electronics

MDPI

Article

Reinforcement Learning-Driven Bit-Width Optimization for the High-Level Synthesis of Transformer Designs on Field-Programmable Gate Arrays

Seojin Jang [1] and Yongbeom Cho [1,2,*]

[1] Department of Electrical and Electronics Engineering, Konkuk University, Seoul 05029, Republic of Korea; seojinygud@konkuk.ac.kr
[2] Deep ET, Seoul 05029, Republic of Korea
* Correspondence: ybcho@konkuk.ac.kr or ybcho@deep-et.com

Abstract: With the rapid development of deep-learning models, especially the widespread adoption of transformer architectures, the demand for efficient hardware accelerators with field-programmable gate arrays (FPGAs) has increased owing to their flexibility and performance advantages. Although high-level synthesis can shorten the hardware design cycle, determining the optimal bit-width for various transformer designs remains challenging. Therefore, this paper proposes a novel technique based on a predesigned transformer hardware architecture tailored for various types of FPGAs. The proposed method leverages a reinforcement learning-driven mechanism to automatically adapt and optimize bit-width settings based on user-provided transformer variants during inference on an FPGA, significantly alleviating the challenges related to bit-width optimization. The effect of bit-width settings on resource utilization and performance across different FPGA types was analyzed. The efficacy of the proposed method was demonstrated by optimizing the bit-width settings for users' transformer-based model inferences on an FPGA. The use of the predesigned hardware architecture significantly enhanced the performance. Overall, the proposed method enables effective and optimized implementations of user-provided transformer-based models on an FPGA, paving the way for edge FPGA-based deep-learning accelerators while reducing the time and effort typically required in fine-tuning bit-width settings.

Keywords: high-level synthesis (HLS); field-programmable gate array (FPGA); reinforcement learning; transformer models; hardware accelerators

check for updates

Citation: Jang, S.; Cho, Y. Reinforcement Learning-Driven Bit-Width Optimization for the High-Level Synthesis of Transformer Designs on Field-Programmable Gate Arrays. *Electronics* **2024**, *13*, 552. https://doi.org/10.3390/electronics13030552

Academic Editors: Nebojsa Bacanin, Robertas Damaševičius, Justyna Patalas-Maliszewska and Lalit Garg

Received: 25 December 2023
Revised: 14 January 2024
Accepted: 22 January 2024
Published: 30 January 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

1.1. Deep Learning and the Rise of Transformers

The field of deep learning has witnessed exponential growth in recent years, fueled by advances in algorithms and computing hardware [1–3]. Deep-learning models, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have demonstrated state-of-the-art performances over a wide range of applications, including image recognition, natural language processing (NLP), and speech recognition [4–7]. However, more recently, the transformer architecture has emerged as a powerful alternative to traditional deep-learning models, particularly for NLP tasks [8,9].

Since their introduction in 2017, transformers have become the foundation for various cutting-edge models, such as bidirectional encoder representations from transformers, generative pretrained transformers, and text-to-text transfer transformers. Transformers offer several advantages over conventional models, including the ability to model long-range dependencies and parallelizable computation [5,8]. Consequently, transformers have demonstrated superior performance in numerous tasks, leading to their extensive use in research and the industry. "Language Models are Few-Shot Learners (Brown, T. B. et al., 2020) [9]" explores the capabilities of Generative Pre-trained Transformer-3 (GPT-3),

a large-scale language model, and highlights how language models can quickly learn from a small amount of data across various tasks. "An Image is Worth 16 × 16 Words: Transformers for Image Recognition at Scale (Dosovitskiy, A. et al., 2020) [10]" introduces the Vision Transformer (ViT) model, applying the transformer model, previously used in NLP tasks, to object detection. "Double Consistency Regularization for Transformer Networks (Wan, Y. et al., 2023) [11]" introduces a new regularization technique that doubles the consistency of transformer networks. The aim is to improve the model's generalization abilities and robustness in noisy data environments. This method is particularly used to enhance performance and stability in fields like natural language processing. "Enhancing Fashion Classification with Vision Transformer (ViT) and Developing Recommendation Fashion Systems Using DINO-VA2 (Abd Alaziz, H.M et al., 2023) [12]" discusses how to apply Vision Transformer (ViT) to a fashion classification and recommendation system that recommends customized fashion items to users in commercial environments such as online shopping.

1.2. Field Programmable Gate Array-Based Acceleration for Deep Learning

The increased complexity of deep-learning models coupled with the demand for real-time processing has necessitated the development of specialized hardware accelerators to improve their efficiency [13]. Field-programmable gate arrays (FPGAs) have emerged as a popular choice for implementing deep-learning models because of their flexibility, energy efficiency, and performance benefits over traditional central processing units (CPUs) and graphic processing units (GPUs).

FPGAs enable developers to create custom hardware tailored to specific applications, leveraging the inherent parallelism and pipelining opportunities in deep-learning models [14,15]. Moreover, their reprogrammable nature enables rapid prototyping and iterative design, which are particularly valuable in the rapidly evolving field of deep learning [16–18].

1.3. Challenges in FPGA-Based Transformer Acceleration

As shown in Figure 1, despite the potential benefits of FPGA-based acceleration for transformer models, several challenges must be addressed to achieve efficient implementation [18–20]. A notable challenge is the long development cycle associated with hardware design, which may be exacerbated by the need to tune various high-level synthesis (HLS) parameters to optimize the final hardware architecture [14,18]. Various parameters, such as parallelism, memory type, and memory size, significantly affect FPGA resource utilization, performance, and power consumption [19,21,22].

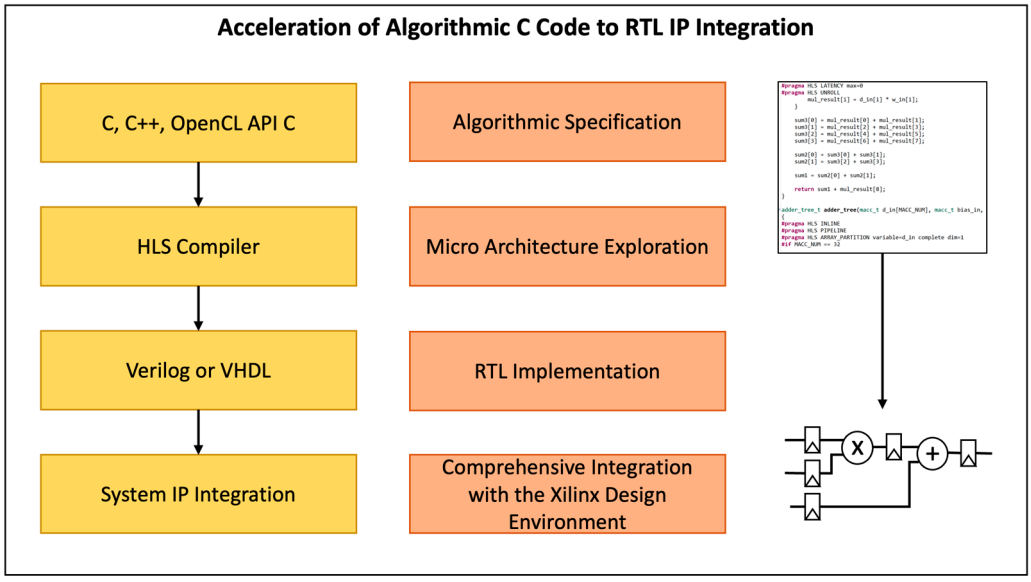


Figure 1. Compilation flow of high-level synthesis tools.

In the realm of hardware design research, many studies focus on how to reduce model size through quantization methods, thereby shrinking the hardware footprint. While this approach can reduce the size and power consumption of hardware designs, it may impact the performance of transformer models, especially in terms of accuracy in visual domains [22,23].

HLS tools have been developed to alleviate complexities in hardware design by enabling developers to work with higher level programming languages, such as C++ or open computing language (OpenCL), rather than traditional hardware description languages (HDLs) [16,17,24]. However, the optimization of HLS parameters remains a manual and time-consuming process that requires domain expertise and trial-and-error experimentation.

1.4. Motivation and Contributions

Despite the benefits of FPGA-based acceleration, technologies that simplify the development process and optimize bit-width settings are crucial for efficient FPGA implementation, given the diverse bit-width requirements across different transformer designs. This necessity becomes more apparent in the light of challenges, where the focus on reducing hardware size through quantization can compromise the transformer model's performance, particularly in accuracy within visual domains. Such a trade-off between hardware efficiency and model effectiveness underlines the need for innovative methods that can balance these aspects more effectively.

To address these hardware implementation challenges and to overcome the limitations of existing approaches, this paper proposes a novel method involving the predesign of an optimal foundational transformer architecture tailored to various types of FPGAs. This architecture is specifically designed to maintain the high performance of transformer models while optimizing hardware resource utilization. Subsequently, to facilitate bit-width optimization, a novel method based on reinforcement learning (RL) is established to automatically tune and optimize the bit-width settings based on user-provided transformer variants. Our research introduces a novel RL-based approach for optimizing bit-width in FPGA-based transformer models. This method stands out in terms of scalability, flexibility, and efficiency compared to existing methods. Unlike traditional FPGA optimization techniques, which often rely on manual tuning and static model configurations, our approach dynamically adapts to various FPGA architectures, significantly speeding up the development process and enhancing model adaptability. This adaptive optimization is crucial for efficiently implementing complex transformer models on diverse FPGA platforms without compromising computational accuracy. Unlike traditional approaches that rely on manual tuning, our RL-based method provides a more adaptive and efficient solution, significantly reducing the time and expertise needed for optimization.

The contributions of this research can be summarized as follows:

Introduction of a predesigned transformer hardware architecture to streamline the FPGA implementation process for various transformer designs, ensuring high model performance while optimizing hardware size and power consumption.

Realization of adaptive bit-width optimization using RL, facilitating more efficient FPGA implementations of user-provided transformer variants, and offering a solution to the challenges of hardware efficiency versus model effectiveness.

By offering a comprehensive analysis of the design of transformer inferences on FPGAs and optimization of bit-width settings, this study seeks to advance FPGA-based deep-learning accelerators, leading to more efficient hardware implementations of complex deep-learning models. This research bridges the gap between the desire for compact hardware designs and the need to maintain the high accuracy and effectiveness of transformer models, particularly in visual applications. Existing methods focus primarily on generating an HLS code by analyzing the structure of transformer models. However, such strategies typically cannot easily optimize hardware design owing to the numerous variables involved in the process, escalating the complexity of the optimization landscape.

In contrast, the proposed method relies on a predesigned transformer hardware architecture as a robust foundation that encapsulates critical hardware configurations, thereby enabling a more controlled and manageable optimization process. With a clear understanding of the user's design intentions, we employ RL to fine-tune the predesigned transformer to achieve optimized performance.

This methodological pivot not only streamlines FPGA implementation but also provides a more direct pathway to achieve optimized hardware performance. By alleviating the complexity associated with traditional HLS code generation from transformer structures, our approach makes the optimization process more efficient, ultimately accelerating and improving the effectiveness of the FPGA implementations of user-provided transformer variants.

1.5. Related Work

This section discusses the influence of HLS parameters on the transformer architecture and FPGA implementations [17,20,21]. Section 1.5.1 explores the fundamental structure and operational principles of the transformer model, highlighting its innovative applications in the fields of language processing and image recognition. Through a comprehensive review of the importance and versatility of this structure, we demonstrate why the transformer model is appropriate for FPGA implementation in this study [20,22]. Section 1.5.2 clarifies the influence of HLS parameters on various layers and operations within the transformer architecture. This subsection focuses on how parallelism, memory type and size, and variable bit-width affect the performance of FPGA-based transformer accelerators, especially in terms of inference speed and power consumption. Additionally, it provides insights into strategies for optimizing parameters for efficient hardware implementation, explaining how such optimizations can enhance hardware performance.

1.5.1. Transformer Architecture

The transformer model, introduced by Vaswani et al. [8], revolutionized language processing tasks by introducing the attention mechanism as an alternative to traditional sequence-to-sequence models based on RNNs or CNNs.

At its core, the transformer uses the scaled dot-product attention and multihead attention mechanisms. The scaled dot-product attention can be formalized as follows:

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

Here, Q, K, and V represent the query, key, and value matrices, respectively, and d_k represents the dimension of the key vectors. This method calculates the weights of the keys for each query and uses them to combine the values.

The multihead attention mechanism enables the model to perform the attention operation from multiple "*head*" simultaneously to learn relationships between inputs in different representation spaces:

$$MultHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \tag{2}$$

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{3}$$

Here, W_i^Q, W_i^K, W_i^V, and W^O are the projection matrices learned by the model.

The transformer model has been largely successful in NLP tasks, and various derivatives have emerged since. One such variant is the vision transformer, which has demonstrated remarkable performance in image processing [10]. The vision transformer divides an image into multiple patches, each of which is embedded and treated like a sequence of tokens. The embedding of image patches is performed as follows:

$$PatchEmbed(x) = x * W_e + b_e \tag{4}$$

where x represents the flattened patches of the input image, W_e is the weight matrix, and b_e represents the bias.

Transformers and their derivatives can model complex relationships between input data through attention mechanisms, this is particularly crucial for sequence or unstructured data. This unique capability underscores the suitability of the transformer architecture for FPGA implementation in this research. However, successful implementation requires a deep understanding of how various HLS parameters influence the performance and efficiency of the transformer model. Thus, in Section 1.5.2, we discuss the influence of HLS parameters on HDL synthesis and FPGA implementation.

Additionally, in contrast to the approach used in "FTRANS: Energy-Efficient Acceleration of Transformers using FPGA (Li, B. et al., 2020)", which focuses on model compression through enhanced block-circulant matrices (BCM), our method employs RL for dynamic bit-width optimization. This strategy allows for real-time, nuanced adaptation of transformer models, leading to improvements in both performance and energy efficiency. Additionally, unlike the quantization analysis method discussed in "Auto-ViT-Acc: An FPGA-Aware Automatic Acceleration Framework for Vision Transformer with Mixed-Scheme Quantization (Li, Z. et al., 2022)", our research specifically tackles the hardware-level challenges of implementing transformer models on FPGAs, with a focus on bit-width optimization. These distinctions underscore the unique contribution of our work in the landscape of FPGA-based deep learning accelerators [22,23].

1.5.2. HLS Variable Bit-Width

In the context of HLS design, bit-width optimization is crucial in artificial intelligence frameworks, especially in transformer models. In particular, bit-width configurations directly affect the performance and efficiency of these models on FPGA, including the computational speed and resource utilization. Bit-width optimization not only aids in reducing resource and power consumption while maintaining the required computational accuracy but also enables a more tailored hardware implementation of transformer models on FPGA. By optimizing the bit-width, developers can customize the hardware implementation of transformer models according to the needs and constraints of specific applications, achieving more efficient and flexible FPGA-accelerated solutions. Hence, this study focuses on bit-width optimization to ensure efficient implementation of transformer models on FPGA, which can promote the development of superior FPGA-based transformer accelerators.

The choice of variable bit-width in HLS design significantly affects the performance, resource utilization, and power consumption of FPGA-based transformer accelerators. Reduced-precision arithmetic, which involves the use of lower bit-widths for the model parameters and intermediate results, can lead to substantial savings in FPGA resources and power consumption [20]. However, this strategy can adversely affect the numerical accuracy and stability of the transformer model [25].

When selecting the appropriate bit-width for a particular transformer model, developers must strike a balance between resource utilization, power consumption, and numerical accuracy. This trade-off can be achieved via empirical analysis and hardware-aware training techniques, ensuring deep-learning models remain robust to reduced-precision arithmetic without reducing performance [19,20,22].

Overall, this section highlights the significant impact of HLS parameters, such as parallelism, memory type and size, and bit-width, on the FPGA implementation of transformer models, specifically, on the various layers and operators within the transformer architecture as well as the entire inference process. The subsequent section introduces the proposed method for optimizing these parameters using Q-learning to develop efficient FPGA-based transformer accelerators.

2. Materials and Methods

2.1. Overall Design Approach and Method Selection

The proposed design approach commences with the predesign of an optimal base transformer architecture tailored to various FPGA types, followed by automatic adaptation and optimization according to the transformer variants provided by users.

The formulation of this design approach is governed by two primary reasons: First, there exist several limitations in directly employing RL or other deep-learning methods for optimizing the original code. These methods focus on optimizing local functions or loops within the constraints of the existing code structure. However, in substantial and complex projects such as transformers, implementing structural code modifications through these methods proves challenging, and these strategies often fail at fully leveraging the hardware capabilities. In contrast, by predesigning an optimal transformer, we can build upon existing research findings and practical experience, laying a solid foundation for subsequent automatic adaptation and optimization. Second, by understanding and adapting the transformer variants provided by users, the proposed framework can offer customized hardware designs to meet different design requirements and performance objectives.

2.2. Predesign of the Optimal Transformer

In the predesign phase, we employed a manual design methodology to construct a sophisticated transformer architecture tailored for FPGA devices. This methodology emphasizes hardware resource allocation, parallelism, and memory layout, ensuring optimal performance and efficiency. The hardware setup includes computation units for encode/decode processes and on-chip memory banks.

Our approach mirrors this by focusing on modular design, facilitating both shallow and deep network implementations. For instance, multihead attention and other modules are reconfigured based on control logic to form either encoder or decoder units. This modularity enables efficient resource usage. Moreover, we optimized the computation sequence within the attention layer for synchronous processing, significantly reducing latency. Our storage architecture mimics a quasi-distributed system, minimizing data transmission costs and power consumption associated with long-distance data movement. This comprehensive approach, combining modular design with optimized data flow and storage, forms the bedrock of our subsequent RL-driven optimization process.

As illustrated in Figure 2, the hardware architecture encapsulates control modules, an on-chip cache, computation arrays, and nonlinear computation modules. The on-chip cache includes input, weight, and intermediate result/output caches, employing ping-pong operations to support continuous data processing. The overall data flow of the accelerator is illustrated in Figure 3. The weight data of an attention sublayer or a feed-forward neural network layer are read in batches, and the next batch of weight data is synchronously loaded during computation. By sequentially loading weight data across network layers and storing interlayer computation results on-chip for the next sublayer computation, the interaction times with off-chip dynamic random-access memory (DRAM) are reduced.

Furthermore, the computation sequence within the attention layer is optimized, facilitating the synchronous computation of the Softmax function with matrix addition and multiplication, thereby mitigating computation latency. Through this task-level scheduling, the computation of all network layers in the transformer model is accelerated. To minimize the data transmission cost between computational and storage units, including the power consumption associated with long-distance transmission and complex address generation read–write cost, a quasi-distributed storage architecture is used, as shown in Figure 4.

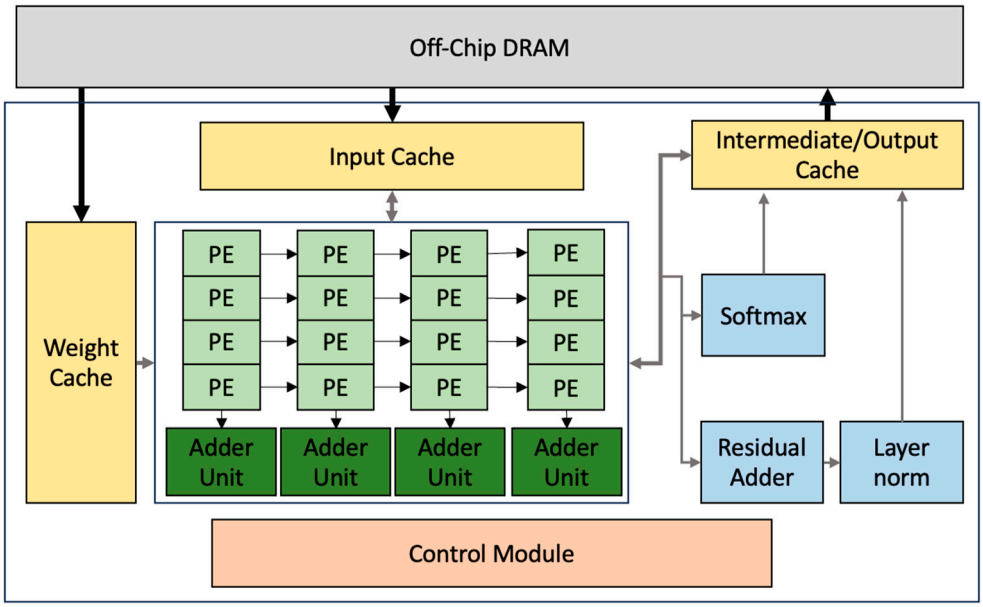


Figure 2. Overall hardware architecture of the proposed accelerator.

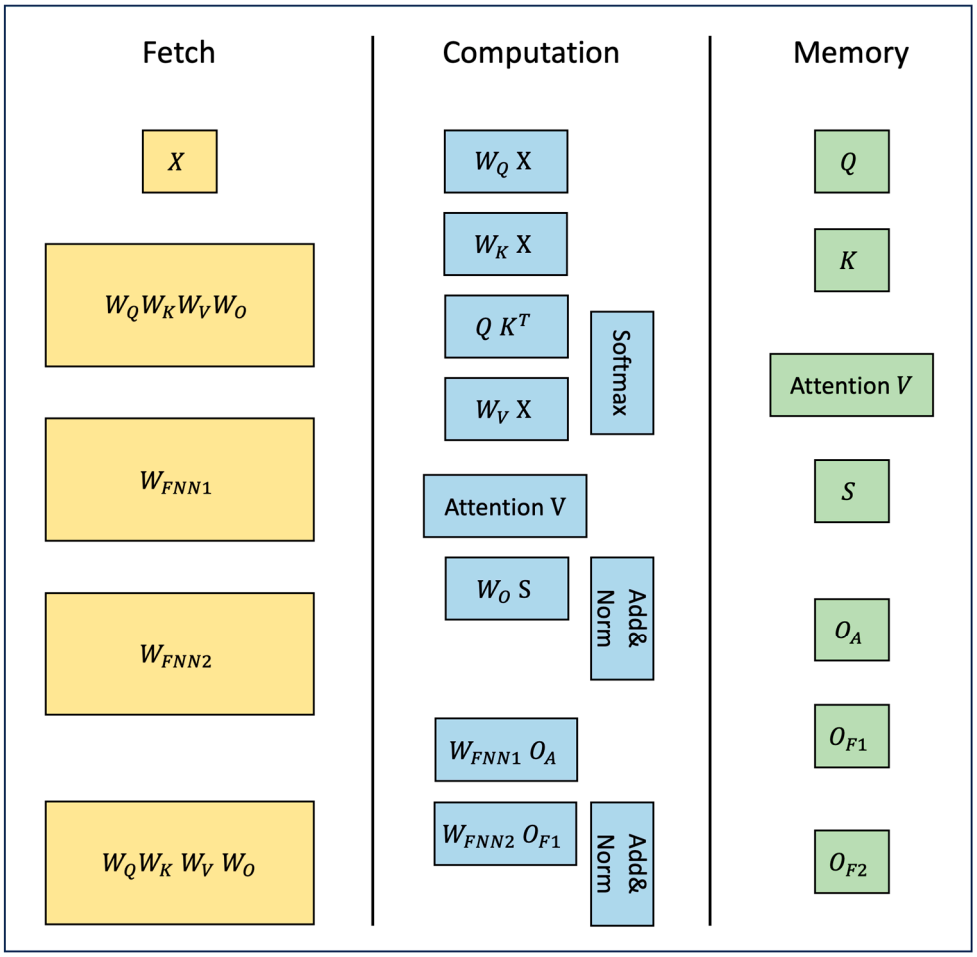


Figure 3. Data flow in the accelerator.

The storage and computation of each neuron are simulated using processing elements (PEs) and a register file (RF). In particular, the PEs are interlinked in a distributed fashion to emulate distributed connections amidst neurons, while multilevel storage entities such as

first in, first out (FIFO), block random-access memory (BRAM), and DRAM are used outside the PE array to reduce data transmission overhead. Specifically, the storage architecture includes PE local registers, internal cache of the computation array (FIFO), on-chip global cache (BRAM), and off-chip DRAM, each having different access costs across multilevel storage hierarchies. The RF within PE, having the lowest access cost, serves as a basic unit point where data can be transferred laterally and longitudinally between each basic unit point. In this manner, the inherent data movement during computation maximally occurs at the RF storage level with the least access overhead, maximizing the consumption of data written by upper-level storage and reducing movement power consumption. When reading computational data from BRAM, the entire computation array is treated as a unit for data input, with basic unit points within the computation array interacting in a distributed manner, transferring input data between points to achieve data reuse across the input, weight, and output dimensions. Unlike the computation array, the local cache can store partial sums and results of matrix computation. Compared with storage in the global cache, adder units can access partial sums and results in a more rapid manner over shorter distances. Additionally, while computing nonlinear functions, this quasi-distributed storage architecture can avoid redundant reads of intermediate result data and reduce redundant computations, thereby accelerating inference.

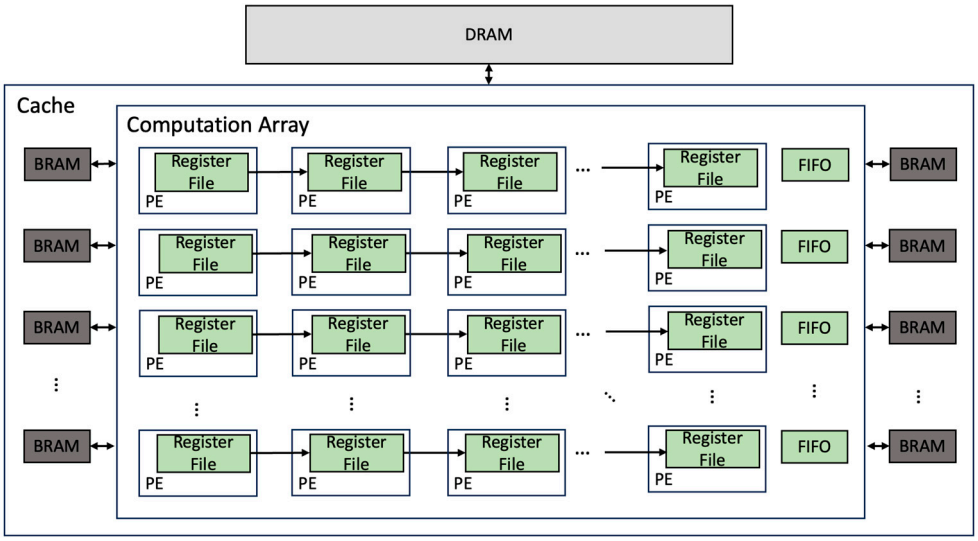


Figure 4. Distributed storage architecture.

As shown in Figure 2, the computation array consists of PEs and adder units, with each column of PEs sharing an adder unit. PEs handle matrix multiplication, while adder units manage the addition of PE results and accumulation of partial sums, which can be expanded based on actual inference demands. The regularity in the pruned offset diagonal matrix, referring to the consistent size and arrangement of submatrices, can be mapped efficiently to the regular computation array. The computation array internally adopts a fixed-weight data flow scheme, as shown in Figure 5, with input and weights fed into computational units in row and sparse block forms, respectively, mitigating the efficiency drops caused by different input sentence lengths and computation modes of encoders and decoders. Input data are transmitted to the right in each cycle, enabling input re-use, while the weight data with the largest data scale only need to be read once from outside the array. The output is accumulated cycle by cycle within the adder unit, achieving output re-use. Each computational unit includes a PE and adder units, with each PE encompassing a multiplication unit and an RF storage unit, serving as regular distributed basic points that can efficiently map the locality of the offset diagonal matrix, i.e., the row and column dimensions of individual submatrices and regular distribution of nonzero values. Each PE contains 16 multipliers and one data distributor, as shown in

Figure 6, with the data distributor responsible for rearranging input data based on offset values to ensure the multiplication of input data and corresponding nonzero value weight data. Sparse decoding outside the PE is unnecessary, and the need to address indexing of partial sums and output or computation results is eliminated, as illustrated in Figure 7. The adder unit is responsible for adding partial sum results or bias data generated by the column PE, as shown in Figure 8, with each adder unit internally equipped with a local cache for caching partial sum results. This configuration helps reduce the data movement distance of partial sums. The dense_en signal controls the data source of the adder unit to switch between dense and sparse data paths, with the adder unit supporting the rectified linear unit activation function.

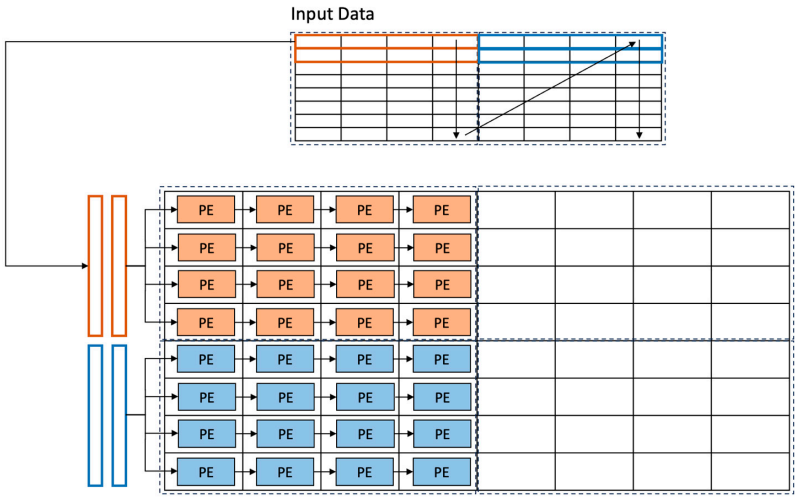


Figure 5. Data flow in the computational unit.

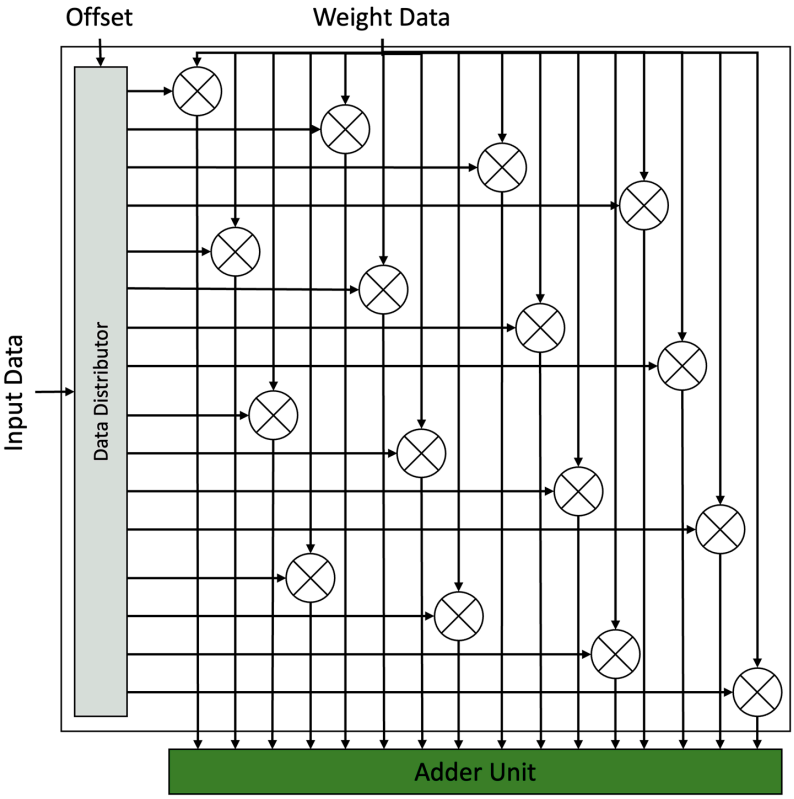


Figure 6. Internal structure of a PE.

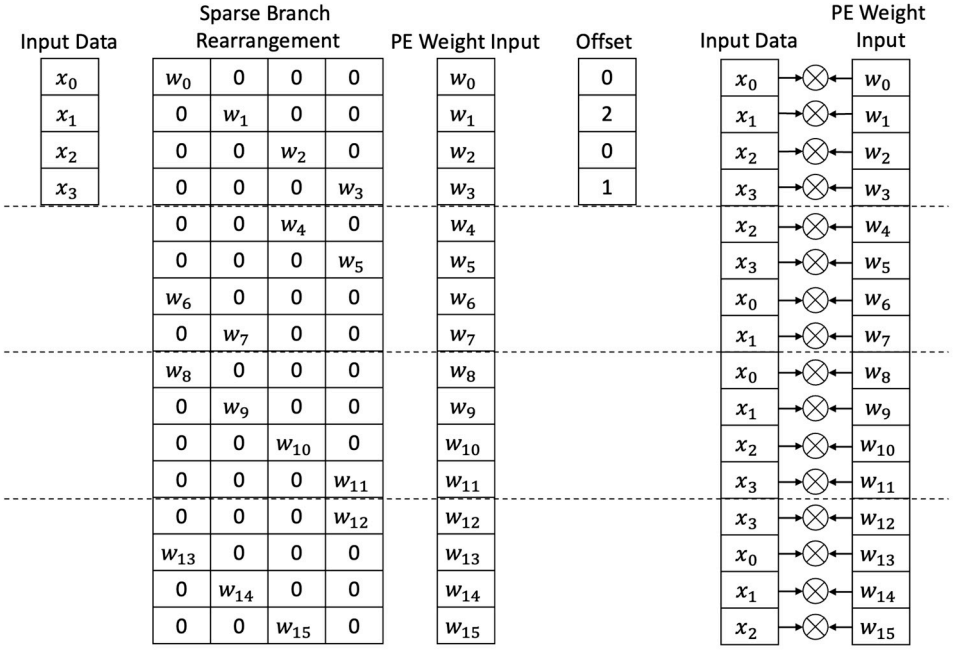


Figure 7. Data organization within the computation unit.

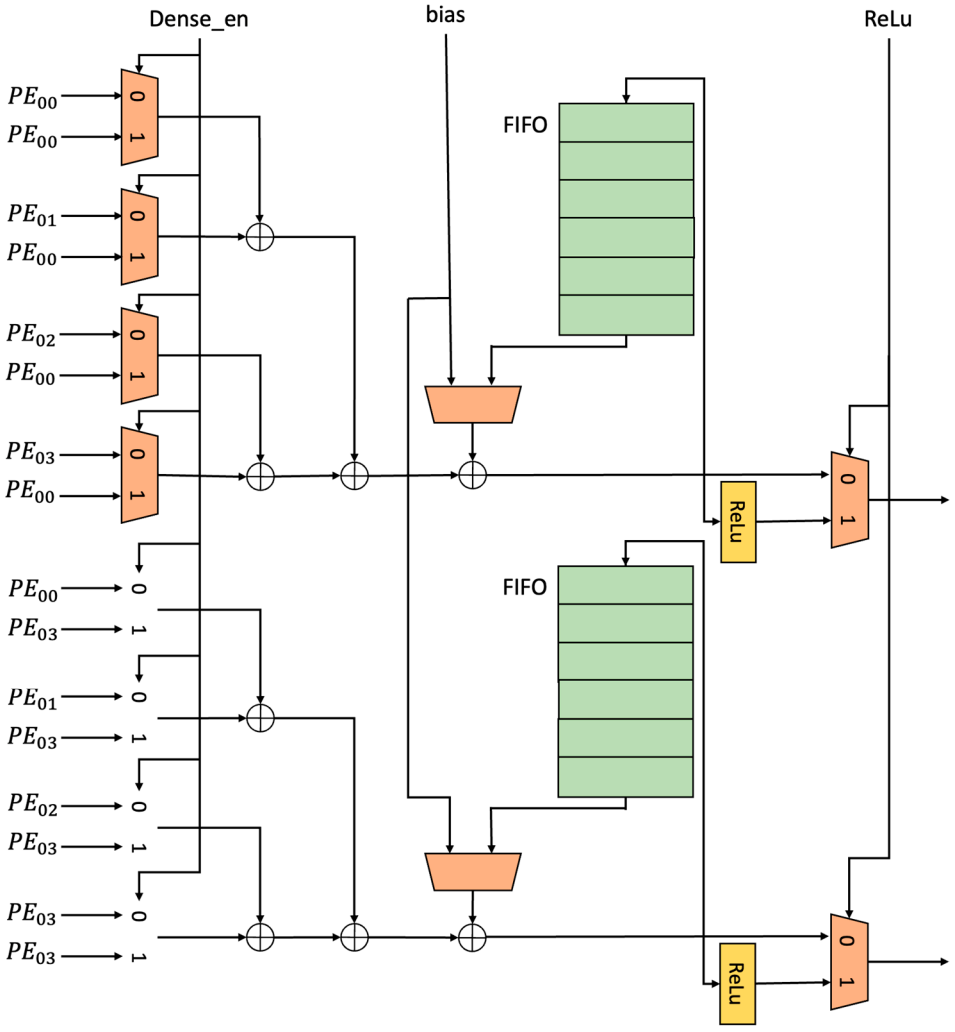


Figure 8. Adder unit configuration.

2.3. RL-Based Adaptive Bit-Width Optimization

The objective of this study is to better adapt to user-specific transformer variants while meeting specific performance and resource efficiency goals, such as reducing computation time and ensuring computational accuracy. This can be achieved by optimizing the bit-width of inputs/outputs and operations while maintaining the bit-width of weights. This assertion has been validated in a previous study [26], which demonstrated that a certain bit-width for weights could ensure a balance between performance and accuracy.

The RL framework is chosen for its ability to dynamically adjust and learn optimal configurations through continuous interaction with the hardware design environment. Specifically, we design an RL framework that dynamically adjusts the bit-width of inputs/outputs and operations through continuous exploration and learning.

In our Q-learning implementation for adaptive bit-width optimization, the hyperparameters were carefully chosen to balance exploration, learning speed, and convergence. The selected hyperparameters are as follows:

- Discount Factor (γ): Set between 0.8 and 0.9, this parameter determines the importance of future rewards. A higher discount factor encourages the algorithm to consider long-term rewards, facilitating a more strategic optimization over time.
- Learning Rate: Chosen within the range of 0.1 to 0.3, the learning rate controls how much new information overrides old information. A moderate learning rate was selected to ensure that the algorithm steadily adapts to new findings without oscillating or diverging.
- ε-Greedy Strategy: The ε value, set between 0.1 and 0.8, dictates the balance between exploration and exploitation. Initially set higher to encourage exploration, ε gradually decreases as the algorithm gains more knowledge about the environment, allowing more exploitation of known information.

The choice of these hyperparameters was based on the need to explore effectively the bit-width configuration space while ensuring stable and efficient learning. The discount factor was set closer to 1 to prioritize long-term effectiveness in bit-width optimization. The learning rate was calibrated to ensure that the algorithm could adapt to new insights at a reasonable pace. The ε-greedy strategy was used to balance exploring new configurations and exploiting known efficient configurations, which is crucial in the dynamically changing environment of FPGA-based transformer models.

Within this RL framework, our reward function is meticulously designed to prioritize computational accuracy, followed by computation time and then hardware resource utilization. This prioritization is reflective of our emphasis on ensuring the high precision of the transformer model, especially crucial in hardware implementations, while also maintaining efficient inference speeds. The specific formulation of the reward function is as follows:

$$\begin{aligned} R(S_t, a_t) = w_1 \cdot Accuracy(S_t, a_t) + w_2 \cdot TimeEfficiency(S_t, a_t) \\ + w_3 \cdot ResourceUtillization(S_t, a_t) \end{aligned} \tag{5}$$

Here, $R(S_t, a_t)$ represents the reward at the state S_t when action a_t is taken. The weights w_1, w_2, w_3 correspond to the importance of computational accuracy, computation time, and hardware resource utilization, respectively, with $w_1 > w_2 > w_3$ signifying their relative importance in the optimization process. Its purpose is to optimize hardware resources and power consumption while ensuring computational accuracy.

The proposed reinforcement learning-based hardware design system includes the following modules:

- Test Design Module: This module serves as the test object for the verification system.
- Agent: This module, consisting of the action control module, transactions, and incentive sequences, is used to build a global action set and control the agent actions.
- Hardware Design Verification Platform: Interaction with this platform is facilitated through a well-defined interface, ensuring seamless communication and data exchange and enabling accurate verification and feedback generation. This platform receives the

incentive inputs from the agent, feeds them into the test design model and reference model according to the timing requirements; monitors the input/output of the test design module, compares simulation results, generates bit positions, and feeds this information back to the reward module.

- Reward Module: The reward function is designed to reflect the achieved performance improvements and resource savings, which are crucial for meeting the specified objectives. This module establishes a return mechanism, with bit positions as the reward return for RL, and sends this information to the global state model.
- Global State Model: The initial state is set based on the current bit-width configuration of the HLS design, providing a starting point for the exploration. The termination conditions are set to ensure a thorough exploration while avoiding excessive computations. The termination criterion may be based on the number of iterations, satisfactory performance/resource level, or no further improvement observed. This module builds a global state set based on the coverage of the test design module and sends the reward return to the state control module.
- State Control Module: The states and actions are carefully defined to provide a manageable yet expressive representation of the bit-width configuration space, enabling efficient exploration and learning. This module builds a global state transition table, determines the current state of the system, and controls state transitions based on the received reward return.

To elucidate how this framework operates in practice, consider the following example of dynamic bit-width selection. In our RL method, the agent dynamically selects the optimal bit-width for various components of the Transformer model during the FPGA optimization process. The action space for the RL agent includes a range of bit-width options, such as 8-bit, 16-bit, and 32-bit configurations. The agent evaluates each configuration based on its impact on performance metrics like processing speed and computational accuracy. For example, in one scenario, the RL agent might choose a 16-bit configuration for certain operations where higher precision is required while selecting an 8-bit configuration for other operations where speed is more critical and high precision is less necessary. This decision-making is guided by the reward function, which balances the trade-off between accuracy, efficiency, and resource utilization. The agent iteratively tests and learns from these bit-width adjustments, converging towards an optimal configuration that meets the specific requirements of the FPGA-based transformer model.

The Q-learning algorithm is designed to ensure efficient learning and convergence to the optimal bit-width configuration, even in the presence of large state and action spaces. The construction of the global action set involves the following steps: According to the input quantity m of the test design module, one input is randomized from m inputs, while other inputs remain unchanged as one action, represented as, a, a_2, …, a_m. Thus, a total of m actions constitute the global action set. The states are defined based on the bit-width: 1 bit corresponds to S_0; greater than 1 and less than 64 coverage states are evenly divided into $n-2$ states, denoted as S_1, S_2, …, S_{n-2}; and 64 bits correspond to S_{n-1}. Thus, a total of n states constitute the global state set. The global state transition table is constructed as follows: With n states as the vertical coordinate and n actions as the horizontal coordinate, the Q-table is constructed based on the maximum future reward expectation, considering the corresponding action under each state as the value, initialized as 0. This value is obtained according to the Q function:

$$Q(S_t,\ a_t) = R(S_t,\ a_t) + \gamma * \max\{Q(S_{t+1},\ a_{t+1})\} \tag{6}$$

where R represents the reward return obtained at the moment t in state S_t by taking action a_t; γ is the discount factor, satisfying $0 \leq \gamma \leq 1$; and $\max\{Q(S_{t+1},\ a_{t+1})\}$ represents the maximum expected Q-value for the next state S_{t+1}. The action a_{t+1}, corresponding to the maximum expected Q-value, is the action taken at the moment $t+1$, pertaining to the state

transition information. The Q-value of the corresponding state and action taken constitute the element of the matrix, and the Q-table is thus constructed, as indicated in Table 1.

Table 1. Q-table of the Q-learning Algorithm.

State/Action	a_1	a_2	a_3	...	a_m
S_0	$Q(S_0, a_1)$	$Q(S_0, a_2)$	$Q(S_0, a_3)$	...	$Q(S_0, a_m)$
S_1	$Q(S_1, a_1)$	$Q(S_1, a_2)$	$Q(S_1, a_3)$	...	$Q(S_1, a_m)$
S_2	$Q(S_2, a_1)$	$Q(S_2, a_2)$	$Q(S_2, a_3)$	...	$Q(S_2, a_m)$
...	...	...	...	...	...
S_{n-1}	$Q(S_{n-1}, a_1)$	$Q(S_{n-1}, a_2)$	$Q(S_{n-1}, a_3)$	...	$Q(S_{n-1}, a_m)$

Table 1 has 11 global states as columns and m actions as rows. Each cell in the table contains the expected reward Q-value calculated using the Q function. Notably, this value does not represent the maximum reward return obtained after taking a certain action; instead, it aims at maximizing the sum of the future discount rewards, i.e., obtaining the maximum discount reward expectation. All values are initialized to 0, thereby obtaining Q-table0. Based on Q-table0, the first row of global states is selected as the starting state S_0, that is, the state with a coverage of 0%. The maximum number of iterations N is set to attain the target point S_{n-1}. In the case of the Q-table, among all columns of the current state S_t, we select the column with the maximum Q-value as action a_t. If multiple columns house the same Q-value and the maximum value, one column is randomly selected from among these columns. After the agent takes action a_t, the generated incentives are sent to the hardware design verification platform and test design module. The state control module determines whether the current state S_{t+1} is the target point in the global state. If so, the bit position of the designer is reached, the hardware design verification target is completed, and the training is terminated. If the target state is not reached in a given iteration, the following steps are implemented: The number of iterations is increased by 1, that is, $t = t + 1$. Next, we check whether the number of iterations has reached the maximum defined value, N. If so, the algorithm proceeds to the final step, and the training is terminated. If not, the algorithm proceeds to the next step.

The complete process for bit adjustment involves the following steps:

- Initialization:

Initialize the Q-table or neural network (if using deep RL) with arbitrary values.

Set the initial state, typically the current bit-width configuration of the HLS design.

Determine the termination conditions to ensure effective exploration and exploitation phases while preventing excessive iterations. Termination may be triggered by reaching a certain number of iterations, achieving a satisfactory performance/resource level, or observing no further improvement over a defined period.

- Exploration and Exploitation:

Use an exploration strategy (e.g., ε-greedy) to decide whether to explore a new action or exploit the current knowledge.

Select an action, changing the bit-width of either the inputs/outputs or variables within operations.

- Interaction:

Apply the selected action to the HLS design.

Obtain the new state and the reward from the environment (HLS design). The reward may be computed based on the achieved performance improvements or resource savings.

- Learning:

Update the Q-value of the state–action pair based on the obtained reward and the highest Q-value of the new state (according to the Q-learning update rule).

- Iteration:

Repeat until a termination condition is met.

- Policy Extraction:

Extract the policy (optimal action in each state) from the Q-table or neural network.

Apply the policy to the HLS design to obtain the optimized bit-width configuration.

The proposed method represents a practical and verifiable process for implementing RL in hardware design. The outcomes can be verified through the hardware design verification platform. The effectiveness of this method is evaluated through extensive experiments, as discussed in the following section.

3. Results and Comparisons

The previous sections described the proposed method, which is aimed at optimizing the bit-width of inputs/outputs and variables within operations for transformer-based designs based on FPGAs in the HLS environment. To demonstrate the effectiveness of the proposed approach, a well-optimized transformer design was developed for FPGAs, leveraging existing architectural principles and optimizing key aspects such as hardware resource allocation, parallelism, and memory layout. This predesigned optimal transformer served as a solid foundation for further adaptive optimization using RL.

We applied our methodologies to ZU7EV and ZU9EG FPGAs, assessing the performance and efficiency of our optimized designs. The implementation was executed using the Xilinx Vivado HLS 2020.1 tool. Our process involved first designing HLS code tailored for a transformer model, ensuring FPGA compatibility, especially in terms of key features like parallelism. Postdesign, the HLS code was compiled into a hardware description language.

The RL algorithm, developed using the PyTorch framework, then analyzed this compiled code to identify optimization opportunities. The focus was on bit-width adjustments, where specific values dictating the transformer model's behavior were examined for potential parallel optimizations. The RL algorithm dynamically adjusted the bit-width in the HLS code, leading to the synthesis of various hardware designs. Each design variation offered different hardware sizes and latencies, presenting multiple avenues for optimization.

This detailed process illustrates the practical application of our RL-driven approach in optimizing FPGA-based transformer models, emphasizing its effectiveness and adaptability for real-world scenarios. Table 2 presents the performance and resource utilization metrics of the predesigned transformer on the two FPGA platforms. Subsequently, the RL framework was introduced to dynamically adjust the bit-width based on the user-specified transformer variant, aiming to meet the desired performance and resource efficiency targets.

Table 2. Performance and resource utilization of predesigned transformer on FPGA platforms.

Parameter	FPGA Platform	FP32 Precision
Frames per second (FPS)	ZU7EV	9.3
	ZU9EG	10.8
Throughput (GOPS)	ZU7EV	331.8
	ZU9EG	341.2
GOPS per DSP	ZU7EV	0.327
	ZU9EG	0.311
GOPS per kLUT	ZU7EV	2.121
	ZU9EG	1.897

By introducing variants with floating point 16-bit (FP16) and integer 8-bit (INT8) precisions, the RL framework effectively fine-tuned the bit-width configurations, enhancing the performance and resource efficiency of the baseline transformer design. The comparison presented in Table 3 demonstrates significant improvements in frame rate, throughput, and

computation efficiency, emphasizing the efficacy of the proposed RL-based optimization approach.

Table 3. Comparative performance analysis post-RL-based optimization.

Parameter	FPGA Platform	FP16 Precision	INT8 Precision
Frames per second (FPS)	ZU7EV	16.74	31.63
	ZU9EG	19.24	36.17
Throughput (GOPS)	ZU7EV	597.24	1110.86
	ZU9EG	641.45	1167.43
GOPS per DSP	ZU7EV	0.591	1.075
	ZU9EG	0.572	1.041
GOPS per kLUT	ZU7EV	3.81	4.11
	ZU9EG	3.45	3.75

The proposed RL framework displayed robustness and adaptability in optimizing various transformer variants, facilitating the realization of real-time inferences and diverse performance objectives. Furthermore, a comparison with other implementations demonstrated the superior performance and efficiency of the proposed design, thereby facilitating FPGA-based transformer acceleration. As depicted in Tables 2 and 3, the application of the proposed RL framework resulted in a significant enhancement in both performance and resource efficiency when transitioning from FP32 to FP16 and INT8 precisions on both FPGA platforms. Specifically, on the ZU7EV, the frame rate (FPS) improved by 80% and 240% for FP16 and INT8 precisions, respectively. On the ZU9EG chip, these improvements were even more pronounced.

The throughput, giga operations per second (GOPS), also increased substantially. For the ZU7EV platform, the throughput increased by 80% and 235% for FP16 and INT8 precisions, respectively, with even higher improvements noted for the ZU9EG chip. Enhancements in GOPS per digital signal processor (DSP) and GOPS per kilo lookup table (kLUT) demonstrated the enhanced resource efficiency achieved through the proposed RL framework. Overall, these improvements validated the effectiveness of the proposed RL framework in dynamically adjusting the bit-width of inputs/outputs and variables within operations while maintaining the bit-width of weights.

From the data presented in Tables 2 and 3, it is evident that the introduction of RL optimization (for FP16 and INT8 precision) led to a significant improvement in frames per second (FPS) and giga operations per second (GOPS), compared to the original design (FP32 precision). The increase in FPS indicates a faster processing speed, which is crucial for applications requiring real-time feedback, such as video processing or live data analytics. This improvement underscores the advantages of dynamic bit-width optimization; by dynamically adjusting the bit-width of inputs/outputs and operations, our RL framework enables the hardware design to use resources more effectively without compromising computational accuracy. A lower bit-width reduces the computational burden of individual operations, thus enhancing overall processing speed. Moreover, the design of our reward function, prioritizing computational accuracy followed by computation time and hardware resource utilization, ensures that the optimization process maintains high accuracy while effectively reducing computation time. The choice of hyperparameters, including the adjustment of the learning rate and the application of the ε-greedy strategy, helps the algorithm strike a balance between exploring new configurations and exploiting known efficient configurations.

Our results demonstrate the advantages of RL, particularly Q-learning, in reducing latency and increasing throughput. The core strength of Q-learning in our design lies in its efficient traversal of the state−action space, swiftly identifying optimal bit-width configurations. Specifically, Q-learning in our proposed framework flexibly adjusts the workload of processing elements (PEs), reducing the time consumption per operation. Crucially, for

FPGA-based systems, our approach enhances hardware resource utilization by adjusting bit-widths, thereby minimizing unnecessary data width and easing computational loads. This leads to an overall increase in processing speed. Moreover, our reward function design, focusing on computational accuracy, time efficiency, and resource utilization $R(S_t, a_t)$, ensures that the algorithm improves throughput without compromising model accuracy or system stability. In summary, our proposed method not only accelerates system response time but also optimizes resource allocation, presenting an effective pathway for optimizing transformer models on FPGA platforms.

Building upon this foundation, we further explored the application of our optimized transformer network in specific tasks like object detection and image segmentation. In the realm of object detection, our hardware design was augmented with a region proposal network (RPN) layer, which identifies and proposes candidate object-bound regions within an image. Additionally, a set of detection heads was integrated for the classification of these proposed regions and for bounding box regression, enabling the network to perform object localization in addition to classification. This demonstrates the practicality and effectiveness of our hardware-optimized approach in real-world scenarios. Similarly, for image segmentation, our approach entailed augmenting the simple transformer network with layers tailored for segmenting images. The hardware-specific adaptations included adding a U-Net-like architecture, incorporating convolutional layers for feature extraction and a series of up-sampling layers for pixel-level classification. This modification allows the network to generate segmentation maps, enabling it to distinguish between different objects and backgrounds at the pixel level. The incorporation of a skip-connection strategy ensures that fine-grained details are preserved in the segmented images. As seen in Table 4, the performance of this detection and segmentation network was evaluated using the COCO dataset with fp16 precision in the ZU7EV platform, where it exhibited notably that the proposed method works well in both segmentation accuracy and processing speed.

Table 4. Performance of Transformer Network for Object Detection and Image Segmentation.

Parameter	Object Detection	Image Segmentation
Mean average precision (mAP)	38.2	34.3
Frames per second (FPS)	13.7	10.5
Throughput (GOPS)	732.2	951.1
Power (W)	9.81	11.1

Additionally, our optimized approach demonstrates commendable power efficiency, a crucial aspect for applications in embedded environments. In our evaluations, the optimized transformer network exhibited power consumption levels of 9.81 W for object detection and 11.1 W for image segmentation. These power levels are well within the acceptable range for most embedded applications, particularly in scenarios where energy efficiency is paramount. For instance, in remote monitoring or mobile devices, such reduced power consumption can significantly extend operational durations while maintaining necessary processing capabilities. This indicates that our approach not only excels in precision and processing speed but also offers notable advantages in energy efficiency, making it suitable for various power-sensitive application environments.

In our endeavor to achieve real-time inference and optimize transformer designs on FPGAs, we leveraged an innovative RL framework while exploring alternative software-based quantization techniques using PyTorch v2.1 with FP16 and INT8 precisions. Although software-based quantization increased the inference speed while maintaining a certain level of accuracy, the hardware optimization enabled by our RL framework significantly surpassed these improvements.

Theoretically, different software quantization schemes exhibit varying levels of accuracy and inference speed. Table 5 presents the accuracy and speed metrics of three software quantization schemes.

Table 5. Theoretical comparison of software quantization schemes based on PyTorch-v2.1.

Scheme	Accuracy	Speed
Quantization aware training	92%	1.5×
DoReFa-Net	90%	2×
Parameterized clipping activation (PACT)	91%	1.7×

In practice, the hardware-accelerated TensorRT 8.6 [27] produces diverse levels of accuracy and inference speed, as indicated in Table 6.

Table 6. Actual performance with TensorRT-v8.6.

Scheme	Accuracy	Speed
TensorRT quantization	93%	2×

Upon comparing the hardware and software optimizations, we observed several key distinctions. Software quantization techniques, such as quantization aware training [28], DoReFa-Net [29], and parameterized clipping activation [30], primarily aim to reduce computational and storage demands by quantizing network weights and activations, thereby enhancing inference speed without considerably sacrificing the accuracy. However, these software quantization schemes often require the original model to be modified, which can adversely affect the accuracy and generalizability of the model.

In contrast, the proposed RL framework adopts a hardware-level optimization approach. It optimizes the performance and resource efficiency of transformer designs on FPGAs by dynamically adjusting the bit-width of operations. The advantage of this method is that the original model does not need to be modified, and instead, performance improvements can be achieved by altering the underlying hardware implementation. Specifically, hardware optimization through the RL framework more effectively leverages the parallel processing capability and memory hierarchy of FPGAs, significantly improving inference speed while maintaining satisfactory accuracy.

We further conducted a comparative analysis between the Q-learning and Deep Q-Network (DQN) [31] approaches on the ZU7EV and ZU9EG FPGA platforms as indicated in Table 7. This comparison aimed to evaluate the efficacy of these methods in optimizing transformer models on FPGAs with respect to various performance metrics.

Table 7. Comparative Performance Analysis: Q-learning vs. DQN.

Parameter	FPGA Platform	Q-Learning (FP32)	DQN (FP32)
Frames per second (FPS)	ZU7EV	9.3	8.4
	ZU9EG	10.8	9.9
Throughput (GOPS)	ZU7EV	331.8	301.7
	ZU9EG	341.2	317.3
GOPS per DSP	ZU7EV	0.327	0.293
	ZU9EG	0.311	0.285
GOPS per kLUT	ZU7EV	2.121	2.43
	ZU9EG	1.897	2.4

In our comparative analysis of Q-learning and DQN methods for optimizing bit-width in FPGA-based transformer models, we observed notable differences in performance. The Q-learning approach maintained a higher frames per second (FPS) rate and better throughput (GOPS), indicating its efficiency in real-time processing and overall throughput management. On the other hand, DQN exhibited superior performance in terms of GOPS per kLUT. This suggests that DQN might be more effective in scenarios where optimizing the use of lookup table (LUT) resources is crucial.

The distinct performance characteristics of these two methods can be attributed to their inherent algorithmic properties. Q-learning, with its direct approach to value estimation, likely navigated the state−action space more efficiently for the task of bit-width optimization. This method, due to its simplicity and tabular approach, might have been particularly effective in the constrained environment of FPGA bit-width configuration.

Conversely, DQN, which generally excels in larger state spaces, might have encountered challenges such as overfitting or less effective generalization in this specific context. This could result in less optimal bit-width adjustments. Additionally, the potential instability and unpredictable convergence behavior of DQN might have also played a role in its comparative underperformance.

Moreover, the exploration strategy employed by Q-learning might have been better aligned with the requirements of bit-width optimization, finding more efficient paths through the configuration space. These findings suggest that while DQN has its merits, especially in more complex or larger state spaces, Q-learning appears to be more suitable for tasks like bit-width optimization in FPGA-based transformer models. This is primarily due to its algorithmic efficiency, stability, and suitable exploration strategy in the context of hardware optimization.

Our research demonstrates significant advancements in FPGA-based transformer models using a Q-learning RL framework. However, it is important to acknowledge certain limitations. Firstly, our experiments are predicated on the use of high-level synthesis (HLS) for FPGA development, implying that our method's applicability is tied to platforms conducive to HLS. This presents a boundary condition for the generalization of our findings across diverse FPGA environments. Secondly, while our approach shows promise in optimizing transformer models, its long-term stability and adaptability, particularly for more complex transformer-based architectures, warrant further investigation. The Q-learning algorithm, while efficient in navigating the state−action space for bit-width optimization, may not fully grasp or address the intricacies of complex, global scenarios inherent in advanced models. This limitation highlights a potential area for future research, where more sophisticated or nuanced RL strategies could be explored to enhance the global decision-making capabilities of the optimization process.

4. Conclusions

This paper introduces an innovative method that adeptly merges a predesigned optimal transformer architecture with a reinforcement learning framework using Q-learning. This approach is designed to refine transformer models implemented on FPGA platforms, addressing the specific needs of these models. Central to our strategy is the formation of a foundational transformer architecture suitable for different FPGA types, subsequently fine-tuned and adapted based on user-modified transformer variants.

While our primary focus has been on enhancing transformer models, owing to their unique balance between accuracy and speed, the underlying principles of our method hold architecture for other deep learning models like CNNs and RNNs. This adaptability is crucial, especially considering the computational demands of transformers, which can hinder real-time applications in resource-constrained environments. By prioritizing the accuracy benefits of transformers and simultaneously optimizing their inference speed, our approach ensures these advanced models are more accessible for edge computing scenarios, maintaining high precision where it is most needed.

Our comparative results highlight the RL framework's efficiency in enhancing frame rate, throughput, and computational efficiency on Xilinx FPGA platforms. This achievement is due to our framework's ability to navigate and tailor optimizations within the HLS space, making it suitable for diverse applications in AI. This paper not only provides valuable insights for FPGA-based transformer implementations but also sets a precedent for broader applications in deep learning, potentially revolutionizing the field of artificial intelligence.

Author Contributions: Methodology, S.J.; Software, S.J.; Validation, S.J.; Formal analysis, S.J.; Investigation, S.J. and Y.C.; Writing—original draft, S.J.; Supervision, Y.C.; Project administration, Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Korea Evaluation Institute of Industrial Technology (KEIT) under the Industrial Embedded System Technology Development (R&D) Program 20016341. The EDA tool was supported by the IC Design Education Center (IDEC), Republic of Korea.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: Author Yongbeom Cho was employed by the company Deep ET. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
2. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going Deeper with Convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
3. TensorFlow. Effective TensorFlow 2. 2021. Available online: https://www.tensorflow.org/guide/effective_tf2 (accessed on 5 November 2021).
4. Hinton, G.; Deng, L.; Yu, D.; Dahl, G.E.; Mohamed, A.-R.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T.N.; et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Process. Mag.* **2012**, *29*, 82–97. [CrossRef]
5. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the NAACL-HLT 2019, Minneapolis, MN, USA, 2–7 June 2019; pp. 4171–4186.
6. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* **2015**, *115*, 211–252. [CrossRef]
7. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
8. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is All You Need. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017.
9. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language Models are Few-Shot Learners. *arXiv* **2020**, arXiv:2005.14165.
10. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An Image is Worth 16 × 16 Words: Transformers for Image Recognition at Scale. In Proceedings of the Ninth International Conference on Learning Representations (ICLR), Vienna, Austria, 3–7 May 2021.
11. Wan, Y.; Zhang, W.; Li, Z. Double Consistency Regularization for Transformer Networks. *Electronics* **2023**, *12*, 4357. [CrossRef]
12. Abd Alaziz, H.M.; Elmannai, H.; Saleh, H.; Hadjouni, M.; Anter, A.M.; Koura, A.; Kayed, M. Enhancing Fashion Classification with Vision Transformer (ViT) and Developing Recommendation Fashion Systems Using DINOVA2. *Electronics* **2023**, *12*, 4263. [CrossRef]
13. Han, S.; Mao, H.; Dally, W.J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv* **2015**, arXiv:1510.00149.
14. Plagwitz, P.; Hannig, F.; Ströbel, M.; Strohmeyer, C.; Teich, J. A Safari through FPGA-based Neural Network Compilation and Design Automation Flows. In Proceedings of the 2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Orlando, FL, USA, 9–12 May 2021; pp. 10–19. [CrossRef]
15. Xilinx Inc. Vitis AI: Development Platform for AI Inference. 2021. Available online: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html (accessed on 1 February 2021).
16. Xilinx Inc. PYNQ: Python Productivity for Zynq. 2021. Available online: http://www.pynq.io/ (accessed on 22 November 2021).
17. Xilinx Inc. SDSoC Development Environment. 2019. Available online: https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html (accessed on 22 May 2019).
18. Xilinx Inc. Vivado Design Suite User Guide: High-Level Synthesis (UG902). 2021. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_1/ug902-vivado-high-level-synthesis.pdf (accessed on 4 May 2021).
19. Qi, P.; Song, Y.; Peng, H.; Huang, S.; Zhuge, Q.; Sha, E.H. Accommodating Transformer onto FPGA: Coupling the Balanced Model Compression and FPGA-Implementation Optimization. In Proceedings of the 2021 on Great Lakes Symposium on VLSI (GLSVLSI '21), Virtual, 22–25 June 2021; pp. 163–168. [CrossRef]

20. Peng, H.; Huang, S.; Geng, T.; Li, A.; Jiang, W.; Liu, H.; Wang, S.; Ding, C. Accelerating Transformer-based Deep Learning Models on FPGAs using Column Balanced Block Pruning. In Proceedings of the 2021 22nd International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 7–9 April 2021; pp. 142–148. [CrossRef]
21. O'Neal, K.; Liu, M.; Tang, H.; Kalantar, A.; DeRenard, K.; Brisk, P. HLSPredict: Cross platform performance prediction for FPGA high-level synthesis. In Proceedings of the International Conference on Computer-Aided Design (ICCAD '18), San Diego, CA, USA, 5–8 November 2018; pp. 1–8.
22. Li, B.; Pandey, S.; Fang, H.; Lyv, Y.; Li, J.; Chen, J.; Xie, M.; Wan, L.; Liu, H.; Ding, C. FTRANS: Energy-efficient acceleration of transformers using FPGA. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '20), Virtual, 10–12 August 2020; pp. 175–180. [CrossRef]
23. Li, Z. Auto-ViT-Acc: An FPGA-Aware Automatic Acceleration Framework for Vision Transformer with Mixed-Scheme Quantization. In Proceedings of the 2022 32nd International Conference on Field-Programmable Logic and Applications (FPL), Belfast, UK, 29 August–2 September 2022; pp. 109–116.
24. Plagwitz, P.; Hannig, F.; Teich, J. TRAC: Compilation-Based Design of Transformer Accelerators for FPGAs. In Proceedings of the 2022 32nd International Conference on Field-Programmable Logic and Applications (FPL), Belfast, UK, 29 August–2 September 2022; pp. 17–23. [CrossRef]
25. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv* **2015**, arXiv:1502.03167.
26. Jang, S.; Liu, W.; Park, S.; Cho, Y. Automatic RTL Generation Tool of FPGAs for DNNs. *Elecstronics* **2022**, *11*, 402. [CrossRef]
27. Nvidia Corp. Available online: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html (accessed on 22 November 2021).
28. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *arXiv* **2017**, arXiv:1712.05877.
29. Zhou, S.; Wu, Y.; Ni, Z.; Zhou, X.; Wen, H.; Zou, Y. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv* **2016**, arXiv:1606.06160.
30. Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P.I.J.; Srinivasan, V.; Gopalakrishnan, K. PACT: Parameterized Clipping Activation for Quantized Neural Networks. In Proceedings of the 6th International Conference on Learning Representations (ICLR), Vancouver Convention Center, Vancouver, BC, Canada, 30 April–3 May 2018.
31. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Playing Atari with Deep Reinforcement Learning. In Proceedings of the Conference on Neural Information Processing Systems (NIPS 2013), Lake Tahoe, NV, USA, 5–10 December 2013.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.