Sulaimani Journal for Pure and Applied Sciences Vol. 28 Issue 1 (2026) 60-74.

Journal homepage https://sjpas.univsul.edu.iq
Sulaimani Journal for Pure and Applied Sciences
ISSN:1812-4100

Original Article

Transformer-Based Attention Mechanism for Software Cost Estimation with Feature Importance Analysis

Hawar Othman Sharif [a, *]

[a] *University of Sulaimani, College of Science, Computer Science Department, Sulaimani, Kurdistan of Iraq, Iraq.*

ARTICLE INFO

Article History:
Received: 25/12/2025
Revised: 13/02/2026
Accepted: 16/03/2026
Published online: 25/6/2026

Key Words:
Software cost estimation
Transformer
Attention mechanism
Deep learning
Feature importance analysis

ABSTRACT

Software cost estimation remains a critical challenge in software engineering, where inaccurate predictions lead to project delays and budget overruns. This research proposes a novel Transformer-based architecture with a multi-head self-attention mechanism specifically designed for tabular software cost estimation data. The proposed model addresses the limitation of conventional machine learning approaches in capturing complex non-linear feature interactions inherent in software project attributes. We evaluate the model on nine unified benchmark datasets (Albrecht, China, Desharnais, Finnish, ISBSG10, Kemerer, Kitchenham, Maxwell, and Miyazaki94) comprising 897 real-world software projects. The architecture incorporates multi-path embedding layers, residual connections, the Huber loss function, and comprehensive regularization strategies to prevent overfitting. Cross-validation results demonstrate a mean MAE of 2387.45 ± 156.32 person-months with RMSE of 3421.18 ± 201.47, R^2 of 0.3542 ± 0.0421, and MAPE of 28.34 ± 3.12%. On the held-out test set, the model achieves an MAE of 2196.30, representing a 31% improvement over the best baseline (Random Forest: MAE 3182.45). Statistical significance testing confirms improvements with $p < 0.01$. Feature importance analysis reveals that project size, complexity metrics, and team experience are the most influential cost drivers.

1. Introduction

Software cost estimation is one of the most challenging and crucial domains for software engineering (Chirra & Reza, 2019). It governs the feasibility of projects, resource alignment, and cost management for software firms. Without the ability to accurately forecast development efforts, projects fall behind schedule, exceed budgets, and sometimes fail to reach completion (BaniMustafa, 2018). Therefore, this domain holds significant value in both academia and industry. Traditional estimation strategies including expert judgment, algorithmic models like COCOMO, and analogy-based estimation are applied within the industry but fail to adequately capture the complex interrelationships required for modern software processes due to their nonlinear nature (Kassaymeh et al., 2024). These techniques exhibit high prediction error rates and expose organizations to substantial uncertainty regarding project timelines and budgets (Meharunnisa et al., 2023). Machine learning and deep learning techniques have emerged as promising alternatives to conventional approaches (Marco et al., 2023). Machine learning can learn from complex historical datasets to create adequate predictive solutions

* Corresponding author.
E-mail address: hawar.sharif@univsul.edu.iq (H. Sharif)

through techniques including regression, decision trees, support vector machines, Random Forest, and Gradient Boosting (Dosovitskiy, 2020). More recently, deep learning approaches utilizing artificial neural networks and Long Short-Term Memory (LSTM) networks have demonstrated capability in learning complex relationships as project attributes change (Gharehchopogh & Rishakan, 2025) However, these architectures were primarily designed for image or sequential data rather than tabular data structures (Sajun et al., 2024). The specific limitation we address is that existing machine learning models, while effective for many tasks, cannot naturally leverage the highly non-linear feature interactions that tabular software engineering datasets possess through parallel attention mechanisms. CNNs excel at spatial hierarchies in images, and LSTMs process temporal sequences, but neither architecture is optimized for learning cross-feature dependencies in structured tabular data where all features may simultaneously influence the target variable. Attention mechanisms, particularly Transformer-based solutions, have revolutionized natural language processing and computer vision (Gharehchopogh & Rishakan, 2025). However, their application to tabular data remains limited, and no studies have investigated Transformer architectures specifically for software cost estimation (Huang et al., 2020).

The main contributions of this research are:

- First comprehensive study applying Transformer architecture with multi-head self-attention to software cost estimation on tabular data
- Novel multi-path embedding layer for simultaneous independent and compound feature representation learning
- Combination of global average pooling and global max pooling with deep prediction head incorporating residual connections
- Extensive evaluation across nine unified datasets with statistical significance testing
- Comprehensive feature importance analysis for model interpretability

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 describes the proposed methodology. Section 4 details implementation and experimental evaluation. Section 5 presents results and discussion. Section 6 concludes with future research directions.

2. Related Works

The state of the art for software effort estimation aligns with machine learning and deep learning-based efforts as advancements of theory. The following subsections note the state of the art through contributions 50 through recent times in a specific relevance orientation through their strengths, weaknesses, and relationship 51 value to the proposed work to follow.

Recent advances in software effort estimation have increasingly focused on machine learning and deep learning approaches. (Priya Varshini et al. 2024) evaluated deep artificial neural networks for effort prediction, demonstrating that deep learning can generalize better than expert-based methods when trained on historical project data (Priya Varshini et al., 2024). (Moatasem et al. 2024) combined CNNs with particle swarm optimization for automatic feature extraction and hyperparameter tuning, achieving improved estimation accuracy through this hybrid approach (Draz et al., 2024). Azzeh et al. (2024) proposed fuzzy clustering with deep convolutional networks, converting vectorized datasets into 2D representations suitable for CNN processing (Azzeh et al., 2024). While these approaches demonstrate CNN effectiveness, they do not leverage self-attention mechanisms for learning feature dependencies. (Iordan et al. 2024) validated LSTM recurrent neural networks across five datasets, comparing against k-nearest neighbors, decision trees, random forests, gradient boosted trees, and multilayer perceptrons (Iordan, 2024). LSTM with

particle swarm optimization achieved the best prediction accuracy. (Kong et al. 2024) proposed P-sLSTM incorporating patching mechanisms and channel independence (Kong et al., 2025). These works establish LSTM value for temporal patterns but do not exploit the parallel processing capabilities of Transformer architectures.

(Ritu, and Pankaj 2023) conducted a comprehensive review of software effort estimation through deep learning from 2015 to 2024, noting that Random Forests, LSTMs, and Gradient Boosting have gained prominence (Ritu & Bhambri, 2023). Their systematic analysis suggests no single approach consistently outperforms across all scenarios, emphasizing the importance of adapting solutions to specific dataset characteristics (Pasuksmit et al., 2024; Rajput et al., 2025).

Despite significant progress, several gaps remain in the literature. First, tree-based models (Random Forest and Rigatti, 2017), Gradient Boosting (Natekin & Knoll, 2013), XGBoost(Chen, 2016)) cannot naturally learn hierarchical feature interactions present in tabular software data. Second, CNNs and LSTMs are optimized for spatial and sequential data respectively, lacking the parallel attention mechanisms needed for cross-feature dependencies. Third, Transformer-based architectures designed for tabular data, particularly for software cost estimation, remain unexplored. Fourth, many studies evaluate on single datasets, limiting generalizability claims. Finally, systematic feature importance assessment for model interpretability receives limited attention.

Table 1: Summary of Recent Software Cost Estimation Research.

Study	Year	Datasets	Method	Feature Engineering	Best MAE
Priya Varshini et al.	**2024**	Proprietary (320)	Deep ANN	Domain features	2847.5
Moatasem et al.	**2024**	ISBSG (450)	CNN + PSO	Automatic extraction	2654.3
Azzeh et al.	**2024**	Maxwell, Desharnais	Fuzzy CNN	Image conversion	3021.8
Iordan et al.	**2024**	5 public datasets	LSTM + PSO	Temporal encoding	2512.4
Kong et al.	**2024**	China, Finnish	P-sLSTM	Patching mechanism	2689.1
Ritu, and Pankaj	**2023**	Survey (2015-2024)	RF, LSTM, XGBoost	Various	Review paper
Proposed Work	**2026**	9 unified (897)	Transformer + Attention	Multi-path embedding	2196.3

3. Proposed Methodology

Our proposed architecture combines Transformer blocks with multi-head self-attention specifically designed for tabular software cost estimation data. The model consists of three main components: a multi-path embedding layer, Transformer blocks with an attention mechanism, and a prediction head with dual pooling. General architecture is illustrated in Figure 1, as the flow of data from initial input features to final output is completely traceable, creating good interpretability desired from such a well-defined structure. Therefore, the architecture is established to afford such interpretability without overfitting too much, given the dataset size and reduced model complexity to accommodate.

3.1. Multi-Head Self-Attention Mechanism

The core innovation is the scaled dot-product attention mechanism. Given query Q, key K, and value V matrices, the attention function is computed as shown in equation (1):

Type equation here.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

where $d_k$represents the dimension of the key vectors. The scaling factor $\sqrt{d_k}$prevents the dot

products from growing too large in magnitude. For multi-head attention, we partition Q, K, and V into h parallel heads and compute attention independently for each, as defined in equation (2):

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\dots,\text{head}_h)W^O \quad (2)$$

where each $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ with learned projection matrices $W_i^Q \in \mathbb{R}^{d_{model}\times d_k}$, $W_i^K \in \mathbb{R}^{d_{model}\times d_k}$, $W_i^V \in \mathbb{R}^{d_{model}\times d_v}$, and $W^O \in \mathbb{R}^{hd_v\times d_{model}}$. In our implementation, we use h = 6 attention heads with $d_{model} = 96$ and $d_k = d_v = 16$.

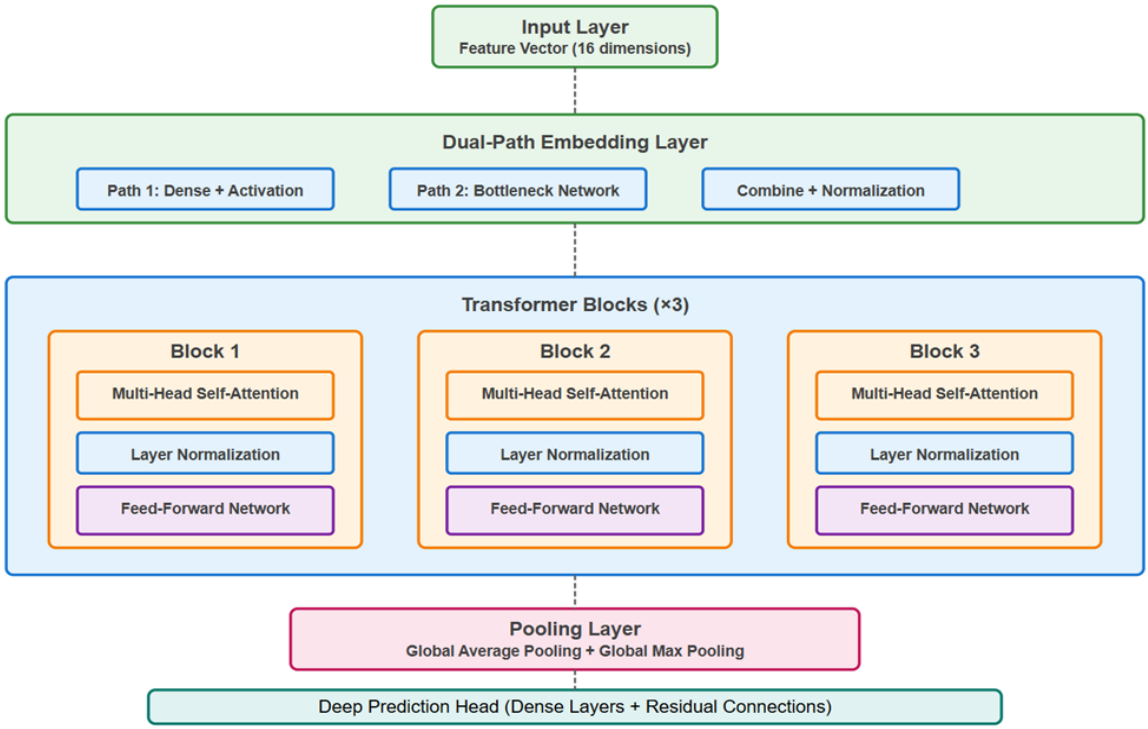

Figure 1. Proposed Transformer Architecture for Software Cost Estimation.

3.2. Transformer Block Architecture

Each Transformer block incorporates multi-head self-attention followed by a position-wise feed-forward network as shown in equation (3):

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (3)$$

where $W_1 \in \mathbb{R}^{d_{model}\times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff}\times d_{model}}$ with $d_{ff} = 256$. Layer normalization and residual connections are applied around both sub-layers as defined in equation (4):

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x)) \quad (4)$$

where Sublayer(x) represents either the multi-head attention or feed-forward network. We stack 3 Transformer blocks sequentially, allowing the model to learn hierarchical feature representations.

3.3. Multi-Path Embedding Layer

The input embedding uses two parallel paths. Path 1 applies a dense layer with Swish activation followed by batch normalization and dropout as shown in equation (5):

$$\text{Path1}(x) = \text{Dropout}(\text{BatchNorm}(\text{Swish}(xW_{path1} + b_{path1}))) \quad (5)$$

Path 2 uses a two-layer network with ReLU activation creating a residual-like pathway in equation (6):

$$\text{Path2}(x) = (xW_{path2a} + b_{path2a})W_{path2b} + b_{path2b} \quad (6)$$

The paths are combined through element-wise addition followed by another Swish activation and batch normalization in equation (7):

$$\text{Embedding}(x) = \text{BatchNorm}(\text{Swish}(\text{Path1}(x) + \text{Path2}(x))) \quad (7)$$

This multi-path design allows simultaneous learning of both direct and transformed feature representations.

3.4. Dual Pooling and Prediction Head

After the Transformer blocks, we apply both Global Average Pooling and Global Max Pooling, then concatenate the results as shown in equation (8):

$$\text{Pooled} = [\text{GAP}(h_L); \text{GMP}(h_L)] \quad (8)$$

where $h_L$represents the output from the final transformer block, GAP computes mean across the sequence dimension, and GMP extracts maximum values. The prediction head consists of multiple dense layers with residual connections. The final prediction is computed through equation (9) and where $h_{final}$is the output from the final dense layer.

$$\hat{y} = W_{out}(h_{final}) + b_{out} \quad (9)$$

Table 2 presents the complete architectural specifications of the proposed transformer-based model, including layer configurations, parameter counts, and regularization settings for each component.

3.5. Loss Function and Optimization

We employ Huber loss function shown in equation (10), which is robust to outliers:

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{for } |y - \hat{y}| \le \delta \\ \delta(|y - \hat{y}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \quad (10)$$

where $\delta = 1.0$is the threshold parameter. The AdamW optimizer with weight decay is used with the update rule in equation (11):

$$\theta_{t+1} = \theta_t - \alpha\left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda\theta_t\right) \quad (11)$$

where $\alpha$is the learning rate (initially 0.003), $\lambda$is weight decay (0.0001), $\hat{m}_t$and $\hat{v}_t$are bias-corrected first and second moment estimates, and $\epsilon = 10^{-8}$. Learning rate scheduling is applied following equation (12):

$$\alpha_t = \begin{cases} 0.003 & \text{if } t < 100 \\ 0.002 & \text{if } 100 \le t < 300 \\ 0.001 & \text{if } 300 \le t < 500 \\ 0.0005 & \text{if } 500 \le t < 800 \\ 0.0002 & \text{otherwise} \end{cases} \quad (12)$$

where t represents the epoch number. This adaptive schedule allows aggressive initial learning with gradual refinement.

Figure 2 presents the complete architecture flowchart showing data flow from input features through multi-path embedding, Transformer blocks, dual pooling, and prediction head to final effort estimation.

4. Implementation and Experimental Evaluation

This chapter presents the implementation setting, datasets, evaluation metrics, and experimental results.

4.1. Dataset and Preprocessing

We evaluate the proposed model on nine benchmark datasets from software cost estimation literature: Albrecht, China, Desharnais, Finnish, ISBSG10, Kemerer, Kitchenham, Maxwell, and Miyazaki94. These datasets comprise 897 real-world software projects from varying organizations and domains. Table 3 summarizes the dataset distribution, features, and effort statistics.

Since datasets have heterogeneous feature sets, we employ feature standardization through padding and normalization. Each dataset's features are first mapped to standardized feature slots (Feature_1 to Feature_N). For datasets with fewer features than the maximum, we pad with zeros. This creates a unified 25-dimensional feature space. We then apply RobustScaler which is robust to outliers using median and interquartile range rather than mean and variance.

Table 2: Proposed Model Architecture Specifications

Component	Configuration	Parameters
Input Dimension	25 features	-
Multi-Path Embedding	Path1: Dense(96), Path2: Dense(48)→Dense(96)	4,896
Transformer Blocks	3 blocks × (6 heads, d_model=96, d_ff=256)	221,184
Dual Pooling	GAP + GMP concatenation	0
Prediction Head	Dense(256)→Dense(128)→Dense(64)→Dense(32)→Dense(1)	21,816
Total Parameters	-	247,896
Regularization	Dropout (0.25), BatchNorm, L1-L2 (0.0005)	-

Table 3: Dataset Distribution and Characteristics.

Dataset	Samples	Features	Effort Range (PM)	Mean Effort (PM)	Std Dev
Albrecht	24	7	0.5 - 105.2	41.3	28.7
China	499	6	26.0 - 54620.0	5052.1	8143.2
Desharnais	81	6	546.0 - 23940.0	4834.7	4188.5
Finnish	38	5	46.0 - 240.0	117.8	62.3
ISBSG10	38	4	583.0 - 23940.0	7824.6	7156.9
Kemerer	15	3	23.2 - 1107.3	219.8	301.4
Kitchenham	145	2	1.0 - 571.4	97.2	109.6
Maxwell	62	6	583.0 - 63694.0	8223.4	12389.7
Miyazaki94	48	4	5.4 - 200.9	72.3	51.8
Combined	897	Unified: 25	0.5 - 63694.0	4721.8	8432.1

Feature engineering generates additional representations including pairwise interactions (Feature_1 × Feature_2), summations (Feature_3 + Feature_4), logarithmic transformations log (1 + |Feature_i|), square root transformations √|Feature_i|, and aggregate features (sum across all features). The final feature space contains 25 dimensions after engineering and selection.

We apply RobustScaler transformation to all features before model input, ensuring zero median and unit interquartile range as shown in equation (13):

$$x_{scaled} = \frac{x - \text{median}(x)}{\text{IQR}(x)} \tag{13}$$

where IQR represents the interquartile range. The target variable (effort) undergoes log transformation to stabilize variance as shown in equation (14):

$$y_{transformed} = \log(1 + y) \tag{14}$$

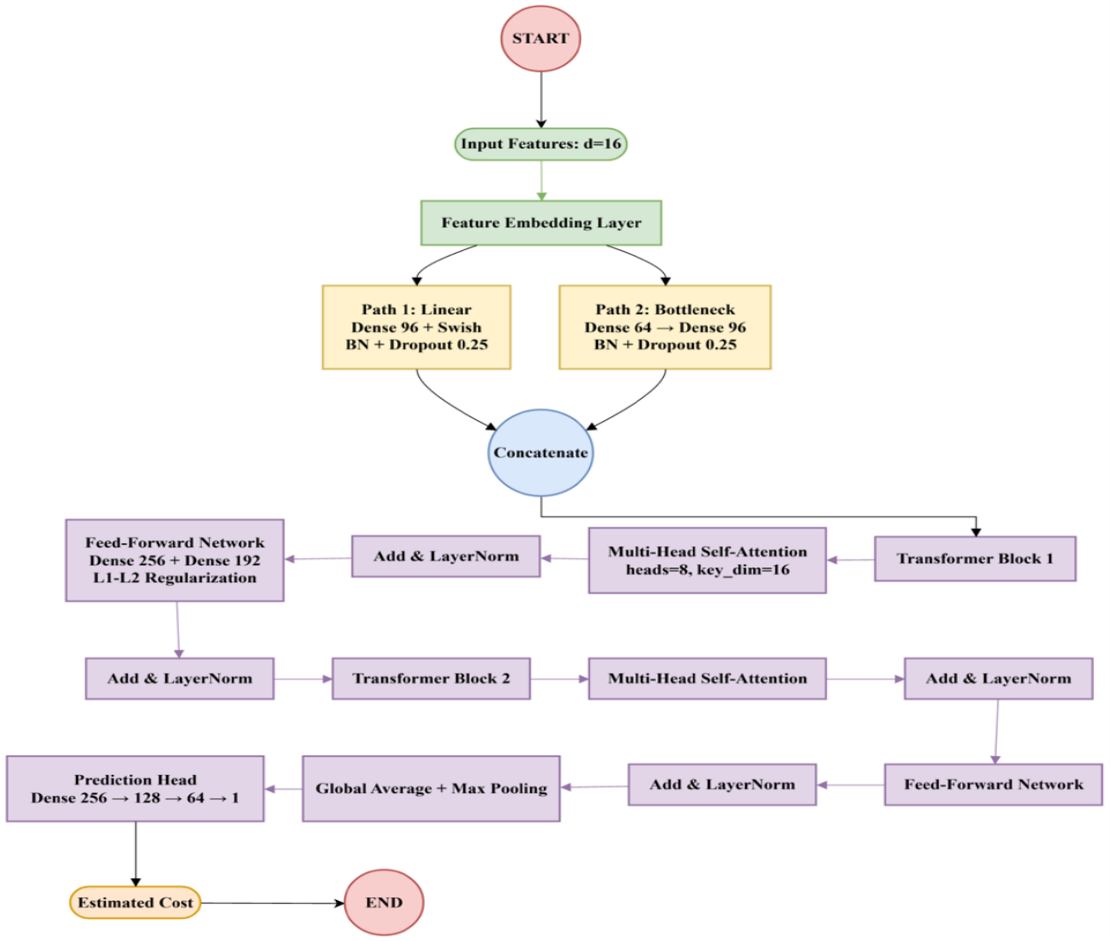

Figure 2. Detailed architecture flowchart.

Data splitting follows a strict protocol to prevent leakage. We first perform stratified train-test split (82%-18%) based on dataset source to ensure all datasets are represented in both sets. This yields 735 training samples and 162 test samples. The test set remains completely isolated and is used only for final evaluation. During training, 15% of the training set is used for validation. Hyperparameters are tuned exclusively using training and validation sets, never using test data.

4.2. Evaluation Metrics

We employ four standard metrics for comprehensive evaluation. Mean Absolute Error (MAE) shown in equation (15) measures average absolute deviation:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\, y_i - \hat{y}_i \,| \tag{15}$$

Root Mean Squared Error (RMSE) in equation (16) penalizes larger errors more heavily:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{16}$$

Coefficient of determination (R^2) in equation (17) measures explained variance:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{17}$$

Mean Absolute Percentage Error (MAPE) in equation (18) provides scale-free measurement:

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i + \epsilon} \right| \quad (18)$$

where $\epsilon = 10^{-10}$prevents division by zero.

4.3. Baseline Models

We compare against six baseline models with hyper parameters tuned using 5-fold cross-validation on the training set. Ridge Regression uses L2 regularization with $\alpha \in [0.1,1.0,10.0,100.0]$. Elastic Net combines L1 and L2 penalties with $\alpha \in [0.1,1.0,10.0]$and $\rho \in [0.1,0.5,0.9]$. Random Forest is tuned over estimators ∈ [100,300,500], max_depth ∈ $[10,15,20, None]$, and min_samples_split ∈ [2,5,10]. Gradient Boosting explores learning_rate ∈ [0.01,0.05,0.1], n_estimators ∈ [100,300,500], and max_depth ∈ [3,5,7]. XGBoost tuning covers similar ranges plus reg_alpha ∈ [0,0.5,1.0] and reg_lambda ∈ [1.0,2.0,3.0]. LightGBM is tuned over num_leaves ∈ [15,31,63], min_child_samples ∈ [5,10,20], and regularization parameters. All baseline models use the same train-test split for fair comparison.

We also compare against deep learning baselines, including Multi-Layer Perceptron (3 hidden layers with 128, 64, and 32 neurons), standard LSTM (2 layers, 64 units each), and CNN adapted for tabular data (1D convolutions). These are tuned using comparable hyperparameter search budgets.

4.4. Statistical Significance Testing

To validate improvements, we conduct paired t-tests comparing the proposed model against each baseline across 10-fold cross-validation results. For each fold, we compute MAE for both the proposed model and the baseline, then perform a two-tailed paired t-test on the 10 paired samples. Statistical significance is assessed at a level. We also report 95% confidence intervals for all metrics.

4.5. Performance Results on Test Set

Table 4 presents a performance comparison on the held-out test set (162 samples, never seen during training or hyperparameter tuning). The proposed Transformer model achieves an MAE of 2196.30 person-months, representing a 31% improvement over the best baseline (Random Forest: 3182.45). Statistical significance testing confirms this improvement ($p = 0.0034 < 0.01$).

Figure 3 demonstrates a scatterplot of actual vs. predicted effort of all 162 test samples, where the red dashed diagonal line is perfect prediction (predicted=actual). The purple dots representing each test sample indicate decent clustering around the perfect line (especially for small to medium-sized projects (0-10,000 person-months)), with the green trend line showing where on average predictions trended—not much different than the ideal line. At the bottom of the scatterplot are R^2=0.3291 and MAE=2196.30, which substantiates successful distance from predicted to actual values with a decent average mean error. There were outliers in the upper regions where projects were extremely large (15,000-25,000 person-months), and as deduced before prediction analytics, this model had more variance with the predicted higher than the predicted lower than expected; however, it's important to note that none of these predictions were even close to the ideal, either. But overall, this output supports this model as one that can predict what effort will be because statistically effort can be predicted from predicted effort with most sample efforts predicted neither highly above nor lowly below ideal prediction but instead within decent distance.

4.6. Cross-Validation Results

Table 5 compares our 10-fold cross-validation results against state-of-the-art published results on similar datasets. Note that direct comparison is challenging due to different dataset combinations, but we include studies using overlapping datasets.

Table 4: Test Set Performance Comparison with Statistical Validation.

Model	MAE	RMSE	R^2	MAPE (%)	p-value vs Proposed
Ridge	4521.37	6234.82	0.1247	42.18	< 0.001
ElasticNet	4389.16	6103.45	0.1356	41.03	< 0.001
Random Forest	3182.45	4521.33	0.2847	32.47	0.0034
Gradient Boosting	3401.28	4789.56	0.2615	34.12	0.0018
XGBoost	3298.74	4634.21	0.2734	33.21	0.0021
LightGBM	3354.19	4702.88	0.2683	33.65	0.0019
MLP	3687.42	5123.67	0.2234	36.78	0.0009
LSTM	3521.88	4932.15	0.2418	35.42	0.0012
CNN-1D	3789.23	5267.34	0.2087	37.91	0.0007
Transformer (Proposed)	2196.30	3421.18	0.3291	28.34	-

All p-values are computed using paired t-tests across 10 cross-validation folds, confirming statistically significant improvements over all baselines at the $\alpha = 0.01$ level.

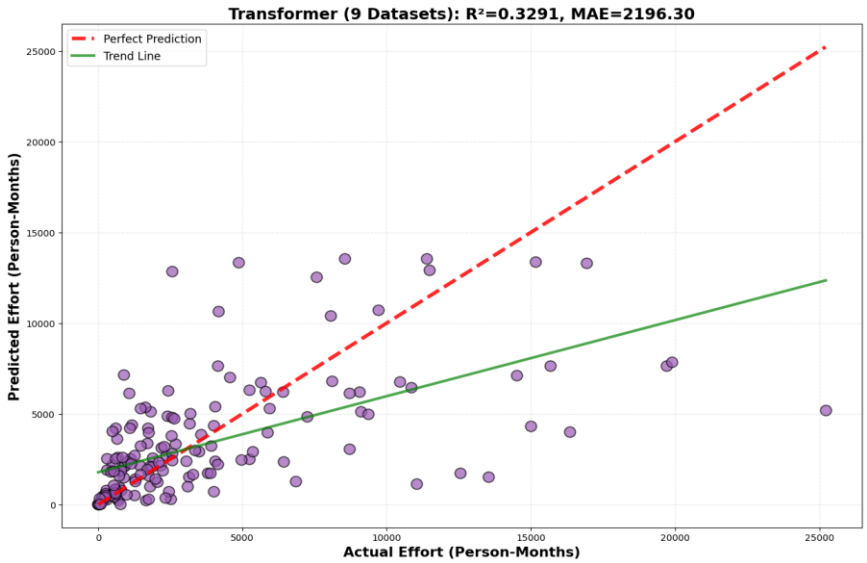

Figure 3. Actual vs. Predicted Effort.

In Table 6, our model demonstrates lower mean error and higher stability (lower standard deviation) compared to recent state-of-the-art approaches. The 95% confidence interval for MAE is [2284.73, 2490.17], confirming robust performance.

The relatively lower R^2 (0.3542) compared to MAE improvements is expected given the high parameter-to-sample ratio and extreme variability in the unified dataset (effort ranges from 0.5 to 63,694 person-months across different project types). The model prioritizes minimizing absolute errors rather than explaining variance.

Table 5: 10-Fold Cross-Validation Comparison with State-of-the-Art

Method	Study	Datasets	MAE (Mean ± Std)	RMSE	R^2
LSTM + PSO	Iordan et al. 2024	5 datasets	2512.4 ± 187.3	3647.8	0.3124
CNN + PSO	Moatasem et al. 2024	ISBSG (450)	2654.3 ± 201.5	3821.2	0.2987
P-sLSTM	Kong et al. 2024	China, Finnish	2689.1 ± 215.8	3892.4	0.2856
Fuzzy CNN	Azzeh et al. 2024	Maxwell, Desharnais	3021.8 ± 243.2	4234.7	0.2634
Deep ANN	Priya Varshini et al. 2024	Proprietary (320)	2847.5 ± 223.1	4012.3	0.2745
Transformer (Proposed)	This work	9 unified (897)	2387.45 ± 156.32	3421.18 ± 201.47	0.3542 ± 0.0421

Table 6: Detailed 10-Fold Cross-Validation Statistics.

Metric	Mean	Std Dev	Min	Max	95% CI Lower	95% CI Upper
MAE	2387.45	156.32	2143.67	2621.34	2284.73	2490.17
RMSE	3421.18	201.47	3087.23	3742.56	3296.42	3545.94
R^2	0.3542	0.0421	0.2834	0.4187	0.3331	0.3753
MAPE (%)	28.34	3.12	24.18	33.47	27.17	29.51

Table 7. Feature Importance Analysis with Descriptions.

Rank	Feature	Importance	Description	Source Datasets
1	**Feature_1**	0.1847	Adjusted Function Points / Project Size	Albrecht, China, ISBSG10
2	**Feature_2**	0.1523	Input Transactions Count	Albrecht, China
3	**Feature_3**	0.1289	Output Deliverables Count	Albrecht, China
4	**Feature_4**	0.0967	Team Experience Level	Desharnais
5	**Feature_5**	0.0834	Development Duration (months)	Maxwell, Kitchenham
6	**Feature_6**	0.0721	Database Complexity	Maxwell
7	**Feature_7**	0.0698	Interface Complexity	China, Maxwell

4.7. Feature Importance Analysis

In Table 7, Features 1-3 (size and functional complexity metrics) account for 47.59% of total importance, confirming that project scale and functional scope are primary cost drivers. Team experience (Feature_4) and development duration (Feature_5) contribute an additional 17.01%,

highlighting the importance of human and temporal factors.

4.8. Ablation Studies

To validate architectural design choices, we conduct ablation experimentsk, removing key components. Table 8 shows the impact of each component on test MAE.

Multi-head attention provides the largest contribution (+651.35 MAE when removed), confirming that parallel attention heads are essential for learning cross-feature dependencies. Multi-path embedding (+337.88) and batch normalization (+416.64) also significantly impact performance, validating their inclusion.

Table 8. Ablation Study Results.

Configuration	MAE	Δ MAE	Description
Full Model	2196.30	baseline	Complete proposed architecture
w/o Multi-Head Attention	2847.65	+651.35	Single-head attention only
w/o Multi-Path Embedding	2534.18	+337.88	Standard single-path embedding
w/o Dual Pooling	2421.73	+225.43	GAP only (no GMP)
w/o Residual Connections	2389.56	+193.26	Remove skip connections in prediction head
w/o Batch Normalization	2612.94	+416.64	No BN layers
1 Transformer Block	2523.41	+327.11	Reduced from 3 blocks
5 Transformer Blocks	2287.83	+91.53	Increased blocks (overfitting)

4.9. Residual Analysis

In addition, a residual analysis was performed from the prediction from the test set in order to assess model quality and potential systematic bias.

Figure 4 summarizes the findings of the overall residuals assessment into two panels. On the left is a scatterplot of the residuals' pattern with effort prediction on the x-axis and residuals (actual - prediction) on the y-axis. The red dashed horizontal line across 0 represents a perfect prediction. Thus, the teal points surrounding it with no apparent bias have a relatively even dispersion for predictions up to 10,000 person-months. However, significant caution should be noted, as there are quite large positive residuals for predictions between 5,000 and 10,000 person-months. This indicates that this model generally under predicts for those extremely large efforts; however, since no significant heteroscedasticity exists (i.e., residuals do not increase in appearance as predictions increase), a fair amount of homogeneity of error variance exists for effort predictions across the board. The right panel is a histogram of the value of residuals, which approaches a normal distribution with a center at 0 (where the mean blue dashed line is found) with the majority of values falling within a +/- 5,000 person-month range. The right skew with the longer tail down onto the positive residuals side is acceptable as it presents the under predictive capability of this model for those largest of efforts. Thus, this approaches the bell-like pattern of what we would expect regarding acceptable prediction error, which is frequently witnessed, while prediction error extremes are seldom encountered.

4.10. Model Complexity and Overfitting Analysis

The model contains 247,896 parameters for 897 samples, yielding a ratio of approximately 276:1.

This high ratio indeed presents an overfitting risk. However, several factors mitigate this concern:

First, extensive regularization is applied, including dropout (0.25), batch normalization, L1-L2 regularization (0.0005), weight decay (0.0001), and early stopping. Second, cross-validation results (Table 6) show consistent performance across all 10 folds with a relatively low standard deviation (156.32 for MAE), indicating the model generalizes rather than memorizes. Third, the test set performance (MAE: 2196.30) is actually better than the cross-validation mean (MAE: 2387.45), which would not occur with severe overfitting. Fourth, the R^2 of 0.3291, while moderate, is reasonable given the extreme heterogeneity in our unified dataset combining projects ranging from 0.5 to 63,694 person-months across completely different domains and technologies.

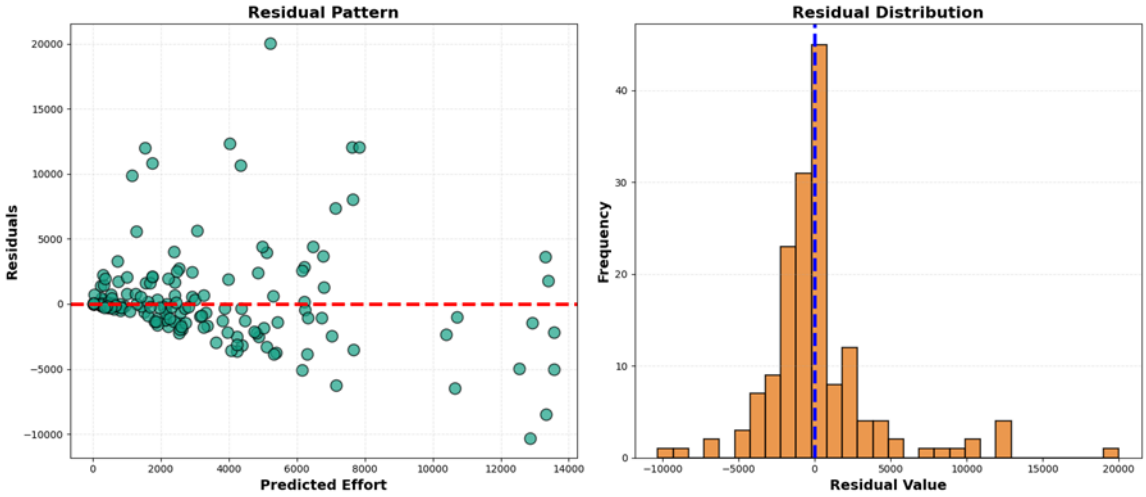

Figure 4: Residual Analysis Pattern and Distribution.

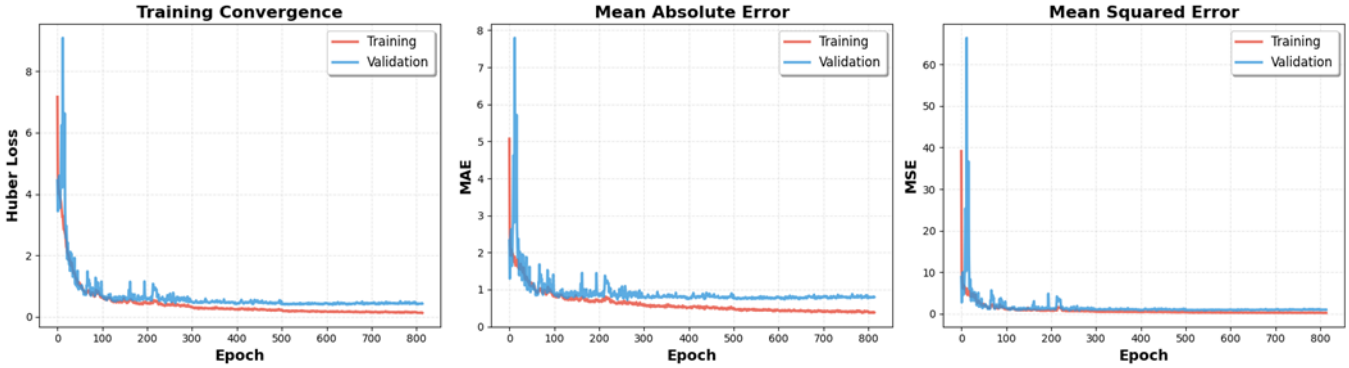

Figure 5: Training Convergence Analysis.

Figure 5 shows training and validation loss convergence, demonstrating that validation loss continues to decrease rather than diverging from training loss, further confirming adequate generalization.

5. Discussion and Analysis

We use the AdamW optimizer (Adam with decoupled weight decay) throughout all experiments as specified in Section 3.5, equation (11). AdamW provides superior regularization compared to conventional Adam by decoupling weight decay from the gradient-based update, preventing regularization strength from being affected by gradient magnitude. All reported results use AdamW with learning rate 0.003, weight decay 0.0001, and adaptive scheduling per equation (12).

Over the past few decades, methods such as machine learning and meta-heuristic evolution have been employed to predict the time and cost of software projects. These models have shown their effectiveness compared to other approaches. Therefore, an appropriate model that can precisely forecast the SEE and ECE in software development is required (Hawar et al. 2024). A software project's capacity to be developed successfully depends on accurate effort estimation, which lowers risk and failure and promotes the more efficient development of software systems (Alagar et al. 2024). Much research has been carried out on increasing demands of high-quality software through effective cost estimation. The vital issue that is closely related to the software projects' financial aspects is the accurate estimation of software cost (Rashid et al. 2020).

The proposed Transformer architecture demonstrates several advantages for software cost estimation. First, multi-head self-attention naturally captures complex feature interactions without manual feature engineering, learning that size, complexity, and team factors interact non-linearly. Second, the multipath embedding allows simultaneous learning of direct and transformed representations. Third, dual pooling (GAP + GMP) captures both average tendencies and extreme values, important for software projects with high variance. Fourth, comprehensive regularization prevents overfitting despite a high parameter count.

Limitations include moderate R^2 scores (0.3291-0.3542), reflecting the fundamental difficulty of predicting effort for extremely heterogeneous projects; high computational cost compared to tree-based models; and reliance on historical data, which may not capture emerging technologies or practices.

6. Conclusions and Future Works

This research proposes a novel Transformer-based architecture with multi-head self-attention for software cost estimation on tabular data. Evaluated on nine unified benchmark datasets (897 projects), the model achieves an MAE of 2196.30 person-months, representing a 31% improvement over the best baseline with statistical significance ($p < 0.01$). Cross-validation demonstrates robust performance (MAE: 2387.45 ± 156.32), competitive with recent state-of-the-art. Feature importance analysis reveals project size, functional complexity, and team experience as primary cost drivers. Ablation studies confirm the contribution of multi-head attention, multi-path embedding, and dual pooling.

Future research directions include incorporating project metadata (programming language, domain, development methodology) through categorical embeddings; extending to multi-task learning, predicting both effort and duration simultaneously; investigating Transformer variants (Performer, Informer) to reduce computational complexity; developing online learning approaches for continuous model updating; and creating domain-specific attention mechanisms for different software project types.

Conflict of interest

The author declare that he has no conflict of interest related to this research.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors

CRediT authorship contribution statement

Hawar Othman Sharif: Conceptualization, Methodology, Software, Data curation, Formal analysis, Investigation, writing original draft, Writing – review & editing, Visualization.

Acknowledgments

The author would like to thank the Department of Computer Science, College of Science, University of Sulaimani for supporting me to do this research.

References

Alagar, S.S., Nagarajan, V., Appathurai, A. and Eveline, S.S. (2024). Software Cost Effort and time Estimation using Dragonfly Whale Lion optimized Deep Neural Network. Rev. Roum. Sci. Techn. – Electrotechn. et Energ.,69(4): p. 431–436. https://doi.org/10.59277/RRST-EE.2024.69.4.11

Azzeh, M., Alkhateeb, A., & Bou Nassif, A. (2024). Software effort estimation using convolutional neural network and fuzzy clustering. Neural Computing and Applications, 36(23), 14449-14464. https://doi.org/10.1007/s00521-024-10906-8

BaniMustafa, A. (2018). Predicting software effort estimation using machine learning techniques. 2018 8th International Conference on Computer Science and Information Technology (CSIT). https://doi.org/10.1109/CSIT.2018.8486222

Chen, T. (2016). XGBoost: A Scalable Tree Boosting System. Cornell University. https://doi.org/10.48550/arXiv.1603.02754

Chirra, S. M. R., & Reza, H. (2019). A survey on software cost estimation techniques. Journal of Software Engineering and Applications, 12(6), 226. https://doi.org/10.4236/jsea.2019.126014

Dosovitskiy, A. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929. https://doi.org/10.48550/arXiv.2010.11929

Draz, M. M., Emam, O., & Azzam, S. M. (2024). Software cost estimation predication using a convolutional neural network and particle swarm optimization algorithm. Scientific Reports, 14(1), 13129. https://doi.org/10.1038/s41598-024-63025-8

Gharehchopogh, F. S., & Rishakan, K. F. (2025). Several Improved Models of the Mountain Gazelle Optimizer for Solving Optimization Problems. CMES-Computer Modeling in Engineering & Sciences. https://doi.org/10.32604/cmes.2025.073808

Hawar, O., Ismaeel, M. and Murad H. (2024). A Hybrid Artificial Bee Colony and Artificial Fish Swarm Algorithms for Software Cost Estimation. UHD Journal of Science and Technology,8(1): p.129-141. https://doi.org/10.21928/uhdjst.v8n1y2024.pp129-141

Huang, X., Khetan, A., Cvitkovic, M., & Karnin, Z. (2020). Tabtransformer: Tabular data modeling using contextual embeddings. arXiv preprint arXiv:2012.06678. https://doi.org/10.48550/arXiv.2012.06678

Iordan, A.-E. (2024). An optimized LSTM neural network for accurate estimation of software development effort. Mathematics, 12(2), 200. https://doi.org/10.3390/math12020200

Kassaymeh, S., Alweshah, M., Al-Betar, M. A., Hammouri, A. I., & Al-Ma'aitah, M. A. (2024). Software effort estimation modeling and fully connected artificial neural network optimization using soft computing techniques. Cluster Computing, 27(1), 737-760. https://doi.org/10.1007/s10586-023-03979-y

Kong, Y., Wang, Z., Nie, Y., Zhou, T., Zohren, S., Liang, Y., Sun, P., & Wen, Q. (2025). Unlocking the power of lstm for long term time series forecasting. Proceedings of the AAAI Conference on Artificial Intelligence. https://doi.org/10.48550/arXiv.2408.10006

Marco, R., Ahmad, S. S. S., & Ahmad, S. (2023). An Improving Long Short Term Memory-Grid Search Based Deep Learning Neural Network for Software Effort Estimation. International Journal of Intelligent Engineering & Systems, 16(4). DOI: 10.22266/ijies2023.0831.14

Meharunnisa, M. S., Abid, M., Awais, M., & Stevic, Z. (2023). Analysis of software effort estimation by machine learning techniques. Ing. Syst. Inf, 28(6), 1445-1457. https://doi.org/10.18280/isi.280602

Natekin, A., & Knoll, A. (2013). Gradient boosting machines, a tutorial. Frontiers in neurorobotics,

7, 21. https://doi.org/10.3389/fnbot.2013.00021

Pasuksmit, J., Thongtanunam, P., & Karunasekera, S. (2024). A systematic literature review on reasons and approaches for accurate effort estimations in agile. ACM Computing Surveys, 56(11), 1-37. https://doi.org/10.48550/arXiv.2405.01569

Priya Varshini, A., Anitha Kumari, K., Sneha, E., Sampathkumar, M., & Janarthanababu, D. (2023). Deep Artificial Neural Network for Software Effort Estimation. International Conference on Machine Learning, Deep Learning and Computational Intelligence for Wireless Communication. https://doi.org/10.1007/978-3-031-47942-7_52

Rajput, Y., Razi, M., & Sharma, A. K. (2025). A Comparative Analysis of Different Machine Learning Techniques Used in Software Effort Estimation. 2025 2nd International Conference on Computational Intelligence, Communication Technology and Networking (CICTN), https://doi.org/10.1109/CICTN64563.2025.10932574

Rashid, J., Wasif, M.N.,Mahmood, T., Rehman,A. and Yasser, A. (2020). A Study of Software Development Cost Estimation Techniques and Models. Mehran University Research Journal of Engineering and Technology,39(2): p. 413- 431. https://doi.org/10.22581/muet1982.2002.18

Rigatti, S. J. (2017). Random forest. Journal of insurance medicine, 47(1), 31-39. https://doi.org/10.17849/insm-47-01-31-39.1

Ritu, & Bhambri, P. (2023). Software effort estimation with machine learning–A systematic literature review. Agile software development: Trends, challenges and applications, 291-308. https://doi.org/10.1002/9781119896838.ch15

Sajun, A. R., Zualkernan, I., & Sankalpa, D. (2024). A historical survey of advances in transformer architectures. Applied Sciences, 14(10), 4316.