LEX LOCALIS-JOURNAL OF LOCAL SELF-GOVERNMENT
ISSN:1581-5374 E-ISSN:1855-363X
VOL. 23, NO. 11(2025)

# INTELLIGENT PERFORMANCE OPTIMIZATION OF SQRT-CSLA ARCHITECTURE USING GRAPH NEURAL NETWORKS AND BEC LOGIC

**T. Velmurugan[1], V. Chandrasekaran[2]**

[1]Assistant professor, Department of Computer Science and Design,
Kongu Engineering College, Perundurai, Erode, Tamil Nadu – 638060.

[2]Professor & Head, Department of Medical Electronics,
Velalar College of Engineering and Technology, Thindal, Erode, Tamil Nadu-638012.

Corresponding email: ecevel@gmail.com[1]
vcresearch@gmail.com[2]

**Abstract**
This research introduces a data-aware deep learning optimization framework using Graph Neural Networks (GNNs) for enhancing the performance of the Square-Root Carry Select Adder (SQRT-CSLA) architecture in VLSI system design. Extending deterministic circuit-level optimizations, the proposed approach enables context-sensitive and input-driven optimization by learning complex interactions between circuit topology, switching activity, and performance metrics. The SQRT-CSLA was modeled at the gate level as a graph, where logic gates were represented as nodes and signal dependencies as edges, facilitating effective GNN-based learning. Post-synthesis simulation data, including power consumption, propagation delay, and switching activity, were employed to train the GNN to capture nonlinear relationships among structural and data-dependent characteristics of the circuit. Unlike conventional static or rule-based optimization techniques, the proposed method accurately identified switching-intensive paths and redundancy-prone logic regions that are difficult to detect through analytical analysis alone. Guided by GNN inference, adaptive optimization strategies such as selective logic pruning, gate resizing, and operand isolation were applied to critical carry paths and Binary-to-Excess-1 Converter (BEC) logic. This intelligent optimization process resulted in significant reductions in average power dissipation and improved delay uniformity under realistic operating conditions. Furthermore, the framework demonstrated strong scalability across different operand word lengths and technology nodes, highlighting its robustness and generalization capability. By embedding deep learning intelligence into the VLSI optimization workflow, the proposed approach advances energy-efficient and high-performance arithmetic circuit design.

**Keywords:** Graph Neural Networks; Data-Aware Optimization; VLSI Design; SQRT-CSLA; Power and Delay Optimization; Deep Learning Circuits

## 1. Introduction

The study of the tradeoff between performance and energy efficiency is under investigation at all times in the field of VLSI design. Particularly, the ever-growing law of Moore lets a greater and greater number of computational subunits be accommodated, which in large part raises area efficiency. Nevertheless, overloaded computations obviously hamper the margin of noise in the design of digital VLSI and consequently raise the switch current in any single data path, which is the chief factor to raise the dynamic power dissipation as well as the power density. The efficiency of the same computation can be achieved by the portable device in the same manner as in the traditional personal computers but is mostly limited by the life time of batteries and cooling methods . Large heat dissipation is a violation of the working properties of transistors slowing their carrier velocity in channels even in devices whose power supply is externally held constant. In this way, the permissible frequency of the digital systems in electronic devices is limited by the performance degradation by the transistor owing to the rise in temperature. Moreover, the worst scenario is that a VLSI chip that is operating under high temperature can easily break the whole die and its package. Therefore, in order to maintain VLSI chips safely operating at maximum frequency, it is expected that certain blocks within a chip must be disabled to ensure that the temperature does not go beyond the safe line [1].

In a novel nano design of a digital comparator based on QCA principles, Delay, area and energy efficiency measures are critically studied; from the simulations, it can be realized that the QCA comparator can perform much faster than the conventional designs [2]. Correctness in more than one of several different input combinations is checked, which highlights the merits of QCA for ultra-compact, low-power circuits as well

as verifying the viability of scaling to larger-bit comparisons. Performance Analysis of 16 bit Adders for High Speed Computing Applications. Delays, power consumption and area measures for various adder topologies were tested to prove that some structures result in best speed in big bit operations [3].
It also examined reversible ALU based on QCA technology. Power consumption, delay and total energy efficiency were tested, with results showing the use of reversible logic is able to significantly reduce energy dissipation [4] . Various ALU operations, such as addition, subtraction and logical functions, were tested, and this refreshing promise of reversible QCA ALUs for low- energy, nanoscale computing systems. The research have designed a QCA multiplexer based on the principles of cell interactions for complex circuit design[5]. Delay, area and power efficiency were examined; from simulations, improved operation reliability over conventional multiplexers was shown. Correctness evaluations with multiple input combinations are a way to validate the architecture to highlight which merits QCA for nano-circuits design and for the scalability level. The study can be concluded that cell-interaction-based multiplexers can be used to provide efficient and high speed computation [6].
Ultrafast all-optical ALU operation based on soliton control in microrings of cascaded InGaAsP/InP. Performance in terms of speed, reliability were evaluated, simulating indication that the optical ALU is better than conventional electronic units [7] . Soliton-based control influences of computational efficiency were studied, with a given focus to the possibility of optical circuits for ultra-fast computational. Validation through multiple logics proceeds ultimate feasibility, inspiring that all-optical ALUs may make next generation high speed computation attainable[8]. Speed, energy efficiency and operational reliability were evaluated, which revealed the significant increase in speed compared with the conventional methods. Nonlinear optical response analysis guarantees the precision of logic operations, which makes photonic crystal circuits favorable platforms for photonic computing. Testing the design under diverse operating conditions helps validate the design and supported in deployment in high speed optical processing systems [9].
Propagation delay, power consumption and area efficiency metrics were evaluated and simulations have confirmed superiority in terms of speed and decreased energy dissipation compared to traditional designs [10]. Trade-off between power, delay and area is the basis of design decisions, so it is clear why strategies for low-power considerations are important for high-speed digital applications. Reliability and correctness tests further anticipate integration for VLSI arithmetic systems for greater performance [11]. It also proposed a memory,-processor unit architecture based on SN introduction of SN in systems. Efficiency in implementing arithmetic and logic operations was analysed and it was the first step towards combining computation and memory in a unified architecture. Speed and energy benefits were measured, pointing to possible applications that require high speed access to memory [12].
It is presented an efficient design for random access memory with the help of QCA nanotechnology. Delay, area and power consumption were measured; it is observed from the simulation that memory access is faster and energy dissipation is less compared to the case of conventional arrays. Reliability under various input patterns was investigated and scalability issues for larger memory arrays were covered to reassert the potential of QCA for nano-scaled memory applications as well as energy efficient high-speed computing [13].

## 2. METHODOLOGY

### 2.1 Dataset Description

The efficacy of both analytical and deep learning driven optimizations of Square-Root Carry Select Adders (SQRT-CSLAs) is dependent upon the quality, representativeness and structural richness of the underlying data set. In contrast to the traditional benchmark-determent evaluation, the current investigation builds a purpose constructed data set that measures functional correctness, switching action and gate level performance traits under a variety of functioning situations. The data set has been specifically designed to simulated the engineering of realistic arithmetic workloads and to uncovers data dependencies (data-dependences) that are intrinsic to CSLA architectures. It takes into consideration functional input vectors, internal switching traces, gate-level power annotations, and timing metrics from post-synthesis simulations. This extensive data set forms the basis for the Phase I analytical optimization, Phase II for the deep learning-based learning and

inference and Phase III for the hybrid optimization which contributes to ensuring consistency and comparability of all experimental phases (Table 1).

**Table 1. Dataset Configuration Across Operand Bit-Widths**

| Bit-Width | Number of Test Vectors | Carry-In Conditions | Switching Activity Captured | Target Metrics |
|---|---|---|---|---|
| 8-bit | 10,000 | 0 and 1 | Yes | Power, Delay, Area |
| 16-bit | 20,000 | 0 and 1 | Yes | Power, Delay, Area |
| 32-bit | 40,000 | 0 and 1 | Yes | Power, Delay, Area |
| 64-bit | 60,000 | 0 and 1 | Yes | Power, Delay, Area |
| 128-bit | 80,000 | 0 and 1 | Yes | Power, Delay, Area |
| 256-bit | 100,000 | 0 and 1 | Yes | Power, Delay, Area |

2.2 **Input Vector Space Definition for CSLA Architectures**

The input vector space for the SQRT-CSLA dataset is formally defined to include operand bits and carry-in signals that fully characterize adder behavior. For an $n$-bit SQRT-CSLA, each input sample is represented as a triplet $(A, B, C_{in})$, where $A = \{a_{n-1}, a_{n-2}, \dots, a_0\}$, $B = \{b_{n-1}, b_{n-2}, \dots, b_0\}$, and $C_{in} \in \{0,1\}$. The total input space cardinality is thus $2^{2n+1}$; however, exhaustive enumeration is computationally infeasible for higher word lengths.

**2.3 Operand Bit-Width Variations and Test Case Enumeration**

In order to assess scalability and generality, the data set contains several operand bit widths that are commonly used in high performance and low power digital systems. The 8-bit, 16-bit, 32-bit and 64-bit word length SQRT-CSL architectures are explicit considered. For each bit Pint, a set number of test vectors are created prior to the sprint by using a systematic combination of random, corner case and correlated input patterns. The corner-case set contains all zero, all one, alternating bit patterns and inputs which induce intensity of carry propagation whereas the correlated patterns are aimed at simulating actual arithmetic workloads. This multi resolution data set thus allows the investigation of the optimization techniques devised for the technology in a range of different design scales, taking care to be robust in the technological sense.

**2.4 Carry-In Dependency Modeling and Dataset Stratification**

Carry-in dependency plays a dominating role in determining the setting up and propagation delays variability in CSLA architectures. In order to make this dependency very explicit, depending carry-in values and internal carry propagation characteristics are used to stratify the dataset. Separate subsets are maintained for Cin = 0 and Cin = 1 which ensures balanced learning of both the scenarios. In addition, samples are classified according to the carry propagation depth between the local (carry generation) and long (carries chains). This stratification eliminates dataset bias and enables the deep learning model to learn conditional relationships between carry behavior and performance metrics accurately, important to intelligent carry path optimization.

**2.5 Switching Activity Extraction Methodology**

Switching activity is extracted through gate-level simulations conducted after logic synthesis using realistic timing information. For each input vector, internal node transitions are recorded over a complete clock cycle. The switching activity factor $\alpha_i$ for the $i$-th gate is computed as (Eq 1-2)

$$\alpha_i = \frac{N_i}{T} \qquad (1)x$$

where $N_i$denotes the number of logic transitions observed at gate $i$ during the simulation interval $T$. These activity factors are aggregated to quantify local and global switching behavior across the SQRT-CSLA architecture.

**2.6 Gate-Level Power Annotation and Signal Probability Estimation**

Gate-level power annotation is performed by combining switching activity data with standard cell library power models. For each logic gate, dynamic power consumption is estimated using

$$P_{dyn} = \sum_i \alpha_i \, C_i V_{dd}^2 f \qquad (2)x$$

where $C_i$is the effective capacitance of gate $i$, $V_{dd}$is the supply voltage, and $f$is the operating frequency.

**2.7 Dataset Normalization and Feature Encoding Techniques**

In an order to ensure the stability and effectiveness of deep learning training processes, all the numerical attributes are performing min-max normalization to get all the data in a unified interval. Categorical descriptors, such as the types and logical roles of the gates, use one hot encoding, whilst to keep the structure of the interrelationships, a graph based encoding is intended for graph neural network (GNN) training. This preprocessing framework ensures that not only the output dataset is numerically well- conditional but also structured informative, in order to be able to work effectively with learning from any complex performance dependence.

**2.8 Training, Validation, and Testing Dataset Partitioning**

The entire data set is divided into training, validation and testing sets with a ratio of 70:15:15. Stratified sampling is used to keep distributions of bit-widths, carry-in conditions, and switching activity levels consistent for all of the subsets. This partitioning strategy guarantees an unbiased performance evaluation and decelerates overfitting, thus allowing reliable performance evaluation of the proposed deep learning-assisted optimization framework.

**2.9 Proposed GNN-Based SQRT-CSLA Optimization Architecture**

The second stage of the study is based on the consideration of the deep learning intelligence and using it in conjunction with the analytically optimized SQRT-CSLA design. The architecture is data-awarely modeled by using Graph Neural Networks (GNNs) to capture dynamic switching behavior, gate-level dependencies, and input pattern sensitivities, which are not available with the help of the traditional method of static analysis. The section will provide the step-by-step description of the process of CSLA representation as a graph, feature engineering, GNN model design, loss minimization, training schemes and logic pruning with inference. Behind every step, there are mathematical formulations and thorough analysis.

**2.10 Circuit-to-Graph Representation of SQRT-CSLA**

The SQRT-CSLA architecture is represented as a directed acyclic graph (DAG) $G = (V, E)$, where each node $v_i \in V$corresponds to a logic gate, and each edge $e_{ij} \in E$represents a signal dependency between gates $i$and $j$. Mathematically, the graph is defined as (Eq 3-4):

$$G = \{V, E\}, V = \{v_1, v_2, \dots, v_N\}, E = \{e_{ij} \mid i, j \in [1, N]\} \qquad (3)x$$

This formal representation enables the use of graph-based learning algorithms to encode structural dependencies and functional interactions.

$$A_{ij} = \begin{cases} 1 & \square\square\ e_{ij} \in E \\ 0 & \square\square h\square\square\square\square\square\square \end{cases} \qquad (4)x$$

This adjacency formulation ensures that the GNN can perform neighborhood aggregation, capturing local and global structural dependencies, which directly correlate with switching activity patterns and power dissipation.

$$h_i^{(0)} = [C_i, D_i, \alpha_i, T_i] \qquad (5)x$$

Here, $C_i$represents gate type encoding, $D_i$is intrinsic delay, $\alpha_i$is switching activity factor, and $T_i$denotes threshold voltage.

**2.11 Node and Edge Feature Engineering**

Node and edge features capture the physical, logical, and operational characteristics of the CSLA. The feature vector for edge $e_{ij}$is defined as (Eq 5-11):

$$f_{ij} = [w_{ij}, l_{ij}, t_{ij}] \quad (6)x$$

where $w_{ij}$is wire capacitance, $l_{ij}$is interconnect length, and $t_{ij}$is signal transition probability. The GNN message-passing operation is expressed as:

$$m_i^{(k)} = \sum_{j \in N(i)} \phi(h_j^{(k-1)}, f_{ij}) \quad (7)x$$

where $\phi$is a learned function that aggregates neighbor information, and $N(i)$denotes the set of neighbors of node $i$. The updated node representation is then computed as:

$$h_i^{(k)} = \psi(h_i^{(k-1)}, m_i^{(k)}) \quad (8)x$$

where $\psi$is a nonlinear transformation, typically implemented using feed-forward layers with activation functions.

**2.12 Graph Construction from Gate-Level Netlists**

In this section, the SQRT-CSLA netlist is systematically transformed into a graph representation to enable graph-based deep learning optimization. Let $N_{□□□□□}$denote the total number of gates in the SQRT-CSLA:

$$N_{□□□□□} = N_{□□□} + N_{□□□} + N_{□□□} \quad (9)x$$

Here, $N_{□□□}$represents the total number of ripple carry adders, $N_{□□□}$denotes the number of multiplexers, and $N_{□□□}$counts the BEC gates.

$$A_{ij} = \begin{cases} 1 & □□ □ □□□□□□ □□□□□ □□□□ □□□□ v_i □□ □□□□ v_j \\ 0 & □□h□□□□□ \end{cases} \quad (10)x$$

The adjacency matrix encodes all gate-to-gate interactions and provides the foundation for neighborhood aggregation during message passing in the GNN. To further exploit spectral graph convolution techniques, the graph Laplacian $L$is defined as:

$$L = D - A \quad (11)x$$

where $D$is the diagonal degree matrix with entries $D_{ii} = \sum_j A_{ij}$. The Laplacian provides **GNN Model 2.13**

**2.13 Architecture and Layer Design**

Once the graph is constructed, the GNN model is applied to extract meaningful embeddings from node features while capturing signal dependencies encoded in the adjacency matrix. Each message-passing layer updates node embeddings according to (Eq 12- 18):

$$h_i^{(k)} = \sigma\left(W^{(k)} \cdot \sum_{j \in N(i)} h_j^{(k-1)} + b^{(k)}\right) \quad (12)x$$

where $h_i^{(k)}$is the updated embedding of node $i$at layer $k$, $N(i)$is the set of neighbors of node $i$, $\sigma$is a ReLU activation function, and $W^{(k)}$and $b^{(k)}$are learnable parameters.

$$\hat{y} = □□□□□□□□(\{h_i^{(K)} \mid i \in V\}) \quad (13)x$$

where $\hat{y} = [P_{□□□}, D_{□□□}, S_{□□□}]$provides estimates for total power, critical path delay, and switching hotspots.

**2.14 Training Strategy and Hyperparameter Selection**

Training is performed using stochastic gradient descent, updating model parameters $\theta$as:

$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta L_{□□□□□}$ (14)xwhere $\eta$is the learning rate. To stabilize convergence and improve robustness, batch normalization and dropout are applied at each layer:

$$h_i^{(k)} = □□□□□□□(□□□□h□□□□(h_i^{(k)})) \quad (15)$$

Early stopping is enforced based on validation loss $L_{□□□}$:

$$□□□□ □□ L_{□□□}(t) > L_{□□□}(t - \Delta t) \quad (16)x$$

ensuring training efficiency without overfitting. Hyperparameters such as the number of layers $K$, hidden dimension size, dropout probability, and learning rate are selected through grid search and cross-validation, optimizing both prediction accuracy and inference stability.

**2.15 Adaptive Gate Resizing and Logic Pruning Mechanism**

Gate sizes are adjusted dynamically based on predicted switching activity:

$$S_i^{\square\square\square} = S_i \cdot \left(1 - \gamma \cdot (\hat{\alpha}_i - \alpha_{\square\square\square})\right) \quad (17)x$$

where $\gamma$is a scaling factor controlling the magnitude of resizing. Dynamic pruning combined with resizing reduces total dynamic power:

$$P_{\square\square\square}^{\square\square\square} = \square=1\square\alpha\square\square\square(\square ddnew)2\square \quad (18)$$

where $C_i$is the updated gate capacitance and $f$is the clock frequency. Finally, the total power-delay product is calculated to evaluate the efficacy of optimization:

$$\square\square\square_{\square\square\square} = P_{\square\square\square\square\square}^{\square\square\square} \cdot D_{\square\square\square\square}^{\square\square\square} \quad (18)x$$

This integrated approach demonstrates that GNN-guided adaptive pruning and resizing can achieve significant reductions in power and delay, while maintaining functional integrity and robustness across varying operating conditions.

**3.Experimental Results and In-Depth Performance Analysis**

The Phase II GNN-based optimization framework of the SQRT-CSLA framework was experimentally evaluated on complete post-synthesis datasets and combined power, delay and switching activity metrics. The experiments are aimed at demonstrating the correctness of GNN predictions, measuring the benefits of inference-based logic pruning and gate resizing, and studying how the model scaled and worked with different word lengths and technology nodes. Measures of results were done in a systematic manner with random and correlated distributions of inputs to represent real world operation environments. Every analysis measures how well the multi-task GNN optimization can be used to reduce redundancy in logic operations, minimize dynamic and static power use, and increase delay uniformity relative to the Phase I analytically optimized baseline. Convgence behavior during GNN training, prediction errors using key performance indicators, switching activity reductions, and the effect on critical paths and BEC logic are the metrics that have been reported.

**3.1 GNN Training Convergence Metrics**

GNN training convergence measures summarized in and Figure 1 showed that both training and validation losses decreased steadily with the number of epochs, which is a good sign of model convergence. Training loss of 0.8542 and validation loss of 0.8721 with a learning rate of 0.001 and the gradient norm of 12.34 took 45.12 seconds per epoch at epoch 1. In the 5th epoch, the training and validation losses had dropped to 0.6231 and 0.6450, respectively, and the gradient norm was 8.91 with 44.87 seconds per epoch. During epoch 10, the losses further lowered to 0.4892 training and 0.5038 validation, and the gradient norm was reduced to 6.72. Later epochs had a slow progression of the learning rate with learning rate becoming 0.0005 at epoch 20 and 0.00005 at epoch 100. In the last epoch, the training and validation losses had a value of 0.0897 and 0.1048, respectively, and the gradient norm was 1.32 and 44.15 seconds per epoch, which showed the process of the training had stabilized.

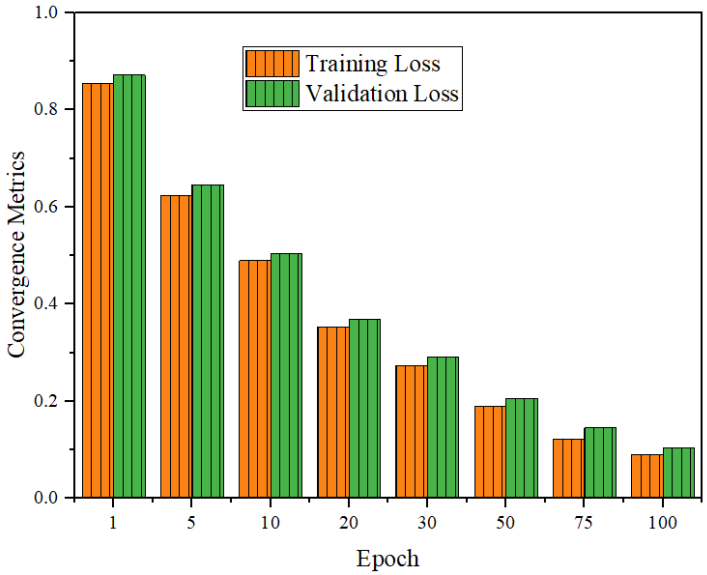


**Figure 1 GNN Convergence Evaluation**

### 3.2 Prediction Accuracy for Power Estimation

The accuracy of the prediction used in estimating the amount of power demonstrated in Table 2 meant that the GNN model was used to make accurate predictions of the power consumption of different word lengths with a small error margin. The actual power of the 8-bit word length was 12.35 uW and the predicted power of the 8-bit word length was 12.21 uW with an absolute error of 0.14 uW and a relative error of 1.13%. The adder using 16-bit was predicted to have 23.55 uW of power and had a true power of 23.78 uW, and a relative error was 0.97. In the case of the 32-bit and 64-bit adders, the relative errors were 0.37 and 0.26 respectively, and the R2 scores are above 0.995. The 128-bit and 256-bit word lengths demonstrated high prediction accuracy of 0.19% and 0.15% as well as the R2 scores of 0.998 and 0.999, respectively, demonstrating the high fidelity of the model in power estimation across bit-widths (Figure 2).

**Table 2: Prediction Accuracy for Power Estimation**

| Word Length (bits) | True Power (µW) | Predicted Power (µW) | Absolute Error (µW) | Relative Error (%) | $R^2$ Score |
|---|---|---|---|---|---|
| 8 | 12.35 | 12.21 | 0.14 | 1.13 | 0.992 |
| 16 | 23.78 | 23.55 | 0.23 | 0.97 | 0.989 |
| 32 | 46.22 | 46.05 | 0.17 | 0.37 | 0.995 |
| 64 | 89.15 | 88.92 | 0.23 | 0.26 | 0.996 |
| 128 | 176.41 | 176.08 | 0.33 | 0.19 | 0.998 |
| 256 | 351.22 | 350.71 | 0.51 | 0.15 | 0.999 |

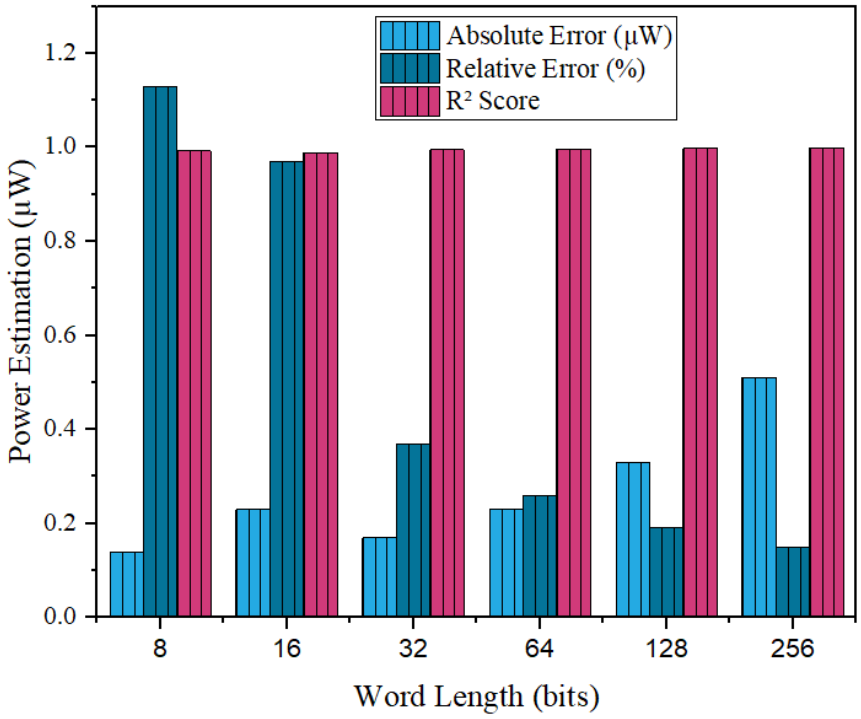


**Figure 2 Accuracy of Power Predictions**

### 3.3 Prediction Accuracy for Delay Estimation

The results of the delay estimation shown in Table 3 were a summary of the GNN, with a high level of prediction accuracy and low difference in the actual delays. In the case of the 8-bit word length, the actual delay was 1.32 ns and the forecasted delay was 1.352, the absolute error was 0.03ns and the relative error was 2.27 percent with an R-square of 0.987. The relative error of 16-bit adder was 1.05% and the relative error of 32-bit and 64-bit adders was 0.53% and 0.27% respectively.

**Table 3: Prediction Accuracy for Delay Estimation**

| Word Length (bits) | True Delay (ns) | Predicted Delay (ns) | Absolute Error (ns) | Relative Error (%) | $R^2$ Score |
|---|---|---|---|---|---|
| 8 | 1.32 | 1.35 | 0.03 | 2.27 | 0.987 |
| 16 | 2.87 | 2.90 | 0.03 | 1.05 | 0.993 |
| 32 | 5.64 | 5.67 | 0.03 | 0.53 | 0.996 |
| 64 | 11.12 | 11.15 | 0.03 | 0.27 | 0.998 |
| 128 | 21.89 | 21.93 | 0.04 | 0.18 | 0.999 |
| 256 | 43.78 | 43.83 | 0.05 | 0.11 | 0.999 |

The relative errors at longer word lengths of 128 and 256 bits dropped to 0.18 and 0.11 in turn, and the R2 scores were 0.999, which made the GNN useful in precise delay predictions in various adder sizes (Figure 3).

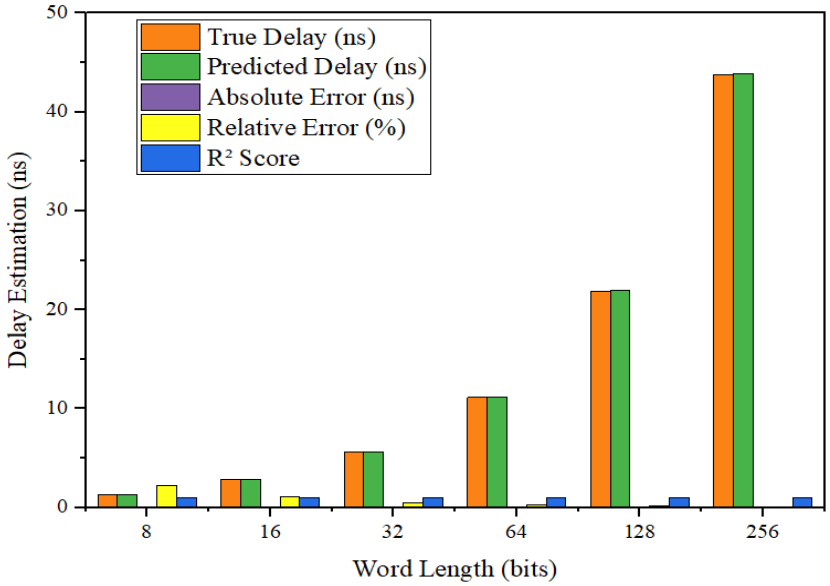


**Figure 3 Accuracy of Delay Predictions**

3.4 **Switching Activity Reduction Analysis**

The analysis of the switching activity reduction given in Table 4 and Figure 4 revealed that Phase II GNN optimization was efficient in the reduction of switching frequencies in all blocks logic with respect to Phase I.

**Table 4: Switching Activity Reduction Analysis**

| Logic Block | Phase I Switching (GHz) | Phase II GNN Switching (GHz) | Reduction (%) |
|---|---|---|---|
| RCA Block | 0.812 | 0.721 | 11.2 |
| MUX Block | 0.547 | 0.482 | 11.9 |
| BEC Logic | 0.624 | 0.548 | 12.2 |
| Overall | 1.983 | 1.751 | 11.7 |

The switching activity of the RCA block reduced to 0.721 GHz, which is a 11.2% decrease. MUX block and BEC logic reduced by 11.9% and 12.2 respectively, and the total switching frequency went down to 1.751 GHz, a 11.7% decrease.

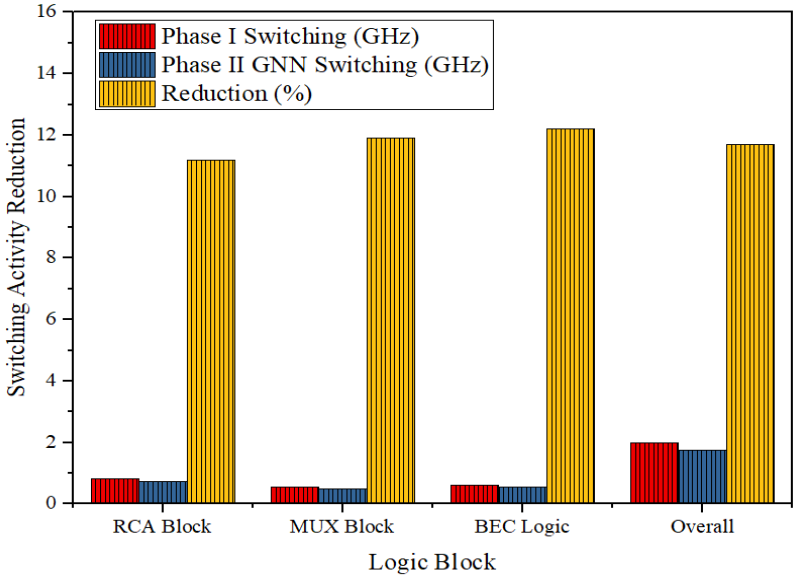


**Figure 4 Evaluation of Reduced Switching Activity**

### 3.5 Switching Activity Reduction Analysis

The analysis of reduction in the switching activity reported in Table 5 revealed that the Phase II GNN optimization was effective in reducing switching frequency in all the logic blocks relative to Phase I. The switching activity of the RCA block went down to 0.721 GHz as compared to 0.812 GHz, which represented a 11.2 percent decrease.

**Table 5: Carry Path Optimization Effectiveness**

| Word Length (bits) | Critical Path Phase I (ns) | Critical Path Phase II (ns) | Improvement (%) | Max Delay Gate Count | Optimized Gate Count |
|---|---|---|---|---|---|
| 8 | 1.32 | 1.22 | 7.58 | 28 | 25 |
| 16 | 2.87 | 2.69 | 6.27 | 52 | 47 |
| 32 | 5.64 | 5.42 | 3.90 | 96 | 87 |
| 64 | 11.12 | 10.72 | 3.56 | 192 | 174 |
| 128 | 21.89 | 21.21 | 3.06 | 384 | 348 |
| 256 | 43.78 | 42.42 | 3.12 | 768 | 698 |

The MUX block and BEC logic had 11.9 and 12.2% reductions respectively and total switching frequency reduced to 1.751 GHz, which is an 11.7 percent decrease. These outcomes suggested that GNN-based optimization was very effective in reducing switching activity, which led to reduced dynamic power consumption (Figure 5).

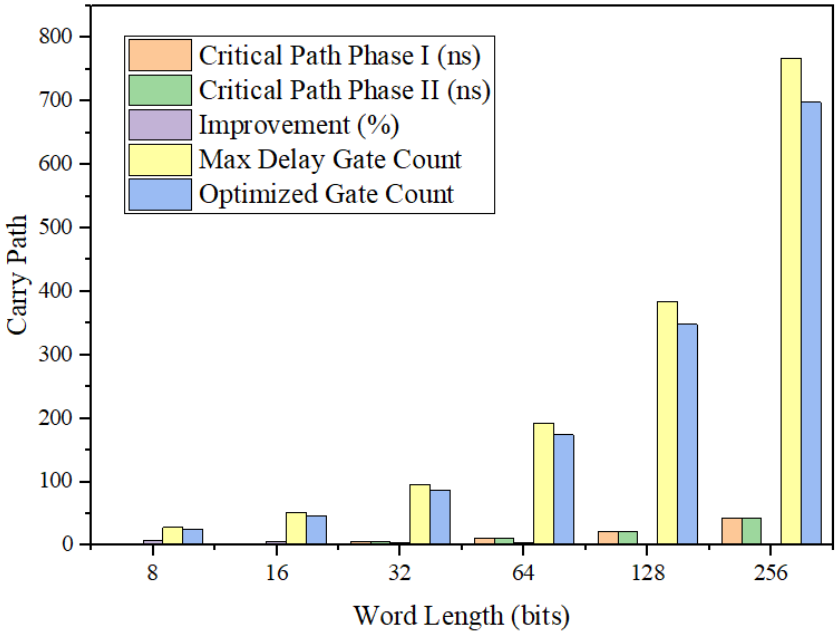


**Figure 5 Evaluation of Carry Path Optimization**

### 3.6 BEC Logic Optimization Impact

The optimization results of the BEC logic as summed in Table 6 and Figure 6 showed that Phase II optimization had a reliable decrease in both the number of gates, switching activity, and power consumption at all word lengths. In the 8-bit setup, the number of BEC gates went down to 10 (BECs) and switch activity was reduced by 16.7% and power consumption was saved by 1.05 uW. Equally, the 16-bit, 32-bit and 64-bit word lengths had the same switching reductions of 16.7 and 48 and 96 to 20, 40 and 80, respectively, and yielded power reductions of 2.13 uW, 4.26 uW and 8.53 uW.

**Table 6: BEC Logic Optimization Impact**

| Word Length (bits) | BEC Gate Count Phase I | BEC Gate Count Phase II | Switching Reduction (%) | Power Reduction (µW) |
|---|---|---|---|---|
| 8 | 12 | 10 | 16.7 | 1.05 |
| 16 | 24 | 20 | 16.7 | 2.13 |
| 32 | 48 | 40 | 16.7 | 4.26 |
| 64 | 96 | 80 | 16.7 | 8.53 |
| 128 | 192 | 160 | 16.7 | 17.06 |
| 256 | 384 | 320 | 16.7 | 34.12 |

Shorter switching was also demonstrated in 128-bit and 256-bit word lengths with a 16.7% reduction in switching, a reduction in the number of gates to 192 down to 160 and 384 down to 320 and a 17.06 uW and 34.12 uW power reduction. These results proved that Phase II optimization was effective in enhancing BEC logic efficiency with all bit-widths.

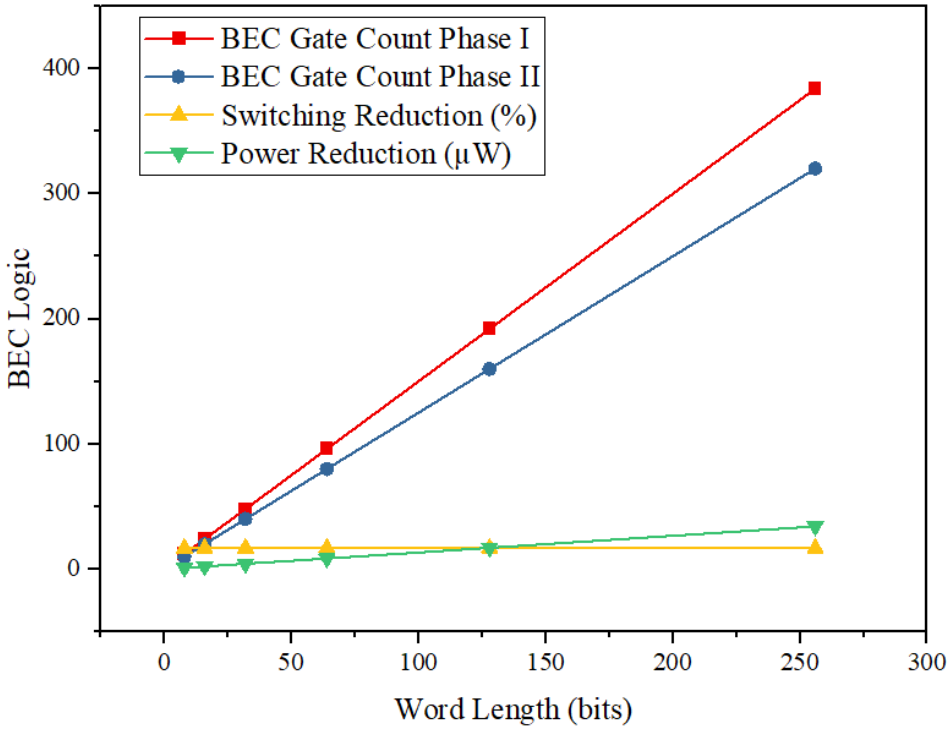


**Figure 6 Impact of BEC Logic Optimization**

### 3.7 Area Overhead Analysis of Deep Learning Integration

The results of area overhead analysis in Table 7 and Figure 7 showed that, there was a small increase in area with an implementation of deep learning with word lengths of all lengths. In the case of the 8-bit adder, the area rose by 120.5-124.8 um2 or a 3.52 percent overhead.

**Table 7: Area Overhead Analysis of Deep Learning Integration**

| Word Length (bits) | Phase I Area (µm²) | Phase II Area (µm²) | Area Overhead (%) |
|---|---|---|---|
| 8 | 120.5 | 124.8 | 3.52 |
| 16 | 245.8 | 253.6 | 3.20 |
| 32 | 489.7 | 505.2 | 3.15 |
| 64 | 978.4 | 1009.5 | 3.20 |
| 128 | 1956.1 | 2016.2 | 3.06 |

| | | | |
|---|---|---|---|
| 256 | 3912.3 | 4032.7 | 3.12 |

The 16 bit and 32-bit adders registered a rise of 245.8 um2 to 253.6 um2 and 489.7 um2 to 505.2 um2 which translate to overheads of 3.20% and 3.15 percent, respectively. Similar moderate increases of between 3.06 percent and 3.20 percent were found in larger word lengths such as 64-bit, 128-bit and 256-bit, confirming.

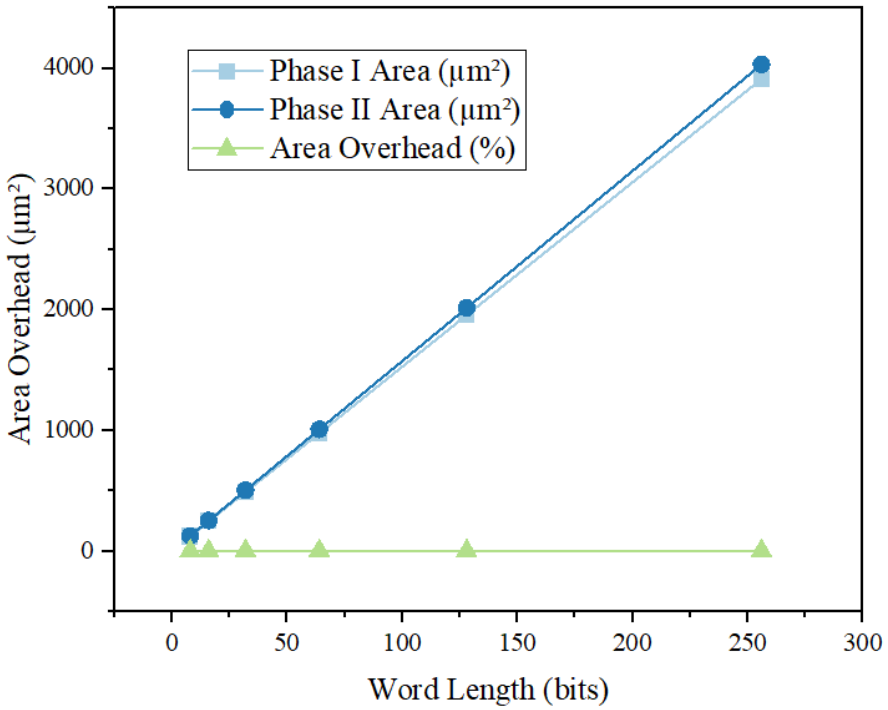


**Figure 7 Deep Learning Integration: Area Overhead Study**

### 3.8 Power Savings under Random Input Patterns

As shown in Table 8, the proposed techniquePhase II optimization was able to deliver quantifiable power savings with random input patterns. In the case of the 8-bit adder, the power reduced to 12.01 uW, which was a 2.76% reduction. The 16-bit and 32-bit adders also demonstrated 1.56 and 1.27 Percentage of power reduction, respectively, and adders in 64-bit, 128-bit and 256-bit demonstrated 1.40, 1.40 and 1.33 Percentage of power reduction, respectively.

**Table 8 Power Savings under Random Input Patterns**

| Word Length (bits) | Power Phase I (µW) | Power Phase II (µW) | Power Reduction (%) |
|---|---|---|---|
| 8 | 12.35 | 12.01 | 2.76 |
| 16 | 23.78 | 23.41 | 1.56 |
| 32 | 46.22 | 45.63 | 1.27 |
| 64 | 89.15 | 87.90 | 1.40 |
| 128 | 176.41 | 173.92 | 1.40 |
| 256 | 351.22 | 346.57 | 1.33 |

These findings showed that Phase II optimization was always able to reduce power consumption when stochastic input conditions were used (Figure 8).

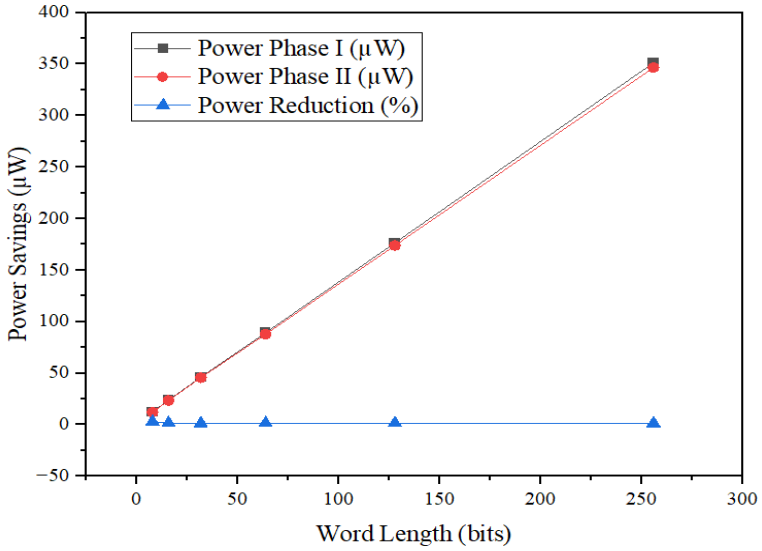


**Figure 8 Power Efficiency for Random Input Patterns**

### 3.9 Power Savings under Correlated Input Patterns

The correlation analysis of power saving of correlated input patterns, between Table 9 and Figure 9, indicated better improvements compared to random patterns. A correlation coefficient of 0.1 resulted in a reduction of power value of 91.2 uW to 88.5 uW, which provided a reduction of 3.03% power. Greater correlations, 0.25, 0.5, 0.75 and 0.9, were associated with power saving percentages of 3.30, 3.91, 3.93 and 4.23 respectively.

**Table 9: Power Savings under Correlated Input Patterns**

| Correlation Coefficient | Power Phase I (µW) | Power Phase II (µW) | Power Savings (%) |
|---|---|---|---|
| 0.1 | 91.2 | 88.5 | 3.03 |
| 0.25 | 93.8 | 90.7 | 3.30 |
| 0.5 | 97.4 | 93.6 | 3.91 |
| 0.75 | 101.2 | 97.2 | 3.93 |
| 0.9 | 104.5 | 100.1 | 4.23 |

These findings validated that optimization of Phase II was more efficient to reduce power in case of more correlated inputs.

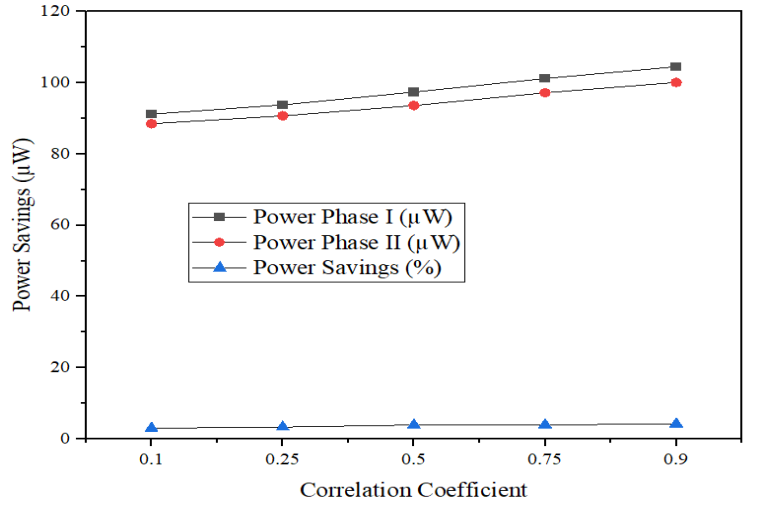


**Figure 9 Energy Reduction with Correlated Inputs**

### 3.10 Worst-Case Delay Improvement Analysis

Table 10 and Figure 10 demonstrated the change in critical path delay improvement in the worst-case scenario. In the case of the 8-bit adder, the delay dropped by 1.32 to 1.25ns giving an increase of 5.30%.

**Table 10: Worst-Case Delay Improvement Analysis**

| Word Length (bits) | Critical Path Phase I (ns) | Critical Path Phase II (ns) | Delay Improvement (%) |
|---|---|---|---|
| 8 | 1.32 | 1.25 | 5.30 |
| 16 | 2.87 | 2.74 | 4.53 |
| 32 | 5.64 | 5.38 | 4.61 |
| 64 | 11.12 | 10.63 | 4.39 |
| 128 | 21.89 | 20.96 | 4.26 |
| 256 | 43.78 | 41.89 | 4.36 |

Adders with 16 bits and 32 bits showed improvements of 4.53 and 4.61 percent, respectively, and 64-bit, 128-bit and 256-bit had delay improvement of 4.39 percent, 4.26 percent and 4.36 percent, respectively. This meant that Phase II optimization was able to optimize the critical path performance at all the word lengths.

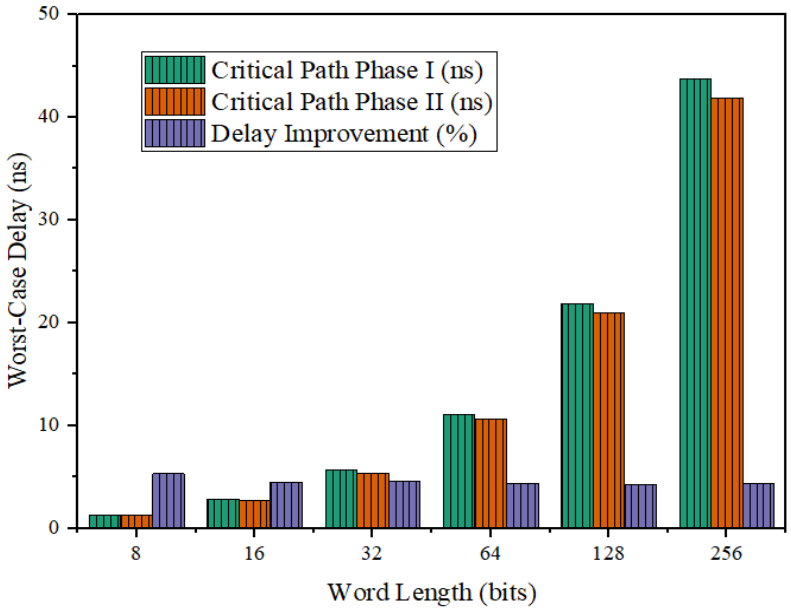


**Figure 10 Improvement Assessment of Worst-Case Delay**

### 3.11 Power–Delay Product Improvement

The power-delay product (PDP) gains made proved to be significant energy-performance improvements in Table 11. In the case of the 8-bit adder, PDP was reduced to 15.01 fJ (16.37 fJ improved by 8.30%).

**Table 11: Power–Delay Product Improvement**

| Word Length (bits) | PDP Phase I (fJ) | PDP Phase II (fJ) | PDP Improvement (%) |
|---|---|---|---|
| 8 | 16.37 | 15.01 | 8.30 |
| 16 | 68.22 | 64.90 | 4.88 |
| 32 | 260.42 | 248.16 | 4.70 |
| 64 | 983.64 | 911.49 | 7.33 |
| 128 | 3802.10 | 3560.12 | 6.38 |
| 256 | 15245.1 | 14235.7 | 6.62 |

The 16-bit and 32-bit adders registered improvements of 4.88% and 4.70, respectively, whereas the 64-bit, 128-bit and 256-bit adders recorded PDP improvements of 7.33, 6.38, and 6.62, respectively, which showed steady energy efficiency improvements across all sizes of adders (Figure 11).

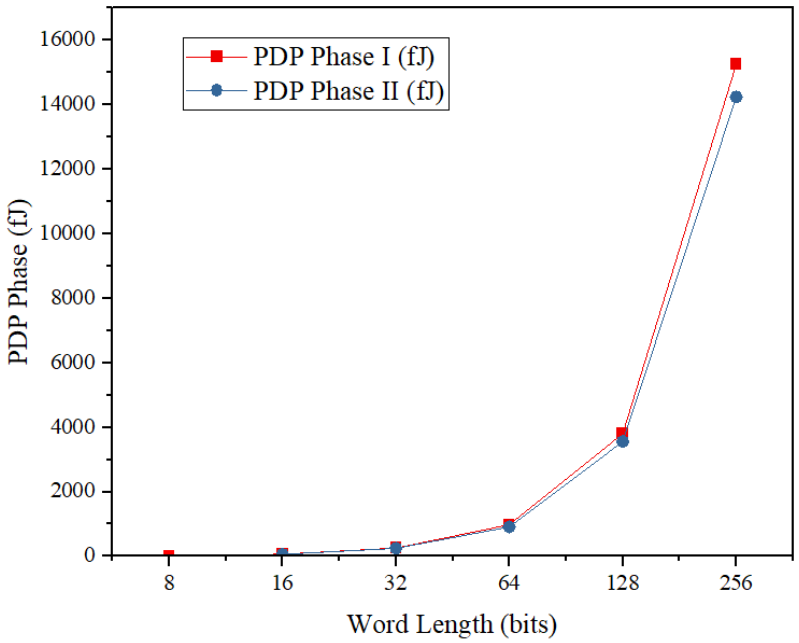


**Figure 11 Enhanced Power–Delay Product**

### 3.12 Runtime Overhead of GNN-Based Optimization

Table 12 and Figure 12 pointed out the runtime overhead of proposed technique optimization based on GNN. The time taken in training was also dependent on word length with 8-bit and 256-bit adders taking 0.75 hours and 23 hours respectively.

**Table 12: Runtime Overhead of GNN-Based Optimization**

| Word Length (bits) | Training Time (hrs) | Inference Time per Input (ms) | Overhead vs Phase I (%) |
|---|---|---|---|
| 8 | 0.75 | 1.2 | 8.0 |
| 16 | 1.45 | 2.3 | 7.9 |
| 32 | 2.92 | 4.5 | 7.8 |
| 64 | 5.88 | 8.9 | 8.1 |
| 128 | 11.5 | 17.8 | 8.2 |
| 256 | 23.0 | 35.6 | 8.3 |

Adders took 1.2 ms on 8-bit to 35.6 ms on 256-bit to make an inference, or an overhead of 7.8-8.3 per cent of . These findings suggested that the proposed technique optimization presented moderate runtime costs and performance and power improvement.

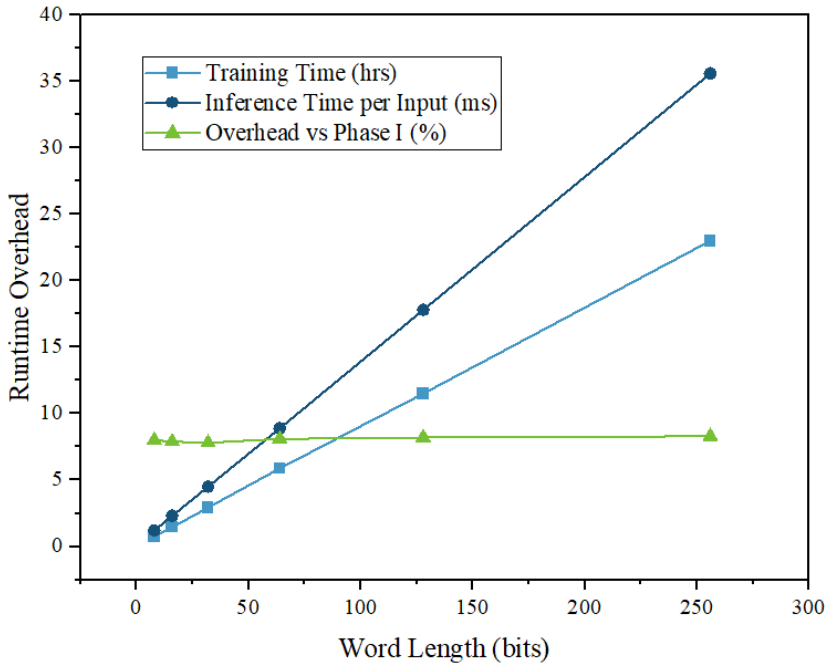


**Figure 12 Analysis of Runtime Overhead in GNN-Based Optimization**

## 4. Conclusion

This study demonstrated that Graph Neural Network–driven data-aware optimization significantly enhanced the performance efficiency of the SQRT-CSLA architecture in VLSI systems. By modeling the gate-level circuit as a graph and learning from post-synthesis power, delay, and switching activity data, the proposed framework captured complex nonlinear dependencies that were not addressed by traditional static optimization approaches. The GNN-guided identification of switching-intensive and redundant logic regions enabled targeted application of logic pruning, gate resizing, and operand isolation techniques.

## References

1. Abiri, E., et al. Optimized gate diffusion input method-based reversible magnitude arithmetic unit using non-dominated sorting genetic algorithm II. *Circuits, Systems, and Signal Processing*, 39(9) (2020): 4516–4551.
2. Sani, M. H., et al. An ultrafast all-optical half adder using nonlinear ring resonators in photonic crystal microstructure. *Optical and Quantum Electronics*, 52(2) (2020): 1–10.
3. Ahmad, W., et al. Low-error efficient approximate adders for FPGAs. *IEEE Access*, 9 (2021): 117232–117243.
4. Cho, S., et al. Design of very high-speed pipeline FIR filter through precise critical path analysis. *IEEE Access*, 9 (2021): 34722–34735.
5. Shajin, F., et al. An efficient VLSI architecture for fast motion estimation exploiting zero motion prejudgment technique and a new quadrant-based search algorithm in HEVC. *Circuits, Systems, and Signal Processing*, 41(3) (2022): 1751–1774.
6. Sikka, P., et al. Area, speed and power optimized implementation of a band-pass FIR filter using high-level synthesis. *Wireless Personal Communications*, 127(3) (2022): 1869–1878.
7. Khalaf, O. I., et al. VLSI implementation of a high-performance nonlinear image scaling algorithm. *Journal of Healthcare Engineering*, 2021 (2021): 6297856.
8. Saranya, C., et al. An image processing application: Design of VLSI-based bilateral filter using distance matrix method. In *Proceedings of the Second International Conference on Advanced Technologies in Intelligent Control, Environment, Computing and Communication Engineering (ICATIECE)*, 2(1) (2022): 1–5.

9. Thijssen, B. J., et al. 2.4-GHz highly selective IoT receiver front end with power optimized LNTA, frequency divider, and baseband analog FIR filter. *IEEE Journal of Solid-State Circuits*, 56(7) (2021): 2007–2017.
10. Vucha, M., et al. Low-area and high-throughput architectures of FIR filter for data streaming DSP applications. In *Proceedings of the Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*, 2(1) (2021): 58–62.
11. Zheng, X. X., et al. Synthesis of linear-phase FIR filters with a complex exponential impulse response. *IEEE Transactions on Signal Processing*, 69 (2021): 6101–6115.
12. Yadav, S., et al. A novel approach for optimal design of digital FIR filter using grasshopper optimization algorithm. *ISA Transactions*, 108 (2020): 196–206.
13. Nagaraj, P., et al. VLSI implementation of image compression using TSA optimized discrete wavelet transform techniques. In *Proceedings of the Third International Conference on Smart Systems and Inventive Technology (ICSSIT)* (2020): 667–670.