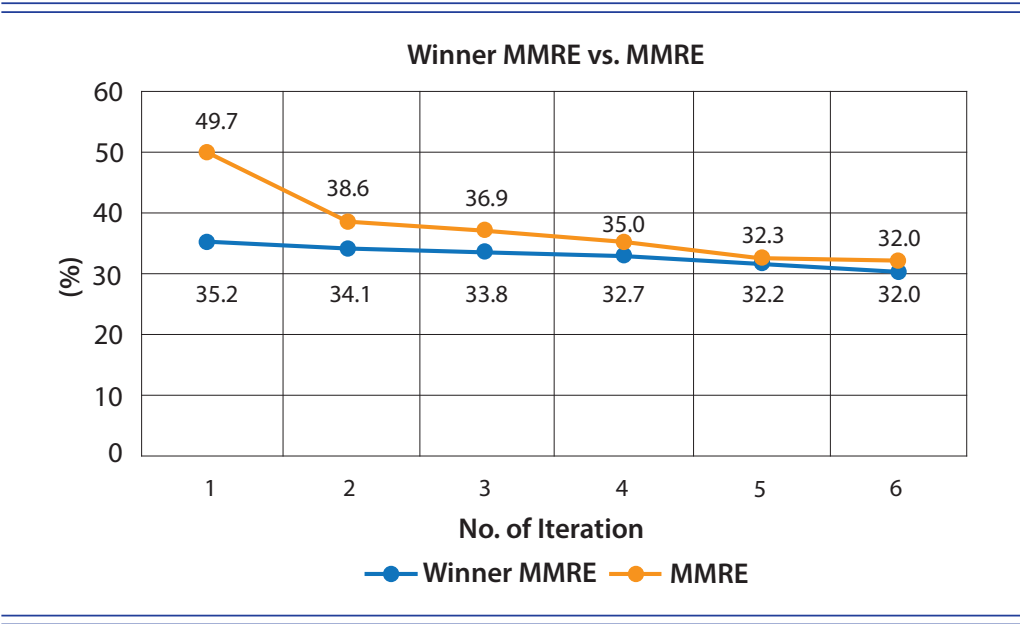


Table 51 presents the results of training the second suggested architecture, ANN-L36prim, on the 3rd cluster of data. As with previous clusters, the evaluation involved analyzing the MRE values for each iteration and selecting the "Winner" ANN network based on the lowest MRE value, indicating superior performance.

In this specific cluster, the "Winner" ANN network, designated as ANN12, achieved an MRE of 16.4%. Additionally, the corresponding MMRE value was also 16.4%. Notably, the Generalization Approximation (GA) criterion was met after 6 iterations, demonstrating that the network's performance met the desired criteria.

Table 51. "Winner" MMRE vs. MMRE Dataset_3 (ANN-L36prim).

	MMRE (%)					
No. of Iteration	1.	2.	3.	4.	5.	6.
Dataset_3	16.7	16.6	16.5	16.4	16.4	16.4
Winner						
Dataset_3	20.6	18.5	17.3	16.9	16.5	16.4

Figure 47 provides a graphical representation of the minimized MMRE values for the "Winner" ANN network relative to the MMRE value on Dataset_3 over 6 iterations. This visual representation allows you to observe how the MMRE values change and compare the performance of the "Winner" network in relation to Dataset_3 during the training process. It provides insights into the network's progress and convergence concerning Dataset_3 throughout these iterations.

Figure 47. "Winner" MMRE vs. MMRE Dataset_3 (ANN-L36prim).

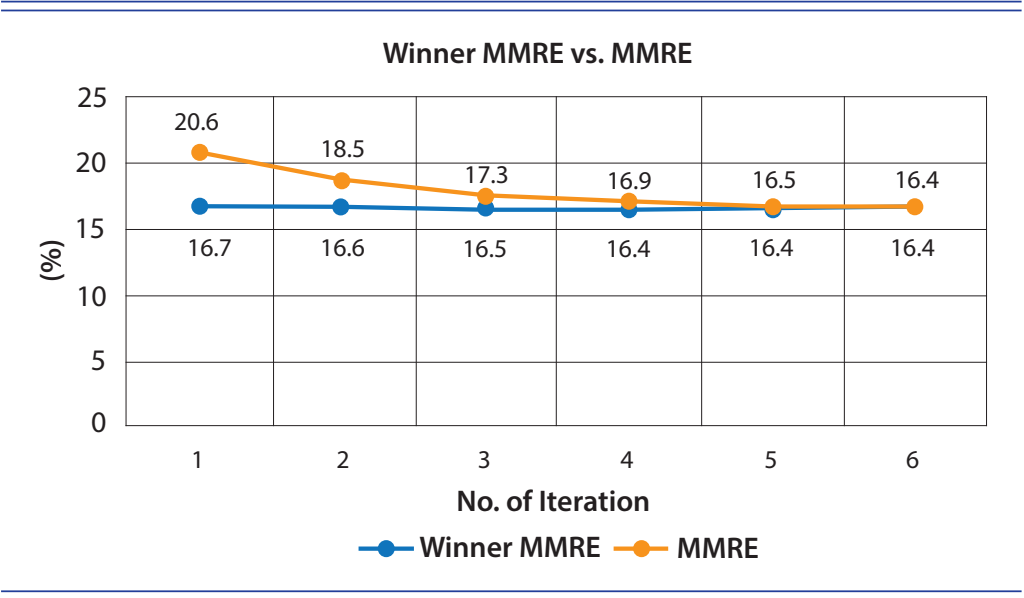


Table 52 indicates the results obtained from training the second suggested architecture, ANN-L36prim, on the 4th cluster of data. Similar to previous clusters, the assessment involved evaluating the minimized MRE values for each iteration and selecting the "Winner" network based on the lowest MRE value, signifying superior performance.

In this specific cluster, the "Winner" network, denoted as ANN5, achieved an MRE of 29.7%. Additionally, the corresponding MMRE value was also 29.7%. Notably, the GA criterion was met after 6 iterations, indicating that the network's performance met the desired criteria.

Table 52. "Winner" MMRE vs. MMRE Dataset_4 (ANN-L36prim).

No. of Iteration	MMRE (%)					
	1.	2.	3.	4.	5.	6.
Dataset_4						
Winner	37.5	33.2	32.1	31.5	30.4	29.7
Dataset_4	51.1	42.7	35.7	32.2	31.1	29.7

Figure 48 presents a graphical representation of the minimized MMRE values for the "Winner" ANN network relative to the MMRE value on Dataset_4 over 6 iterations. This visual representation allows you to observe how the MMRE values change and compare the performance of the "Winner" ANN network in relation to Dataset_4 during the training process. It provides insights into the network's progress and convergence concerning Dataset_4 throughout these iterations.

Figure 48. "Winner" MMRE vs. MMRE Dataset_4 (ANN-L36prim).

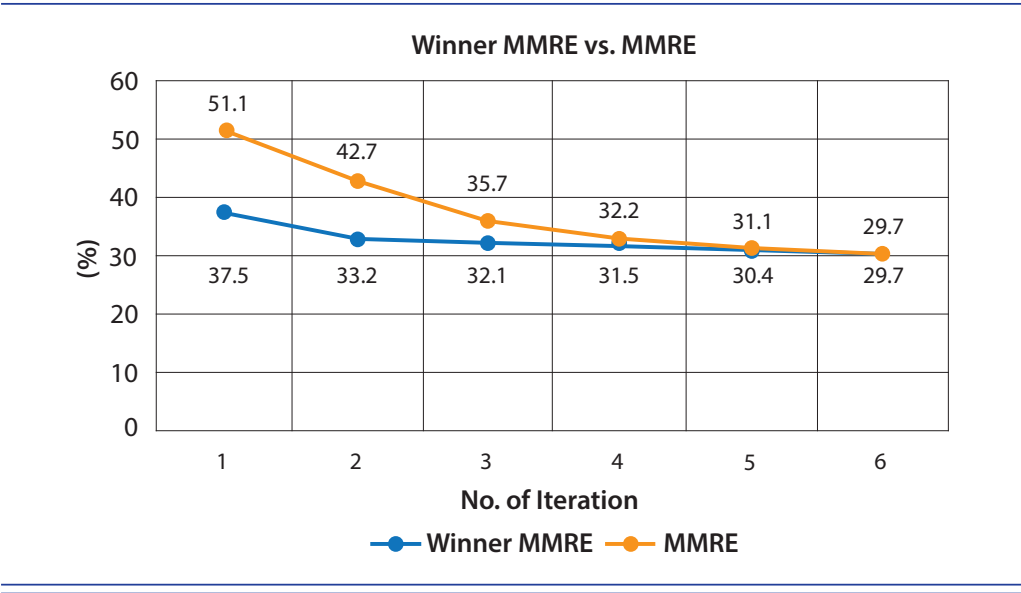


Table 53 provides the results obtained from training the second suggested architecture, ANN-L36prim, on the 5th cluster of data. Similar to previous clusters, the evaluation process involved assessing the MRE values for each iteration and selecting the "Winner" network based on the lowest MRE value, signifying superior performance.

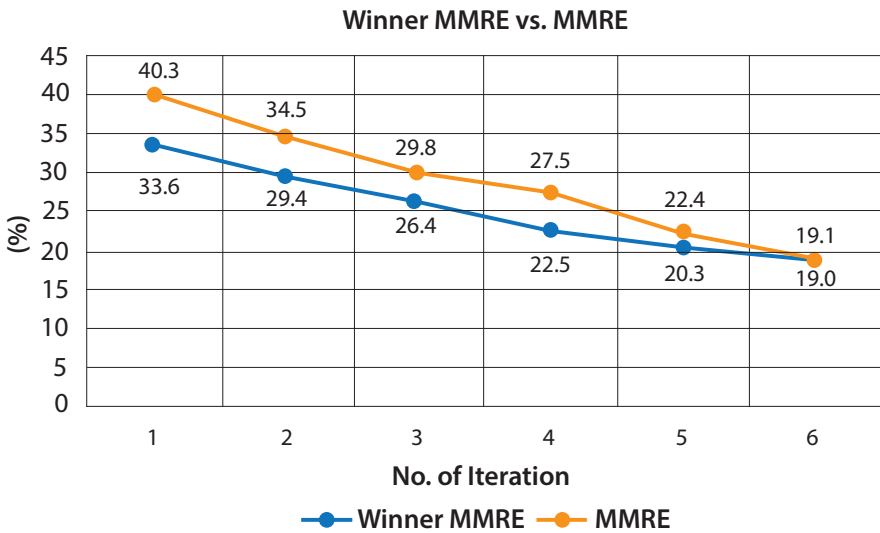
In this specific cluster, the "Winner" ANN network, designated as ANN5, achieved an MRE of 19.0%. Additionally, the corresponding MMRE value was 19.1%. Notably, the GA criterion was met after 6 iterations, demonstrating that the network's performance met the desired criteria.

Table 53. "Winner" MMRE vs. MMRE Dataset_5 (ANN-L36prim).

	MMRE (%)					
No. of Iteration	1.	2.	3.	4.	5.	6.
Dataset_5 Winner	33.6	29.4	26.4	22.5	20.3	19.0
Dataset_5	40.3	34.5	29.8	27.5	22.4	19.1

Figure 49 provides a graphical representation of the minimized MMRE values for the "Winner" ANN network related to the MMRE value on Dataset_5 across 6 iterations. This visual representation allows you to observe how the MMRE values change and compare the performance of the "Winner" network in relation to Dataset_5 during the training process. It provides insights into the network's progress and convergence concerning Dataset_5 throughout these iterations.

Figure 49. "Winner" MMRE vs. MMRE Dataset_5 (ANN-L36prim).



One hidden layer and two nodes are used in the first proposed ANN-L12 architecture in this method, while one hidden layer and three nodes are also used in the second proposed ANN-L36prim design. The models' complexity was not greatly increased by the decision to use ANN topologies with a single hidden layer. This led to 11 factors in the ANN-L12 architecture while 16 factors were produced in the ANN-L36prim architecture. Within the COSMIC FFP methodology, this degree of complexity was found to be acceptable. Curiously, tests showed that this architectural change resulted in a 0.9% decrease in the MMRE score. This shows that adding more sophisticated structures wasn't essential because processing time was not considerably longer than in earlier experimented architectures.

The MMRE value was the outcome that was being watched as the necessary iterations were being completed. Depending on the characteristics of the dataset, this value showed stability for the two suggested designs in all three of the experiment's components. In comparison to datasets with medium functional sizes (such as Dataset_3), datasets with small functional sizes (such as Dataset_1, Dataset_2) showed greater MMRE values. Datasets with large functional sizes (such as Dataset_4, Dataset_5) fall within the middle range of MMRE values. Higher MMRE values were seen in validation datasets with a variety of functional magnitudes.

The chosen ANN-L12 architecture achieved the lowest MMRE value on Dataset_3 with a value of 17.0%, whereas ANN-L36prim had an MMRE value of 16.4%. As shown in Figure 50 and Table 54, the average MMRE value across all datasets in all three phases of the experiment was 29.7% for ANN-L12 and 28.8% for ANN-L36prim. These data represent a significant 14.5% decrease in MMRE when compared to the COCOMO method, as seen in Table 55.

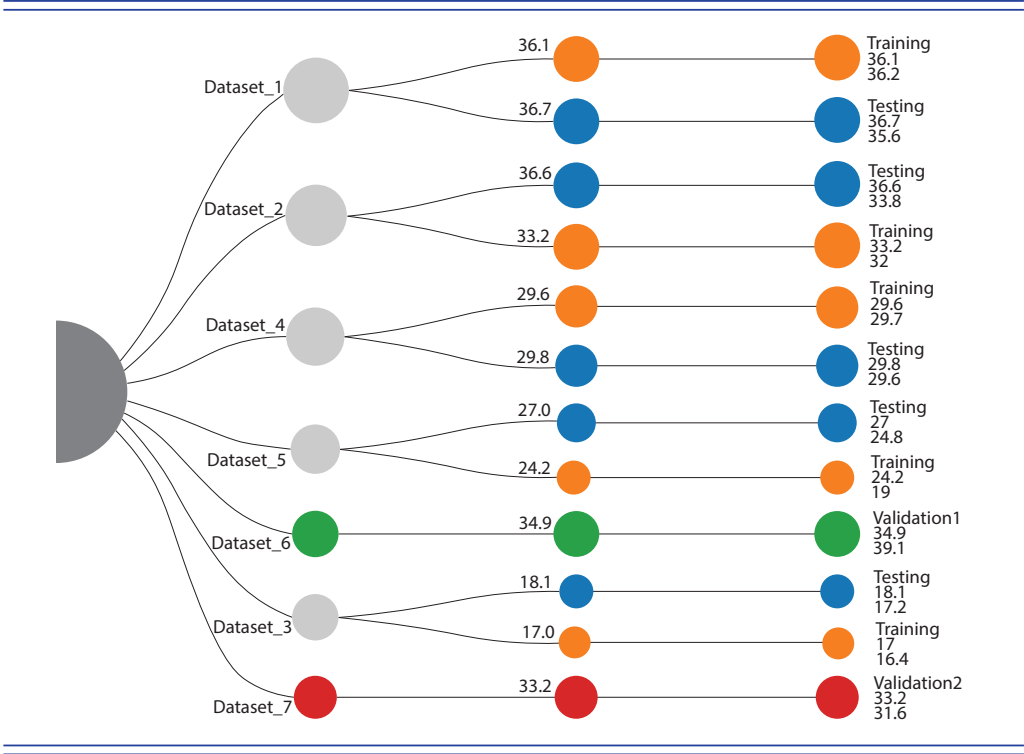
Table 54. MMRE value for each proposed ANN architecture (COSMIC FFP).

Datasets	ANN-L12 MMRE(%)	ANN-L36prim MMRE(%)	Part of experiment
Dataset_1	36.1	36.2	Training
	36.7	35.6	Testing
Dataset_2	33.2	32.0	Training
	36.6	33.8	Testing
Dataset_3	17.0	16.4	Training
	18.1	17.2	Testing
Dataset_4	29.6	29.7	Training
	29.8	29.6	Testing
Dataset_5	24.2	19.0	Training
	27.0	24.8	Testing
Dataset_6	34.9	39.1	Validation1
Dataset_7	33.2	31.8	Validation2
AVERAGE (MMRE)	29.7	28.8	

Table 55. MMRE values COCOMO2000 vs. COSMIC FFP.

COCOMO model		Functional point - COSMIC model		
ANN L-18 MMRE (%)	ANN L-27 MMRE (%)	ANN L-36 MMRE (%)	ANN L-12 MMRE (%)	ANN L-36prim MMRE (%)
59.7	45.3	43.3	29.7	28.8

Figure 50. Graphical representation of MMRE value for each proposed ANN architecture (COSMIC FFP).



In addition to evaluating the MMRE score, the convergence speed of the two ANN architectures, ANN-L12 and ANN-L36prim, was assessed across all 5 datasets. It is observed that both architectures exhibit their fastest convergence in Dataset_5 and their steady convergence in Dataset_4, as depicted in Figure 51 and Figure 52. This information provides insights into how quickly the models reach a stable and accurate prediction level for each dataset, with Dataset_5 demonstrating the fastest convergence and Dataset_4 requiring more iterations to reach convergence.

Figure 51. Convergence rate ANN-L12 (COSMIC FFP).

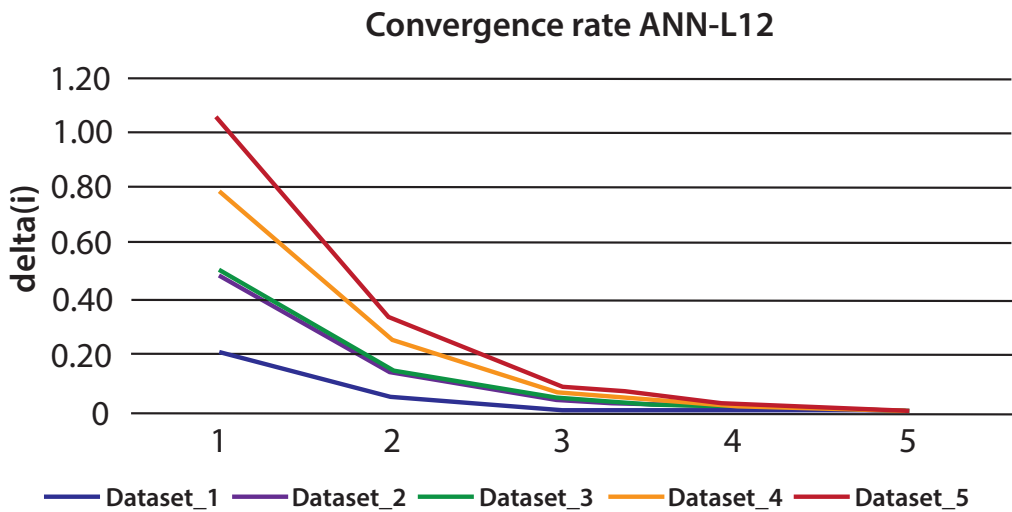
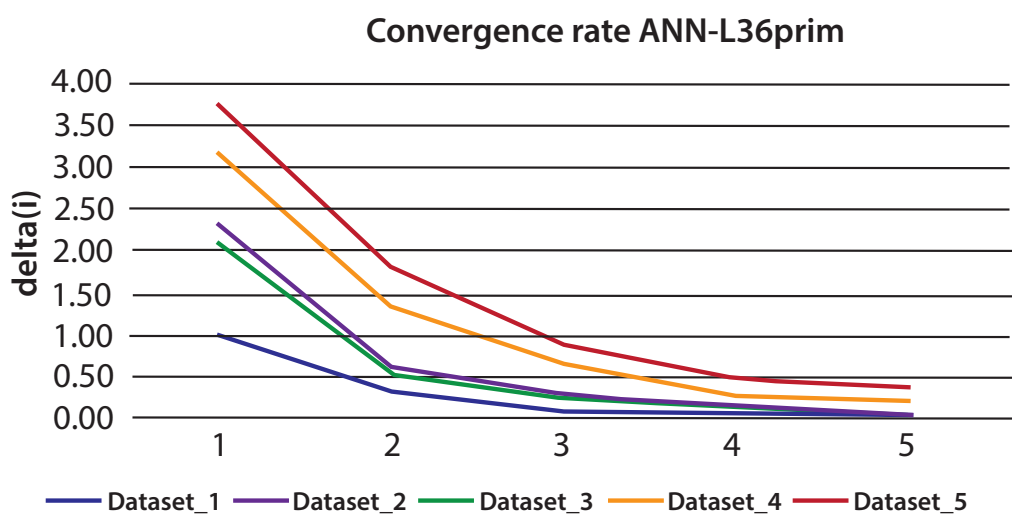


Figure52. Convergence rate ANN-L36prim (COSMIC FFP).



A useful indicator of the consistency and strength of the relationship between observed values is the correlation coefficient. The dependability, accuracy, and precision of the suggested approach can be assessed by calculating the correlation coefficient between the estimated and real values. Correlation coefficients determined in this context using the Pearson and Spearman methods both show a very high level of correlation.

According to the experimental findings, Dataset_2 and Dataset_4 have the highest correlation coefficients, demonstrating a strong and persistent link between the estimated and real values. Conversely, the lowest correlation coefficients are observed on Dataset_6 and Dataset_7, suggesting weaker relationships between the estimated and actual values, as shown in Table 56. These correlation coefficients provide valuable insights into the performance of the proposed approach on different datasets.

Table 56. Correlation coefficients (COSMIC FFP).

Correlation Datasets	Pearson		Spearman's rho	
	ANN-L12	ANN-L36prim	ANN-L12	ANN-L36prim
Dataset_1	0.859	0.756	0.884	0.785
Dataset_2	0.984	0.788	0.984	0.808
Dataset_3	0.734	0.688	0.745	0.678
Dataset_4	0.912	0.911	0.918	0.917
Dataset_5	0.767	0.618	0.715	0.918
Dataset_6	0.628	0.586	0.612	0.534
Dataset_7	0.674	0.622	0.654	0.618

Monitoring the forecast at various confidence levels, such 25%, 30%, 50%, offers a way to judge the validity and suitability of the suggested strategy. It aids in deciding which of the ANN architectures that are shown is the best model based on degree of certainty. A good model has a high prediction value. The ANN-L36prim architecture in your method yields the highest prediction value for the 25% criterion on Dataset_3 (84.1%). On Dataset_3, the ANN-L36prim architecture achieves the highest prediction value for the 30% criterion (86.4%). The ANN-L36prim design likewise has the highest prediction value for the 50% criterion in Dataset_1 (98.1%).

In this context, the prediction value represents the percentage of projects with an MRE value < 25%, 30%, 50%, respectively. This information is summarized in Table 57 and provides valuable insights into the performance of the different models across various datasets and confidence levels.

Table 57. Prediction measured on three criteria (COSMIC FFP).

PRED (%)	P(25)		P(30)		P(50)	
	ANN -L12	ANN -L36prim	ANN -L12	ANN -L36prim	ANN -L12	ANN -L36prim
Dataset_1	39.6	37.7	49.1	52.8	83.0	98.1
Dataset_2	33.3	31.7	41.3	44.4	69.8	82.5
Dataset_3	79.5	84.1	86.4	86.4	95.5	97.7
Dataset_4	42.3	42.3	51.3	51.3	78.2	80.8
Dataset_5	50.0	59.1	54.5	63.6	81.8	86.4
Dataset_6	28.6	28.6	28.6	35.7	78.6	78.6
Dataset_7	30.3	30.3	42.4	57.6	81.8	84.8

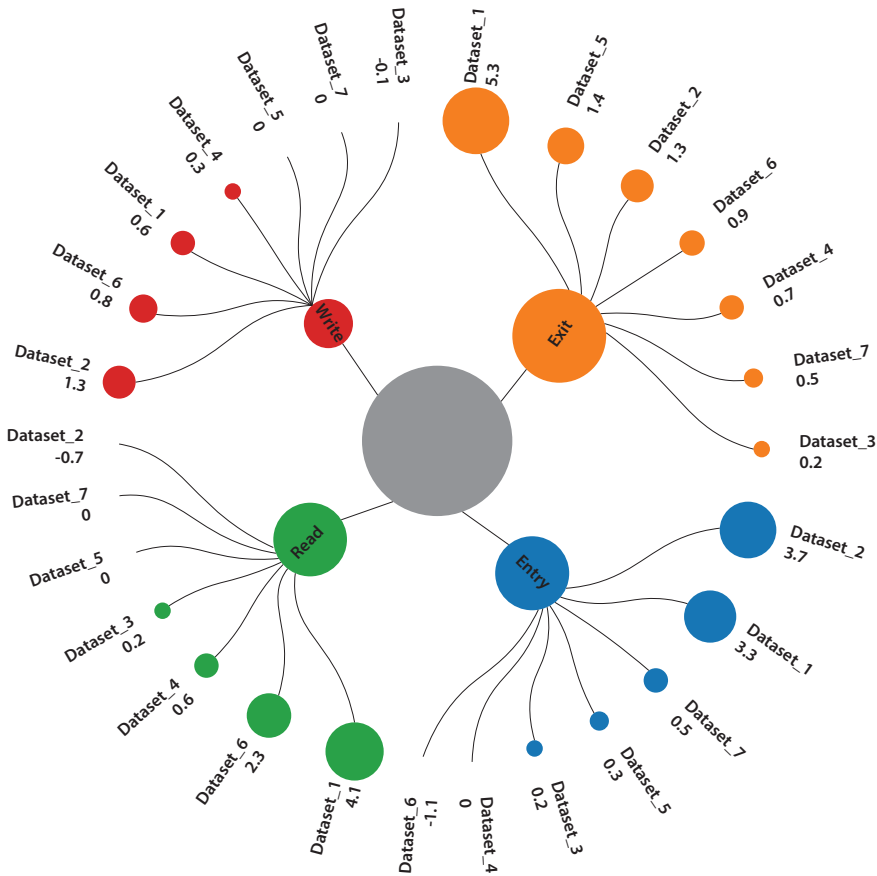
The impact of the input values of the COSMIC FFP method on the MMRE value was also examined, in addition to the criteria offered for measuring the efficacy, efficiency, and precision of the suggested approach. This was accomplished by individually deleting the 4 input values (Entry, Exit, Read, and Write) while monitoring the change in MMRE relative to the mean (MMRE) for the proposed architecture on each chosen dataset. The impact of each input value on the final projected project value is ascertained through this experimental research.

It can be deduced that with the ANN-L12 design, the second input value (Exit) has the greatest impact on Dataset_1, resulting in a 5.3% rise in the MMRE value. On the other hand, Dataset_6 is least affected by the first input value (Entry), which causes a -1.1% change in the MMRE score. In other words, Dataset_1's Exit value raises the MMRE value by 5.3%, whereas Dataset_6's Entry value lowers the MMRE value by 1.1%. The influence of all four input values across all seven datasets falls within the interval of [-1.1%, 5.3%]. A value of 0.0% implies that the input value does not significantly affect the change in the MMRE value, as illustrated in Figure 53 and Table 58. This analysis provides insights into the sensitivity of the model to different input values.

Table 58. Influence of input values on MMRE value ($\delta(\%)$) - ANN-L12 architecture.

$\delta(\%)$	Dataset_1	Dataset_2	Dataset_3	Dataset_4	Dataset_5	Dataset_6	Dataset_7
Entry	3.3	3.7	0.2	0.0	0.3	-1.1	0.5
Exit	5.3	1.3	0.2	0.7	1.4	0.9	0.5
Read	4.1	-0.7	0.2	0.6	0.0	2.3	0.0
Write	0.6	1.3	-0.1	0.3	0.0	0.8	0.0

Figure 53. Graphical representation of the influence of input values on MMRE value (ANN-L12 COSMIC FFP).

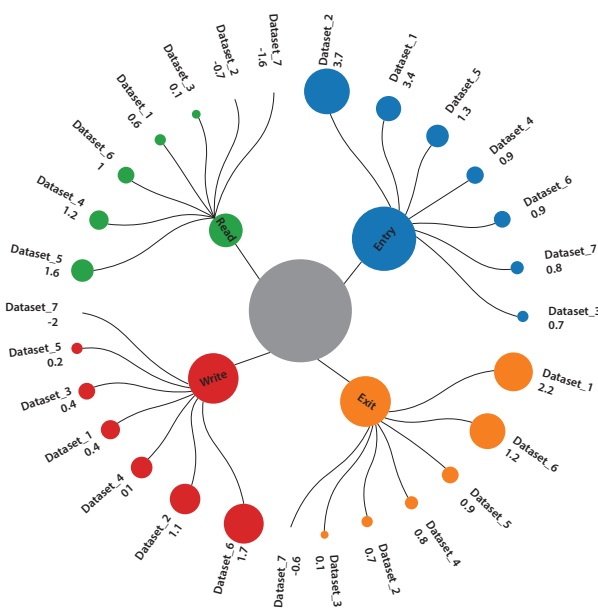


When examining how the input values for the ANN-L36prim architecture affect Dataset_2, it is clear that the first input value (Entry) has the biggest effect, increasing the Mean Magnitude of Relative Error (MMRE) value by 3.7%. On the other hand, Dataset_7 is least affected by the fourth input value (Write), which causes a -2.0% change in the MMRE score. To put it another way, the Write value on Dataset_7 lowers the MMRE value by 2.0% while the Entry value on Dataset_2 raises it by 3.7%. Figure 54 and Table 59 show that the influence of each of the four input values individually on each of the seven datasets is between [-2.0%, 3.7%]. This analysis provides information on how the performance of the model varies depending on the input value across several datasets.

Table 59. Influence of input values on MMRE value ($\delta(\%)$) - ANN-L36prim architecture.

$\delta(\%)$	Dataset_1	Dataset_2	Dataset_3	Dataset_4	Dataset_5	Dataset_6	Dataset_7
Entry	3.4	3.7	0.7	0.9	1.3	0.9	0.8
Exit	2.2	0.7	0.1	0.8	0.9	1.2	-0.6
Read	0.6	-0.7	0.1	1.2	1.6	1	-1.6
Write	0.4	1.1	0.4	1	0.2	1.7	-2.0

Figure 54. Graphical representation of the influence of input values on MMRE value (ANN-L36prim COSMIC FFP).



Software functionality is evaluated using a model called COSMIC FFP. Based on particular user needs and functional specifications for the program being built, it is possible to further improve the accuracy of the MMRE by monitoring the influence of input values such as Entry, Exit, Read, and Write. When the individual effects of each of these 4 input values are examined, it becomes clear that Entry and Exit values have the most influence on MMRE. Therefore, it is critical to undertake additional research and testing on these 2 input values to ascertain whether they meet all of their requirements.

There are ways to improve and optimize the Entry value, which reflects user messages to the system, to lessen its effect on MMRE. The Exit value, which also represents system replies, file reads, and the outcomes of logical and mathematical processes, can be adapted and altered to reduce its impact on MMRE. It becomes feasible to improve the effectiveness, accuracy, stability, and reliability of the suggested technique by making these modifications and adjustments that are matched to the system's user needs, finally resulting in the success of software project development.

The models and tools created in this context are intended to estimate the time required to develop software projects based on functional points that represent the program's functional size. Compared to previously utilized FPA methods, recent research and many businesses prefer the COSMIC FFP method because it enables the early evaluation of development effort.

Together with your strategy and applied methodology, the findings of the COSMIC FFP method show that it is possible to estimate functional size with a high level of accuracy. These results imply that it is possible to obtain more precise functional size estimates based on a set of base functional components (BFC) when working with a group of projects inside a particular application domain, requiring particular computing requirements and implementation technologies. When time restrictions make it difficult to use other conventional ways before delivering full needs specifications, this approach might be quite helpful.

The results shown in the previous tables and figures of this approach (COCOMO2000) were processed in the R programming language (available at: <https://www.r-project.org/>) and checked in the Python programming language (available at: <https://www.python.org/>) within the RStudio environment (available at: <https://www.rstudio.com/products/rstudio/>). The data required for statistical analysis were processed in the IBM SPSS Statistical 25 software tool (available at: <https://www.ibm.com/support/pages/downloading-ibm-spss-statistics-25>).

6.3. UCP HYBRID MODEL AND ANN

Like the functional points model, the UCP (Use Case Points) paradigm enables the measuring of system size. This methodology determines the effort and expense necessary to realize a software project by evaluating system characteristics and using use cases. Due of the outstanding evaluation outcomes UCP may produce, it is commonly employed. However, this model has a flaw in that it fails to take into consideration the data structure of the system because use cases rarely incorporate such data..

The results of utilizing the provided dataset to train the first recommended ANN-L16 design are shown in Table 60. It was kept track of how many iterations were required to satisfy the specified GA criterion. After four iterations, the GA requirement was met. The ANN network with the minimized MRE value based on all conducted iterations was named the "Winner" ANN network. Each iteration also included a calculation of the MMRE value. The "Winner" ANN network's (ANN6) achieved value is 6.7%, and the MMRE value is 7.1%. These findings shed light on how well the ANN model performs when estimating effort and cost using the UCP model.

Table 60. ANN-L16 results of training part.

ANN-L16	Iteration1	Iteration2	Iteration3	Iteration4
ANN1	0.0838	0.0795	0.0756	0.0721
ANN2	0.1960	0.1125	0.0820	0.0728
ANN3	0.1883	0.1128	0.0814	0.0724
ANN4	0.0847	0.0771	0.0742	0.0717
ANN5	0.1609	0.1053	0.0805	0.0726
ANN6	0.0694	0.0676	0.0672	0.0669
ANN7	0.0785	0.0726	0.0707	0.0700
ANN8	0.1508	0.1053	0.0811	0.0729
ANN9	0.1908	0.1199	0.0758	0.0721
ANN10	0.0730	0.0797	0.0741	0.0728
ANN11	0.0785	0.0797	0.0749	0.0721
ANN12	0.1304	0.0836	0.0738	0.0703
ANN13	0.1130	0.0830	0.0735	0.0712

ANN14	0.0944	0.0802	0.0725	0.0706
ANN15	0.0943	0.0797	0.0727	0.0706
ANN16	0.1022	0.0815	0.0739	0.0714
GA	16	10	5	0
"Winner" ANN	0.0694	0.0676	0.0672	0.0669
MMRE	0.1181	0.0887	0.0753	0.0714

The results of training the second suggested ANN-L36prim architecture using the UCP model's given dataset are shown in Table 61. The number of iterations needed to satisfy the stated GA criterion was tracked, and it was discovered that after six iterations, the condition was satisfied. The "Winner" ANN network, which is the ANN network with the lowest MRE value, was chosen based on the MRE values recorded in each executed iteration. Each cycle also included the calculation of the MMRE value. The "Winner" ANN network's (ANN10) achieved value is 6.9%, and the MMRE value is 7.0%. These findings shed light on how well the ANN model performs when assessing effort and cost utilizing.

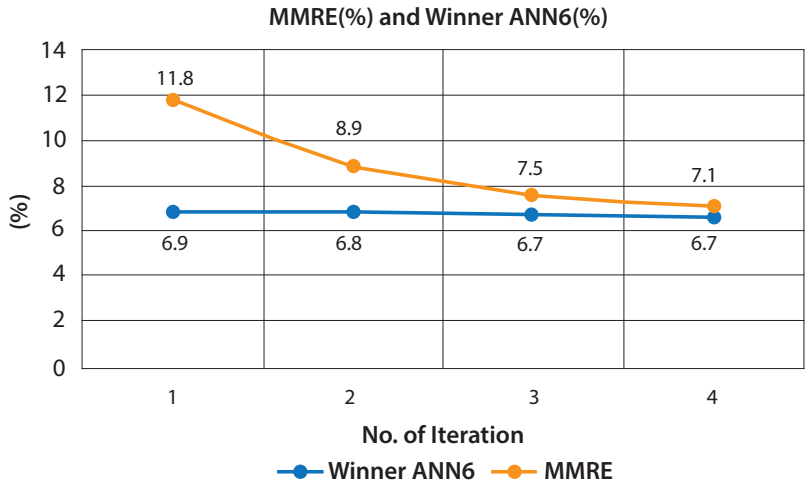
Table 61. ANN-L36prim results of training part.

ANN-L36prim	Iteration1	Iteration2	Iteration3	Iteration4	Iteration5	Iteration6
ANN1	0.085	0.072	0.060	0.045	0.043	0.041
ANN2	0.085	0.073	0.060	0.045	0.043	0.041
ANN3	0.096	0.083	0.071	0.056	0.054	0.052
ANN4	0.095	0.083	0.070	0.055	0.053	0.051
ANN5	0.097	0.085	0.072	0.057	0.055	0.053
ANN6	0.084	0.071	0.059	0.044	0.041	0.039
ANN7	0.100	0.087	0.075	0.060	0.057	0.055
ANN8	0.090	0.078	0.065	0.050	0.047	0.045
ANN9	0.080	0.067	0.055	0.040	0.037	0.035
ANN10	0.073	0.060	0.048	0.033	0.030	0.028
ANN11	0.093	0.080	0.068	0.053	0.050	0.048
ANN12	0.109	0.097	0.084	0.069	0.066	0.064
ANN13	0.077	0.065	0.052	0.037	0.034	0.032
ANN14	0.077	0.065	0.052	0.037	0.034	0.032

ANN15	0.078	0.046	0.043	0.028	0.025	0.023
ANN16	0.116	0.084	0.081	0.066	0.063	0.061
ANN17	0.143	0.110	0.108	0.093	0.090	0.088
ANN18	0.127	0.095	0.092	0.077	0.074	0.072
ANN19	0.117	0.084	0.072	0.057	0.053	0.051
ANN20	0.103	0.070	0.068	0.053	0.049	0.047
ANN21	0.161	0.128	0.126	0.111	0.107	0.105
ANN22	0.155	0.123	0.120	0.105	0.102	0.100
ANN23	0.102	0.070	0.067	0.052	0.049	0.047
ANN24	0.132	0.100	0.097	0.082	0.079	0.077
ANN25	0.090	0.057	0.055	0.040	0.037	0.035
ANN26	0.085	0.052	0.050	0.045	0.042	0.040
ANN27	0.094	0.062	0.059	0.054	0.051	0.049
ANN28	0.187	0.154	0.152	0.147	0.143	0.141
ANN29	0.180	0.148	0.145	0.140	0.137	0.135
ANN30	0.172	0.140	0.137	0.122	0.119	0.117
ANN31	0.180	0.137	0.135	0.130	0.127	0.125
ANN32	0.174	0.142	0.139	0.134	0.131	0.129
ANN33	0.186	0.153	0.151	0.146	0.142	0.140
ANN34	0.197	0.145	0.142	0.137	0.134	0.132
ANN35	0.163	0.111	0.108	0.103	0.100	0.098
ANN36	0.175	0.123	0.120	0.115	0.112	0.110
GA	36	36	23	14	3	0
"Winner" ANN23	0.073	0.072	0.070	0.070	0.070	0.069
MMRE	0.121	0.094	0.088	0.075	0.072	0.070

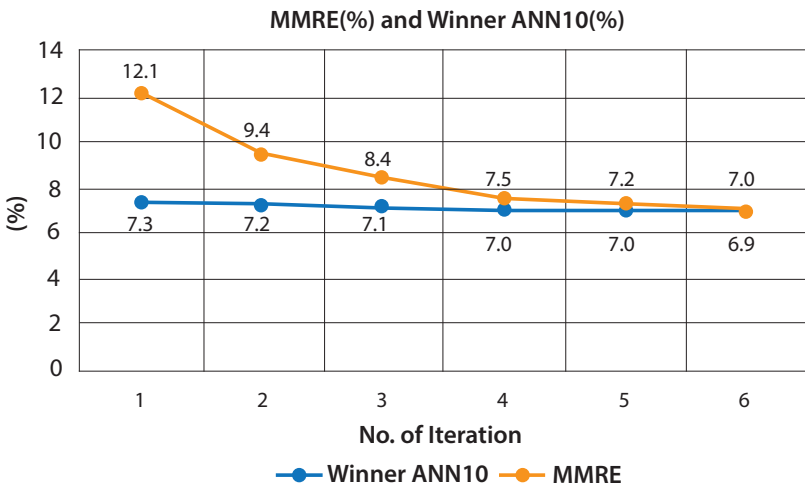
The MRE value for the "Winner" ANN network in relation to the MMRE value on the training dataset across the four iterations is shown graphically in Figure 55. This image gives an impression of the convergence and performance of the ANN model in relation to the UCP model by showing how the MRE value for the "Winner" ANN network varies over the course of the iterations.

Figure 55. "Winner" MRE vs. MMRE on the training dataset (ANN-L16).



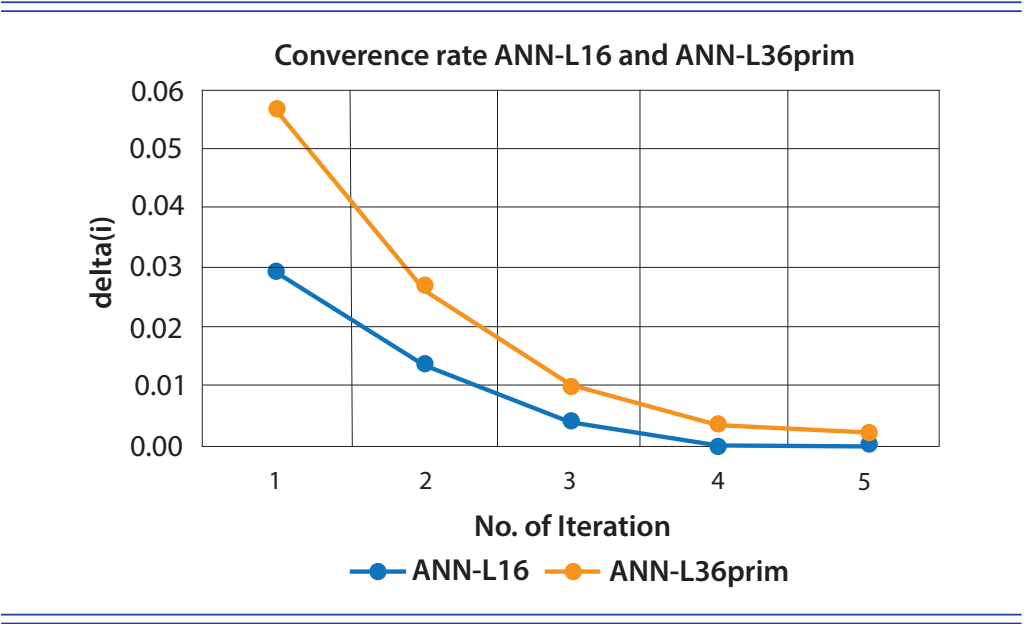
The MRE value for the "Winner" ANN network in relation to the MMRE value on the training dataset throughout the course of four iterations is shown graphically in Figure 56. This graph shows the evolution of the MRE value for the "Winner" ANN network over time and how it relates to the total MMRE value. Understanding the convergence and effectiveness of the ANN model in calculating effort and cost using the UCP model is made possible by these graphics.

Figure 56. "Winner" MRE vs. MMRE on the training dataset (ANN-L36prim).



The examination of the convergence rate on all training data for both ANN architectures, ANN-L36prim and ANN-L16, reveals that the ANN-L36prim architecture converges more quickly to the minimum MMRE compared to the ANN-L16 architecture. This observation is depicted in Figure 57, which visually demonstrates the convergence behavior of the two architectures. A faster convergence to the minimum MMRE indicates that the ANN-L36prim architecture is more efficient and effective in estimating effort and cost using the UCP model.

Figure 57. Convergence rate ANN-L16 vs. ANN-L36prim (UCP).



The findings for the two suggested architectures, ANN-L16 and ANN-L36prim, in each of the three portions of the experiment show that the complexity of the architecture employed is not greatly impacted by the dataset's nature. Additionally, the values of the input parameters have little impact on how well these systems work.

In the first suggested architecture, ANN-L16, all six input values were utilized, with four of them being linearly dependent and two being linearly independent. The MMRE value remained consistent at 7.5% across all three parts of the experiment, as shown in Table 62.

The MMRE value for the second architecture, ANN-L36prim, which used four distinct input values, likewise held steady at 7.5% throughout the experiment's three phases, as shown in Table 64. The small error differences that were only 0.5% or less in each portion of the experiment further demonstrate the validity of the suggested model. These findings indicate that the suggested ANN designs, when applied to various datasets and input values, offer reliable and consistent estimations of effort and cost using the UCP model.

Table 62. MMRE value in all three parts of the experiment (UCP).

Datasets	ANN-L16	ANN-L36prim	Part of experiment
	MMRE(%)	MMRE(%)	
Dataset_1	6.7	7.0	Training
	7.1	7.1	Testing
Dataset_2	8.0	7.5	Validation1
Dataset_3	8.3	8.4	Validation2
AVERAGE(MMRE)	7.5	7.5	

The Pearson's and Spearman's rho correlation coefficients, which have large values, show that the estimated and real values obtained using the suggested models are highly consistent. In particular, Pearson's correlation coefficient is stated to be 0.983 for the ANN-L36prim architecture. The suggested model's accuracy and dependability in estimating effort and cost using the UCP model are highlighted by the high Pearson's correlation value, which shows an exceptional and strong association between the observed values, as shown in Table 63.

Table 63. Correlation coefficients (UCP).

Correlation	ANN-L16	ANN-L36prim
Pearson's	0.875	0.983
Spearman's rho	0.784	0.962

The prediction meter, which measures the proportion of projects with errors below a given standard, enables to verify the accuracy and dependability of the used models. The prediction value is constantly 100% for all three proposed criteria, PRED (25), PRED (30), and PRED (50), and for all three experimental phases (training, testing, and validation), when utilizing both of the suggested structures, as shown in Table 64. This steady 100% prediction rate highlights the models' ability to accurately and successfully estimate effort and cost using the UCP model, across a variety of criteria and experimentation phases.

Table 64. Prediction values (UCP).

Training		
PRED(%)	ANN-L16(%)	ANN-L36prim(%)
PRED(25)	100.0	100.0
PRED(30)	100.0	100.0
PRED(50)	100.0	100.0
Testing		
PRED(25)	100.0	100.0
PRED(30)	100.0	100.0
PRED(50)	100.0	100.0
Validation1		
PRED(25)	100.0	100.0
PRED(30)	100.0	100.0
PRED(50)	100.0	100.0
Validation2		
PRED(25)	100.0	100.0
PRED(30)	100.0	100.0
PRED(50)	100.0	100.0

Through the change in the MMRE value, it has been demonstrated that using a 4-dimensional vector instead of a 6-dimensional vector is sufficient. The errors associated with dependent input values on the four datasets used range from -0.3 to 0.5, all within the range of less than 1%.

Among the input values, the most significant influence is observed with the input value of AUCP (Dataset_3), resulting in an increase in the MMRE value by 0.5%. On the other hand, the least significant influence is observed with the input value of UUCW (Dataset_4), leading to a decrease in the MMRE value by 0.5%. This indicates that further analysis of observed values could potentially reduce or increase the error.

The architecture with 6 input quantities can, therefore, be successfully substituted by the design with 4 input quantities, it can be said. In this situation, the suggested strategy, shown in Table 65, can use the ANN-L36prim architecture in place of the ANN-L16 architecture. This replacement simplifies the input vector and lowers computational complexity while maintaining or increasing the model's accuracy..

Table 65. Influence of the input values on the MMRE change (UCP).

Dataset	MMRE	UAW	UUCW	UUCP	TCF	ECF	AUCP
Dataset_1	6.7%	7.1%	6.7%	7.0%	6.7%	6.7%	7.1%
Dataset_2	7.0%	7.1%	7.0%	7.2%	6.9%	7.1%	7.2%
Dataset_3	8.0%	7.9%	8.1%	7.9%	8.1%	8.1%	7.5%
Dataset_4	8.3%	7.9%	8.4%	7.9%	8.3%	8.2%	8.0%

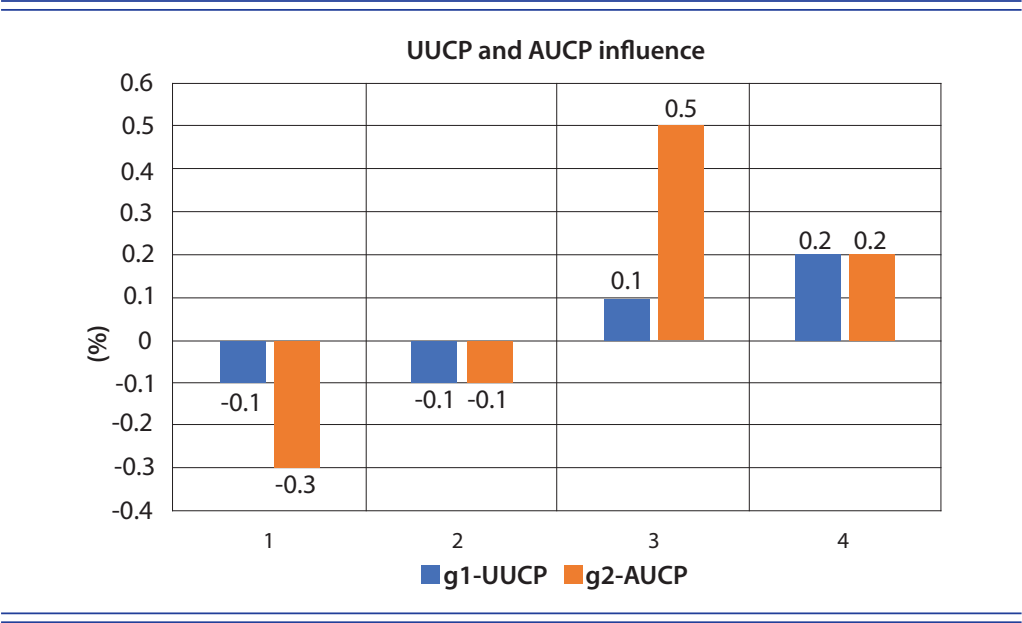
According to Table 66, the dependent variable UUCP has a smaller effect than the dependent variable AUCP. The MMRE value changes by 0.5% as a result of the strongest influence, which is AUCP (Dataset_3). On the other hand, AUCP (Dataset_1) exhibits the least significant influence, causing a -0.3% change in the MMRE value. In the context of the observed datasets, this study further highlights the diverse effects of various dependent factors on the MMRE value.

Table 66. Influence of dependent variables (UUCP and AUCP) on the change of MMRE value.

Dataset	UUCP	g-UUCP=MMRE-UUCP	AUCP	g-AUCP=MMRE-AUCP
Dataset_1	6.9	-0.1	6.8	-0.3
Dataset_2	7.1	-0.1	7.1	-0.1
Dataset_3	8.0	0.1	8.0	0.5
Dataset_4	8.2	0.2	8.2	0.2
	max	0.2%	max	0.5%
	min	-0.1%	min	-0.3%

The accompanying input values for UUCP and AUCP, as well as their respective error values, are shown graphically in Figure 58.

Figure 58. Influence of dependent variables (UUCP and AUCP) on the change of MMRE value.



The results shown in the previous tables and figures of this approach (COCOMO2000) were processed in the R programming language (available at: <https://www.r-project.org/>) and checked in the Python programming language (available at: <https://www.python.org/>) within the RStudio environment (available at: <https://www.rstudio.com/products/rstudio/>). The data required for statistical analysis were processed in the IBM SPSS Statistical 25 software tool (available at: <https://www.ibm.com/support/pages/downloading-ibm-spss-statistics-25>).

6.4. COMPARING THREE NOVEL HYBRID MODELS

It is clear that the enhanced COCOMO2000 approach with ANN reduces model error by 4.5 times ($193.1/43.3$) as compared to the parametric COCOMO2000 method. The model error reduction in the second suggested approach is 6.7 times ($193.1/28.8$) when comparing the parametric COCOMO2000 method with the enhanced COSMIC FFP using ANN. The model error reduction is a significant 25.7 times ($193.1/7.5$) less than the parametric COCOMO2000 technique with UCP and ANN, Figure 59.

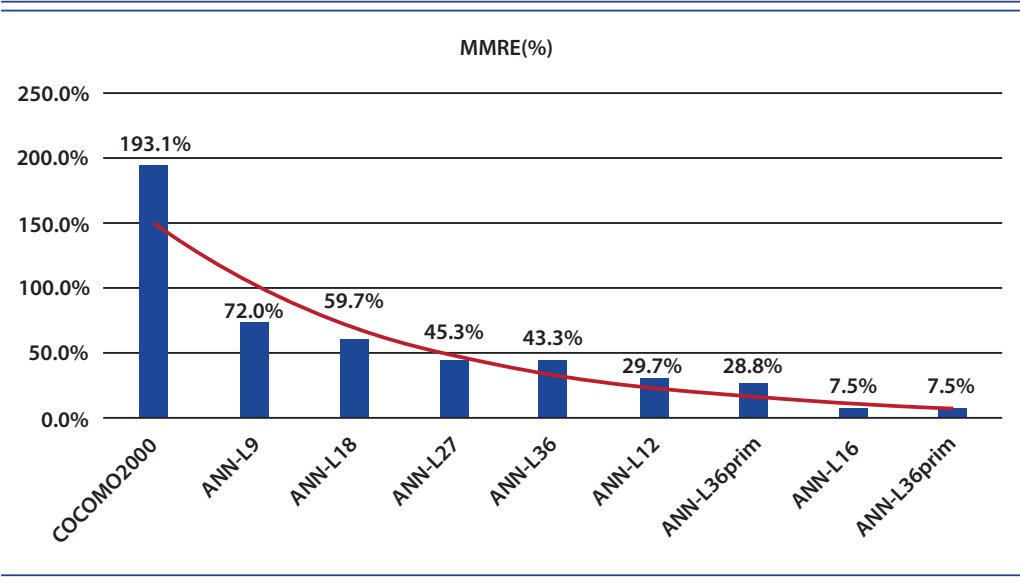
The ANN-L36 architecture has the first proposed method's lowest model error value of 43.3%. The ANN-L36prim algorithm yields a 28.8% error value in the second recommended strategy. The ANN-L16 and ANN-L36prim suggested architectures produce the lowest model error value of 7.5% in the third recommended strategy. This illustrates that the lowest MMRE value is attained with the third recommended UCP technique. Additionally, this approach's ANN-L16 architecture converges quickly, meeting the "stop criterion" after only 4 iterations, the fewest iterations of any architecture employed in any of the three described ways.

Less than 0.5% of the MMRE value change in the ANN-L16 architecture is significantly impacted by dependent variables. As a result, it can be said that the best estimator for effort and cost in software project development is the enhanced UCP model employing the ANN-L16 architecture.

Table 67. MMRE values for used approaches.

MMRE(%)	COCOMO2000 and ANN				COSMIC FFP and ANN		UCP and ANN	
COCOMO2000	ANN-L9	ANN-L18	ANN-L27	ANN-L36	ANN-L12	ANN-L36prim	ANN-L16	ANN-L36prim
193.1%	72.0%	59.7%	45.3%	43.3%	29.7%	28.8%	7.5%	7.5%

Figure 59. MMRE value for used approaches.



By selecting the best ANN architectures, which achieved the lowest MMRE value for each of the three proposed, improved models, it can be concluded that: COSMIC FFP with ANN is $43.3/28.8=1.5$ times better than COCOMO2000 with ANN; UCP with ANN is $48.8/7.5=5.8$ times better than COCOMO2000 with ANN; UCP with ANN is $28.8/7.5=3.8$ times better than COSMIC FFP with ANN, see Table 68.

Table 68. COCOMO2000 and ANN vs. COSMIC FFP and ANN vs. UCP and ANN.

	COCOMO2000 and ANN	COSMIC FFP and ANN	UCP and ANN	
	ANN-L36	ANN-L36prim	ANN-L16	ANN-L36prim
MMRE(%)	43.3%	28.8%	7.5%	7.5%

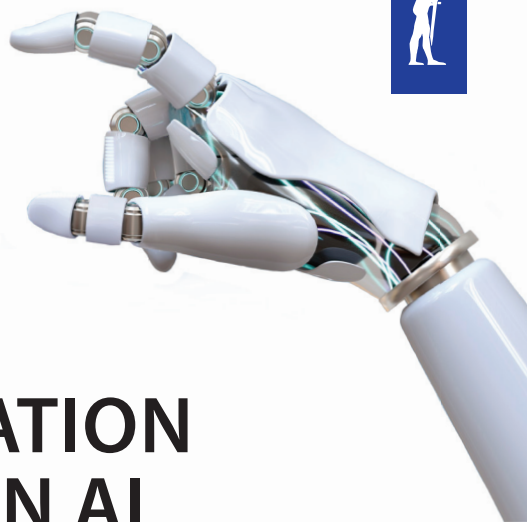
Throughout the software product development lifecycle, data collection and utilization play pivotal roles. Employing various machine learning tools, an array of analyses is conducted, leading to specific conclusions that inform critical decisions related to project success [114], [115]. Many tasks within the assessment process can be framed as machine learning problems. Artificial neural networks, in particular, offer a multitude of avenues for addressing these challenges during training [119]. Evaluating new proposed solutions and projects accurately necessitates experimentation with a sufficient number of realistic projects.

The dynamic information environment demands rapid and efficient adjustments to the effort and cost estimation process. Challenges that persist in the context of ANN estimation encompass dataset clustering, determining the optimal number of projects within each dataset, and employing diverse algorithms for cluster size assessment through various statistical analyses.

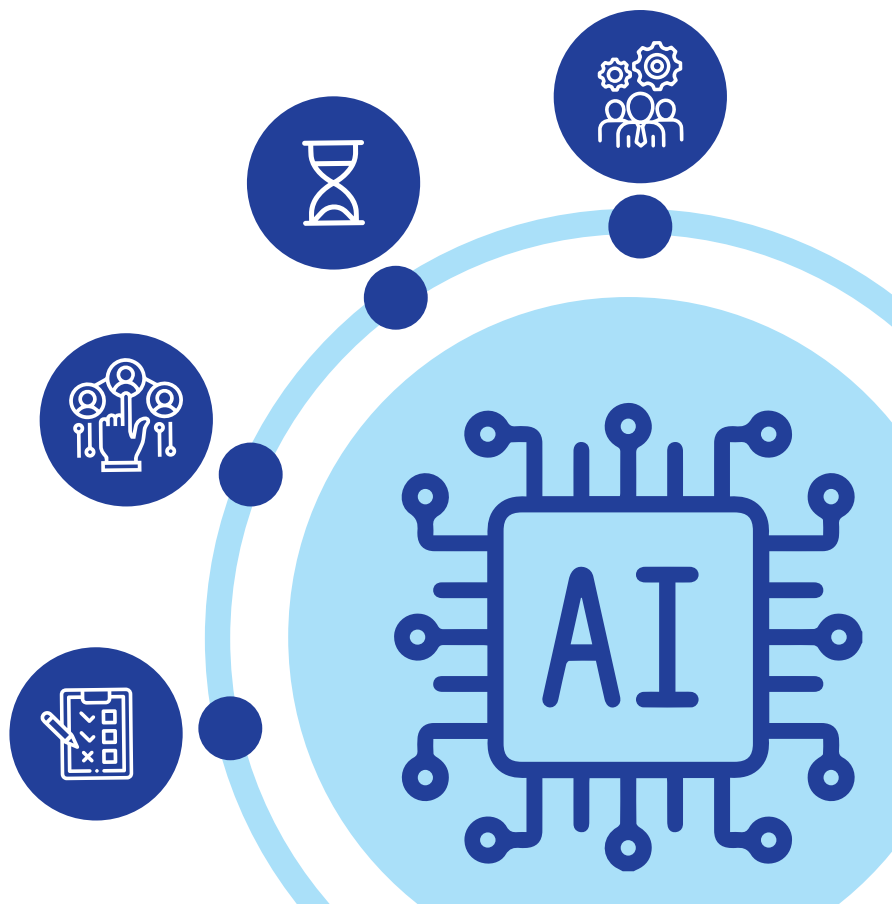
Additional advancements in cost and effort estimation can be made beyond the experiments done with the three new models that have been proposed and different ANN architectures by looking into different activation functions, encoding/decoding techniques, and analyzing the effect of input variables on relative error variations, among other things. Although the suggested models outperform conventional approaches in terms of advantages, each parameter influencing estimation necessitates careful consideration and numerous experimental iterations to achieve stable and trustworthy findings.

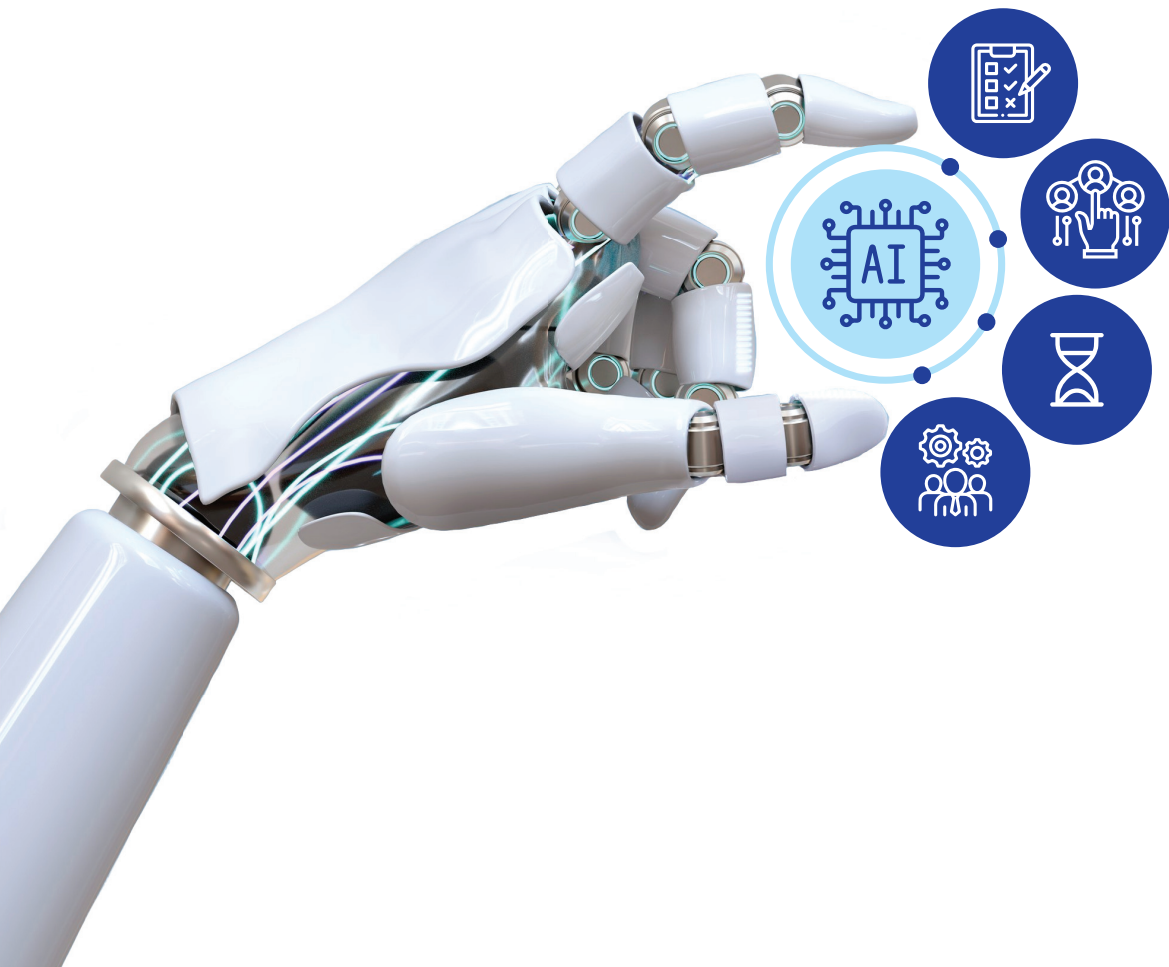


CHAPTER 7



GENERALIZATION PROBLEM IN AI





7. GENERALIZATION PROBLEM IN AI

7.1. WHAT IS THE REQUIRED NUMBER OF PROJECT IN DATASET?

The choice of suitable datasets is essential for the training process involving various ANN topologies, depending on the suggested methodologies. The model, input dimensions, and accessibility of publicly available databases all influence how suitable a dataset is.

The COCOMO2000 dataset was used for the ANN training process in the first proposed hybrid model. The validation method included the COCOMO81, NASA60, and Kemerer datasets, while the same dataset was utilized to test ANN performance on other projects. Each chosen dataset must have more projects than there are weighting elements in the suggested ANN architecture in order to enable robust training and evaluation.

The minimum project count criterion was satisfied for the ANN-L9, ANN-L18, and ANN-L27 architectures across all employed datasets, as shown in Table 71. The requisite requirement was not met for Dataset_2, where there were 20 testing projects, for the ANN-L36 architecture, which has 23 weighting coefficients. The project count was insufficient for the third validation using Dataset_5, which included 23 weighting coefficients, at 15.

In summary, it can be inferred that the first proposed hybrid model met the conditions in 90% of all experiment phases, as outlined in Table 69.

Table 69. The number of projects greater than the number of weighting factors (COCOMO2000).

Datasets		No. of projects	Experiment	Weighting factors (coefficients)			
				ANN-L9 (4)	ANN-L18 (8)	ANN-L27 (13)	ANN-L36 (23)
Dataset_1	COCOMO2000	100	Training	+	+	+	+
Dataset_2	COCOMO2000	20	Testing	+	+	+	/
Dataset_3	COCOMO81	51	Validation1	+	+	+	+
Dataset_4	NASA	60	Validation2	+	+	+	+
Dataset_5	Kemerer	15	Validation3	+	+	+	/

In the second suggested hybrid model, the ISBSG dataset was divided into five distinct clusters, and these clusters were utilized for the ANN training process. The same dataset, partitioned into these five clusters, was subsequently employed for ANN testing. In this setup, the ratio of projects in both training and testing phases was 70:30, with 70 projects allocated for training and 30 for testing. For the validation process, additional datasets, including the Desharnais dataset and a dataset aggregated from various real-world industrial projects, were used. It's crucial that each selected dataset contains more projects than the number of weighting factors in the proposed ANN architecture.

One dataset (Dataset_5) for the ANN-L12 architecture, which has 11 weighting coefficients, did not meet the requirement because there were only 7 projects available for testing.

The requirement was not met for Dataset_1, which included 15 testing projects using the ANN-L36prim architecture, which has 16 weighting coefficients. Similar to the third cluster, where there were 13 testing projects, the necessary criterion for Dataset_3 was not satisfied. In addition, 14 projects in Dataset_5 that were accessible for both training and testing did not satisfy the requirement. The requirement was not met in the 14 projects comprising the Desharnais dataset (Dataset_6).

Overall, the requirements were satisfied across all phases of the experiment in 7 cases out of 12 datasets, showing a fulfillment rate of 75% in the second suggested technique, as shown in Table 70.

Table 70. The number of projects greater than the number of weighting factors (COSMIC FFP).

		Weighting factors (coefficients)			
	Datasets	No. of projects	Experiment	ANN-L12 (11)	ANN-L36prim (16)
Dataset_1	ISBSG (Functional Size<10)	37	Training	+	+
	ISBSG (Functional Size<10)	15	Testing	+	/
Dataset_2	ISBSG (10<Functional Size<50)	45	Training	+	+
	ISBSG (10<Functional Size<50)	17	Testing	+	+
Dataset_3	ISBSG (50<Functional Size<100)	30	Training	+	+
	ISBSG (10<Functional Size<100)	13	Testing	+	/
Dataset_4	ISBSG (100<Functional Size<500)	60	Training	+	+
	ISBSG (10<Functional Size<500)	17	Testing	+	+
Dataset_5	ISBSG (Functional Size>500)	14	Training	+	/
	ISBSG (Functional Size>500)	7	Testing	/	/
Dataset_6	Desharnais	14	Validation1	+	/
Dataset_7	Combined	33	Validation2	+	+

In the third suggested hybrid model, the Benchmark (Mendeley) dataset was utilized for the ANN training process, following a 70:30 ratio, where 70% of the projects were allocated for training and 30% for testing. For the validation process, a combination of real, industrial projects (Dataset_3 and Dataset_4) was employed. It's imperative that each selected dataset contains more projects than the number of weighting factors in the proposed ANN architecture.

The criteria was satisfied for the ANN-L16 design, which has 15 weighting coefficients, ensuring that there were more projects than weighting coefficients for all phases of the experiment.

Similar to the ANN-L36prim architecture, which has 16 weighting coefficients, the criterion was met in all areas of the experiment because there were more projects than there were weighting coefficients.

As a result, the conditions were met in every phase of the experiment using the third suggested strategy, exhibiting a 100% fulfillment rate, as shown in Table 71.

Table 71. The number of projects greater than the number of weighting factors (UCP).

		Weighting factors (coefficients)			
	Datasets	No. of projects	Experiment	ANN-L16 (15)	ANN-L36prim (16)
Dataset_1	UCP Benchmark Dataset	50	Training	+	+
Dataset_2	UCP Benchmark Dataset	21	Testing	+	+
Dataset_3	Combined	18	Validation1	+	+
Dataset_4	Combined Industrial projects	17	Validation2	+	+

7.2. HYBRID MODELS VS. SVM ALGORITHM

The Support Vector Machine (SVM) technique was used to verify the suggested method and evaluate it against other AI tools. A well-known machine learning (ML) algorithm called SVM trains its models using observed data. The robustness and efficiency of SVM in both classification and regression tasks are well known. Additionally, it deals with the overfitting problem by reducing the predicted error [253], [254], [255]. The SVM machine algorithm divides the plane by the function f_{SVM} into two parts, so that the points (project input values) lie above or below the function f_{SVM} . Three different functions inside a kernel in SVM with radial basis function (RBF) were used:

- linear kernel function,
- quadratic kernel function,
- cubic kernel function.

The results obtained indicate a strong correlation (deterministic coefficient- R^2) between the estimated values in the training phase of the experiment for the ANN-L27 and ANN-L36 architectures, which are based on three input variables (E, PEMi, and KLOC). Graphical representations illustrating the relationship between the actual and estimated values using the SVM algorithm and the corresponding kernel functions for both architectures can be seen in Figure 60, Figure 61, and Figure 62.

Figure 60. Graphical representation using different kernel functions based on SVM (RBF) for ActEffort on the training dataset (COCOMO2000).

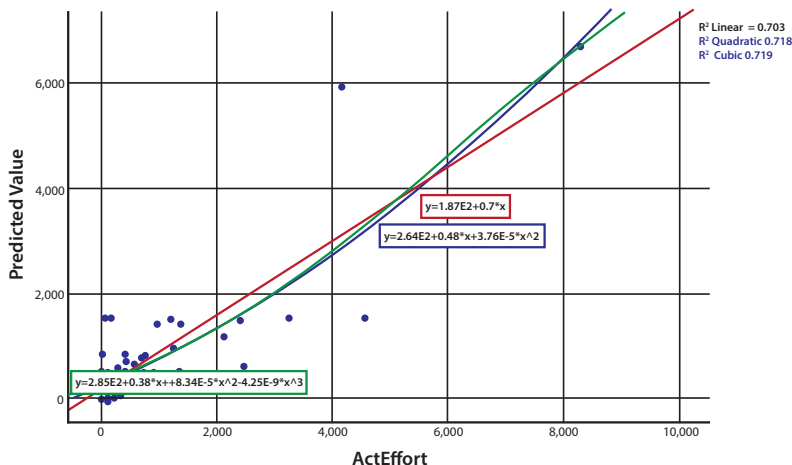


Figure 61. Graphical representation using different kernel functions based on SVM (RBF) for EstEffort on the training dataset (ANN-L27).

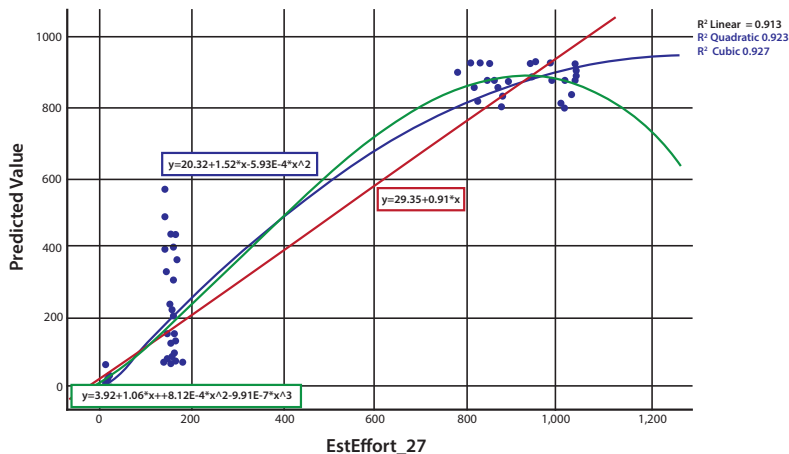
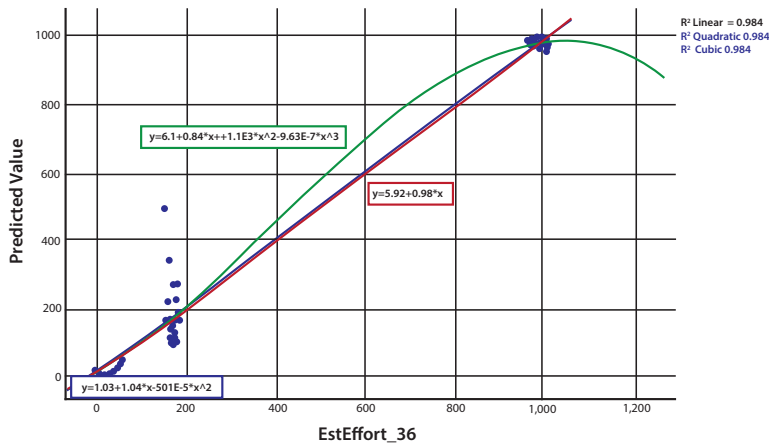


Figure 62. Graphical representation using different kernel functions based on SVM (RBF) for EstEffort on the training dataset (ANN-L36).



In comparison to the suggested method employing ANN and Taguchi Orthogonal Arrays, the MMRE values for all three kernel functions utilized in the SVM algorithm are higher (60% for the ANN-L27 architecture and 56.3% for the ANN-L36 architecture), as shown in Table 72. The MMRE value is 45.3% for the ANN-L27 design employing Taguchi Orthogonal Arrays and the COCOMO2000 model, and it is 43.3% for the ANN-L36 architecture.

Additionally, the high value of the deterministic coefficient, R^2 , employing three distinct kernel functions, confirmed the estimated values obtained using the revised COCOMO2000 model. This further demonstrates the suitability of the dataset's clustering.

Table 72. R^2 values using different kernel functions based on SVM (RBF) - COCOMO2000.

	Act Effort	EstEffort_27	EstEffort_36
R^2 Linear	0.703	0.913	0.984
R^2 Quadratic	0.718	0.923	0.984
R^2 Cubic	0.719	0.927	0.984
MMRE (%)		60.0	56.3

Based on 4 input quantities (Entry, Exit, Read, and Write), the estimated values for the ANN-L12 and ANN-L36prim designs in the training phase of the experiment show a very high degree of correlation (deterministic coefficient - R^2). Figures 63, 64, and 65, which use the SVM algorithm and the corresponding kernel functions for both architectures, provide graphical representations of the actual and estimated values that highlight these connections.

Figure 63. Graphical representation using different kernel functions based on SVM (RBF) for Functional Size on the training dataset (COSMIC FFP).

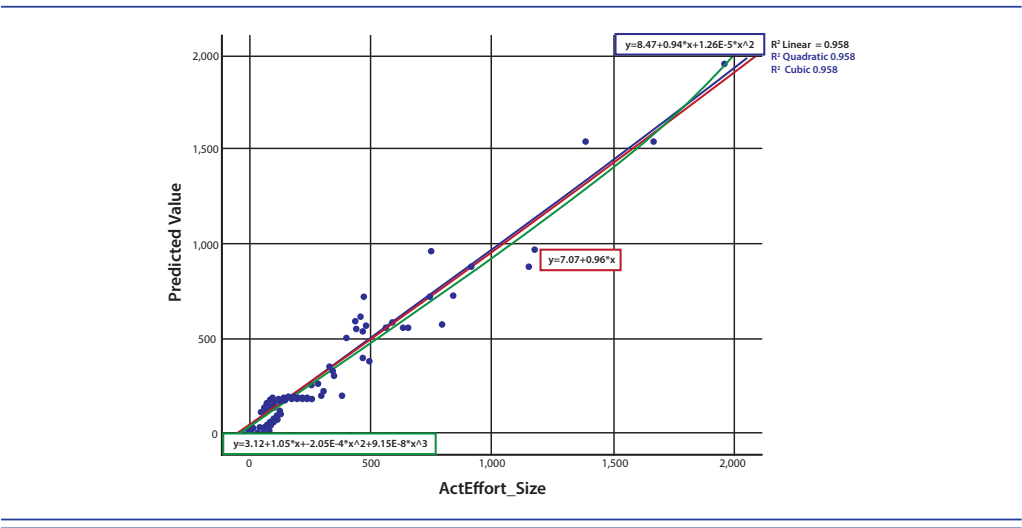


Figure 64. Graphical representation using different kernel functions based on SVM (RBF) for EstEffort on the training dataset (ANN-L12).

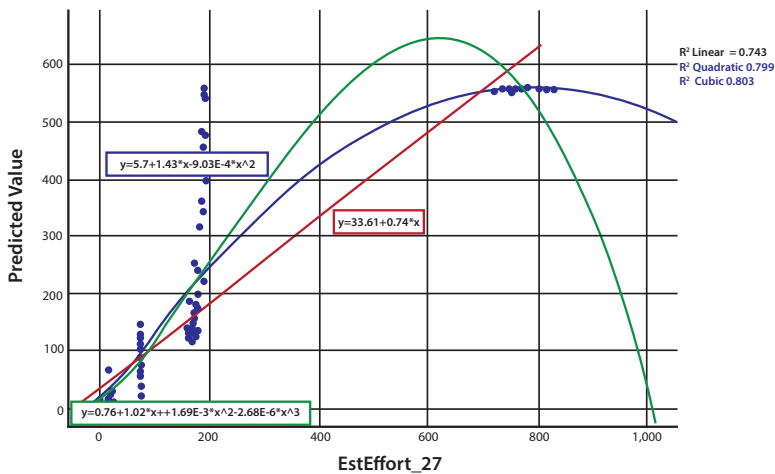
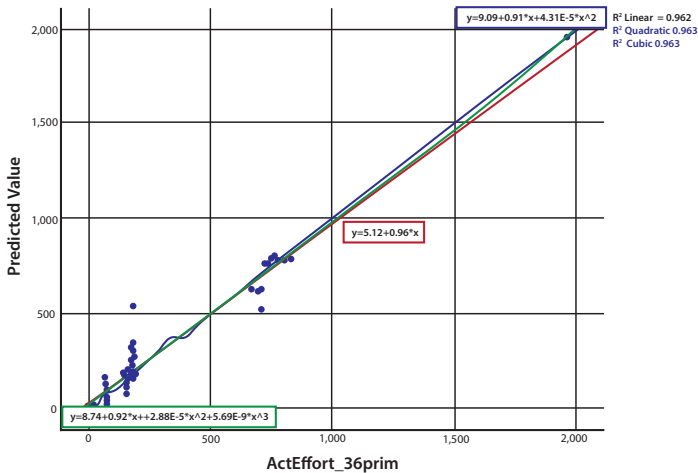


Figure 65. Graphical representation using different kernel functions based on SVM (RBF) for EstEffort on the training dataset (ANN-L36prim COSMIC FFP).



The MMRE values for both ANN-L12 and ANN-L36prim architectures using three different kernel functions in the SVM algorithm are higher (38.1% for ANN-L12 and 35.7% for ANN-L36prim) compared to the proposed approach utilizing ANN and Taguchi Orthogonal Arrays. In the case of the ANN-L12 architecture using Taguchi Orthogonal Arrays and the COSMIC FFP model, the MMRE value is 29.7%, while for ANN-L36prim, it's 28.8%, which indicates the superior performance of the proposed approach.

Furthermore, the high values of the deterministic coefficient (R^2) obtained using the improved COSMIC FFP model and three different kernel functions in the SVM algorithm reaffirm the adequacy of dataset clustering in the proposed approach.

Table 73. R^2 values using different kernel functions based on SVM (RBF) - COSMIC FFP.

	Act Effort	EstEffort_12	EstEffort_36prim
R^2 Linear	0.958	0.743	0.962
R^2 Quadratic	0.958	0.799	0.963
R^2 Cubic	0.959	0.803	0.963
MMRE (%)		38.1	35.7%

The obtained results show that there is a very high degree of correlation (deterministic coefficient - R^2) between the estimated values in the training phase of the experiment for both the ANN-L16 and ANN-L36prim architectures, which are based on 6 input variables (UAW, UUCW, TCF, ECF, UUCP, and AUCP). Figures 66, 67, and 68 provide graphic representations of the actual and estimated values obtained by the SVM method employing the respective kernel functions for both architectures.

Figure 66. Graphical representation using different kernel functions based on SVM (RBF) for Real Effort on the training dataset (UCP).

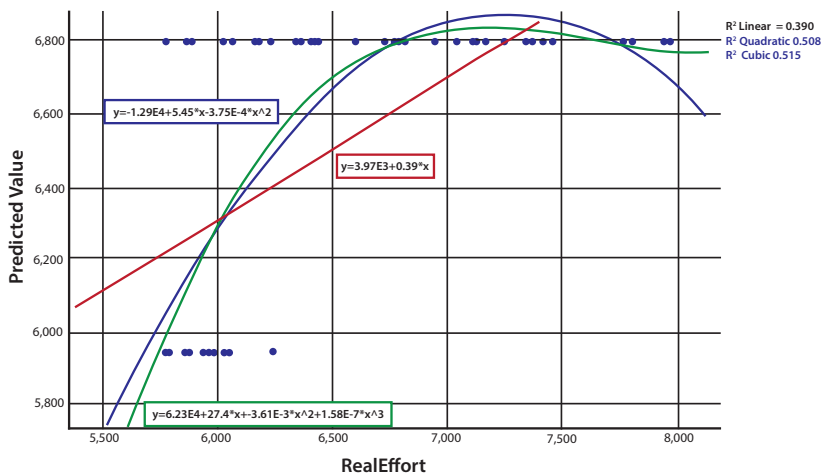


Figure 67. Graphical representation using different kernel functions based on SVM (RBF) for EstEffort on the training dataset (ANN-L16).

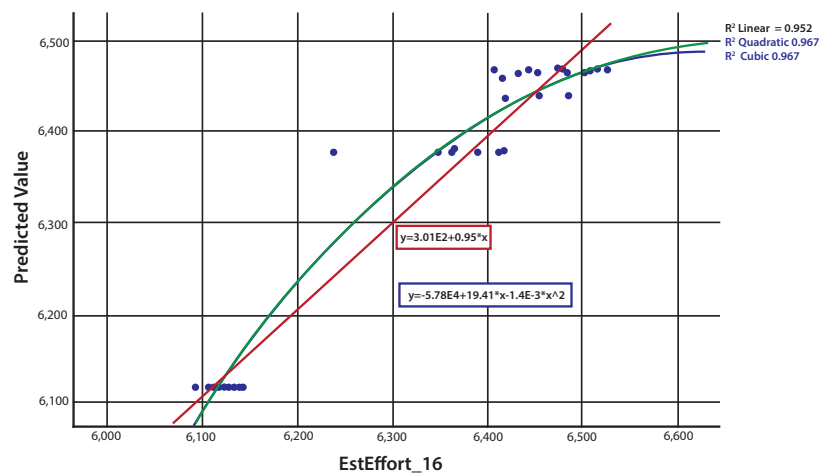
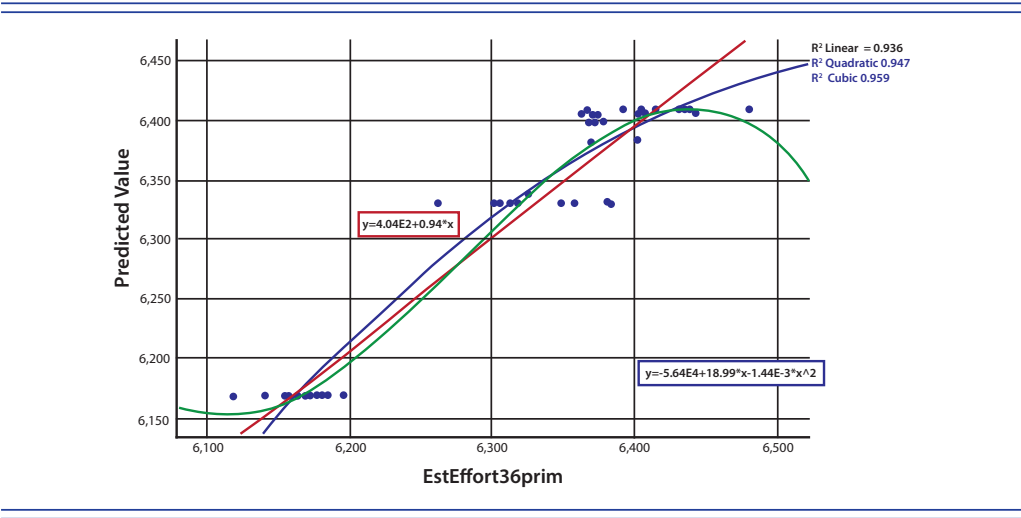


Figure 68. Graphical representation using different kernel functions based on SVM (RBF) for EstEffort on the training dataset (ANN-L36prim UCP).



In comparison to the suggested method employing ANN and Taguchi Orthogonal Arrays, the MMRE values achieved for the ANN-L16 and ANN-L36prim designs using the SVM algorithm with three distinct kernel functions are higher (13.2% for ANN-L16 and 13.5% for ANN-L36prim), as shown in Table 74. The MMRE value is 7.5% for the ANN-L36prim architecture and 7.5% for the ANN-L16 architecture employing Taguchi Orthogonal Arrays and the UCP FFP model, respectively.

Furthermore, the high value of the deterministic coefficient (R^2) obtained using the improved UCP model once again confirms the accuracy and reliability of the estimated values, with datasets being adequately divided into clusters

Table 74. R^2 values using different kernel functions based on SVM (RBF) - UCP.

	Act Effort	EstEffort_16	EstEffort_36prim
R^2 Linear	0.390	0.952	0.936
R^2 Quadratic	0.508	0.967	0.947
R^2 Cubic	0.515	0.967	0.959
MMRE (%)		13.2%	13.5%

The results show that even when compared to other machine-learning tools like the SVM method, our suggested approach is consistent, dependable, and effective.

7.3. ARCHITECTURAL INTRICACY OF ANNS

One of the primary objectives across all three proposed models is to choose the simplest possible ANN architecture while satisfying several additional criteria:

- Minimize the number of iterations, thus reducing estimation time.
- Opt for the architecture that converges most rapidly.
- Minimize the MMRE value.
- Utilize clustering and fuzzification techniques along with a sigmoid activation function.
- Employ a GA stop criterion to control the number of iterations.

The amount of input values, the count of weight coefficients, and the associated orthogonal vector plan are some of the variables that affect the architectural choice. Each suggested model offers either a combined orthogonal vector plan or a two- or three-level architecture. The adoption of the simplest architecture is continually prioritized in experiments, with nodes in the hidden layer being increased as needed and the number of hidden layers finally being increased.

Table 75 makes it clear that the COCOMO2000 model utilizes three input values, the COSMIC FFP model uses four, and the UCP model either uses six or four input values. Although the ANN-L9 design is built around the L9 orthogonal vector plan, it does not have a hidden layer. In comparison, the ANN-L36 architecture uses two hidden layers, the first of which has three nodes and the second of which has two nodes. The only hidden layer in the other ANN topologies that were used in the trials had two or three nodes. The estimated outcome is computed as a single output value by each of the three proposed models.

Table 75. Characteristics of ANN architecture.

Architecture	ANN -L9	ANN -L18	ANN -L27	ANN -L36	ANN -L12	ANN -L36prim	ANN -L16	ANN -L36prim
Input values	3	3	3	3	4	4	6	4
No. of hidden layers	0	1	1	2	1	1	1	1
No. of nodes	0	2	3	3+12	3	3	2	3
Output values	1	1	1		1	1	1	1

7.4. LINK BETWEEN CONVERGENCE RATE AND NUMBER OF ITERATIONS IN ANNS

One of the key objectives of the new and improved models is to minimize the number of iterations required, thereby reducing the time needed for estimation. Table 76 presents the number of iterations needed to meet the set GA criterion for each of the four specified ANN architectures.

The minimum number of iterations needed is for the ANN-L27 and ANN-L36 architectures, particularly for small and medium clusters. Conversely, the ANN-L18 architecture necessitates the highest number of iterations, requiring nine iterations in total. This suggests that the UCP model, when used with the ANN-L27 and ANN-L36 architectures, allows for rapid estimation, with only a minimal number of iterations.

In the second COSMIC FFP approach, both proposed architectures require six iterations, except for the ANN-L12 architecture, where five iterations are needed for Dataset_1 and Dataset_3. This demonstrates that the COSMIC FFP model enables rapid estimation, with a minimal number of iterations, as shown in Table 77.

For the first proposed ANN-L16 architecture used with the UCP model, 4 iterations are necessary to meet the set GA criterion. Conversely, the ANN-L36prim architecture requires six iterations. This indicates that the UCP model, when applied with the ANN-L16 architecture, facilitates even faster estimation with a minimal number of iterations, as illustrated in Table 78.

Table 76. Number of iterations performed for each ANN architecture - COCOMO2000.

COCOMO2000 models												
No. of Iteration GA<0.01	ANN-L9			ANN-L18			ANN-L27			ANN-L36		
	Small cluster	Medium cluster	Large cluster	Small cluster	Medium cluster	Large cluster	Small cluster	Medium cluster	Large cluster	Small cluster	Medium cluster	Large cluster
	7	6	8	7	6	9	5	5	8	5	5	7

Table 77. Number of iterations performed for each ANN architecture - COSMIC FFP.

COSMIC FFP models										
ANN	ANN-L12					ANN-L36prim				
Datasets	D_1	D_2	D_3	D_4	D_5	D_1	D_2	D_3	D_4	D_5
GA<0.01	5	6	5	6	6	6	6	6	6	6

Table 78. Number of iterations performed for each ANN architecture - UCP.

No. of Iterattion	UCP model	
	ANN-L16	ANN-L36prim
	Training	Training
GA<0.01	4	6

In all three suggested approaches, the ANN-L16 architecture for the UCP model requires a minimum of 4 iterations to meet the GA criterion, indicating that this architecture achieves the fastest convergence to the minimum MMRE value.

7.5. ACTIVATION FUNCTION CHOICE AND ENCODING/DECODING METHOD

In all studies in the three suggested techniques, different activation functions for the hidden and output layers were investigated. Sigmoid, hyperbolic tangent, Gaussian, and other functions were among them. All three of the proposed improved models used the sigmoid function since it produced the minimum MMRE value and consistently produced the best outcomes.

To address the diverse nature of input values, different coding methods were employed, such as fuzzification, logarithmic, combined methods, among others. Among these, the fuzzification method consistently delivered the best results across all three proposed models

7.6. POTENTIAL MODELS LIMITATIONS

We will examine potential risks to the suggested models' validity for each of the three approaches in more detail in this chapter. We intend to identify potential problems that may affect and advance future research in the area of software project estimation by addressing these risks. The main elements of validity are: Validity of the construction, external validity, internal validity, and conclusion validity.

Internal validity

The selection of methodologies employed across all three proposed models serves as a critical factor contributing to the internal validity of software effort estimation. Specifically, the methods introduced, including clustering, fuzzification, and the allocation of weight coefficients to distinct ANN architectures, represent potential sources of internal validity. These potential threats have been effectively mitigated through the implementation of hyperparameter optimization techniques employing the Taguchi method, which relies on orthogonal vector plans.

A fresh and useful strategy for optimizing software products is the Taguchi method, which is well-known for its effectiveness in resilient design. The weight coefficients are combined with various input values that are specific to the needs of the three models being used in this methodology. The hidden and output layers use the sigmoid activation function to calculate the estimated value.

To achieve an MRE value of less than 1%, a spectrum of ANN architectures, ranging from the simplest to the most complex, was systematically employed. The utilization of ANNs constructed based on Taguchi's orthogonal vector plans serves as a confirmation of the reliability, stability, and efficiency of the proposed models.

With the advent of the Orthogonal Array Tuning Method (OATM), fewer tests are now necessary to get the minimal MMRE value. The three proposed models use either two levels (L1 and L2) or three levels (L1, L2, and L3) depending on the chosen orthogonal vector plan. Hyperparameters are represented as distinct levels within the OATM framework. By using this method, an ideal parameter configuration is produced that serves as the basis for each experiment, increasing the model's plausibility.

External validity

The results obtained from all three proposed models hold the potential for broader applications beyond the domain of software project estimations. These findings, derived from a comprehensive experiment comprising training, testing, and validation phases, were based on the utilization of publicly available datasets containing real values associated with various projects. These datasets encompassed diverse sources, including COCOMO2000, COCOMO81, NASA60, NASA93, Desharnais, Kemerer, ISBSG, UCP Benchmark (Mendeley), and others, including industrial datasets composed of real projects.

The consistent efficiency of the proposed models was observed across different ANN architectures within the three distinct approaches. The method employed, which involves iteratively halving the interval and adjusting weight coefficients, facilitates rapid convergence towards minimal cost and effort values. This approach is straightforward and leads to the attainment of a minimum MMRE value while requiring fewer than 10 iterations. By reducing the search interval by three orders of magnitude over less than 10 steps, this technique showcases its simplicity and effectiveness. The robustness of the obtained results suggests their potential applicability to new datasets and domains in the future.

Construction validity

A range of ANN architectures based on Taguchi's orthogonal vector plans were used throughout the three proposed enhanced models. The input values and weight coefficients related to the orthogonal vector plan were accommodated in these designs, which were made specifically for the given model. Chapter 3 presents and elaborates on the comprehensive step-by-step algorithm for each method.

One of the primary objectives underlying the model development was to select the simplest architecture, ensuring fast convergence towards minimal MMRE values. This pursuit of simplicity is driven by the desire to achieve stability and reliable estimation accuracy surpassing other software development effort estimation models. A significant element introduced to regulate the iterative process is the GA criterion, which serves as an indicator of when to continue iterations as the ANN architecture converges towards the final error value.

It is worth noting that in all experiments, across all ANN architectures, the "stop criterion" was consistently met after a maximum of nine iterations. Remarkably, the best-performing architecture within the UCP model achieved convergence in a mere four iterations, showcasing the efficiency and effectiveness of the approach.

Conclusion validity

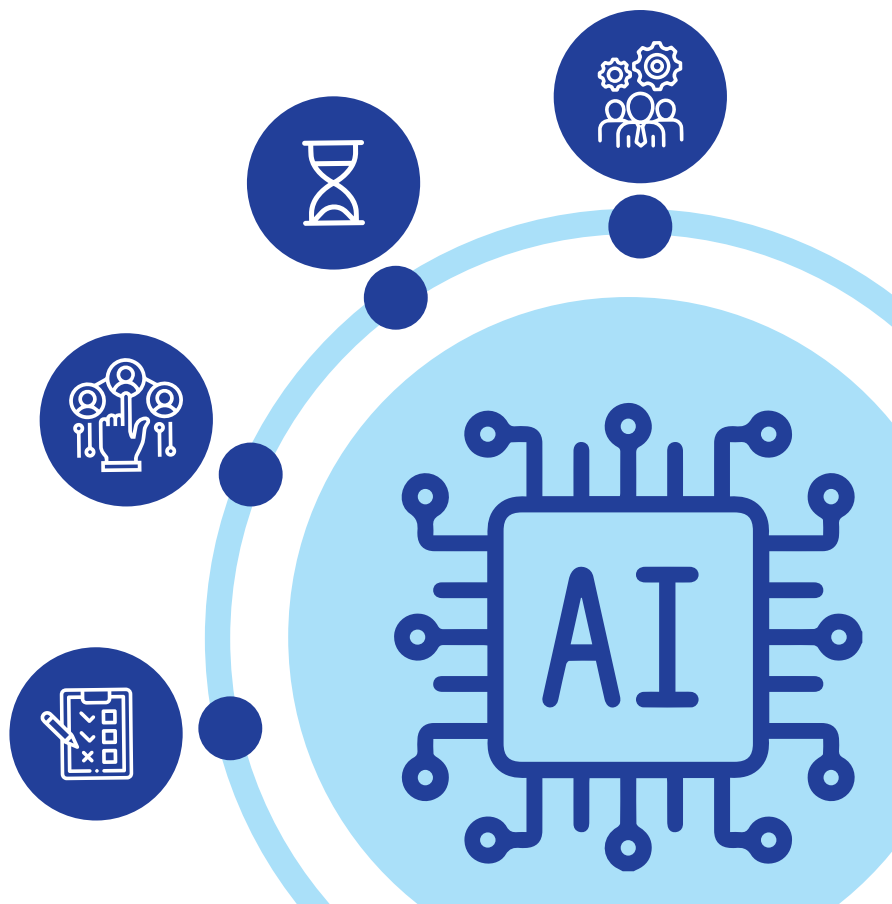
The studies provided in this book have undergone extensive verification and validation utilizing various data sets, ANN structures, and a minimum number of iterations.

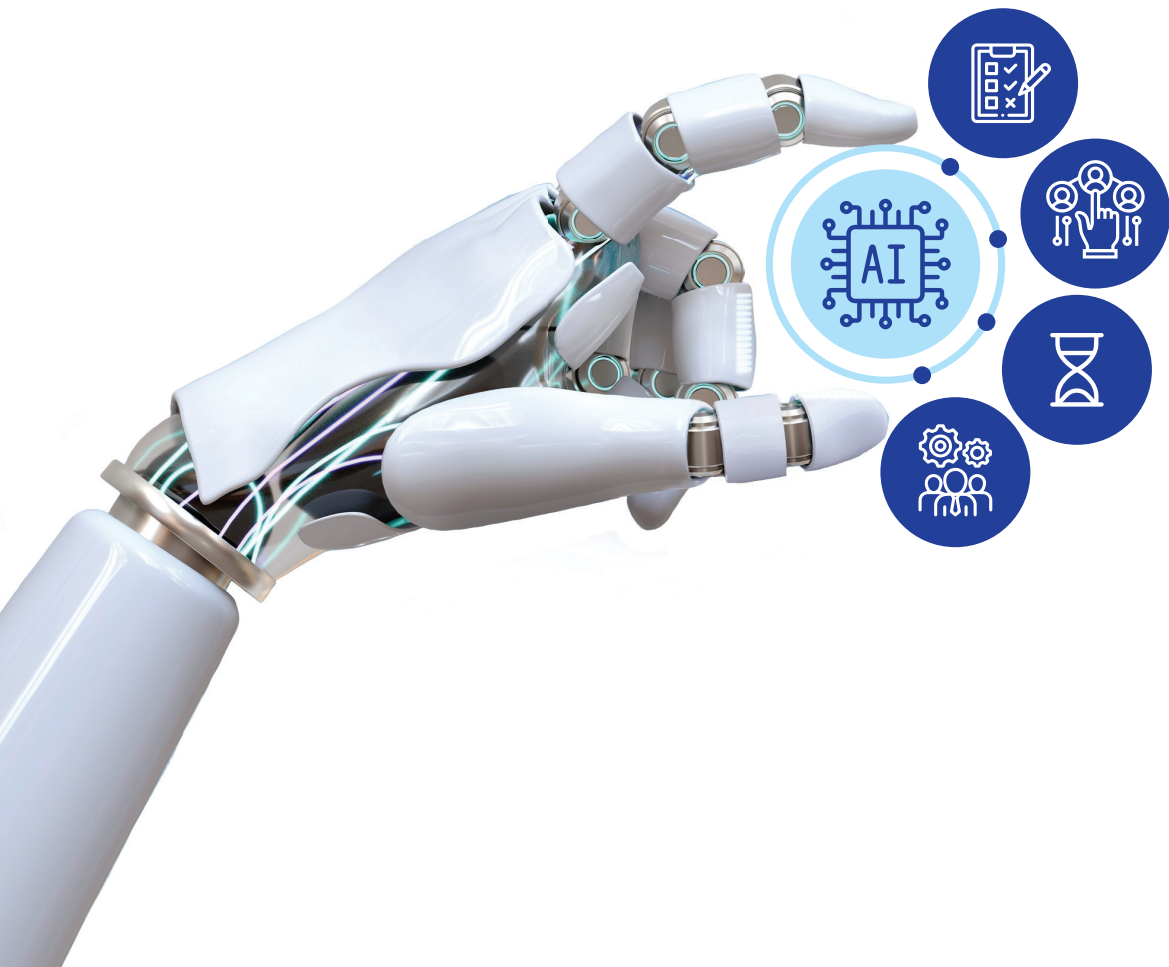


CHAPTER 8



MODELS CONTRIBUTION: SCIENCE VS. INDUSTRY





8. MODELS CONTRIBUTION: SCIENCE VS. INDUSTRY

The research conducted in this book has demonstrated the potential for creating improved models for estimating effort and costs in software development, yielding superior results compared to existing methods. The scientific approaches employed herein have the capacity to advance the field of software engineering. It's important to note that the newly applied methodology does not preclude the use of subjective estimation of effort and costs but rather complements it. Instead, it assists project teams by identifying potential discrepancies among various methods, enabling teams to conduct additional analyses for more accurate and timely results. Without such a methodology, project teams would bear the burden of subjectivity without a safety net.

The findings presented in this book offer valuable insights for project teams, software engineers, and test engineers. They empower these teams to formulate both short-term and long-term plans with heightened confidence in estimation reliability. Consequently, teams can efficiently navigate all phases of analysis and design, ensuring on-time project completion with a high level of certainty. Moreover, these results enable teams to proactively anticipate effort and costs for each project task, facilitating quality and optimal project realization.

This book introduces three distinct models for estimating effort and costs in software development, all of which exhibit improved accuracy compared to existing methods (as detailed in Chapter 3):

- novel, hybrid COCOMO2000 with ANN,
- novel, hybrid COSMIC FFP with ANN, and
- novel, hybrid UCP with ANN.

The selection of one of the three suggested ways enables precise, effective, and trustworthy assessment of the accomplishment of planned tasks, depending on a client's historical data. The use of scientific techniques and the suggested models has the potential to advance software engineering to a more developed state. It's important to note that the likelihood of software projects being completed successfully is not at its ideal level at the moment. Therefore, the use of trustworthy scientific techniques, particularly in the field of software work estimation, can yield useful insights.

The case study in this book has shown that new and improved models for estimating effort and costs in software project implementation can produce better results than previously used methods and models. These models were constructed utilizing three different methodologies. Additionally, these proposed models may be used to produce excellent software effort assessment.

All three of the proposed novel, hybrid models are grounded in artificial neural networks, enabling their application to a wide range of problems beyond the scope of software engineering. These models can be utilized in diverse fields and challenges that may be beyond the reach of classical computer technologies. They may be used in fields including text recognition, medical diagnosis, weather forecasting, nuclear research, robotics, automatic control, telecommunications, financial services, and banking. They may also be used in areas like signal processing, image and speech recognition, natural language processing, and robotics.

Organizations or individuals can select from the three proposed models based on their data and specific requirements. Additionally, all three approaches can be simultaneously employed to obtain more stable, reliable, and accurate estimates of effort and costs.

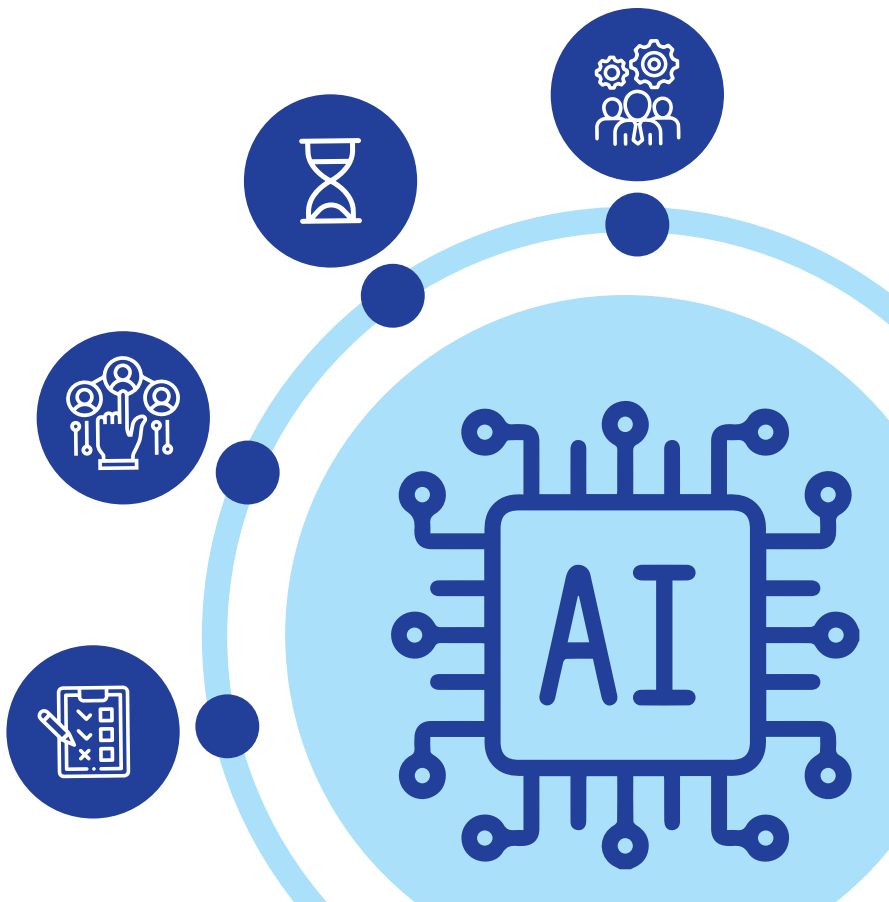
It is anticipated that the proposed artificial intelligence models will find numerous new applications in the future. Future research may focus on developing models to address cybersecurity challenges.

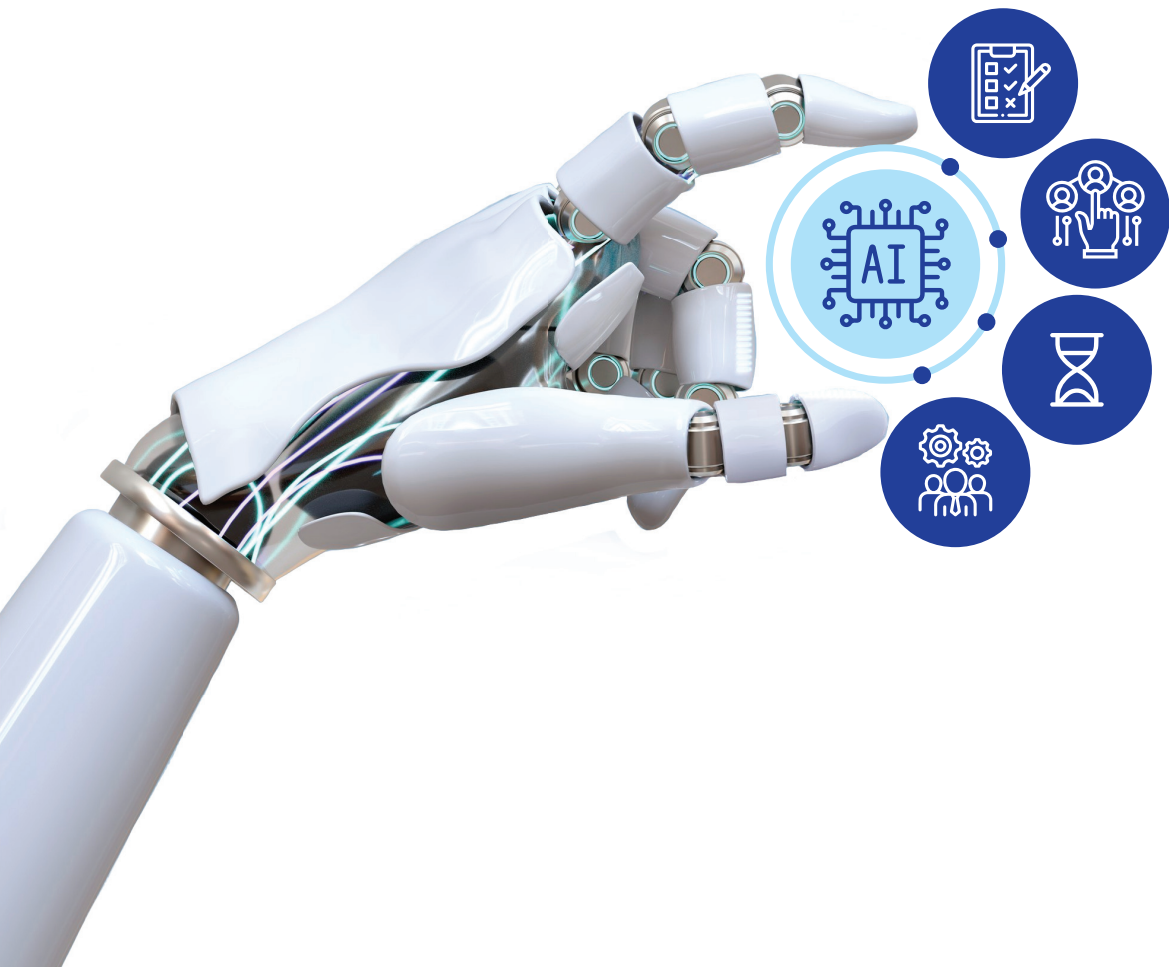


CHAPTER 9



CONCLUDING REMARK





9. CONCLUDING REMARK

To the best of our knowledge, the three novel, hybrid models presented in the proposed approaches have consistently achieved significantly better results across all phases of the experiments compared to previous research and studies.

A combination of 4 different ANN architectures, 5 unique datasets grouped into three clusters, the sigmoid activation function, a fuzzification approach, and the Taguchi method for calculating effort and cost were used in the first proposed, innovative, hybrid COCOMO2000 model. As shown by tracking the values of MMRE, the findings from all trials show that this strategy, when built in this way, provides the dependability and stability of the outcomes. It can be inferred that the cost-effect function and the characteristics of the projects in the various datasets used affect the convergence rate of each suggested architecture. Generally speaking, more sophisticated ANN designs have faster rates of convergence, which means it takes less time to complete the necessary iterations to reach the minimum MMRE, and vice versa. The division of each dataset into clusters and the application of the fuzzification method help mitigate the heterogeneous nature of the projects, rendering the proposed model sufficiently stable, adaptable, and dependable.

In conclusion, the ANN-L36 architecture consistently delivered the best results, specifically yielding the lowest MMRE values within the COCOMO2000 approach across all phases of the experiment.

The main advantages of this model are:

- the number of iterations is in the range of five to ten - which means shorter effort estimation time thanks to the convergence speed and simple architectures of each proposed ANN,
- use of simple ANN architectures,
- high actual effort coverage on used datasets, and
- minimal MMRE.

While these advantages are notable, one potential limitation is the need for further research to discover new methods that could lead to even lower MMRE values. However, there are no specific limitations that prevent the application of this approach, thanks to the experimentation with various ANN architectures and the model's ability to provide reliable estimates across a wide range of project values.

The use of two separate ANN architectures based on Taguchi Orthogonal Arrays resulted in notable improvements for the second suggested novel, hybrid COSMIC FFP model reported in this paper. This model uses four input parameters to predict software effort based on user functional needs. The model made use of the ISBSG dataset, a clustering approach for input values, and a fuzzification method to improve performance, which helped control and reduce fluctuations in project structures.

The main advantages of the proposed approach are:

- the number of iterations is in the range of five to six - which significantly reduces the estimation time,
- use of simple ANN architectures,
- optimization of proposed ANN architectures using Taguchi Orthogonal Arrays,
- high coverage of different values of functional size of software projects,
- ISBSG repository of data on real projects collected from different companies.

Overall, these findings underscore the effectiveness and improved performance of the enhanced COSMIC FFP model, offering better accuracy and reliability in software effort estimation.

The third novel, hybrid UCP model uses four different datasets, two different ANN architectures, a sigmoid activation function, a fuzzification method, and the Taguchi method to estimate the cost and labor of software development. The monitoring of the MMRE value and convergence rate for each of the architectures has shown the model to have made significant improvements over the prior two models.

The main advantages of this model are:

- the number of iterations is in the interval from four to six - which means reduced
- effort estimation time thanks to the exceptional convergence rate of both architectures,
- two simple proposed ANN architectures,
- high coverage of different real effort values and the lowest MMRE value of 7.5%.

Finding new ways to further reduce the MMRE score presents one potential hurdle. However, there are no innate restrictions that would prevent the use of this strategy. Depending on a company's historical data and particular demands for software implementation projects, this model offers flexibility and can be used separately or in conjunction with the previous two models. Although it may not be as standardized as the earlier models, it is becoming more and more popular among software firms, software engineers, and project teams looking for efficient ways to gauge the effort necessary for a software project's successful execution.

9.1. WHO WILL BENEFIT THE MOST FROM THESE MODELS?

The innovative hybrid models presented here can provide inspiration for the creation of tools aimed at precise, efficient, and dependable estimation of effort and costs throughout different phases of software project development. Moreover, these models are valuable resources for software companies, software engineers, and technical project managers, enabling them to swiftly and accurately determine all the necessary requirements for a dependable and consistent project estimation. This, in turn, can help mitigate the prevalent challenges faced by professionals and teams within the realm of software engineering.

Additionally, it can be concluded that artificial intelligence in the future will be most developed in the following way in these areas:

Blockchain is a machine learning technology with applications in the use, use, and transactions of cryptocurrencies, such as Bitcoin. This technology uses a decentralized database as a decentralized ledger in which all transactions are recorded without intermediaries. Blockchain transactions cannot be changed or deleted after updating the database. In addition, they provide the participants with all the required transparency. Techniques in machine learning are expected to be applied in the future to increase transparency and enforcement. Today, banks use machine learning algorithms and blockchain to determine the creditworthiness of potential borrowers for obtaining loans.

Self-service tools are already using technical solutions based on artificial intelligence to simplify or improve their operations. Searching for a specific location in the browser offers us the location, and different routes to the destination, saving time and money. It is expected that the application of machine learning will be able to contribute even more and offer new automated experiences for

customers who are looking for services and products. Voice assistants perform various tasks such as booking plane tickets, paying bills, and even messaging colleagues. Over time, it is expected that using NLP techniques of artificial intelligence; these check bots will provide even more personalized experiences, such as example, giving legal advice, helping in making business decisions, providing medical treatments, and the like. Today, there are already options with specific search engines. By searching for a particular product or service in the following search, you get specific results requested in the previous search.

Wearable devices and sensors as medical devices today make up a large number of medical devices that are already used in everyday life. They can measure heart rate, glucose levels, and salt levels. In the near future, with the application of various artificial intelligence tools and techniques, new devices will have a much greater application in healthcare in general and medical sciences. For example, wearable devices will be able to analyze personalized health data in real-time and, based on this, provide an appropriate diagnosis, then suggest a specific treatment tailored to the needs of each individual. Soon, wearable sensors will also be able to suggest a series of health tests based on health data and schedule an appointment with a specialist doctor nearby.

Self-driving cars and vehicles can drive in complex urban environments without human intervention and have already been tested on the streets. In the future, their wider use is expected and will not be limited only to passenger vehicles but will be extended to other means of transport, such as buses for the transport of passengers, trucks for the transport of goods, and other means of transport without human participation.

Deep learning is applied through applications for recognizing speech, writing, images, autonomous movement, and the like. Its increasing application in medical diagnostics is expected to develop algorithms to understand the code and write programs based on the input data.

To conclude...

The first time we introduced the term "*artificial intelligence*" was for over one century, and since then, we have constantly been talking about its application. Today, without even noticing, we are using it in our everyday life activities, and it will expand even more than the human brain could imagine.

Bibliography

- [1] Zhang, C., & Lu, Y. (2021). Study on artificial intelligence: The state of the art and future prospects. *Journal of Industrial Information Integration*, 23, 100224.
- [2] Haenlein, M., & Kaplan, A. (2019). A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California management review*, 61(4), 5-14.
- [3] Collins, G. S., & Moons, K. G. (2019). Reporting of artificial intelligence prediction models. *The Lancet*, 393(10181), 1577-1579.
- [4] Kh Wisamova, Z. I., Begishev, I. R., & Gaifutdinov, R. R. (2019). On methods to legal regulation of artificial intelligence in the world. *International Journal of Innovative Technology and Exploring Engineering*, 9(1), 5159-5162.
- [5] Zimmermann, A., Schmidt, R., Jügel, D., & Möhring, M. (2020, September). Evolution of enterprise architecture for intelligent digital systems. In *International Conference on Research Challenges in Information Science* (pp. 145-153). Springer, Cham.
- [6] Ilin, I., Levina, A., Borremans, A., & Kalyazina, S. (2019, December). Enterprise architecture modeling in digital transformation era. In *Energy management of municipal transportation facilities and transport* (pp. 124-142). Springer, Cham.
- [7] Stecher, P., Pohl, M., & Turowski, K. (2020). Enterprise architecture's effects on organizations' ability to adopt artificial intelligence-A Resource-based perspective.
- [8] Hannemann, I., Rodrigues, S., Loures, E., Deschamps, F., & Cestari, J. (2022). Applying a decision model based on multiple criteria decision making methods to evaluate the influence of digital transformation technologies on enterprise architecture principles. *IET Collaborative Intelligent Manufacturing*.
- [9] Tyagi, A. K., & Chahal, P. (2022). Artificial intelligence and machine learning algorithms. In *Research Anthology on Machine Learning Techniques, Methods, and Applications* (pp. 421-446). IGI Global.
- [10] Lewowski, T., & Madeyski, L. (2022). Code smells detection using artificial intelligence techniques: A business-driven systematic review. *Developments in Information & Knowledge Management for Business Applications*, 285-319.
- [11] Sestino, A., & De Mauro, A. (2022). Leveraging artificial intelligence in business: Implications, applications and methods. *Technology Analysis & Strategic Management*, 34(1), 16-29.
- [12] Sarker, I. H. (2022). Ai-based modeling: Techniques, applications and research issues towards automation, intelligent and smart systems. *SN Computer Science*, 3(2), 1-20.

- [13] Ramkumar, G., Othman, B., Malviya, B., Mohamma, A. J., Narayana, M. S., & Verma, D. (2022, April). A Conceptual Analysis on the Impact of Machine Learning Towards on Digital Marketing Transformation. In 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE) (pp. 2274-2278). IEEE.
- [14] Zheng, M. (2022). Advanced Artificial Intelligence Model for Financial Accounting Transformation Based on Machine Learning and Enterprise Unstructured Text Data. *Mobile Information Systems*, 2022.
- [15] Malik, H., Chaudhary, G., & Srivastava, S. (2022). Digital transformation through advances in artificial intelligence and machine learning. *Journal of Intelligent & Fuzzy Systems*, 42(2), 615-622.
- [16] Khoa, B. T., Anh, H. N., Ly, N. M., & Truong, N. X. (2022). A Study on Buying Attitude on Facebook in the Digital Transformation Era: A Machine Learning Application. In *Data Engineering for Smart Systems* (pp. 497-510). Springer, Singapore.
- [17] Tu, W., & He, J. (2022). Can Digital Transformation Facilitate Firms' M&A: Empirical Discovery Based on Machine Learning. *Emerging Markets Finance and Trade*, 1-16.
- [18] Jia, D., & Wu, Z. (2022). Application of Machine Learning in Enterprise Risk Management. *Security and Communication Networks*, 2022.
- [19] Romele, A. (2022). Images of Artificial Intelligence: a Blind Spot in AI Ethics. *Philosophy & Technology*, 35(1), 1-19.
- [20] Siddig, O. M., Al-Afnan, S. F., Elkatatny, S. M., & Abdulraheem, A. (2022). Drilling data-based approach to build a continuous static elastic moduli profile utilizing artificial intelligence techniques. *Journal of Energy Resources Technology*, 144(2).
- [21] Aggarwal, S., & Chugh, N. (2022). Review of machine learning techniques for EEG based brain computer interface. *Archives of Computational Methods in Engineering*, 1-20.
- [22] Abualigah, L., Zitar, R. A., Almotairi, K. H., Hussein, A. M., Abd Elaziz, M., Nikoo, M. R., & Gandomi, A. H. (2022). Wind, solar, and photovoltaic renewable energy systems with and without energy storage optimization: a survey of advanced machine learning and deep learning techniques. *Energies*, 15(2), 578.
- [23] Palimkar, P., Shaw, R. N., & Ghosh, A. (2022). Machine learning technique to prognosis diabetes disease: random forest classifier approach. In *Advanced Computing and Intelligent Technologies* (pp. 219-244). Springer, Singapore.
- [24] Singh, B., Vijayvargiya, A., & Kumar, R. (2022). Kinematic modeling for biped robot gait trajectory using machine learning techniques. *Journal of Bionic Engineering*, 19(2), 355-369.
- [25] Mukherjee, D., Chakraborty, S., & Ghosh, S. (2022). Power system state forecasting using machine learning techniques. *Electrical Engineering*, 104(1), 283-305.

- [26] Liao, L. D., Ferrara, A., Greenberg, M. B., Ngo, A. L., Feng, J., Zhang, Z., ... & Zhu, Y. (2022). Development and validation of prediction models for gestational diabetes treatment modality using supervised machine learning: a population-based cohort study. *BMC medicine*, 20(1), 1-13.
- [27] Pargent, F., Pfisterer, F., Thomas, J., & Bischl, B. (2022). Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics*, 1-22.
- [28] Alsahli, A., & Alsulmi, M. (2022). Automatic Detection of Sand Fouling Levels in Railway Tracks Using Supervised Machine Learning: A Case Study from Saudi Arabian Railway. *Arabian Journal for Science and Engineering*, 1-11.
- [29] Ouadah, A., Zemmouchi-Ghomari, L., & Salhi, N. (2022). Selecting an appropriate supervised machine learning algorithm for predictive maintenance. *The International Journal of Advanced Manufacturing Technology*, 119(7), 4277-4301.
- [30] Koltcov, S. (2022). Application of Duality Properties of Renyi Entropy for Parameter Tuning in an Unsupervised Machine Learning Task. In *International Conference on Applied Informatics* (pp. 189-203). Springer, Cham.
- [31] Verkerken, M., D'hooge, L., Wauters, T., Volckaert, B., & De Turck, F. (2022). Towards model generalization for intrusion detection: Unsupervised machine learning techniques. *Journal of Network and Systems Management*, 30(1), 1-25.
- [32] Usmani, U. A., Happonen, A., & Watada, J. (2022). A Review of Unsupervised Machine Learning Frameworks for Anomaly Detection in Industrial Applications. In *Science and Information Conference* (pp. 158-189). Springer, Cham.
- [33] Tarbani, N., & Wadhwa, K. (2022). Aggregation of Semantically Similar News Articles with the Help of Embedding Techniques and Unsupervised Machine Learning Algorithms: A Machine Learning Application with Semantic Technologies. In *Integrating Meta-Heuristics and Machine Learning for Real-World Optimization Problems* (pp. 107-120). Springer, Cham.
- [34] Bridgelall, R. (2022). Applying unsupervised machine learning to counterterrorism. *Journal of Computational Social Science*, 1-30.
- [35] Mandapati, S., Kadry, S., Kumar, R. L., Sutham, K., & Thinnukool, O. (2022). Deep learning model construction for a semi-supervised classification with feature learning. *Complex & Intelligent Systems*, 1-11.
- [36] Tu, H., & Menzies, T. (2022). DebtFree: minimizing labeling cost in self-admitted technical debt identification using semi-supervised learning. *Empirical Software Engineering*, 27(4), 1-37.
- [37] Malik, S., Tyagi, A. K., & Mahajan, S. (2022). Architecture, Generative Model, and Deep Reinforcement Learning for IoT Applications: Deep Learning Perspective. In *Artificial Intelligence-based Internet of Things Systems* (pp. 243-265). Springer, Cham.

- [38] Oladipo, I. D., AbdulRaheem, M., Awotunde, J. B., Bhoi, A. K., Adeniyi, E. A., & Abiodun, M. K. (2022). Machine Learning and Deep Learning Algorithms for Smart Cities: A Start-of-the-Art Review. *IoT and IoE Driven Smart Cities*, 143-162.
- [39] Tanveer, J., Haider, A., Ali, R., & Kim, A. (2022). An overview of reinforcement learning algorithms for handover management in 5G ultra-dense small cell networks. *Applied Sciences*, 12(1), 426.
- [40] Ahsan, M., Nygard, K. E., Gomes, R., Chowdhury, M. M., Rifat, N., & Connolly, J. F. (2022). Cybersecurity Threats and Their Mitigation Approaches Using Machine Learning—A Review. *Journal of Cybersecurity and Privacy*, 2(3), 527-555.
- [41] Singh, S., Singh, N., & Singh, V. (2022). Comparative Analysis of Open Standards for Machine Learning Model Deployments. In *ICT Systems and Sustainability* (pp. 499-507). Springer, Singapore.
- [42] Maulud, D. H., Zeebaree, S. R., Jacksi, K., Sadeeq, M. A. M., & Sharif, K. H. (2021). State of art for semantic analysis of natural language processing. *Qubahan Academic Journal*, 1(2), 21-28.
- [43] Meera, S., & Geerthik, S. (2022). Natural language processing. *Artificial Intelligent Techniques for Wireless Communication and Networking*, 139-153.
- [43] Lauriola, I., Lavelli, A., & Aioli, F. (2022). An introduction to deep learning in natural language processing: Models, techniques, and tools. *Neurocomputing*, 470, 443-456.
- [45] Zini, J. E., & Awad, M. (2022). On the Explainability of Natural Language Processing Deep Models. *ACM Computing Surveys (CSUR)*.
- [46] Nagarhalli, T. P., Vaze, V., & Rana, N. K. (2021, February). Impact of machine learning in natural language processing: A review. In *2021 third international conference on intelligent communication technologies and virtual mobile networks (ICICV)* (pp. 1529-1534). IEEE.
- [47] Bitterman, D. S., Miller, T. A., Mak, R. H., & Savova, G. K. (2021). Clinical natural language processing for radiation oncology: a review and practical primer. *International Journal of Radiation Oncology* Biology* Physics*, 110(3), 641-655.
- [48] Karhade, A. V., Bongers, M. E., Groot, O. Q., Cha, T. D., Doorly, T. P., Fogel, H. A., ... & Schwab, J. H. (2021). Development of machine learning and natural language processing algorithms for preoperative prediction and automated identification of intraoperative vascular injury in anterior lumbar spine surgery. *The Spine Journal*, 21(10), 1635-1642.
- [49] Li, H., & Li, Z. (2022). Text Classification Based on Machine Learning and Natural Language Processing Algorithms. *Wireless Communications and Mobile Computing*, 2022.
- [50] Murthy, M. Y. B., Mastanbi, S., Sujitha, B., & Babu, K. R. (2023). Evaluating Deep Learning Algorithms for Natural Language Processing. In *Proceedings of International Conference on Communication and Computational Technologies* (pp. 709-720). Springer, Singapore.

- [51] Allen, K. S., Hood, D. R., Cummins, J., Kasturi, S., Mendonca, E. A., & Vest, J. R. (2023). Natural language processing-driven state machines to extract social factors from unstructured clinical documentation. *JAMIA open*, 6(2), ooad024.
- [52] Sharma, H. (2021, October). Improving Natural Language Processing tasks by Using Machine Learning Techniques. In 2021 5th International Conference on Information Systems and Computer Networks (ISCON) (pp. 1-5). IEEE.
- [53] Locke, S., Bashall, A., Al-Adely, S., Moore, J., Wilson, A., & Kitchen, G. B. (2021). Natural language processing in medicine: a review. *Trends in Anaesthesia and Critical Care*, 38, 4-9.
- [54] Wu, C., Li, X., Guo, Y., Wang, J., Ren, Z., Wang, M., & Yang, Z. (2022). Natural language processing for smart construction: Current status and future directions. *Automation in Construction*, 134, 104059.
- [55] Xu, X., & Cai, H. (2021). Ontology and rule-based natural language processing approach for interpreting textual regulations on underground utility infrastructure. *Advanced Engineering Informatics*, 48, 101288.
- [56] Patel, R., & Patel, S. (2021). Deep Learning for Natural Language Processing. In *Information and Communication Technology for Competitive Strategies (ICTCS 2020)* (pp. 523-533). Springer, Singapore.
- [57] Demner-Fushman, D., Elhadad, N., & Friedman, C. (2021). Natural language processing for health-related texts. In *Biomedical Informatics* (pp. 241-272). Springer, Cham.
- [58] Dhar, T., Dey, N., Borra, S., & Sherratt, R. S. (2023). Challenges of Deep Learning in Medical Image Analysis—Improving Explainability and Trust. *IEEE Transactions on Technology and Society*, 4(1), 68-75.
- [59] Cunliffe, D., Vlachidis, A., Williams, D., & Tudhope, D. (2022). Natural language processing for under-resourced languages: Developing a Welsh natural language toolkit. *Computer Speech & Language*, 72, 101311.
- [60] Samarawickrama, C., de Almeida, M., Perera, A. S., de Silva, N., & Ratnayaka, G. (2021). Identifying legal party members from legal opinion texts using natural language processing. *EasyChair, Tech. Rep.*
- [61] Ribeiro, J., Lima, R., Eckhardt, T., & Paiva, S. (2021). Robotic process automation and artificial intelligence in industry 4.0—a literature review. *Procedia Computer Science*, 181, 51-58.
- [62] Rajawat, A. S., Rawat, R., Barhanpurkar, K., Shaw, R. N., & Ghosh, A. (2021). Robotic process automation with increasing productivity and improving product quality using artificial intelligence and machine learning. In *Artificial Intelligence for Future Generation Robotics* (pp. 1-13). Elsevier.

- [63] Jha, N., Prashar, D., & Nagpal, A. (2021). Combining artificial intelligence with robotic process automation—an intelligent automation approach. In *Deep Learning and Big Data for Intelligent Transportation* (pp. 245-264). Springer, Cham.
- [64] Das, S., Das, I., Shaw, R. N., & Ghosh, A. (2021). Advance machine learning and artificial intelligence applications in service robot. In *Artificial Intelligence for Future Generation Robotics* (pp. 83-91). Elsevier.
- [65] Frolov, D., Radziewicz, W., Saienko, V., Kuchuk, N., Mozhaiev, M., Gnusov, Y., & Onishchenko, Y. (2021). Theoretical and technological aspects of intelligent systems: problems of artificial intelligence. *International Journal of Computer Science & Network Security*, 21(5), 35-38.
- [66] Marge, M., Espy-Wilson, C., Ward, N. G., Alwan, A., Artzi, Y., Bansal, M., ... & Yu, Z. (2022). Spoken language interaction with robots: Recommendations for future research. *Computer Speech & Language*, 71, 101255.
- [67] Shah, D., Yang, B., Kriegman, S., Levin, M., Bongard, J., & Kramer-Bottiglio, R. (2021). Shape changing robots: bioinspiration, simulation, and physical realization. *Advanced Materials*, 33(19), 2002882.
- [68] Cox, A. (2021). The impact of AI, machine learning, automation and robotics on the information professions: A report for CILIP.
- [69] Kumar, K., Shah, R., Kumar, N., & Singh, R. P. (2021). Application of Robotic Process Automation. In *Applications of Artificial Intelligence in Engineering* (pp. 929-937). Springer, Singapore.
- [70] Arents, J., & Greitans, M. (2022). Smart industrial robot control trends, challenges and opportunities within manufacturing. *Applied Sciences*, 12(2), 937.
- [71] Wang, E. Z., Lee, C. C., & Li, Y. (2022). Assessing the impact of industrial robots on manufacturing energy intensity in 38 countries. *Energy Economics*, 105, 105748.
- [72] Raviola, A., De Martin, A., Guida, R., Jacazio, G., Mauro, S., & Sorli, M. (2021, June). Harmonic drive gear failures in industrial robots applications: An overview. In *PHM Society European Conference* (Vol. 6, No. 1, pp. 11-11).
- [73] Mesmer, P., Neubauer, M., Lechler, A., & Verl, A. (2022). Robust design of independent joint control of industrial robots with secondary encoders. *Robotics and Computer-Integrated Manufacturing*, 73, 102232.
- [74] Bogue, R. (2022). The role of robots in entertainment. *Industrial Robot: the international journal of robotics research and application*.
- [75] Javaid, M., Haleem, A., Singh, R. P., & Suman, R. (2021). Substantial capabilities of robotics in enhancing industry 4.0 implementation. *Cognitive Robotics*, 1, 58-75.
- [76] Doyle-Kent, M. (2021). Collaborative robotics in industry 5.0 (Doctoral dissertation, Wien).

- [77] Vrontis, D., Christofi, M., Pereira, V., Tarba, S., Makrides, A., & Trichina, E. (2022). Artificial intelligence, robotics, advanced technologies and human resource management: a systematic review. *The International Journal of Human Resource Management*, 33(6), 1237-1266.
- [78] Galaz, V., Centeno, M. A., Callahan, P. W., Causevic, A., Patterson, T., Brass, I., ... & Levy, K. (2021). Artificial intelligence, systemic risks, and sustainability. *Technology in Society*, 67, 101741.
- [79] Abioye, S. O., Oyedele, L. O., Akanbi, L., Ajayi, A., Delgado, J. M. D., Bilal, M., ... & Ahmed, A. (2021). Artificial intelligence in the construction industry: A review of present status, opportunities and future challenges. *Journal of Building Engineering*, 44, 103299.
- [80] Amritha Ashok, K., Savy, A., Shijoh, V., Shaw, R. N., & Ghosh, A. (2021). Hospital assistance robots control strategy and machine learning technology. In *Machine Learning for Robotics Applications* (pp. 35-46). Springer, Singapore.
- [81] YanRu, L. (2021). An artificial intelligence and machine vision based evaluation of physical education teaching. *Journal of Intelligent & Fuzzy Systems*, 40(2), 3559-3569.
- [82] Smith, M. L., Smith, L. N., & Hansen, M. F. (2021). The quiet revolution in machine vision-a state-of-the-art survey paper, including historical review, perspectives, and future directions. *Computers in Industry*, 130, 103472.
- [83] Ren, Z., Fang, F., Yan, N., & Wu, Y. (2021). State of the art in defect detection based on machine vision. *International Journal of Precision Engineering and Manufacturing-Green Technology*, 1-31.
- [84] Al-Shoqran, M., & Shorman, S. (2021). A review on smart universities and artificial intelligence. *The Fourth Industrial Revolution: Implementation of Artificial Intelligence for Growing Business Success*, 281-294.
- [85] Zhang, L., Pan, Y., Wu, X., & Skibniewski, M. J. (2021). Artificial intelligence in construction engineering and management (pp. 95-124). Springer.
- [86] Pimenov, D. Y., Bustillo, A., Wojciechowski, S., Sharma, V. S., Gupta, M. K., & Kuntoğlu, M. (2022). Artificial intelligence systems for tool condition monitoring in machining: Analysis and critical review. *Journal of Intelligent Manufacturing*, 1-43.
- [87] Soheli, S. J., Jahan, N., Hossain, M., Adhikary, A., Khan, A. R., & Wahiduzzaman, M. (2022). Smart Greenhouse Monitoring System Using Internet of Things and Artificial Intelligence. *Wireless Personal Communications*, 1-32.
- [88] Ulus, Ş., & Eski, İ. (2021). Neural network and fuzzy logic-based hybrid attitude controller designs of a fixed-wing UAV. *Neural Computing and Applications*, 33(14), 8821-8843.

- [89] Zangeneh, M., Aghajari, E., & Forouzanfar, M. (2022). Design and implementation of an intelligent multi-input multi-output Sugeno fuzzy logic controller for managing energy resources in a hybrid renewable energy power system based on Arduino boards. *Soft Computing*, 26(3), 1459-1473.
- [90] Meli, D., Sridharan, M., & Fiorini, P. (2021). Inductive learning of answer set programs for autonomous surgical task planning. *Machine Learning*, 110(7), 1739-1763.
- [91] Albahri, A. S., Alnoor, A., Zaidan, A. A., Albahri, O. S., Hameed, H., Zaidan, B. B., ... & Yass, A. A. (2022). Hybrid artificial neural network and structural equation modelling techniques: a survey. *Complex & Intelligent Systems*, 8(2), 1781-1801.
- [92] Malinda, M., & Chen, J. H. (2022). The forecasting of consumer exchange-traded funds (ETFs) via grey relational analysis (GRA) and artificial neural network (ANN). *Empirical Economics*, 62(2), 779-823.
- [93] Ghani, S., & Kumari, S. (2022). Liquefaction behavior of Indo-Gangetic region using novel metaheuristic optimization algorithms coupled with artificial neural network. *Natural Hazards*, 111(3), 2995-3029.
- [94] Narkhede, M. V., Bartakke, P. P., & Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 55(1), 291-322.
- [95] Bülbül, M. A., & Öztürk, C. (2022). Optimization, modeling and implementation of plant water consumption control using genetic algorithm and artificial neural network in a hybrid structure. *Arabian Journal for Science and Engineering*, 47(2), 2329-2343.
- [96] Zhou, J., Huang, S., Zhou, T., Armaghani, D. J., & Qiu, Y. (2022). Employing a genetic algorithm and grey wolf optimizer for optimizing RF models to evaluate soil liquefaction potential. *Artificial Intelligence Review*, 1-33.
- [97] Guido, R., Groccia, M. C., & Conforti, D. (2022). Hyper-Parameter Optimization in Support Vector Machine on Unbalanced Datasets Using Genetic Algorithms. In *Optimization in Artificial Intelligence and Data Sciences* (pp. 37-47). Springer, Cham.
- [98] Ince, M. (2022). Automatic and intelligent content visualization system based on deep learning
- [99] Fang, H., Zhou, J., Wang, Z., Qiu, Z., Sun, Y., Lin, Y., ... & Pan, M. (2022). Hybrid method integrating machine learning and particle swarm optimization for smart chemical process operations. *Frontiers of Chemical Science and Engineering*, 16(2), 274-287.
- [100] Bhardwaj, T., Mittal, R., Upadhyay, H., & Lagos, L. (2022). Applications of Swarm Intelligent and Deep Learning Algorithms for Image-Based Cancer Recognition. In *Artificial Intelligence in Healthcare* (pp. 133-150). Springer, Singapore.

- [101] Awotunde, J. B., Adeniyi, A. E., Ajagbe, S. A., Jimoh, R. G., & Bhoi, A. K. (2022). Swarm Intelligence and Evolutionary Algorithms in Processing Healthcare Data. In *Connected e-Health* (pp. 105-124). Springer, Cham.
- [102] Fu, S. (2022). Research and Development of Learning Early Warning System Based on Artificial Intelligence Technology. In *International Conference on Cognitive based Information Processing and Applications (CIPA 2021)* (pp. 884-889). Springer, Singapore.
- [103] Pan, R., Bagherzadeh, M., Ghaleb, T. A., & Briand, L. (2022). Test case selection and prioritization using machine learning: a systematic literature review. *Empirical Software Engineering*, 27(2), 1-43.
- [104] Moin, A., Challenger, M., Badii, A., & Günnemann, S. (2022). A model-driven approach to machine learning and software modeling for the IoT. *Software and Systems Modeling*, 21(3), 987-1014.
- [105] Berry, D. M. (2022, March). Requirements Engineering for Artificial Intelligence: What Is a Requirements Specification for an Artificial Intelligence?. In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (pp. 19-25). Springer, Cham.
- [106] Kumar, P., Obaid, A. J., Cengiz, K., Khanna, A., & Balas, V. E. (Eds.). (2022). *A Fusion of Artificial Intelligence and Internet of Things for Emerging Cyber Systems*. Springer.
- [107] Brunotte, W., Chazette, L., Klös, V., & Speith, T. (2022, March). Quo Vadis, Explainability?—A Research Roadmap for Explainability Engineering. In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (pp. 26-32). Springer, Cham.
- [108] Herbold, S., & Haar, T. (2022). Smoke testing for machine learning: simple tests to discover severe bugs. *Empirical Software Engineering*, 27(2), 1-30.
- [109] Ellul, J. (2022). Should we regulate Artificial Intelligence or some uses of software?. *Discover Artificial Intelligence*, 2(1), 1-6.
- [110] Bouskela, D., Falcone, A., Garro, A., Jardin, A., Otter, M., Thuy, N., & Tundis, A. (2022). Formal requirements modeling for cyber-physical systems engineering: An integrated solution based on FORM-L and Modelica. *Requirements Engineering*, 27(1), 1-30.
- [111] Goyal, S. (2022). Handling class-imbalance with KNN (neighbourhood) under-sampling for software defect prediction. *Artificial Intelligence Review*, 55(3), 2023-2064.
- [112] Gupta, A., Asad, A., Meena, L., & Anand, R. (2023). IoT and RFID-Based Smart Card System Integrated with Health Care, Electricity, QR and Banking Sectors. In *Artificial Intelligence on Medical Data* (pp. 253-265). Springer, Singapore.

- [113] Sree Harsha, S., Sohoni, A. C., & Chandrasekaran, K. (2022). NeuralDoc-Automating Code Translation Using Machine Learning. In *Artificial Intelligence Driven Circuits and Systems* (pp. 125-138). Springer, Singapore.
- [114] Schüle, M. E., Springer, M., Kemper, A., & Neumann, T. (2022). LLVM Code Optimisation for Automatic Differentiation. *DEEM@ SIGMOD*. ACM.
- [115] Burgueño, L., Cabot, J., Li, S., & Gérard, S. (2022). A generic LSTM neural network architecture to infer heterogeneous model transformations. *Software and Systems Modeling*, 21(1), 139-156.
- [116] Bogina, V., Hartman, A., Kuflik, T., & Shulner-Tal, A. (2022). Educating software and AI stakeholders about algorithmic fairness, accountability, transparency and ethics. *International Journal of Artificial Intelligence in Education*, 32(3), 808-833.
- [117] Verbruggen, M. (2022). No, Not That Verification: Challenges Posed by Testing, Evaluation, Validation and Verification of Artificial Intelligence in Weapon Systems. In *Armament, Arms Control and Artificial Intelligence* (pp. 175-191). Springer, Cham.
- [118] Misra, S., & Arumugam, C. (2022). Illumination of Artificial Intelligence in Cybersecurity and Forensics.
- [119] Zednik, C., & Boelsen, H. (2022). Scientific Exploration and Explainable Artificial Intelligence. *Minds and Machines*, 32(1), 219-239.
- [120] Zeng, X., & Long, L. (2022). Introduction to Artificial Intelligence. In *Beginning Deep Learning with TensorFlow* (pp. 1-45). Apress, Berkeley, CA.
- [121] Liao, C., Wang, A., Georgakoudis, G., de Supinski, B. R., Yan, Y., Beckingsale, D., & Gamblin, T. (2022). Extending OpenMP for Machine Learning-Driven Adaptation. In *International Workshop on Accelerator Programming Using Directives* (pp. 49-69). Springer, Cham.
- [122] Elghaish, F., Chauhan, J. K., Matarneh, S., Rahimian, F. P., & Hosseini, M. R. (2022). Artificial intelligence-based voice assistant for BIM data management. *Automation in Construction*, 140, 104320.
- [123] Khaliq, Z., Farooq, S. U., & Khan, D. A. (2022). Transformers for GUI Testing: A Plausible Solution to Automated Test Case Generation and Flaky Tests. *Computer*, 55(3), 64-73.
- [124] Tetaly, M., & Kulkarni, P. (2022, October). Artificial intelligence in cyber security—A threat or a solution. In *AIP Conference Proceedings* (Vol. 2519, No. 1, p. 030036). AIP Publishing LLC.
- [125] Faustino, J., Adriano, D., Amaro, R., Pereira, R., & da Silva, M. M. (2022). DevOps benefits: A systematic literature review. *Software: Practice and Experience*.
- [126] Jadhav, A., Kaur, M., & Akter, F. (2022). Evolution of software development effort and cost estimation techniques: five decades study using automated text mining approach. *Mathematical Problems in Engineering*, 2022.

- [127] Duch-Brown, N., Gomez-Herrera, E., Mueller-Langer, F., & Tolan, S. (2022). Market power and artificial intelligence work on online labour markets. *Research Policy*, 51(3), 104446.
- [128] Mizuta, T. (2022). Artificial intelligence (ai) for financial markets: a good ai for designing better financial markets and a bad ai for manipulating markets. In *Digital Designs for Money, Markets, and Social Dilemmas* (pp. 305-329). Springer, Singapore.
- [129] Feuerriegel, S., Shrestha, Y. R., von Krogh, G., & Zhang, C. (2022). Bringing artificial intelligence to business management. *Nature Machine Intelligence*, 4(7), 611-613.
- [130] A Guide to the Project Management Body of Knowledge (PMBOK Guide). Third Edition, Project Management Institute, Inc. 2004. ISBN: 1-930699-45-X.
- [131] B. W. Boehm, C. Abts, and S. Chulani. Software development cost estimation approaches-A survey. *Annals of software engineering*, 10 (1): 177-205, 2000.
- [132] https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/GENREF/ChaosManifest_2011.pdf
- [133] https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf
- [134] M. Ayat et al. Current trends analysis and prioritization of success factors: a systematic literature review of ICT projects. *International Journal of Managing Projects in Business*, 14 (3): 652-679, 2021.
- [135] <https://www.standishgroup.com/>
- [136] A. Stoica and J. Blosiu. Neural learning using Orthogonal arrays. *Advances in Intelligent Systems*, 41:418, 1997.
- [137] J. F. Khaw, B. Lim, and L. E.Lim. Optimal design of neural networks using the taguchi method. *Neurocomputing*, 7 (3): 225-245, 1995.
- [138] A. BaniMustafa. Predicting software effort estimation using machine learning techniques. In *8th International Conference on Computer Science and Information Technology (CSIT)*, pages 249-256, IEEE, 2018.
- [139] P. S. Kumar et al. Advancement from neural networks to deep learning in software effort estimation: Perspective of two decades. *Computer Science Review*, 28 (11):100288, 2020.
- [140] P. S. Kumar and H. Behera. Estimating software effort using neural network: An experimental investigation. *Computational Intelligence in Pattern Recognition, Proceedings of CIPR*, pages 165-180, Springer, 2020.
- [141] P. S. Kumar and H. Behera. Role of soft computing techniques in software effort estimation: an analytical study. *Computational Intelligence in Pattern Recognition. Advances in Intelligent Systems and Computing*, pages 807-831, Springer, 2020.

- [142] J. Popović. Enhancing methods for effort estimation in software projects. Doctoral dissertation, University of Belgrade, School of Electrical Engineering, Belgrade, Serbia, 2016.
- [143] S. J. Huang and N. H. Chiu. Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and software technology*, 48 (11): 1034-1045, 2006.
- [144] M. Shepperd, C. Schofield, and B. Kitchenham. Effort estimation using analogy. *In Proceedings of IEEE 18th International Conference on Software Engineering*, pages 170-178, IEEE, 1996.
- [145] D. S. Cruzes and T. Dybå. Research synthesis in software engineering: A tertiary study. *Information and Software Technology*, 53 (5): 440-455, 2011.
- [146] P. Pandey. Analysis of the techniques for software cost estimation. *In 3rd International Conference on Advanced Computing and Communication Technologies (ACCT)*, pages 16-19, IEEE, 2013.
- [147] J. J. Ryan et al. Quantifying information security risks using expert judgment elicitation. *Computers & Operations Research*, 39 (4): 774-784, 2012.
- [148] J. G. Borade and V. R. Khalkar. Software project effort and cost estimation techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(8): 730-739, 2013.
- [149] O. Fedotova, L. Teixeira, and H. Alvelos. Software Effort Estimation with Multiple Linear Regression: Review and Practical Application. *Journal of Information Science and Engineering*, 29(5): 925-945, 2013.
- [150] N. Condori-Fernandez and P. Lago. Characterizing the contribution of quality requirements to software sustainability. *Journal of Systems and Software*, 137: 289-305, 2018.
- [151] C. Pacheco et al. Reusing functional software requirements in small-sized software enterprises: a model oriented to the catalog of requirements. *Requirements Engineering*, 22 (2): 275-287, 2017.
- [152] T. Hovorushchenko and O. Pomorova. Methodology of evaluating the sufficiency of information on quality in the software requirements specifications. *In 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pages 370-374, IEEE, 2018.
- [153] A. Dar and N. Anuradha. Use of Orthogonal arrays and design of experiment via Taguchi L9 method in probability of default. *Accounting*, 4 (3): 113-122, 2018.
- [154] M. Alshibli et al. A robust robotic disassembly sequence design using Orthogonal arrays and task allocation. *Robotics*, 8 (1): 20, 2019.
- [155] G. Madhoo and M. Shilpa. Optimization of process parameters of stir casting technique using Orthogonal arrays. *International journal of advanced research methodology in engineering & technology*, 1 (2), 2017.

- [156] P. Singh, V. Sarin, and N. Midha. Mixture Designs Constructed Using Taguchi's Mixed Element Orthogonal Arrays. *Journal of Statistical Theory and Practice*, 14 (4): 1-12, 2020.
- [157] K. Suresh and R. Dillibabu. Designing a machine learning based software risk model using Naïve Bayes algorithm. *TAGA Journal*, 14: 3141-3147, 2018.
- [158] P. Pospieszny, B. Czarnacka-Chrobot and A. Kobylinski. An effective approach for software project effort and duration estimation with machine learning algorithms. *Journal of Systems and Software*, 137: 184-196, 2018.
- [159] M. Thrun, F. Pape, and A. Ultsch. Interactive machine learning tool for clustering in visual analytics. In *7th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 479-487, IEEE, 2020.
- [160] T. Tajti. Fuzzification of training data class membership binary values for neural network algorithms. In *Annales Mathematicae et Informaticae*, Eszterházy Károly Egyetem Líceum Kiadó, 52: 217-228, 2020.
- [161] T. D. Khang et al. Fuzzy C-Means Clustering Algorithm with Multiple Fuzzification Coefficients. *Algorithms*, 13 (7): 158, 2020.
- [162] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. arXiv preprint, arXiv:1710.05941, 2017.
- [163] C. Nwankpa et al. Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint, arXiv:1811.03378, 2018.
- [164] L. H. Chen and S. H. Nien. A new approach to formulate fuzzy regression models. *Applied Soft Computing*, 86 (01): 105915, 2020.
- [165] van Smeden M et al. Sample size for binary logistic prediction models: beyond events per variable criteria. *Statistical methods in medical research*, 28 (8): 2455-2474, 2019.
- [166] M. Nilashi et al. A recommender system for tourism industry using cluster ensemble and prediction machine learning techniques. *Computers & industrial engineering*, 109: 357-368, 2017.
- [167] S. Scalabrino et al. Automatically assessing code understandability: How far are we? In *32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 417-427, IEEE, 2017.
- [168] B. W. Boehm. Safe and simple software cost analysis. *IEEE software*, 17 (5): 14-17, 2000.
- [169] A. J. Albrecht and J. E. Gaffney. Software function, source lines of code, and development effort prediction: a software science validation. *IEEE transactions on software engineering*, (6): 639-648, 1983.
- [170] <https://www.ifpug.org/about-function-point-analysis/?lang=de>

- [171] C. R. Symons. Function point analysis: difficulties and improvements. *IEEE transactions on software engineering*, 14 (1): 2-11, 1988.
- [172] N. Board. Software Measurement in the Netherlands - The 25th Anniversary of Nesma. In *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, pages 125-126, IEEE, 2014.
- [173] <https://www.totalmetrics.com/function-point-resources/downloads/COSMIC-Versus-IFPUG-Similarities-and-Differences.pdf>
- [174] R. Meli et al. On the applicability of COSMIC-FFP for measuring software throughout its life cycle. In *Proceedings of the 11th European Software Control and Metrics Conference*, pages 18-20, Springer, 2000.
- [175] L. C. Briand, K. El Emam, and F. Bomarius. COBRA: a hybrid method for software cost estimation, benchmarking, and risk. In *Proceedings of the 20th international conference on Software engineering*, pages 390-399, IEEE, 1998.
- [176] S. Diev. Use cases modeling and software estimation: applying use case points. *ACM SIGSOFT Software Engineering Notes*, 31 (6): 1-4, 2006.
- [177] L. Lavazza and S. Morasca. Software effort estimation with a generalized robust linear regression technique. In *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, pages 206-215, IET, 2012.
- [178] V. Nguyen, B. Steece, and B. W. Boehm. A constrained regression technique for COCOMO calibration. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 213-222, ACM-IEEE, 2008.
- [179] S. Devnani-Chulani et al. Calibration Approach and Results of the COCOMO II Post-Architecture Model. In *Proceedings of the 20th Annual Conference of the International Society of Parametric Analysts (ISPA) and the 8th Annual Conference of the Society of Cost Estimating and Analysis (SCEA)*, 1998.
- [180] N. Mittas and L. Angelis. Ranking and clustering software cost estimation models through a multiple comparisons algorithm. *IEEE Transactions on software engineering*, 39 (4): 537-551, 2012.
- [181] E. Kocaguneli et al. Exploiting the essential assumptions of analogy-based effort estimation. *IEEE transactions on software engineering*, 38 (2): 425-438, 2011.
- [182] M. P. Fay and Y. Malinovsky. Confidence intervals of the Mann-Whitney parameter that are compatible with the Wilcoxon-Mann-Whitney test. *Statistics in medicine*, 37 (27): 3991-4006, 2018.
- [183] http://info.usherbrooke.ca/llavoie/enseignement/References/Cocomoll_Reference_Manual.pdf

- [184] L. H. Putnam and W. Myers. Measures for excellence: reliable software on time, within budget. *Prentice Hall Professional Technical Reference*, First Edition, 1991. ISBN-10: 0135676940
- [185] D. D. Galorath and M. W. Evans. Software sizing, estimation, and risk management: when performance is measured performance improves, *Auerbach Publications*, First Edition 2006. ISBN-10: 0849335930.
- [186] B. Boehm et al. Software engineering economics. *New York*, 197, 1981.
- [187] B. W. Boehm et al. Software cost estimation with COCOMO II. First Edition, *Prentice Hall Press*, 2009.
- [188] A. A. Fadhil, R. G. Alsarraj, and A. M. Altaie. Software cost estimation based on dolphin algorithm. *IEEE Access*, 8: 75279–75287, 2020.
- [189] G. Kumar and P. K. Bhatia. Automation of software cost estimation using neural network technique. *International Journal of Computer Applications*, 98 (20): 11-17, 2014.
- [190] S. Goyal and A. Parashar. Machine learning application to improve cocomo model using neural networks. *International Journal of Information Technology and Computer Science (IJITCS)* 3: 35–51, 2018.
- [191] M. Nandini, P. Bhargavi, and G. R. Sekhar. Face recognition using neural networks. *International Journal of Scientific and Research Publications*, 3 (3): 1-5, 2013.
- [192] S. Mukherjee and R. K. Malu. Optimization of project effort estimate using neural network. In: *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, pages 406–410, IEEE, 2014.
- [193] N. Couellan. Probabilistic robustness estimates for feed-forward neural networks. *Neural Networks*, 142: 138-147, 2021.
- [194] M. Pandey, R. Litoriya, and P. Pandey. Validation of existing software effort estimation techniques in context with mobile software applications, *Wireless Personal Communications*, 110 (4):1659–1677, 2020.
- [195] G. Boetticher, An of metric contribution in the construction of a neural network-based effort estimator. In: *Second International Workshop on Soft Computing Applied to Software Engineering*, Enschede, NL, 2001.
- [196] K. Kamaraj, C. Arvind, and K. Srihari. A weight optimized artificial neural network for automated software test oracle. *Soft Computing*, 24 (17):13501–13511, 2020.
- [197] M. Albarqi, R. Alsulami, and J. Graham. Automated data processing of neutron depth profiling spectra using an artificial neural network. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 953: 163217, 2020.
- [198] M. A. Hamada. Neural Network Estimation Model to Optimize Timing and Schedule of Software Projects. In *2021 IEEE International Conference on Smart Information Systems and Technologies (SIST)*, pages 1-7, IEEE, 2021.

- [199] C. L. C. Roxas. Modeling road construction project cost in the philippines using the artificial neural network approach. In: *2019 IEEE 11th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, pages 1–5, IEEE, 2019.
- [200] J. Popović and D. Bojić. A Comparative Evaluation of Effort Estimation Methods in the Software Life Cycle. *Computer Science and Information Systems*, 9 (1): 455-484, 2012. ISSN: 1820-0214.
- [201] G. De Vito, F. Ferrucci, and C. Gravino. Design and automation of a COSMIC measurement procedure based on UML models. *Software and Systems Modeling*, 19 (1): 171-198, 2020.
- [202] C. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30 (5): 416-429, 1987.
- [203] V. T. Ho and A. Abran. A Framework for automatic function point counting from source code. In *International Workshop on Software Measurement (IWSM'99)*, 1999.
- [204] C. Symons. Software Sizing and Estimating, Mk II Function Point Analysis. *John Wiley & Sons*, Chichester, England, 1991.
- [205] https://cosmicsizing.org/wpcontent/uploads/2015/07/Measurement_Patterns_Guideline_v1.01-1.pdf
- [206] M. Salmanoglu, T. Hacaloglu, and O. Demirors. Effort estimation for agile software development: comparative case studies using COSMIC functional size measurement and story points. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (IWSM Mensura '17)*, pages 41-49, Association for Computing Machinery, New York, USA, 2017.
- [207] I. Hussain, L. Kosseim, and O. Ormandjieva. Approximation of COSMIC functional size to support early effort estimation in Agile. *Data & Knowledge Engineering*, 85: 2-14, 2013.
- [208] Z. Sakhrawi, A. Sellami, and N. Bouassida. Investigating the Impact of Functional Size Measurement on Predicting Software Enhancement Effort Using Correlation-Based Feature Selection Algorithm and SVM Method. In: Ben Sassi S., Ducasse S., Mili H. (eds) *Reuse in Emerging Software Engineering Practices. ICSR 2020. Lecture Notes in Computer Science*, 12541. Springer, Cham, 2020.
- [209] T. Fehlmann and L. Santillo. From story points to cosmic function points in agile software development—a six sigma perspective. In *Metrikon-Software Metrik Kongress*, page 24, 2010.
- [210] M. Ochodek, S. Kopczyńska, and M. Staron. Deep learning model for end-to-end approximation of COSMIC functional size based on use-case names. *Information and Software Technology*, 123 (07):106310, 2020.

- [211] S. Bagriyanik and A. Karahoca. Using Data Mining to Identify COSMIC Function Point Measurement Competence. *International Journal of Electrical and Computer Engineering*, 8 (6): 5253, 2018.
- [212] W. Xia et al. A new calibration for Function Point complexity weights. *Information and Software Technology*, 50 (7-8): 670-683, 2008.
- [213] W. Xia, D. Ho, and L. F. Capretz. A neuro-fuzzy model for function point calibration. *arXiv preprint*, arXiv:1507.06934, 2015.
- [214] https://www.cs.cmu.edu/~jhm/DMS%202011/Presentations/Cohn%20%20Estimating%20with%20Use%20Case%20Points_v2.pdf
- [215] M. Kassab, C. Neill, and P. Laplante. State of practice in requirements engineering: contemporary data. *Innovations in Systems and Software Engineering*, 10:(4): 235-241, 2014.
- [216] M. Ochodek, J. Nawrocki, and K. Kwarciak. Simplifying effort estimation based on Use Case Points. *Information and Software Technology archive*, 53 (3): 200-213, 2011.
- [217] E. R. Carroll. Estimating software based on use case points. *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 257-265, San Diego, CA, USA, 2005.
- [218] A.B. Nassif, L.F. Capretz, and D. Ho. Enhancing Use Case Points Estimation Method using Soft Computing Techniques. *Journal of Global Research in Computer Science*, 1 (4): 12- 21, 2010.
- [219] A.B. Nassif. Software Size and Effort Estimation from Use Case Diagrams Using Regression and Soft Computing Models. Doctoral dissertation, Western University, London, Ontario, Canada, 2012.
- [220] M. Azzeh. Fuzzy Model Tree for Early Effort Estimation Machine Learning and Applications. *12th International Conference on Machine Learning and Applications*, pages 117-121, Miami, Florida, USA, IEEE, 2013.
- [221] T. Urbanek et al. Using Analytical Programming and UCP Method for Effort Estimation. Modern Trends and Techniques in Computer Science, *Advances in Intelligent Systems and Computing*, 285: 571-581, 2014.
- [222] A. Kaur and K. Kaur. Effort estimation for mobile applications using use case point (UCP). *In Smart Innovations in Communication and Computational Sciences*, pages 163-172, Springer, Singapore, 2019.
- [223] Z. C. Ani, S. Basri, and A. Sarlan. A reusability of UCP-based effort estimation framework using object-oriented approach. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9 (3-5): 111-114, 2017.
- [224] Y. Mahmood, N. Kama, and A. Azmi. A systematic review of studies on use case points and expert-based estimation of software development effort. *Journal of Software: Evolution and Process*, 32 (7), e2245, 2020.

- [225] K. K. Gebretsadik and W. T. Sewunetie. Designing Machine Learning Method for Software Project Effort Prediction. *Computer Science and Engineering*, 9(1): 6-11, 2019.
- [226] R. Alves, P. Valente, and N. J. Nunes. Improving software effort estimation with human-centric models: a comparison of UCP and iUCP accuracy. *In Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 287-296, 2013.
- [227] E. Guresen and G. Kayakutlu. Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 3: 426-433, 2011.
- [228] J. Zupan. Introduction to artificial neural network (ANN) methods: what they are and how to use them. *Acta Chimica Slovenica*, 41: 327-327, 1994.
- [229] S. Lek and J. F. Guégan. Artificial neural networks as a tool in ecological modelling, an introduction. *Ecological modelling*, 120 (2-3): 65-73, 1999.
- [230] A. R. Venkatachalam. Software cost estimation using artificial neural networks. *In Proceedings of 1993 International Conference on Neural Networks (IJCNN-93 Nagoya, Japan*, pages 987-990, IEEE, 1993.
- [231] M. H. Sazli. A brief review of feed-forward neural networks. Communications Faculty of Sciences University of Ankara Series A2-A3 *Physical Sciences and Engineering*, 50 (01), 2006.
- [232] I. A. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43 (1): 3-31, 2000.
- [233] B. J. Yum et al. The Taguchi robust design method: Current status and future directions. *Journal of Korean Institute of Industrial Engineers*, 39 (5): 325-341, 2013.
- [234] K. L. Tsui. An overview of Taguchi method and newly developed statistical methods for robust design. *IIE Transactions*, 24 (5): 44-57, 1992.
- [235] J. D. Kechagias et al. A comparative investigation of Taguchi and full factorial design for machinability prediction in turning of a titanium alloy. *Measurement*, 151: 107213, 2020.
- [236] N. Rankovic. et al. A new approach to software effort estimation using different Artificial Neural Network architectures and Taguchi Orthogonal Arrays. *IEEE Access*, 9: 26926-26936, 2021.
- [237] N. Rankovic et al. Improved effort and cost estimation model using Artificial Neural Networks and Taguchi method with different activation functions. *Entropy*, 23 (7), 854, 2021.
- [238] A. Jeang and C. L. Chang. Combined robust parameter and tolerance design using Orthogonal arrays. *The International Journal of Advanced Manufacturing Technology*, 19 (6), 442-447, 2002.

- [239] E. Kolaiti and C. Koukouvinos. On the use of three level Orthogonal arrays in robust parameter design. *Statistics & probability letters*, 76 (3): 266-273, 2006.
- [240] R. Fontana, F. Rapallo, and H. P. Wynn. Circuits for robust designs. *arXiv preprint*, arXiv:2106.11213, 2021.
- [241] D. Rankovic et al. Convergence rate of Artificial Neural Networks in software development projects. *Information and Software Technology Journal*, 138 (10), 2021.
- [242] <http://promise.site.uottawa.ca/SERepository/>
- [243] H. K. Lam. A review on stability analysis of continuous-time fuzzy-model-based control systems: From membership-function-independent to membership-function-dependent analysis. *Engineering Applications of Artificial Intelligence*, 67: 390-408, 2018.
- [244] L. Ghosh, S. Saha, and A. Konar. Decoding emotional changes of android-gamers using a fused Type-2 fuzzy deep neural network. *Computers in Human Behavior*, 116: 106640, 2021.
- [245] A. B. Nassif et al. Software development effort estimation using regression fuzzy models. *Computational intelligence and neuroscience*, 2019.
- [246] M. V. Boldin. On the power of Pearson's test under local alternatives in autoregression with outliers. *Mathematical Methods of Statistics*, 28(1): 57-65, 2019.
- [247] D. Blum and H. Holling. Spearman's law of diminishing returns. A meta-analysis. *Intelligence*, 65: 60-66, 2017.
- [248] W. Li. A new approach to solve uncertain multidisciplinary design optimization based on conditional value at risk. *IEEE Transactions on Automation Science and Engineering*, 18 (1): 356-368, 2020.
- [249] L. Qiao. Deep learning based software defect prediction. *Neurocomputing*, 385: 100-110, 2020.
- [250] M. A. Shah. Ensembling artificial bee colony with analogy-based estimation to improve software development effort prediction. *IEEE Access*, 8: 58402-58415, 2020.
- [251] <https://www.isbsg.org/software-project-data/>
- [252] <https://data.mendeley.com/datasets/2rfkjh3cn/1>
- [253] P. Manali et al. Long-lead prediction of ENSO modoki index using machine learning algorithms. *Scientific reports*, 10 (1): 1-13, 2020.
- [254] H. Liang. Dynamic evaluation of drilling leakage risk based on fuzzy theory and PSO-SVM algorithm. *Future Generation Computer Systems*, 95: 454-466, 2019.
- [255] Y. Liu, L. Wang, and K. Gu. A support vector regression (SVM)-based method for dynamic load identification using heterogeneous responses under interval uncertainties. *Applied Soft Computing*, 107599, 2021.

Author Biographies



Dr Nevena Ranković, PhD

Dr. Nevena Ranković, PhD was born in Valjevo in 1994. She completed her secondary education at the "Mathematical Gymnasium," a unique school in Serbia specialized for students talented in mathematics, physics, and computer science, in Belgrade in **2012**.

Her academic journey then led her to pursue a Bachelor's degree in Informatics, specializing in Information Technology, which she successfully completed in **2015**. She studied at the Department of Computer Science within the Faculty of Natural Sciences at the University of Novi Sad.

From **2015** to **2016**, Nevena continued her studies by attending the Master's program in Software Engineering at the Technical Faculty in Stuttgart, Germany. During the same period, she secured her first employment as a Junior Software Engineer at Daimler-Mercedes-Benz in Singelfinden, Stuttgart, Germany.

In **2016**, Nevena returned to Serbia and enrolled in the Master's program in Informatics, specializing in Software Engineering, at the Department of Computer Science within the Faculty of Natural Sciences at the University of Novi Sad. Her Master's thesis, titled "Risk Analysis in Software Project Management," was successfully defended in collaboration with the research station "*Petnica*" in Valjevo in **2018**.

Undeterred by challenges, Nevena embarked on her doctoral studies in 2018 at the same institution. She successfully defended her doctoral dissertation in **2022**, focusing on the application of artificial intelligence in software engineering, using special techniques of optimization through Latin extraction robust design of the experiment.

Between **2018** and **2019**, Nevena was appointed as a teaching associate and assistant at the Department of Mathematics and Informatics, Faculty of Natural Sciences, University of Novi Sad, specializing in Computer Science. She also earned the titles of teaching assistant, associate and assistant professor at the Department of Computer Science, Faculty of Computing, University Union in Belgrade.

Nevena's expertise extended beyond academia as she worked **at Software AG in Berlin**, Germany, within the Alphabet Center, excelling as an architect of software solutions and transformations.

Throughout her career, Nevena actively contributed to research projects, including two internal projects at the Faculty of Computing, University Union in Belgrade, focusing on the application of artificial intelligence in software engineering. She also served as a research collaborator on two bilateral projects.

In addition to her research endeavors, Nevena co-authored a textbook on statistical analysis and data processing for students at the Faculty of Applied Sciences, University Union "Nikola Tesla."

Currently, Nevena Ranković holds the position of **Assistant Professor** at the Department of Cognitive Science and Artificial Intelligence at Tilburg University, Netherlands. She is the lead coordinator for the Master's program in **Business Intelligence for Data Science** and teaches multiple courses in the field of artificial intelligence.

Nevena's extensive research spans various domains, including the application of artificial intelligence in software engineering, AI medical-diagnostics, cybersecurity, business intelligence, agile development, web technologies, software quality, requirements engineering, software project management, software testing, automotive engineering, and optimization problems. She has authored or co-authored 22 scientific papers, with 14 published in reputable international journals and presented several at international conferences. Nevena has been honored with the **Crystal Award** and with the "**Best Paper Award**" price at two international conferences within the IEEE association and CEUR PROCEEDING in 2019 and 2021, respectively.



Dr Dragica Ranković, PhD

Dr. Dragica Ranković, PhD was born in Valjevo in 1963. She completed her secondary education at Gymnasium in Valjevo in **1990**. She completed her undergraduate studies in 1990 at the Department of Mathematics, Faculty of Natural Sciences and Mathematics, University of Kragujevac. She completed her master's studies in the field of technical sciences in **2007** at the Technical Faculty, University of Novi Sad. At the same faculty, in **2009**, she defended her doctoral dissertation in the field of the application of fuzzy theory in mathematics education.

She is the author of several books: "Mathematics 1," "Mathematics 2," and "Mathematics for Experimental Classes." She is also the author of textbooks for undergraduate students related to statistical analysis and data processing at Union University "Nikola Tesla."

She has participated in several national projects, serving as the main co-laborator and external advisor to the school administration for the Kolubara District, with representation in Valjevo. She worked as a mathematics teacher in a high school in Valjevo for 30 years, where she also held the position of school director. She was appointed as an assistant professor with a doctoral degree, lecturer, and associate professor at the Department of Mathematics, Faculty of Computer Science, Union University in Belgrade. Currently, she works as an associate professor at Union University "Nikola Tesla."

Her areas of scientific interest include statistics, fuzzy theory, mathematical analysis, applied artificial intelligence (especially medical AI diagnostics), linear algebra, and analytic geometry.

She has authored or co-authored 20 scientific papers, with 13 published in reputable international journals, and has presented several at international conferences. Additionally, she co-authored a paper that received the "**Best Paper Award**" at international conferences within the IEEE association in 2021.



Dr Jelena Kaljević, PhD

Dr Jelena Kaljević, PhD, was born in 1982 in Valjevo, Republic of Serbia. She lives in Valjevo and works as Assistant Professor at the Faculty of Health and Business Studies, Singidunum University, Valjevo.

She graduated from the Faculty of Business Informatics at Singidunum University, study program *Programming and Design*, which she successfully completed in **2007** to hold the title of Bachelor of Business Informatics. She completed her bachelor academic studies and earned a 9.68 grade point average (max.10) and a grade of 10 in the graduation exam. She completed the Master's study program *Modern Information and Communication Technologies* in **2010** with an average grade of 9.33 (max.10), defending the Master's thesis entitled "*Artificial Intelligence Model through Game Theory*" with a grade of 10 and obtained the title of Graduated Engineer of Business Informatics - Master.

She completed doctoral academic studies of the third degree in the *Engineering Systems in Management* in **2018** defending the doctoral dissertation entitled "*Statistical evaluation of the improvement of the success of students who studied through the DLS platform*" with an average grade of 10 (max.10) and obtained the title of **Doctor of Science - Engineering Management**.

She has been working at the Faculty since **2005**, first as a Teaching associate, then as an Assistant and since **2019** as an Assistant professor. In **2022**, she was elected as the Head of the Department of Healthcare at the Faculty of Health and Business Studies.

With many years of teaching experience and student mentoring, she manages courses in the fields that link computer science and business field: IT solutions, applied mathematics and statistics, business communication and decision-making, data science, machine learning and contemporary information systems at undergraduate and master studies.

She has authored or co-authored **22 scientific papers**, with 5 of them published in international journals, and 17 presented at international and national conferences. Also, she has participated in several projects, the most important of which are **AITREATMED - "Artificial Intelligence for Enhancing Treatment Choices in Medicine"** funded by the Science Fund of the Republic of Serbia, as well as the project **"New Skills For New Industries - National IT Retraining Program"** organized by the Prime Minister's Office in partnership with the Ministry of Labor, Employment, Veterans and Social affairs and the Office of Information Technologies and Electronic Government in cooperation with the United Nations Development Program (UNDP). She is a member of organisation comitee for IT science conference **"Network"** at the Faculty of Health and Business Studies, Singidunum University, 2013-2023. Since 2016, she has been a member of the department for IT and health studies, and she has participated in numerous conferences and congresses organized by the Association of Nurses of Serbia, where the topics were related to the latest IT solutions in healthcare.

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.8
004.41

RANKOVIĆ, Nevena, 1994-

Artificial Intelligence Techniques in Software Engineering / Nevena Ranković,
Dragica Ranković, Jelena Kaljević. - 1st ed. - Begrade : Singidunum University,
2023 (Belgrade : Publikum). - X, 241 str. : ilustr. ; 24 cm

Slike autora. - Tiraž 300. - Author Biographies: str. 236-240. - Bibliografija: str.
216-234.

ISBN 978-86-7912-809-6

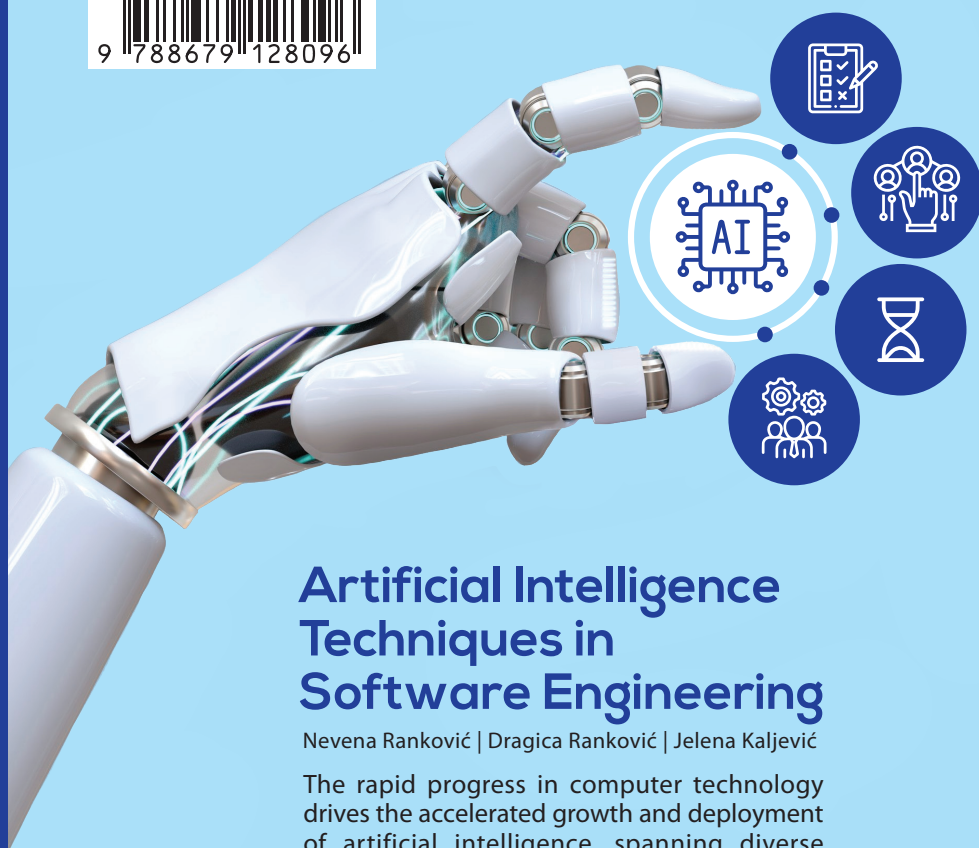
1. Ranković, Dragica, 1963- [autor] 2. Kaljević, Jelena, 1982- [autor]

a) Вештачка интелигенција b) Софтверско инжењерство

COBISS.SR-ID 131060745

© 2023

All rights reserved. No part of this work covered by the copyright herein may be reproduced, transmitted, stored or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, without the prior written permission of the publisher.



Artificial Intelligence Techniques in Software Engineering

Nevena Ranković | Dragica Ranković | Jelena Kaljević

The rapid progress in computer technology drives the accelerated growth and deployment of artificial intelligence, spanning diverse techniques and models tailored for specific purposes. Today, software pervades all aspects of life, often unnoticed, from basic mobile apps to advanced systems that replace human functions. This book explores fundamental AI techniques integrated into everyday life and business, emphasizing their adaptable nature to address various challenges. It delves into the intersection of AI and software engineering, particularly in software estimation. This includes artificial neural networks, Taguchi's method, software quality metrics, and innovative models designed to enhance estimation accuracy. Targeted at students and professionals, it enriches AI application knowledge in software engineering and project management.