Exploring the applicability of Graph Attention Networks in computer vision and their hardware acceleration

Abdolvahab Khalili Sadaghiani
Linköping University
Linköping, Sweden
0000-0003-4870-2768

Jose Nunez-Yanez
Linköping University
Linköping, Sweden
0000-0002-5153-5481

Abstract— **Edge detection is a fundamental task in computer vision, crucial for object recognition, segmentation, and scene understanding. Traditional methods often fail to capture complex edge structures due to their inability to model intricate relationships between pixels. Graph Neural Networks (GNNs), particularly Graph Attention Networks (GATs), have shown promise in addressing these limitations by leveraging graph structures to model pixel relationships. This paper explores the applicability of Graph Attention Networks in edge detection, highlighting their advantages over ordinary Graph convolutional Networks (GCNs) through rigorous mathematical reasoning. We integrate GATs into an edge detection framework based on an encoder-decoder structure with U-Net architecture and provide detailed theoretical and implementation insights. Furthermore, we discuss the hardware acceleration of GCNs and GATs with a reconfigurable dataflow architecture integrated in the Pytorch framework. The experimental results demonstrate the superior performance of GAT-based edge detection and the potential acceleration possible on reconfigurable edge platforms with limited resources. The key advantage of our proposed method is its hardware-friendly design, making it highly suitable for FPGA acceleration while also enabling efficient optimization through pruning of the network.**

Keywords—edge detection, GNN, Graph Attention Networks, encoder-decoder structure, U-Net

I. Introduction

Edge detection is a critical step in many computer vision applications, serving as the foundation for tasks such as object recognition, image segmentation, and scene interpretation. Traditional edge detection algorithms, like Sobel, Prewitt, and Canny operators, rely on gradient calculations and thresholding techniques. While effective in simple scenarios, these methods often struggle with complex images containing noise, texture variations, and intricate edge patterns.

GNNs have emerged as a powerful tool for modeling relational data by representing input data as graphs and learning representations that capture both local and global structures. GATs, a variant of GNNs, introduce an attention mechanism that assigns learnable weights to the edges in the graph, allowing the model to focus on the most relevant neighboring nodes during message passing.

In this paper, we focus on the following contributions:

1. **Theoretical Advantages of GATs over ordinary GCNs in Edge Detection**: We provide an in-depth mathematical analysis of how GATs improve edge detection performance compared to traditional GCNs.
2. **Implementation Details**: We present a detailed description of integrating GATs into an edge detection framework, including architectural design and training procedures.
3. **Hardware Acceleration**: We discuss how a PYNQ overlay can be used to accelerate the computation required for GCNs/GATs and hardware-aware quantization enhances computational efficiency.

Recent advancements in contour detection have introduced a variety of novel approaches, each addressing specific challenges. Ren *et al.* proposed Sparse Code Gradients (SCG), leveraging sparse coding for improved accuracy but with high computational demands [1]. Isola *et al.* introduced a pointwise mutual information-based affinity measure for crisp boundary detection, achieving pixel-level precision but with sensitivity to noise [2]. Hallman *et al.* utilized random forests to model edge orientation, offering interpretability and efficiency, though limited by training data [3]. Shen *et al.* employed deep convolutional networks with a positive-sharing loss function, capturing semantic features but requiring extensive resources [4]. Bertasius *et al.* combined multi-level features for boundary detection, enhancing performance but with design complexity [5]. Together, these works highlight diverse strategies to improve contour detection while presenting trade-offs in computational efficiency, generalizability, and implementation complexity. In contrast, our method combines the strengths of deep learning with adaptive edge refinement, enabling better generalization and computational efficiency across diverse datasets.

The method proposed in [6] excels in capturing multi-scale features, ensuring robust and accurate edge detection under various conditions. However, its high model complexity can result in increased computational costs, limiting its applicability in real-time or resource-constrained systems. On the other hand, [7] enhances precision and reduces false positives, particularly in complex scenes, by leveraging innovative fusion difference convolution techniques. Despite its accuracy, this method may struggle with high computational demands and scalability for large datasets. Both the approaches in [6] and [7] aim to balance trade-offs between performance and computational efficiency, depending on the application requirements.

The rest of the paper is organized as follows. Section II gives a background and introductory information about important concepts used in this work. Section III explains the GAT mechanism and its function. Section IV explains the algorithmic aspects of the work. Section V presents the experimental results of the algorithm. Section VI introduces initial results of our efforts towards hardware acceleration. Finally, section VII presents the conclusions and future work.

II. Background

A. Edge Detection

Edge detection aims to identify points in an image where the intensity changes sharply, indicating boundaries of objects or textures. Mathematically, edges correspond to discontinuities in the image intensity function I(x,y). Traditional methods compute the gradient magnitude and direction using convolutional kernels, such as (1) and (2).

- Prewitt Operator:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * I, \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * I \quad (1)$$

- Gradient Magnitude:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \quad (2)$$

These methods are limited by their local nature and fixed kernels, which cannot adapt to complex patterns.

B. Graph Convolutional Network (GCN)

GNNs generalize neural networks to graph-structured data. A graph $G = (V, E)$consists of nodes V and edges E. Each node v∈V has a feature vector h_v. The standard GNN updates node features through message passing as (3).

$$m_{uv} = M(h_u, h_v, e_{uv}) \quad (3)$$

where e_{uv} is the edge feature, and M is a message function. Aggregation of the neighboring nodes is as follows in (4).

$$a_v = \sum_{u \in N(v)} m_{uv} \quad (4)$$

where *N(v)* denotes the neighbors of node v. Later nodes ought to be updated as follows in (5).

$$h'_v = U(h_v, a_v) \quad (5)$$

where U is an update function. Two main Limitations of GNN are as follows.

- **Uniform Treatment of Neighbors**: All neighbors contribute equally, which may not capture the importance of specific nodes.
- **Over-Smoothing**: Repeated aggregations can make node features indistinguishable.

C. Graph Attention Network (GAT)

GATs address these limitations by introducing an attention mechanism that assigns different weights to neighbor contributions. Attention coefficient computation is defined as

$$e_{uv} = LeakyRelu(a^T [W h_u \parallel W h_v]),$$

Where W is a weight matrix, **a** is an attention vector, and ‖ denotes concatenation. Normalization is also an important step that needs to be taken into account (6).

$$\alpha_{uv} = \frac{\exp(e_{uv})}{\sum_{k \in N(v)} \exp(e_{uv})} \quad (6)$$

Lastly, feature Update rewrites the values as (7).

$$h'_v = \sigma\left(\sum_{u \epsilon N(v)} \alpha_{uv} W h_u\right) \quad (7)$$

where σ is an activation function. The attention coefficient computed in (6) for each neighbor assigns a degree of importance that can emphasize strong, informative edges while downplaying irrelevant or weak signals. For edge detection, this capacity to differentially weight each node's contribution based on similarity is essential, as it helps in identifying true boundaries while suppressing noise and texture variations that often cause false positives in traditional methods.

As for enhanced expressiveness, GATs are able to model complex relationships and capture high-frequency components (edges) more effectively. High-frequency components correspond to rapid changes in node features (edges) and they are preserved by GATs by assigning lower weights to dissimilar neighbors, thus retaining edge information. Also, the attention mechanism prevents over-smoothing by limiting the aggregation of irrelevant information [8].

III. Attention-based edge detection framework

In this section, we provide an overview of the proposed edge detection framework, which integrates GAT into an encoder-decoder architecture based on U-Net. U-Net is a convolutional neural network architecture designed primarily for image segmentation tasks. The block diagram of the workflow and components of the architecture is depicted in Fig. 1. The implementation is designed to leverage the strengths of both convolutional neural networks and graph neural networks to enhance edge detection performance.

The first step involves constructing a graph representation from the input image or feature maps extracted by the encoder. Each pixel or a group of pixels (super-pixels) in the feature map is considered a node in the graph. The node features are derived from the corresponding pixel's feature vector obtained from the encoder's output at the bottleneck layer. Edges are established between nodes based on spatial adjacency or feature similarity. For spatial adjacency, we can use a 4-connected or 8-connected neighborhood, where each node is connected to its immediate neighbors. Alternatively, feature similarity can be used to connect nodes with similar feature representations, which can help in capturing long-range dependencies. The graph *G= (V, E)* is thus defined, where *V* is the set of nodes and E is the set of edges connecting these nodes.

After the GAT layer updates the node features, the graph is transformed back into a grid structure to be processed by the decoder. The decoder consists of up-sampling operations, typically implemented using transposed convolutions, which increase the spatial dimensions of the feature maps. At each up-sampling stage, skip connections from the encoder are utilized, concatenating high-resolution features from the encoder with the up-sampled features in the decoder. This mechanism preserves spatial details that may have been lost during down-sampling and enriches the decoder's inputs with multi-scale information.

The decoder progressively processes the feature maps, integrating the GAT-enhanced representations with spatially detailed encoder features. This combination enables the model to reconstruct high-resolution edge maps that accurately localize edges while maintaining contextual coherence. To further enhance edge detection performance, the model incorporates a complementary operation using the Prewitt operator. The Prewitt operator is applied to the input image to compute a basic edge map based on intensity gradients. This edge map captures fundamental edge information that might be missed by learned representations, particularly in regions where the model's predictions are uncertain.

The final edge map is obtained by combining the GAT-based edge map with the Prewitt edge map. This combination is performed using weighted summation, where the weights are hyperparameters that can be tuned to balance the contributions of each method. By integrating traditional image processing techniques with advanced neural network outputs, the model leverages the strengths of both approaches, improving robustness and accuracy.

The encoder part of the U-Net consists of several convolutional blocks, each followed by a max-pooling layer, progressively reducing the spatial dimensions of the feature maps while increasing the number of channels.

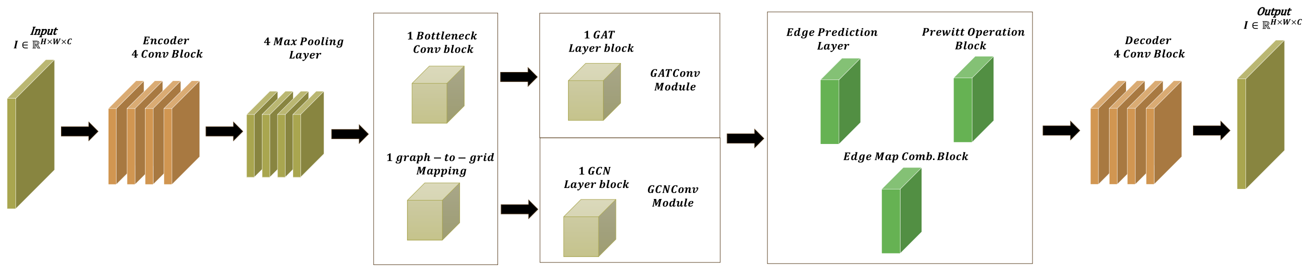

Fig. 1. Block diagram of the workflow and components of the architecture.

The output of the encoder at the bottleneck layer is a feature map $F \epsilon \mathbb{R}^{H_b \cdot W_b \cdot C_b}$, where H_b and W_b are the height and width, and C_b is the number of channels. The feature map F is reshaped into a two-dimensional array $H \epsilon \mathbb{R}^{N.C_b}$, where $N = H_b . W_b$ is the total number of nodes. Each node v_i has a feature vector $h_i \epsilon \mathbb{R}^{C_b}$. Multiple GAT layers are stacked to capture higher-order relationships (8).

$$h'_v = \left\{ \begin{matrix} K \\ k = 1 \end{matrix} \sigma(\textstyle\sum_{u \in N(v)} \frac{1}{|N(v)|} W h_u) \right. \tag{8}$$

where K is the number of attention heads.
h' is the updated feature of node i.
σ is an activation function (e.g., ELU).
α_{ij} is the attention coefficient between nodes i and j.
W is a learnable weight matrix.
N(i) is the set of neighboring nodes of node i.

Edge Prediction Layer outputs a probability map indicating the likelihood of each pixel being an edge [9]. The decoder part of the U-Net reconstructs the spatial dimensions by up-sampling the feature maps and combining them with the corresponding feature maps from the encoder via skip connections. Transposed convolution method is used to up-sample and increase the spatial dimensions. The feature maps from the encoder are concatenated with the up-sampled feature maps at each level, providing the decoder with high-resolution features that aid in precise localization. The GAT-enhanced feature maps are used to generate an edge map E_{GAT} . The Prewitt operator is applied to the input image to obtain $E_{Prewitt}$, capturing basic edge information based on intensity gradients. The final edge map is obtained by combining the two edge maps. To enhance edge detection, we combine the GAT output with the Prewitt operator as (9).

$$E_{Final} = \lambda_{GAT} . E_{GAT} + \lambda_{Prewitt} . E_{Prewitt} \tag{9}$$

where λ_{GAT} and $\lambda_{Prewitt}$ are weighting factors satisfying $\lambda_{GAT} + \lambda_{Prewitt} = 1$. A combination of Binary Cross-Entropy (BCE) loss and Dice loss is used to handle class imbalance and improve edge detection performance as (10).

$$L = L_{BCE}(E_{final}, E_{ground\ truth}) + \beta L_{Smooth}(E_{final}) \tag{10}$$

where L_{BCE} is the Binary Cross-Entropy loss, and L_{Smooth} encourages spatial smoothness.

IV. Novel Encoder-Decoder Structure with U-Net and GNN Integration

In this section we highlight the novelties of the proposed framework in detail. As previously indicated, the proposed edge detection model introduces a novel integration of GATs into the traditional U-Net architecture, forming an encoder-decoder structure that leverages both convolutional and graph-based operations. The encoder path of the U-Net captures local spatial features through successive convolutional layers and pooling operations, effectively reducing the spatial dimensions while increasing the depth of feature representations. This process extracts hierarchical features that are essential for identifying edges at multiple scales [10].

The novelty lies in the incorporation of a GAT layer at the bottleneck of the U-Net architecture, where the feature maps have the lowest spatial resolution but richest feature representations. By converting these feature maps into a graph structure, each pixel (or group of pixels) becomes a node with associated features, and edges are established based on spatial adjacency or feature similarity. The GAT operates on this graph to perform attention-based message passing, allowing the model to capture complex, non-local relationships between distant pixels that standard convolutional operations might miss.

Another key innovation is the seamless fusion of the GAT-enhanced features back into the decoder path of the U-Net. After the GAT processes the graph and updates the node features, these features are reshaped back into a grid format to match the decoder's expected input. The decoder then progressively up-samples these feature maps, using transposed convolutions and skip connections from the encoder layers. The skip connections ensure that fine-grained lost spatial information during down-sampling is preserved, while the GAT-enhanced features provide enriched contextual information. This combination allows the decoder to reconstruct high-resolution edge maps that are both precise and contextually informed.

A. Customized U-Net Architecture

The model introduces a complementary operation by integrating the output of the GAT-enhanced U-Net with the traditional Prewitt operator. U-Net consists of two main parts. First, the encoder path (Contracting Path) captures context by progressively down-sampling the input image through convolutional and pooling layers, extracting high-level features. While the GAT captures complex patterns and non-local interactions, the Prewitt operator provides a simple yet effective method for detecting basic edge structures based on intensity gradients. By combining the outputs of both methods, the model benefits from the strengths of deep learning and classical image processing techniques.

This hybrid approach improves robustness and accuracy in edge detection, particularly in challenging scenarios with noise, texture variations, or subtle edges that might be missed by either method alone. The encoder comprises several convolutional blocks, each consisting of two convolutional layers with a small kernel size (e.g., 3×3), each followed by a rectified linear unit (ReLU) activation. Also, a max-pooling layer that reduces the spatial dimensions by a factor of 2. At each down-sampling step, the number of feature channels is doubled to capture more complex features. The bottleneck is the deepest part of the network where the feature maps have the smallest spatial dimensions but the highest number of channels. This is where the GAT is integrated to enhance the feature representations by capturing global context and complex relationships between features.

The decoder path (Expanding Path) enables precise localization by up-sampling the features and combining them with corresponding features from the encoder via skip connections. The decoder mirrors the encoder structure and consists of an up-sampling step that increases the spatial dimensions, often implemented using transposed convolutions. Also, a concatenation with the corresponding feature map from the encoder via skip connections.

There are two convolutional layers with ReLU activations in which at each up-sampling step, the number of feature channels is halved. Mathematically, the encoder applies a series of convolutional operations as (11).

$$F_{enc}^{(l+1)} = \sigma(W_{enc}^{(l)} * F_{enc}^{(l)} + b_{enc}^{(l)}) \quad (11)$$

where $F_{enc}^{(l+1)}$ is the feature map at layer l, and * denotes convolution$W_{enc}^{(l)}$ and $b_{enc}^{(l)}$ are the weights and biases, and σ is an activation function.

B. *Integration of GAT into the U-Net Bottleneck*

To leverage the advantages of GATs, we integrate a GAT layer into the bottleneck of the U-Net architecture, where the feature maps are at their lowest spatial resolution but richest in features. The bottleneck layer contains rich feature representations but lacks spatial resolution. Incorporating GAT at this stage allows the model to capture complex dependencies and relationships between features, which is particularly beneficial for edge detection where contextual information is crucial. In order to convert and map the features to Graph network, the bottleneck feature-map $F_{bottleneck} \epsilon \mathbb{R}^{H_b \cdot W_b \cdot C_b}$ is reshaped into a graph representation. Each spatial location *(i,j)* corresponds to a node v_{ij} with feature vector $h_{y_i} = F_{bottleneck} \epsilon \mathbb{R}^{C_b}$. Edges are established based on spatial adjacency (e.g., connecting each node to its 4 or 8 neighbors).

C. *Decoder with Skip connections*

The decoder reconstructs the high-resolution edge map by up-sampling the feature maps and combining them with corresponding encoder feature maps through skip connections as (12).

$$F_{dnc}^{(l)} = \sigma(W_{dnc}^{(l+1)} * \left(Upsamle(F_{enc}^{(l+1)}) \oplus F_{enc}^{(l)}\right) + b_{dnc}^{(l)})$$
(12)

where ⊕ denotes concatenation, and *Upsample* is an up-sampling operation (e.g., transposed convolution). The following steps summarize the algorithm:

Algorithm Summary

Input: Image $I \in R^{H \times W \times C}$.

Encoder: Extract feature maps $\left\{F_{enc}^{(l)}\right\}$ through convolutional layers.

Bottleneck:

a. Convert Feature Map to Graph: Nodes V, edges E.

b. Apply GAT Layer: Update node features h'_v.

c. Convert Graph Back to Feature Map: F_{GAT}.

Decoder: Reconstruct feature maps $\{F_{dec}^{(l)}\}$ using up-sampling and skip connections.

Edge Prediction:

a. Decoder Output: Edge map E_{GATE}.

b. Compute Prewitt Edge Map: $E_{Prewitt}$.

c. Combine Edge Maps:

$$E_{Final} = \lambda_{GAT} . E_{GAT} + \lambda_{Prewitt} . E_{Prewitt}$$

The final result is a robust edge detection framework that significantly outperforms traditional methods and standard GNN-based models as shown in the next section, showcasing the effectiveness of integrating GATs within an encoder-decoder architecture for computer vision tasks.

V. Experimental Results

The experimental evaluation of the proposed model focuses on demonstrating its effectiveness in edge detection tasks compared to traditional methods and standard GNN-based models. The experiments are designed to assess the model's performance quantitatively and qualitatively, highlighting the advantages conferred by the integration of GAT within the U-Net architecture and the complementary use of the Prewitt operator.

The BSDS500 dataset is employed for training and testing the model. This dataset is a standard benchmark for edge detection, containing a diverse collection of natural images with manually annotated ground truth edges. The variety in image content and complexity provides a rigorous testbed for evaluating edge detection algorithms. The proposed method is evaluated using the widely accepted ODS, OIS, and AP metrics. It is compared against several competing methods, including [1], [2], [3], [4], [5], [6], and [7] to demonstrate its effectiveness and robustness in edge detection tasks as depicted in table 1.

Table 1. Comparison with other methods on BSDS500 image bank.

Methods	ODS	OIS	AP
[1]	0.739	0.803	0.773
[2]	0.741	0.769	0.799
[3]	0.746	0.770	0.820
[4]	0.757	0.776	0.800
[5]	0.767	0.788	0.795
[6]	0.758	0.771	0.673
[7]	0.730	0.778	0.747
Proposed GCN-based	0.747	0.791	0.788
Proposed GAT-based	**0.771**	**0.809**	**0.827**

These quantitative gains reflect the model's ability to accurately detect edges while minimizing false positives and negatives. Visual inspections of the edge maps produced by the different models reveal that the proposed model generates more precise and continuous edges. The GAT-based model captures fine details and complex edge structures that are often missed by traditional methods and standard GNNs. Edges in regions with subtle intensity variations or intricate textures are detected more reliably. The U-Net structure further complements this approach by preserving spatial details through skip connections, allowing the decoder to reconstruct precise edge maps that combine both global context and fine-grained information. Fig 2,3 and 4 show

how, when applied to different types of images, the GAT-based method particularly excels with colored images containing high-frequency components, such as the Baboon image. These images present a challenge due to their intricate textures and rapid intensity variations. The attention mechanism in GAT allows the model to effectively discern significant edges amidst the high-frequency noise by prioritizing important features and suppressing less relevant information.

This results in a more pronounced improvement over the GCN-based method for the Baboon image compared to grayscale images like Lena and Boat, which have lower frequency components and simpler structures. The GCN-based approach, lacking the adaptive focus provided by attention, struggles to capture the detailed edges in such complex images, leading to less accurate edge detection.

Compared to other competing methods, the proposed GAT-based approach demonstrates enhanced capabilities in handling both colored and grayscale images with varying degrees of complexity. Traditional edge detection algorithms, such as Sobel, Prewitt, or even more advanced methods like Canny, rely on fixed operators and are limited in their ability to adapt to the diverse features present in different images. In contrast, the proposed GAT-based method learns to adaptively weight features based on their importance for edge detection, resulting in more accurate and robust edge maps.

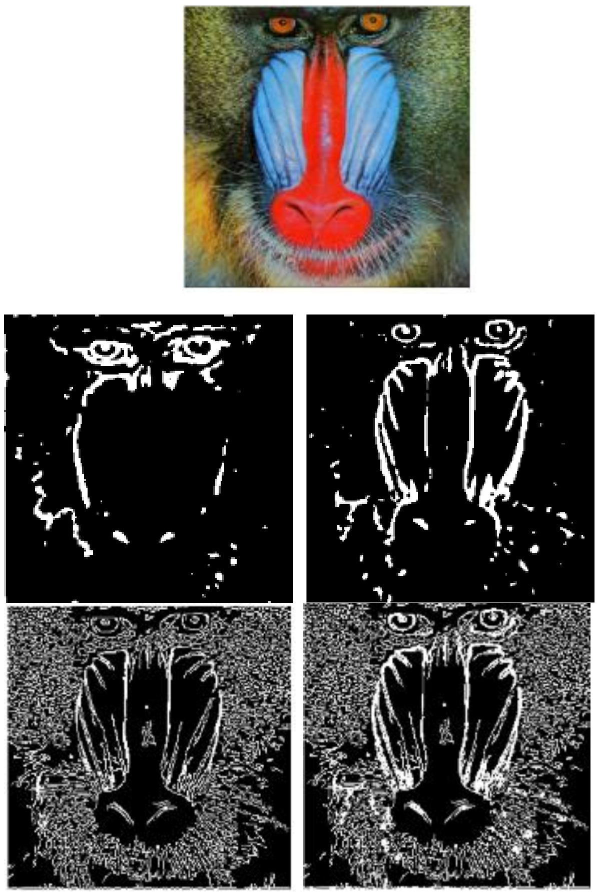
Fig. 2. Input image(top), estimated ground truth, and detected output image of 'Baboon' (GCN on left, GAT on right).

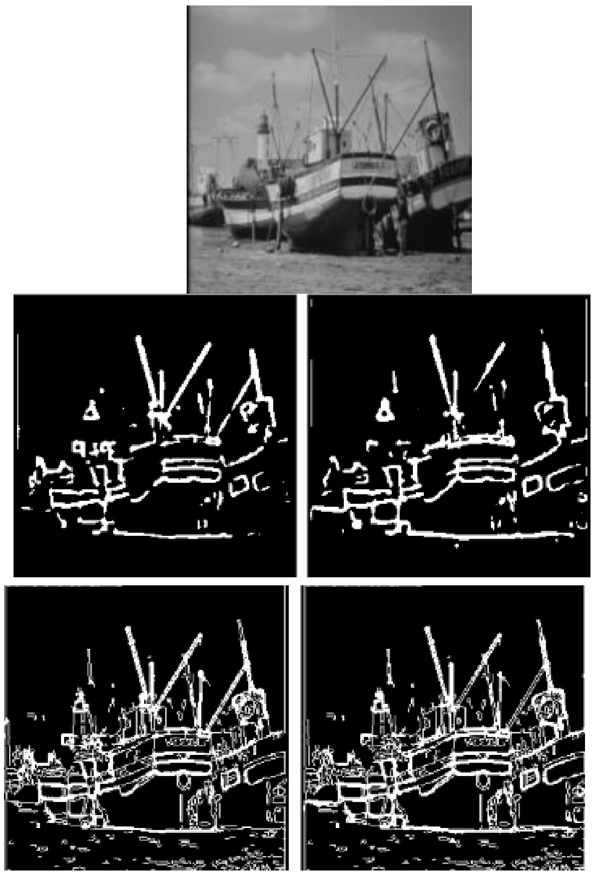
Fig. 3. Input image(top), estimated ground truth, and detected output image of 'Boat' (GCN on left, GAT on right).

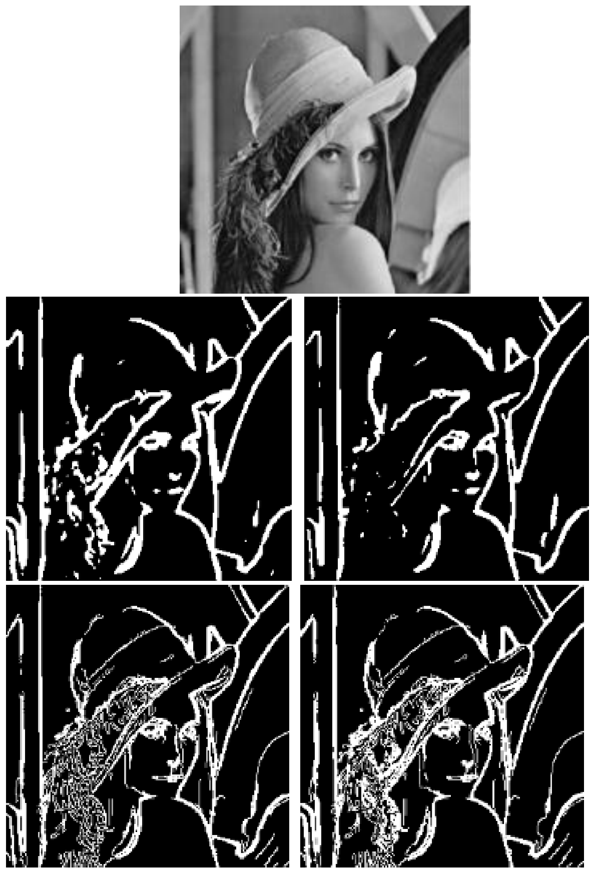

Fig. 4. Input image(top), estimated ground truth, and detected output image of 'Lena' (GCN on left, GAT on right).

The adaptability, combined with the strengths of the U-Net architecture and the attention mechanism, allows the proposed method to outperform traditional techniques and provides a significant advancement in the field of edge detection.

VI. Hardware Acceleration

The algorithm research presented in the previous sections have illustrated the benefits of graph neural networks. These networks are, on the other hand, very compute-intensive as previously shown in [12]. Fig. 1 shows how the algorithm uses Pytorch Geometric layers GATConv and GCNConv. To tackle the compute complexity we have been working on a hardware accelerator integrated in the Pytorch framework that can be used to map these layers transparently to hardware execution using a PYNQ overlay [12]. A Pytorch hardware library exports hardware layers GATConvPYNQ and GCNConvPYNQ and takes care of the data preparation, kernel execution and quantization/dequantization. Fig. 5 depicts a block diagram overview of the accelerator that includes engines for graph aggregation, combination and attention linked via a dataflow paradigm that keeps high hardware utilization and minimizes external memory accesses.

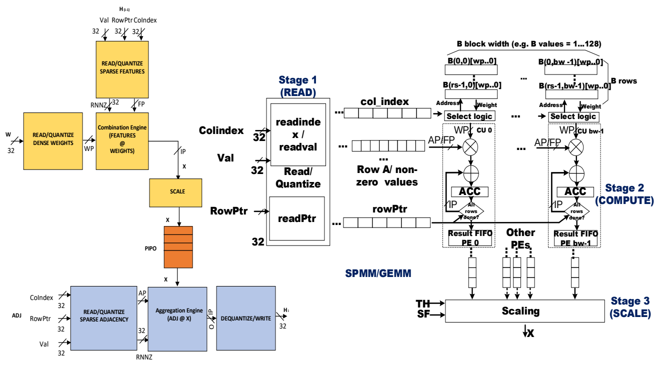

Fig. 5. GNN layer overlay architecture.

We target a Zynq Ultrascale MPSOC device in which the processor runs standard Pytorch/Python code offloading the layer execution to the programmable logic when needed. The hardware is highly configurable and scalable with multiple options possible such as quantization level, the number of hardware threads and the number of computation units per thread. Table 2 shows the complexity of several configurations with 8-bit precision, 4 hardware threads for aggregation and combination and 8,16 or 32 compute units per thread. Each compute unit takes care of processing one column of weight data independently and each tensor computes a fraction of the graph independently.

Table 2. Hardware complexity of 8-bit GNN accelerator.

Configuration	LUTs(K)	FF(K)	BRAM-18Ks	DSP48Es
(4t4t8c)	58	57	160	191
(4t4t16c)	70	68	256	351
(4t4t32c)	93	85	448	671

Initial results obtained with GCNConvPYNQ layers are shown in Fig 6 for two graph layers for different levels of graph sparsity at 99.82%, 99%, 90% sparse. The acceleration factor compares execution time with running the layers in software using the 4-core ARM Cortex A52 processor with optimized SIMD/Neon instructions. We can see that the more parallel configuration with 64 compute units is significantly faster than the narrow 32 unit as the sparsity of graph is reduced. This shows that the computation intensity of the layer increases and the additional hardware can be utilized better.

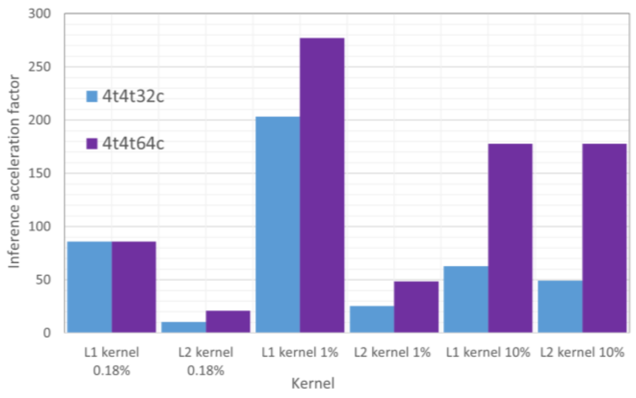

Fig. 6. Comparison of results between two configurations.

Capability of hardware implementation in short word lengths and coefficient lengths is crucial for systems with high computational intensity, offering faster processing and lower latency compared to software-based approaches. It enhances energy efficiency, making it ideal for power-constrained environments. Additionally, it ensures real-time performance and scalability for complex applications [13]. This proposed method stands out due to its exceptional hardware friendliness, which makes it particularly suitable for FPGA acceleration. This characteristic allows for efficient implementation on hardware platforms, ensuring high performance and low power consumption. Additionally, the method's high pruning capability enables significant model optimization by reducing unnecessary complexity without sacrificing accuracy. These features collectively enhance the adaptability and efficiency of the method, making it an ideal choice for hardware-based machine learning applications and real-time processing tasks.

VII. Conclusion

In this paper, we have presented a novel edge detection framework that integrates Graph Neural Networks within a U-Net architecture, leveraging the strengths of both convolutional neural networks and graph-based methods. The theoretical advantages of GATs, such as adaptive neighbor weighting and enhanced expressiveness, were analyzed and demonstrated to be particularly beneficial for edge detection tasks. By combining the GAT-enhanced features with the traditional Prewitt operator, the model capitalizes on both learned representations and classical image processing techniques, achieving superior performance compared to baseline methods.

The experimental results validated the effectiveness of the proposed approach, showing significant improvements in edge detection accuracy and robustness. Additionally, initial evaluation of the potential for acceleration of the graph layers in edge devices highlighted the model's practical viability, enabling efficient computation and performance. The detailed architectural design and implementation strategies outlined in this work provide valuable insights for future research in integrating advanced graph neural network techniques and hardware acceleration for computer vision tasks. These benefits highlight the method's focus on adaptability and performance for hardware-based implementations, aligning with the paper's broader exploration of applying GNNs in computer vision.

REFERENCES

[1] X. Ren and L. Bo, "Discriminatively trained sparse code gradients for contour detection," in *Proceedings of the Neural Information Processing Systems (NIPS)*, 2012, pp. 5–6.

[2] P. Isola, D. Zoran, D. Krishnan, and E. H. Adelson, "Crisp boundary detection using pointwise mutual information," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014, p. 6.

[3] S. Hallman and C. C. Fowlkes, "Oriented edge forests for boundary detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1732–1740.

[4] W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang, "Deep-contour: A deep convolutional feature learned by positive-sharing loss for contour detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–2, 6.

[5] B. Bertasius, J. Shi, and L. Torresani, "High-for-low and low-for-high: Efficient boundary detection from deep object features and its applications to high-level vision," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1–2, 6.

[6] K. Li, Y. Tian, B. Wang, Z. Qi, and Q. Wang, "Bi-Directional Pyramid Network for Edge Detection," *Electronics*, vol. 10, no. 3, p. 329, Feb. 2021. https://doi.org/10.3390/electronics10030329

[7] Z. Yin, Z. Wang, C. Fan, X. Wang, and T. Qiu, "Edge Detection via Fusion Difference Convolution," *Sensors*, vol. 23, no. 15, p. 6883, Aug. 2023. https://doi.org/10.3390/s23156883

[8] D. Z. and S. T., "Edge detection techniques—An overview," *Pattern Recognition and Image Analysis: Advances in Mathematical Theory and Applications*, vol. 8, no. 4, pp. 537–559, 1998.

[9] T. Lindberg, "Edge detection and ridge detection with automatic scale selection," *International Journal of Computer Vision*, vol. 30, no. 2, pp. 117–154.

[10] H. Shu and G. P. Qiu, "More precise edge detection," *arXiv preprint*, arXiv:2407.19992v3, Oct. 2024. Accessed: Nov. 2024. [Online]. Available: https://arxiv.org/abs/2407.19992.

[11] D. A. Mely, J. Kim, M. McGill, Y. Guo, and T. Serre, "A systematic comparison between visual cues for boundary detection," *Vision Research*, vol. 120, pp. 93–107, 2016.

[12] J. L. Nunez-Yanez, "Accelerating Graph Neural Networks in Pytorch with HLS and Deep Pipelined Dataflows," in *Proceedings of the 2023 International Conference on Field-Programmable Technology (ICFPT)*, 2023, pp. 1-8.

[13] Abdolvahab Khalili Sadaghiani and Behjat Forouzandeh, "High-performance power spectral/bispectral estimator for biomedical signal processing applications using novel memory-based FFT processor," *Integration*, vol. 99, p. 102241, Nov. 2024. https://doi.org/10.1016/j.vlsi.2024.102241