Software Cost Estimation using neural networks.

Robin Ramaekers$^{1[0000-1111-2222-3333]}$

[1] Thomas More Geel, Kleinhoefstraat 4, 2440 Geel, Belgium
robin.ramaekers2@gmail.com

Abstract. Software Cost Estimation (SCE) is one of the most vital parts when starting a new software engineering project, it helps with allocating resources, managing risks, making informed decisions, and stands in correlation with the success or the failure of a project. Because Software Cost Estimation (SCE) is prone to human bias, solutions started being researched with the aid of Artificial Intelligence (AI) and Machine Learning (ML). This paper will investigate the importance of Software Cost Estimation (SCE). Further, the existing taxonomies and methodologies regarding the use of neural networks with Software Cost estimation will be compared (COCOMO, GEHO-ANN, OLCE, and -ANN-NEAT). This will be done with the use of evaluation metrics such as RMSE, MMRE, PRED, MAE, etc. After, a proposal for further research is made on why **using Deep Reinforcement Learning (DRL)** could be very beneficial for the development of Software Cost Prediction Models. This technique combines Deep Learning (DL) and Machine Learning (ML) and can solve complex tasks with many variables and a rapidly developing environment.

Keywords: Software Cost Estimation, Artificial Intelligence, Machine Learning, Deep Reinforcement Learning, Neural Networks.

1 Introduction

The importance of Software Cost Estimation (SCE) cannot be understated in a new software development project. The core goal of Software Cost Estimation (SCE) is to perform a prognostic analysis of the cost of the development of software projects before the project phases are initiated. The term cost refers to the number of resources that will be engaged in the development, time consumed in development, and all approximated effort to achieve the end-state of the development phase. The process of Software Cost Estimation (SCE) stands in correlation with the success or the failure of a project [1]. The core challenge of Software Cost Estimation is lack of expertise, lack of historical data, overconfidence, human bias, etc. this is the reason why Software Cost Estimation (SCE) Models using Artificial intelligence (AI) have been developing over the last decade. Artificial Intelligence (AI) has been rapidly growing and shows great promise in automatization and making predictions based on the data that is presented. This rapid evolution is thanks to more powerful hardware and data that has become publicly available. The most frequently adopted model in the software industry is COCOMO because of its simplicity, flexibility, historical data, and wide applicability but this model has its strengths and weaknesses, this is the reason that researchers have been experimenting with diverse methods. The COCOMO model is based on the COCOMO dataset. Many more data sources regarding SCE are now publicly available, this caused many researchers to take interest in the topic. Many machine learning techniques have been applied to do SCE e.g., GEHO-based NFN, OLSE, ANN-NEAT, these algorithms can achieve decent accuracy and minimal loss. In this paper, the above-mentioned algorithms are summarized and compared to each other. The research on these techniques where done in the last year (2022-2023) and where found on Google scholar. Finally, it was noted that while there is a lot of information about the use of machine learning regarding Software Cost Estimation, Deep Learning a more capable form of Machine Learning was rarely mentioned or researched that is why I propose a Deep Reinforcement Learning approach, the reasoning being that it combines Machine Learning and Deep Learning techniques to solve complex problems with a lot of variables and an adaptive environment similar to that of software engineering. This paper is divided into 10 sections: 1 Introduction, 2 Related works, 3 Software Cost Estimation, 4 Data sources, 5 Neural Network implementation, 6 Evaluation metrics, 7 Methodologies, 8 Future proposal, 9 Conclusion, and 10 Literature cited. The use of Artificial Intelligence in Software Cost Estimation is becoming increasingly important, as it has the potential to improve accuracy, reduce costs, and enhance decision-making processes in software development projects, highlighting the need for greater adoption of Deep Learning approaches in the field.

2 Related works

Several studies have investigated the use of Neural Networks in Software Cost Estimation focusing on Machine Learning. For example, Sudhir Sharma & Shripal Vijayvargiya (2022) have developed several Neural Network models using Machine Learning to do Software Cost Estimation, they benchmarked all the models on different data sources to see which model and technique performed the best. Meanwhile, K Nitalaksheswara Rao, Jhansi Vazram Bolla, Satyanarayana Mummana, CH. V. Murali Krishna, O. Gandhi & M James Stephen (2022) created an optimized learning-based Cost Estimation model and scheme. This approach achieved a balance between accurate predictions and efficiency for computational power. Other studies have explored the use of Artificial Neural Networks (ANN), with Puppala Ramya, M. Sai Mokshith, M. Abdul Rahman & N. Nithin Sai (2023) developing a model that can comprehend complex relationships and patterns found in the data sources for Software Cost Estimation.

3 Software Cost Estimation (SCE)

As mentioned in section 1, the importance of Software Cost Estimation (SCE) cannot be understated in a new software development project. The core goal of Software Cost Estimation (SCE) is to perform a prognostic analysis of the cost of the development of software projects before the project phases are initiated. The term cost refers to the number of resources that will be engaged in the development, time consumed in development, and all approximated effort to achieve the end-state of the development phase. In the figure, Figure 1, a typical cost estimation practice is shown. In this case, the project manager is responsible for the target software cost. The process below cycles and the various attributes are modified until the cost of the target software is found justified. [4]

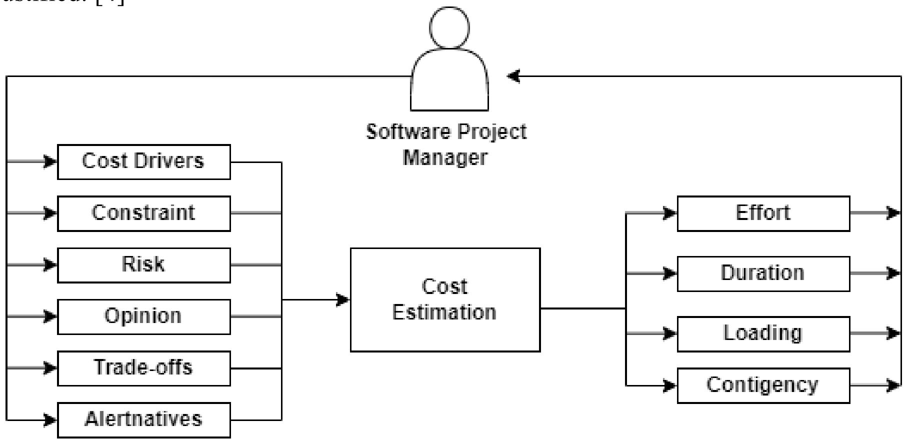


Figure 1: The typical process of Software Cost Estimation [4]

The process shown in Figure 1 is a critical practice to perform analysis and predict all the possible risks and trade-offs in the costing procedure. This process gives the software project manager the possibility to investigate the software project and filter out the evaluated risk possibilities to accomplish the cost of the target software. It`s worth keeping in mind that the project manager and the various department teams work together to realize the final target cost. There are a variety of input variables that are used in cost estimation as shown in Figure 1, cost drivers, constraint, risk, opinion, trade-offs, and alternatives. On the other hand, the outcome of the estimation is evaluated concerning effort, duration, loading, and contingency. In this next section, there will be a focus on the different taxonomies and challenges that are linked with Software Cost Estimation. [4]

3.1 Taxonomies

Currently, several cost estimation models are already available for Software Cost Estimation e.g., COCOMO (Constructive Cost Model), COCOMO II, Function Point Analysis (FPA), PERT (Program Evaluation and Review Technique), etc. Each of these models has its strengths and weaknesses, and the suitability of a particular model depends on the project's specific requirements, constraints, and scope. Further detail on the advantages and disadvantages are described in the section below. Out of all these models, the most frequently adopted in the software industry is COCOMO because of its simplicity, flexibility, historical data, and wide applicability. Apart from this Popular SCE model, standard taxonomy cost prediction methods also exist:

- Analogous Estimating: Analogous Estimating is a method used in project management to estimate the cost or duration of a current project by comparing it to a similar completed project. This method is also known as top-down estimating because it involves using data from a previous project to estimate the cost of the current project. Analogous Estimating is often used when limited information about the current project is available.

- Bottom-Up Estimating: Bottom-Up Estimating is a method used in project management to estimate the cost or duration of a project by breaking it down into individual components and estimating the cost or duration of each component. The total cost or duration of the project is then calculated by adding up the costs or durations of each component. This method is more accurate than Analogous Estimating but can be more time-consuming.

- Three-Point Estimating: Three-Point Estimating is a method that involves estimating three scenarios for each project component: an optimistic estimate, a pessimistic estimate, and a most likely estimate. The final estimate is calculated using a weighted average of the three estimates.

These are just a few examples; the appropriate method depends on the specific use case and characteristics of the project.

3.2 Challenges of existing methods

As previously stated, there are still several challenges that must be tackled when utilizing these methods. Some of the key challenges linked to the current approaches are:

Analogous Estimating

- Finding a suitable reference project: The accuracy of Analogous Estimating is dependent on the similarity between the current project and the reference project used for estimation. Finding a suitable reference project can be challenging, especially if the current project is unique or complex.

- Historical data: Analogous Estimating relies on historical data from a finished project to estimate the cost of the current project. However, there may be limited historical data available, making it difficult to accurately estimate the cost of the current project.

- Project details: Analogous Estimating relies on the assumption that the details of the current project are similar to the reference project used for estimation. However, if there are significant differences in the details of the two projects, the estimate may not be accurate.

- Assumptions and biases: Analogous Estimating involves making assumptions about the current project based on the reference project used for estimation. These assumptions can be subject to biases, such as optimism bias, which can lead to inaccurate estimates.

- Lack of transparency: Analogous Estimating can be less transparent than other cost estimation methods, as it relies on the expertise of the estimator to produce accurate estimates. This can make it difficult for stakeholders to understand and validate the estimates.

Bottom-Up Estimating

- Time-consuming: Bottom-Up Estimating can be a time-consuming process, as it involves breaking down the project into smaller tasks and estimating the cost of each task individually. This can be a significant challenge for large and complex software development projects, as it can require a lot of time and effort to produce accurate estimates.

- Lack of expertise: Bottom-Up Estimating requires a deep understanding of software development processes and practices. If the project team lacks expertise in cost estimation, it can be challenging to produce accurate estimates.

- Changing requirements: Software development projects often have changing requirements, which can make it challenging to estimate costs accurately. As requirements change, estimates may need to be revised, which can add time and cost to the project.

- Historical data: See the explanation in the section above.

- Human bias: See the explanation in the section above.

Three-Point Estimating

- Subjectivity: Three-Point Estimating relies on the judgment of the estimator to determine the optimistic and pessimistic estimates of a task. This subjectivity can lead to inaccurate estimates if the estimator is biased or lacks expertise.

- Overconfidence: Estimators may be overconfident in their ability to estimate the cost of a task, leading to overly optimistic estimates. This can lead to cost overruns and project delays later in development.

- Lack of transparency: See explanation in section 'Analogous Estimating'.

- Historical data: See explanation in section 'Analogous Estimating'.

- Changing requirements: See the explanation section above.

4 Data sources

Several data sources regarding Software Cost Estimation are already available, they are used to train artificial intelligence models e.g., the cocomo81, cocomonasa1, cocomonasa v2, Desharnais, China, etc. datasets. For my further research, I propose to work on a variety of data sources to evaluate the model's performance in different scenarios because this is the foremost representation of the real world.

4.1 Most common

The most used data sources are cocomo'81, NASA, and Desharnais. In 2019 the number of projects using these data sources looks like this:

Table 1: Comparison data sources with their number of projects and methodologies.

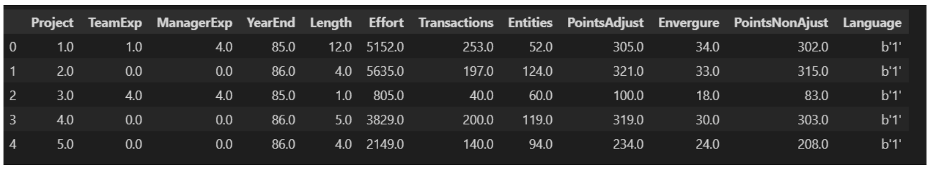
Data source	# of projects	Methodologies
Desharnais	Unk	GEHO-NFN
NASA + industrial	Ninety-nine projects	FNN-SEERSEM, SEERSEM
COMOMO`81	Sixty-nine projects	FNN,COCOMO

4.2 EDA Data sources

In this section, a clarification regarding the contents of the listed data sources will be available. Understanding the data in a dataset is an essential first step to creating an Artificial Intelligence model. This also allows for a thorough Exploratory Data Analysis (EDA) of the provided data.

Desharnais

The Desharnais data source is a cost estimation dataset that contains eighty records. The first 5 records in the dataset are shown in the Figure (Figure 2) below.



	Project	TeamExp	ManagerExp	YearEnd	Length	Effort	Transactions	Entities	PointsAdjust	Envergure	PointsNonAjust	Language
0	1.0	1.0	4.0	85.0	12.0	5152.0	253.0	52.0	305.0	34.0	302.0	b'1'
1	2.0	0.0	0.0	86.0	4.0	5635.0	197.0	124.0	321.0	33.0	315.0	b'1'
2	3.0	4.0	4.0	85.0	1.0	805.0	40.0	60.0	100.0	18.0	83.0	b'1'
3	4.0	0.0	0.0	86.0	5.0	3829.0	200.0	119.0	319.0	30.0	303.0	b'1'
4	5.0	0.0	0.0	86.0	4.0	2149.0	140.0	94.0	234.0	24.0	208.0	b'1'

Figure 2: First 5 records in the data source.

The table below will explain the meaning of the "effort multipliers".

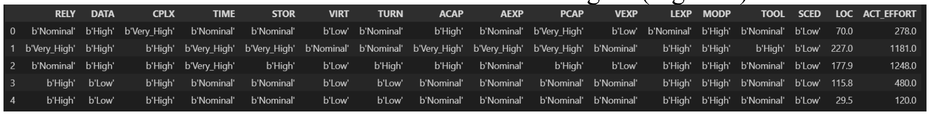
Table 2: Columns and their explanations.

Project	This column indicates the **required software reliability**.
TeamExp	This column indicates the **team experience (in years)**.
ManagerExp	This column indicates the **manager's experience (in years)**.
YearEnd	
Length	
Effort	This column indicates the **Actual effort (in person-hours)**.
Transactions	This column indicates the **count of basic logical transactions in the system**.
Entities	This column indicates the **number of entities in the systems data model**.
PointsAdjust	This column indicates the **application experience**.
Envergure	
PointsNonAdjust	
Language	This column indicates the **programming languages**.

NASA

The COCOMO NASA data source is based on the COCOMO`81 but instead of classifying the "effort multipliers" in 'low', 'nominal', 'high', and 'very_high' instead of numeric values. Note these classifications are based on the standard numeric values that are shown in Figure 3 in the section below.

The first five records in the dataset are shown in the Figure (Figure 3) below.



	RELY	DATA	CPLX	TIME	STOR	VIRT	TURN	ACAP	AEXP	PCAP	VEXP	LEXP	MODP	TOOL	SCED	LOC	ACT_EFFORT
0	b'Nominal'	b'High'	b'Very_High'	b'Nominal'	b'Nominal'	b'Low'	b'Nominal'	b'High'	b'Nominal'	b'Very_High'	b'Low'	b'Nominal'	b'High'	b'Nominal'	b'Low'	70.0	278.0
1	b'Very_High'	b'High'	b'High'	b'Very_High'	b'Very_High'	b'Nominal'	b'Nominal'	b'Very_High'	b'Very_High'	b'Very_High'	b'Nominal'	b'High'	b'High'	b'High'	b'Low'	227.0	1181.0
2	b'Nominal'	b'High'	b'High'	b'Very_High'	b'High'	b'Low'	b'High'	b'High'	b'Nominal'	b'High'	b'Low'	b'High'	b'High'	b'Nominal'	b'Low'	177.9	1248.0
3	b'High'	b'Low'	b'High'	b'Nominal'	b'Nominal'	b'Low'	b'Low'	b'Nominal'	b'Nominal'	b'Nominal'	b'Nominal'	b'High'	b'High'	b'Nominal'	b'Low'	115.8	480.0
4	b'High'	b'Low'	b'High'	b'Nominal'	b'Nominal'	b'Low'	b'Low'	b'Nominal'	b'Nominal'	b'Nominal'	b'Nominal'	b'High'	b'High'	b'Nominal'	b'Low'	29.5	120.0

Figure 3: First 5 records in the data source.

The table below will explain the meaning of the "effort multipliers."

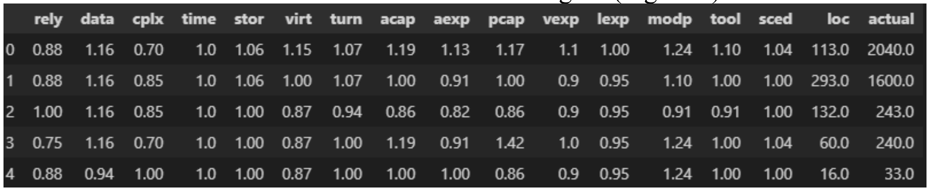
Table 3: Columns and their explanations.

rely	This column indicates the **required software reliability**.
data	This column indicates the **database size**.
cplx	This column indicates the **process complexity**.
time	This column indicates the **time constraint for the CPU**.
stor	This column indicates the **main memory constrain**t.
virt	This column indicates the **machine volatility**.
turn	This column indicates the **turnaround time**.
acap	This column indicates the **analyst's capability**.
aexp	This column indicates the **application experience**.
pcap	This column indicates the **programmer's capability**.
vexp	This column indicates the **virtual machine experience**.
lexp	This column indicates the **language experience**.
modp	This column indicates **modern programming practices**.
tool	This column indicates the **use of software tools**.
sced	This column indicates the **schedule constraint**.
loc	This column indicates the **Lines of code**.
act	This column indicates the **actual value**.

COCOMO`81

The COCOMO software cost model measures effort in calendar months of 152 hours (and includes development and management hours). COCOMO assumes that the effort grows more than linearly on software size. ("Comparative Study of the Performance of M5-Rules Algorithm with ...") Following the formula:
months= a*(KSLOC^b)*(EM1*EM2*EM3*...)
"a" and "b" are domain-specific parameters; "KSLOC" is estimated directly or computed from a function point analysis; and "c" is the product of over a dozen "effort multipliers." [2]

The first 5 records in the dataset are shown in the Figure (Figure 4) below.



	rely	data	cplx	time	stor	virt	turn	acap	aexp	pcap	vexp	lexp	modp	tool	sced	loc	actual
0	0.88	1.16	0.70	1.0	1.06	1.15	1.07	1.19	1.13	1.17	1.1	1.00	1.24	1.10	1.04	113.0	2040.0
1	0.88	1.16	0.85	1.0	1.06	1.00	1.07	1.00	0.91	1.00	0.9	0.95	1.10	1.00	1.00	293.0	1600.0
2	1.00	1.16	0.85	1.0	1.00	0.87	0.94	0.86	0.82	0.86	0.9	0.95	0.91	0.91	1.00	132.0	243.0
3	0.75	1.16	0.70	1.0	1.00	0.87	1.00	1.19	0.91	1.42	1.0	0.95	1.24	1.00	1.04	60.0	240.0
4	0.88	0.94	1.00	1.0	1.00	0.87	1.00	1.00	1.00	0.86	0.9	0.95	1.24	1.00	1.00	16.0	33.0

Figure 4: First 5 records in the data source.

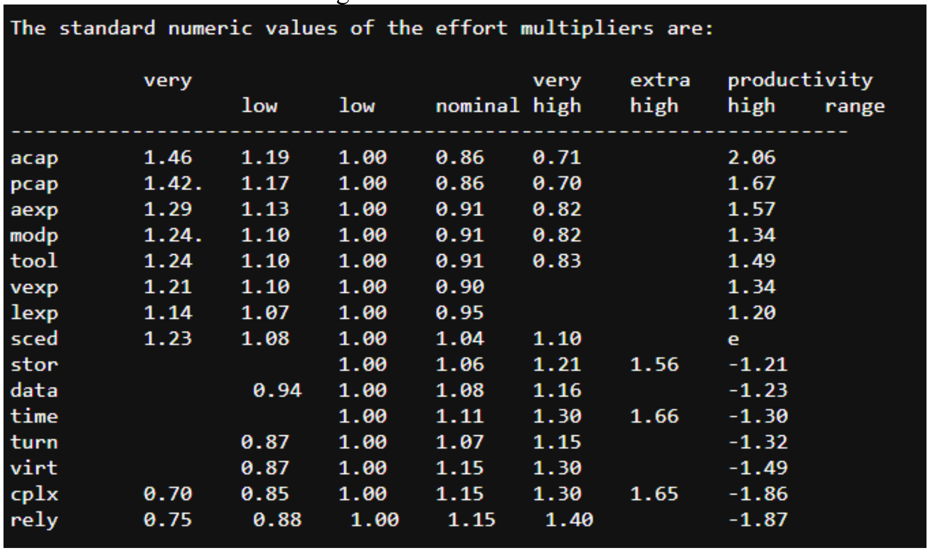
The table below will explain the meaning of the "effort multipliers."

Table 4: Columns and their explanations.

rely	This column indicates the **required software reliability**.
data	This column indicates the **database size**.

cplx	This column indicates the **process complexity**.
time	This column indicates the **time constraint for the CPU**.
stor	This column indicates the **main memory constrain**t.
virt	This column indicates the **machine volatility**.
turn	This column indicates the **turnaround time**.
acap	This column indicates the **analyst's capability**.
aexp	This column indicates the **application experience**.
pcap	This column indicates the **programmer's capability**.
vexp	This column indicates the **virtual machine experience**.
lexp	This column indicates the **language experience**.
modp	This column indicates **modern programming practices**.
tool	This column indicates the **use of software tools**.
sced	This column indicates the **schedule constraint**.
loc	This column indicates the **Lines of code**.
act	This column indicates the **actual value**.

Note: For Columns **acap, pcap, Aexp, modp, tool** and **vexp** a higher value indicates decreased effort as shown in Figure 5.



```
The standard numeric values of the effort multipliers are:

           very                            very    extra   productivity
                   low     low     nominal high    high    high    range
---------------------------------------------------------------------
acap       1.46    1.19    1.00    0.86    0.71            2.06
pcap       1.42.   1.17    1.00    0.86    0.70            1.67
aexp       1.29    1.13    1.00    0.91    0.82            1.57
modp       1.24.   1.10    1.00    0.91    0.82            1.34
tool       1.24    1.10    1.00    0.91    0.83            1.49
vexp       1.21    1.10    1.00    0.90                    1.34
lexp       1.14    1.07    1.00    0.95                    1.20
sced       1.23    1.08    1.00    1.04    1.10            e
stor                       1.00    1.06    1.21    1.56    -1.21
data                0.94   1.00    1.08    1.16            -1.23
time                       1.00    1.11    1.30    1.66    -1.30
turn               0.87    1.00    1.07    1.15            -1.32
virt               0.87    1.00    1.15    1.30            -1.49
cplx       0.70    0.85    1.00    1.15    1.30    1.65    -1.86
rely       0.75     0.88    1.00    1.15    1.40           -1.87
```

Figure 5: Value classifications.

5 Neural Network implementation

The methodologies as stated in section "3.2 Challenges of existing methods" have some critical challenges to overcome from whom most of which are human error and lack of experience. A solution to this problem is by using the power of Artificial Intelligence

(AI). The AI field is one of the fastest-growing sectors in IT at the moment. some examples of recent advancements in this field are ChatGPT and DALL-E both of which use Natural language Processing (NLP) techniques.

5.1 Evolution

The use of Artificial Intelligence in Software Cost Estimation has been a topic of research for several decades, but its practical implementation started gaining traction in the early 2000s.

One of the earliest examples of the use of AI in Software Cost Estimation was the COCOMO II model developed by Barry Boehm in 2000. COCOMO II is a constant software cost estimation model that uses a set of equations and algorithms to estimate the effort, time, and cost required to develop a software project. The model uses Machine Learning techniques to learn from historical data and refine its estimates over time.

Since then, many researchers and practitioners have explored the use of different AI techniques such as Neural Networks, decision trees, fuzzy logic, and genetic algorithms for Software Cost Estimation. These techniques have been applied to various software development processes, including agile development, traditional waterfall development, and iterative development.

Today, many Software Cost Estimation tools and platforms use AI and Machine Learning algorithms to provide accurate and reliable estimates. These tools leverage large datasets of historical project data to identify patterns and correlations and make predictions based on those patterns.

5.2 Advantages

There are several advantages of using AI in Software Cost Estimation, some examples are listed below:

- Improved accuracy: AI can analyze large datasets and identify patterns and correlations that may not be obvious to humans. This allows for more accurate cost estimations based on historical data.

- Increased efficiency: AI can automate many of the tasks involved in software cost estimation, such as data analysis and modeling. This reduces the time and effort required to generate estimations and allows teams to focus on other critical tasks.

- Scalability: Using AI it is possible to scale for handling large and complex software projects, which may be challenging for humans to estimate accurately. This allows organizations to estimate costs for projects of any size and complexity.

• Consistency: AI algorithms can provide consistent estimates based on predefined rules and parameters, which can reduce the variability that may occur when humans estimate costs.

• Continuous learning: AI algorithms can learn from new data and adjust their estimates over time, which can improve the accuracy of cost estimates as more data becomes available.

Overall, the use of AI in software cost estimation can improve accuracy, efficiency, scalability, and consistency, and can use continuous learning as time passes, which can lead to better project planning and decision-making.

6 Evaluation metrics

In this section, the evaluation metrics that are used to see how accurately a model will perform will be discussed. The metrics that are listed here will frequently occur in the next section.

6.1 MRE (Magnitude of relative error)

$$\text{Formula MRE: } \frac{\text{actual-estimated}}{\text{actual}} \tag{1}$$

6.2 MMRE (Mean Magnitude of Relative Error)

The MMRE calculates the average relative difference between the actual values and the predicted values, expressed as a percentage of the actual values. The lower the MMRE, the better the accuracy of the prediction model.

$$\text{Formula MMRE: } \frac{1}{n}\sum_{x=1}^{n} \text{MRE} \tag{2}$$

6.3 MSE (Mean Squared Error)

The MSE is a commonly used metric in statistics and machine learning to measure the average squared difference between the predicted values and the actual values. In other words, it is a way to quantify how far off the predicted values are from the actual values.

$$\text{Formula MSE: } \frac{1}{n}\sum_{x=1}^{n}(Y_i - Y_i^{\wedge})^2 \text{ (3)}$$

6.4 RMSE (Root Mean Squared Error)

The RMSE measures the standard deviation of the residuals and is expressed in the same units as the response variable. The lower the RMSE, the better the accuracy of the prediction model.

$$\text{Formula RMSE: } \sqrt{\sum_{i=1}^{n} \frac{(\text{Predicted}_i\text{-Actual}_i)^2}{n}} \tag{4}$$

6.5 MdMRE (Median Magnitude of Relative Error)

The MdMRE measures the median relative difference between the actual values and the predicted values expressed as a percentage of the actual values. The lower the MdMRE, the better the accuracy of the prediction model.

The use of median instead of mean in this metric makes it less sensitive to outliers, which may be useful in some contexts where the data is skewed or has extreme values.

$$\text{Formula MdMRE: median(MRE)} \tag{5}$$

6.6 PRED (Prediction accuracy)

The PRED is the accuracy measure; thus, the greater the PRED value, the higher the model's accuracy score. ("Modeling of software project effort estimation: a comparative ...") It is defined as the percentage of projects whose predicted values are within (N%) of their actual values.[3]

$$\text{Formula PRED: } \frac{100}{N} * \sum_{n=1}^{N} \left\{ 1, \text{ if } \text{MRE}_n \leq \frac{T}{100} \text{ (0, if not 1)} \right. \tag{6}$$

6.7 MAE (Mean Absolute Error)

The MAE measures the average absolute difference between the actual and predicted values of the target variable. It is a commonly used metric for regression problems, where the goal is to predict a continuous numerical value. A lower MAE value indicates better accuracy of the model.

$$\text{Formula MAE: } \frac{1}{n} * \sum_{i=1}^{n} \left\lfloor (y\text{-}y') \right\rfloor \tag{7}$$

7 Methodologies

In this section, the focus will lay on existing machine learning methodologies that have already been researched will be considered e.g., GEHO-NFN, AA-SEE, and ANN. These are just some examples of relevant studies.

7.1 Genetic elephant herding optimization (GEHO) based Neuro-fuzzy network (GEHO-based NFN).

In the first research paper, the most efficient method for solving software cost estimation problems is the GEHO-based NFN. In this proposed method, a software cost estimation model is devised and developed using the Genetic elephant herding optimization-based Neuro-fuzzy network (GEHO-based NFN) model. Here, the software data was collected from the database, and to select the prominent features, they used a correlation coefficient-based feature selection. After, they designed a Neuro-fuzzy system that is further trained to get the optimum weights using genetic algorithm-based. For the entire formulated method, an implemented Neuro-fuzzy model and GA-based EHO were combined to get the optimized results. [3]

The EHO (Elephant Herding Optimization) technique is a swarm-based meta-heuristic optimization algorithm used for solving optimization problems. It is based on the behavior of elephant herds, where the leader elephant, or the "matriarch", leads the herd toward food sources and water. The algorithm simulates the behavior of elephant herds by creating a population of solutions, where each solution represents an elephant. The algorithm uses a herd leader, which is the solution with the best fitness value, to guide the other solutions toward the optimal solution. During the optimization process, the algorithm uses two main strategies: exploitation and exploration. Exploitation involves exploiting the best solution found so far, while exploration involves searching for new solutions in the search space. These two strategies are used to balance convergence towards the global optimum and diversification of the search process.

Using this method, these results for the evaluation metrics ranging over 60% of the training data are achieved. Note, the model was trained on the COCOMONASA2 dataset.

Table 5: Evaluation metrics with their corresponding values. [3]

MMRE (6.2)	0.261
RMSE (6.3)	0.236
MdMRE (6.4)	0.246
PRED (6.5)	52.63%

7.2 Optimized Learning-based Cost Estimation (OLCE)

In the second research paper, an Optimized Learning-based Cost Estimation (OLCE) approach is proposed. the main purpose of OLCE is to implement automation towards software development as well as perform further training for Neural Network platforms using a search-based optimization scheme. This scheme is meant for evolving the units of a Neural Network with its structure and parameters of learning to perform a prognostic evaluation of the stability of Software Cost Estimation. According to their proposed search-based optimization
scheme, the scheme constructs an initial set of populations followed by scoring and scaling it. It further retains the best outcome while parents are selected to generate an outcome that is further used for scoring and scaling the population. In all the above-mentioned steps of operation, the scheme follows three core rules to finetune the outcome of the population: [4]

- Rule for Selection: The main principle that selects the individual referred to as the parent, resulting in a modified population in successive rounds of operation.
- Rule for Aggregation: This is the second principle that generates an updated population by merging information from two parents.
- Rule for Transformation: This is the tertiary principle that transforms a single parent to produce a modified set of the single population randomly.

The unique features of the OLCE scheme are as follows:

1. The scheme is capable of producing multiple outcomes of a population, with the stopping point of iteration determined by achieving the optimal score according to the fitness function.
2. The scheme employs a randomly selected number to determine the next cycle of population, which enhances the system's responsiveness and speed.
3. It is capable of processing multiple evaluation functions aimed at increasing the convergence rate.

To understand the unique features of their proposed scheme, Figure 6 will elaborate on the operation of the working in comparison to the conventional technique. [4]

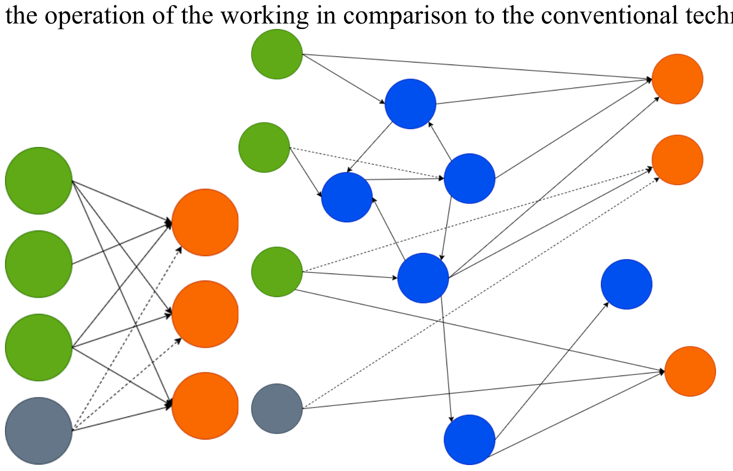


Figure 6: Standard Neural Network architecture and OLCE architecture [4]

For further information regarding the working of this scheme:
OLCE: Optimized Learning-based Cost Estimation
for Global Software Projects by Research Square

7.3 ANN with NEAT optimization (Artificial Neural Network with Neuro-Evolution of Augmenting Topology).

In the third research paper, an Artificial Neural Network with Neuro-Evolution of Augmenting Topology is proposed. Artificial Neural Networks (ANNs) are a type of Machine Learning model inspired by the structure and function of the human brain. ("About Artificial Neural Network - Engine Related input/output data?") ANNs consist of interconnected nodes or neurons organized into layers. The input layer receives the input data, and the output layer produces the output. ("Neural Networks: Sigmoid Functions and Output Layers") There may also be one or more hidden layers in between. Each neuron applies a non-linear activation function to the sum of its weighted inputs, and this activation value is then passed on to the next layer.

NEAT is an algorithm for optimizing ANNs through the process of evolution. NEAT starts with a minimal network structure and evolves it over time by adding or removing neurons and connections. NEAT maintains a population of different network structures, and each structure is evaluated for its fitness based on how well it performs on a given task. The fittest networks are then selected to produce offspring through crossover and mutation, which are added back to the population for the next generation. NEAT allows the topology of the network to evolve along with the weights and biases, which can lead to more efficient and effective networks. [5]

For further information regarding the working of this scheme: Effective ANN Model based on Neuro-Evolution Mechanism for Realistic Software Estimates in the Early Phase of Software Development

Using this method, these results for the evaluation metrics ranging over 60% of the training data are achieved. Note, the model was trained on the COCOMONASA2 dataset.

Table 6: Evaluation metrics with their corresponding values. [5]

MMRE (6.2)	0.113581
RMSE (6.3)	39.335067
MSE ()	1547.247493
MAE (6.6)	22.151230
PRED (6.5)	68.91522

8 Gap in knowledge

One of the gaps in knowledge with utilizing Machine Learning algorithms in Software Cost Estimation is the lack of large-scale, high-quality datasets for training and testing the Machine Learning models. Most Software Cost Estimation datasets are relatively small, and some may be of questionable quality, which can affect the overall performance and generalizability of the AI models. This also makes it harder to use more powerful techniques like Deep Learning CNNs (Convolutional Neural Networks).

Another gap is the interpretability and transparency of Machine Learning models. Machine Learning algorithms are considered "black box" models, meaning that it can be difficult to understand on what assumptions and variables they made their prediction. This makes it challenging to explain the reasoning behind cost estimates to stakeholders, this can be significant in software development projects. The use of decision trees can eliminate this problem, but decision trees are the most basic form of classification thus they are not optimized for a large set of variables.

Furthermore, there may be challenges in identifying relevant features or variables to include in Machine Learning models for Software Cost Estimation. This is because many factors can influence software development costs, some of which may be difficult to measure or quantify. As a result, identifying the most important features to include in Machine Learning models can be a challenging task.

8.1 Future proposal

My future proposal is to explore the possibilities of Deep Reinforcement Learning (DRL) in correlation to Software Cost Estimation. Deep Reinforcement Learning is a technique that uses Reinforcement Learning (RL) and Deep Learning (DL). Reinforcement Learning (RL) is a learning process where an agent learns to make decisions in challenging environments by interacting with the environment and receiving feedback in the form of a reward signal. The agent takes actions that affect the environment's state and receives feedback on how well it is performing. The aim is to learn a policy that maximizes the accumulated reward over time. It would be theoretically possible to use in Software Cost Estimation (SCE) because it is a complex problem with many variables and uncertainties. DRL can learn from experience and adapt to changing conditions which is important in software engineering which is a rapidly changing environment.

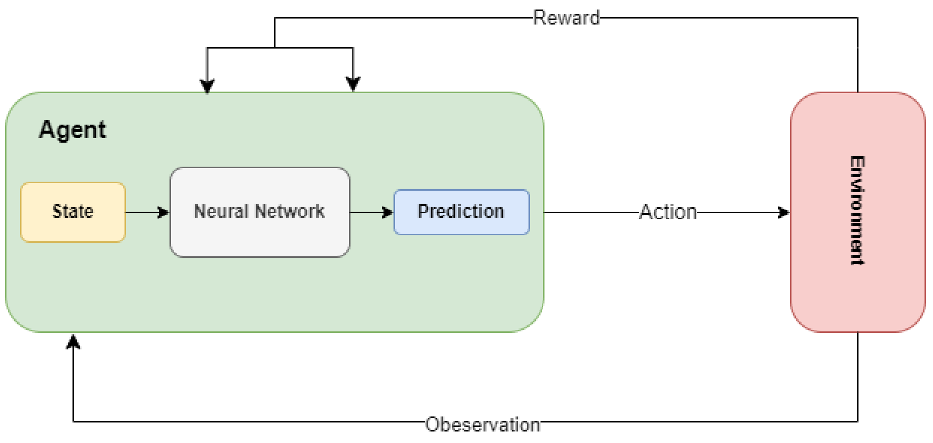


Figure 7: generalized DLR process.

9 Conclusion

In conclusion, developing algorithms and models to aid the process of Software Cost Estimation has come a long way. I have discovered that the most used technique to solve SCE problems are Machine Learning solutions. These solutions can yield fairly good results in terms of accuracy and values. But with the recent advancements of the rapid development in Deep Learning, it may be interesting to conduct research where the two techniques (ML and DL) are combined. That is why I propose a Deep Reinforcement Learning (DLR) architecture which is commonly used in solving very complex problems, seeing how many variables and environmental factors are used in Software Cost Estimation. It would be a perfect fit. The use of DLR in Software Cost Estimation is new and could be a huge advancement over the traditional machine learning approaches.

10 Literature cited

1. Kumar, K.H., Srinivas, K. An accurate analogy-based software effort estimation using hybrid optimization and machine learning techniques. Multimed Tools Appl (2023). https://doi.org/10.1007/s11042-023-14522-x
2. H. Duggal and P. Singh, "Comparative Study of the Performance of M5-Rules Algorithm with Different Algorithms," Journal of Software Engineering and Applications, Vol. 5 No. 4, 2012, pp. 270-276. doi: 10.4236/jsea.2012.54032.
3. Sharma, S., Vijayvargiya, S. Modeling of software project effort estimation: a comparative performance evaluation of optimized soft computing-based methods. Int. j. inf. tecnol. 14, 2487–2496 (2022). https://doi.org/10.1007/s41870-022-00962-5
4. K Nitalaksheswara Rao, Jhansi Vazram Bolla, Satyanarayana Mummana et al. OLCE: Optimized Learning-based Cost Estimation for Global Software Projects, 06

September 2022, PREPRINT (Version 1) available at Research Square https://doi.org/10.21203/rs.3.rs-2024296/v1

5. Ramya, P., Sai Mokshith, M., Abdul Rahman, M., Nithin Sai, N. (2023). Software Development Estimation Cost Using ANN. In: Ogudo, K.A., Saha, S.K., Bhattacharyya, D. (eds) Smart Technologies in Data Science and Communication. Lecture Notes in Networks and Systems, vol 558. Springer, Singapore. https://doi.org/10.1007/978-981-19-6880-8_23