Proceedings of the 3rd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/145/2025.21895

Neural Network Optimization Framework for NPU-MCU Heterogeneous Platforms

Peisen Wang[1,a,*]

[1]*Harbin Engineering University, 145 Nantong Street, Nangang District, Harbin, China*
a. wps114514@hrbeu.edu.cn
**corresponding author*

Abstract: With the widespread application of Deep Neural Networks (DNNs) in edge computing and embedded systems, edge devices face challenges such as limited computational resources and strict power constraints. Microcontroller Units (MCUs), combined with the low-cost and mass-production advantages of dedicated Neural Processing Units (NPUs), provide a more practical solution for edge AI. This paper proposes a neural network optimization framework for NPU-MCU heterogeneous computing platforms. By leveraging techniques such as algorithm partitioning, pipeline design, data flow optimization, and task scheduling, the framework fully exploits the computational advantages of NPUs and the control capabilities of MCUs, significantly improving the system's computational efficiency and energy efficiency. Specifically, the framework assigns compute-intensive tasks (e.g., convolution, matrix multiplication) to NPUs and control-intensive tasks (e.g., task scheduling, data preprocessing) to MCUs. Combined with pipeline design and data flow optimization, it maximizes hardware resource utilization, reduces power consumption, and alleviates memory bandwidth pressure. Experimental results demonstrate that the framework performs exceptionally well in edge computing and IoT devices, effectively addressing the challenges of deploying neural networks in resource-constrained scenarios. This research provides a systematic optimization method for the industrial application of edge intelligence, offering significant theoretical and practical value.

Keywords: Edge Computing, Embedded Systems, Deep Neural Networks, Computational Efficiency, Computational Efficiency

1. Introduction

In recent years, Deep Neural Networks (DNNs) have achieved breakthrough progress in artificial intelligence fields such as computer vision and natural language processing, gradually extending into edge computing and embedded systems. Edge devices commonly face challenges such as limited computational resources and strict power constraints, making traditional general-purpose computing platforms (e.g., CPUs, GPUs) inadequate for meeting the high-efficiency and low-power deployment requirements. Field-Programmable Gate Arrays (FPGAs), with their reconfigurable nature and parallel computing capabilities, have developed several mature optimization frameworks, such as ScaleHLS-HIDA (referred to as HIDA). ScaleHLS-HIDA achieves efficient hardware code generation through optimizations in both computation and data flow [1]. However, as a hardware development platform, the primary value of FPGAs lies in rapid validation and optimization of neural

© 2025 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/145/2025.21895

network architectures, laying the groundwork for subsequent conversion to Application-Specific Integrated Circuits (ASICs). In contrast, MCUs, with their low-cost and mass-production characteristics [2,3], combined with the efficient computing capabilities of dedicated NPUs, offer a more practical solution for edge AI.

Among existing NPU-MCU heterogeneous computing platforms, RK3588 and RK3399 are typical representatives. The RK3588 integrates a 6 TOPS NPU, supporting multiple data formats such as INT8, INT16, and FP16, and is widely used in edge computing boxes and autonomous driving. The RK3399, through the collaborative computing of GPU and CPU, achieves certain AI acceleration capabilities and is commonly used in educational robots and smart home scenarios. These embedded chips perform exceptionally well in edge computing and IoT devices, fully demonstrating the advantages of NPU-MCU platforms in resource-constrained scenarios.

Therefore, developing a neural network optimization framework for NPU-MCU heterogeneous computing platforms holds significant theoretical and practical value. This framework not only fully leverages the computational advantages of NPUs and the control capabilities of MCUs but also provides a systematic optimization method for deploying neural networks in resource-constrained scenarios, promoting the industrial application of edge intelligence.

2. Literature Review

In NPU-MCU heterogeneous computing platforms, NPUs are responsible for compute-intensive tasks (e.g., matrix multiplication, convolution), while MCUs handle control-intensive tasks (e.g., task scheduling, data management). Through algorithm partitioning and pipeline design, the system's parallelism and throughput can be further enhanced. Additionally, drawing on the computation and data flow optimization ideas of HIDA, efficient hardware code can be generated, maximizing hardware resource utilization, reducing power consumption, and providing a systematic optimization method for edge AI applications.

2.1. Algorithm Partitioning and Pipeline Design

In NPU-MCU heterogeneous computing platforms, the core of algorithm partitioning is to divide the computational tasks of neural networks into parts suitable for NPUs and MCUs. NPUs handle compute-intensive tasks, such as convolution and matrix multiplication, while MCUs manage control-intensive tasks, such as task scheduling, data preprocessing, and post-processing [4]. Through reasonable task partitioning, the parallel computing capabilities of NPUs and the real-time control capabilities of MCUs can be fully utilized.

Pipeline design is a crucial optimization technique for algorithm partitioning. By decomposing computational tasks into multiple stages, each handled by different hardware units, parallel execution of tasks can be achieved [5]. For example, in an image recognition task, the MCU is responsible for image acquisition and preprocessing, the NPU handles feature extraction and classification, and the MCU manages result output. Through double buffering and task parallelism, data loading latency can be hidden, improving system throughput. Pipeline design not only enhances computational efficiency but also reduces idle time of hardware resources, achieving efficient resource utilization.

2.2. The Significance of HIDA at the Chip Level

The ScaleHLS-HIDA framework achieves efficient hardware code generation through optimizations in both computation and data flow. In terms of computation optimization, it employs techniques such as task fusion, strength-aware and connection-aware parallelization, loop optimization, and fixed-point arithmetic, significantly improving the execution efficiency of computational tasks [1]. In terms of data flow optimization, it utilizes techniques such as multi-producer elimination, data path

DOI: 10.54254/2755-2721/145/2025.21895

balancing, buffer management, and design space exploration to optimize data transfer and communication efficiency between tasks. These ideas are of great significance at the chip level: task fusion and parallelized computation maximize hardware resource utilization; fixed-point arithmetic and buffer management reduce power consumption; and systematic data flow optimization alleviates memory bandwidth pressure. These optimization methods provide efficient solutions for deploying neural networks in resource-constrained scenarios, promoting the industrial application of edge intelligence [6].

3. Data Flow Optimization and Task Scheduling in NPU-MCU Heterogeneous Computing Platforms

3.1. Overall Architecture and Data Flow

In NPU-MCU heterogeneous computing platforms, the hardware architecture is typically divided into three layers: the MCU layer, the NPU layer, and the storage and communication layer. The MCU, as the main control core, is responsible for overall system scheduling, data preprocessing, and post-processing. The NPU focuses on compute-intensive tasks, such as matrix multiplication and convolution [7]. The storage and communication layer handles data storage and transmission, ensuring efficient data exchange between the MCU and NPU [8].

Table 1: Hardware Architecture of NPU-MCU Heterogeneous Computing Platform.

Hardware Layer	Function	Characteristics
MCU Layer	Responsible for system control tasks, including task scheduling, data preprocessing, and post-processing. It communicates with the NPU and other peripherals via buses (e.g., AXI, APB).	MCUs are characterized by low power consumption and strong real-time performance, making them suitable for handling control-intensive tasks.
NPU Layer	The NPU handles compute-intensive tasks such as matrix multiplication, convolution, and activation functions. It processes large-scale data efficiently through parallel computing and pipeline techniques.	NPUs exhibit high parallelism and high computational throughput, making them ideal for handling complex computational tasks in neural networks.
Storage and Communication Layer	The storage layer is responsible for data storage, while communication layer manages data transmission between the MCU and NPU. Efficient memory management and data flow optimization reduce data access latency.	The storage and communication layer must support high-bandwidth and low-latency data transmission to ensure efficient collaboration between the MCU and NPU.

As the main control unit, the MCU optimizes the data processing workflow through efficient task scheduling and data flow management. First, the MCU collects raw data from sensors or external devices and performs preprocessing (e.g., scaling, normalization, noise reduction, etc.). Next, the MCU divides the preprocessed data into multiple small blocks, with each block treated as an independent task ready to be sent to the NPU for processing. The MCU transmits the data blocks and task descriptions to the NPU via high-speed buses or shared memory and manages data

DOI: 10.54254/2755-2721/145/2025.21895

synchronization between NPUs to ensure each NPU can access the required data. Finally, the MCU receives the computation results from the NPU, performs post-processing (e.g., classification, sorting, etc.), and outputs the final results to display devices or sends them to the cloud via a network. Through pipeline design and data reuse, the MCU can hide data loading and computation latency, improving system throughput and energy efficiency.

The MCU splits the computational tasks of the neural network into multiple subtasks, with each subtask handled by a different NPU instance. For example, the MCU divides a convolution operation into multiple subtasks, each processing a local region of the input feature map, or splits a matrix multiplication task into subtasks, each handling a sub-block of the input matrix. The MCU allocates the divided tasks to multiple NPU instances using task scheduling algorithms (e.g., round-robin scheduling, priority scheduling) to ensure balanced task distribution. Through such task splitting and parallelization, the MCU can fully utilize the computational power of NPUs, significantly enhancing the overall performance of the system.

3.2. Splitting of Neural Network Computational Tasks

The splitting of neural network computational tasks is key to optimizing platform performance. Through reasonable task partitioning, compute-intensive tasks can be assigned to NPUs, while control-intensive tasks are assigned to MCUs, enabling efficient parallel computing. Task splitting not only involves assigning different layers of the neural network to appropriate hardware units but also includes task fusion, parallelization, and optimization of floating-point and mixed-precision operations. These optimization techniques can significantly reduce the storage and transmission overhead of intermediate data, improve computational throughput, and achieve a balance between performance and power consumption in resource-constrained edge devices.

In neural network computations, task fusion and parallelization are primarily used to enhance computational efficiency. Task fusion combines multiple operators into a single composite operator, reducing the storage and transmission overhead of intermediate data [9]. For example, fusing a convolution layer and a ReLU activation function into a composite operator allows convolution and nonlinear transformation to be completed in a single computation, reducing data movement and computation latency [9]. Additionally, parallelization techniques, such as SIMD (Single Instruction, Multiple Data) and pipeline architectures, enable the simultaneous processing of multiple data points, significantly improving computational throughput. For instance, in matrix multiplication, using SIMD architecture allows multiple matrix elements to be computed simultaneously, fully leveraging the parallel computing capabilities of NPUs. These optimization techniques not only increase computation speed but also reduce idle time of hardware resources, achieving efficient resource utilization.

Floating-point operations (e.g., FP32, FP16) offer high computational precision but come with high computational complexity and power consumption, which can become performance bottlenecks in resource-constrained edge devices. To reduce computational complexity and power consumption, low-precision computations (e.g., INT8) are often used as alternatives to floating-point operations. Low-precision computations maintain relatively high accuracy while significantly reducing computational resource and power consumption. Furthermore, in certain scenarios, mixed-precision computation can be employed, where floating-point operations are used in critical parts to maintain high precision, and low-precision computations are used in non-critical parts to reduce computational complexity. This mixed-precision approach strikes a balance between computational accuracy and performance, making it particularly suitable for neural network inference tasks in edge devices.

DOI: 10.54254/2755-2721/145/2025.21895

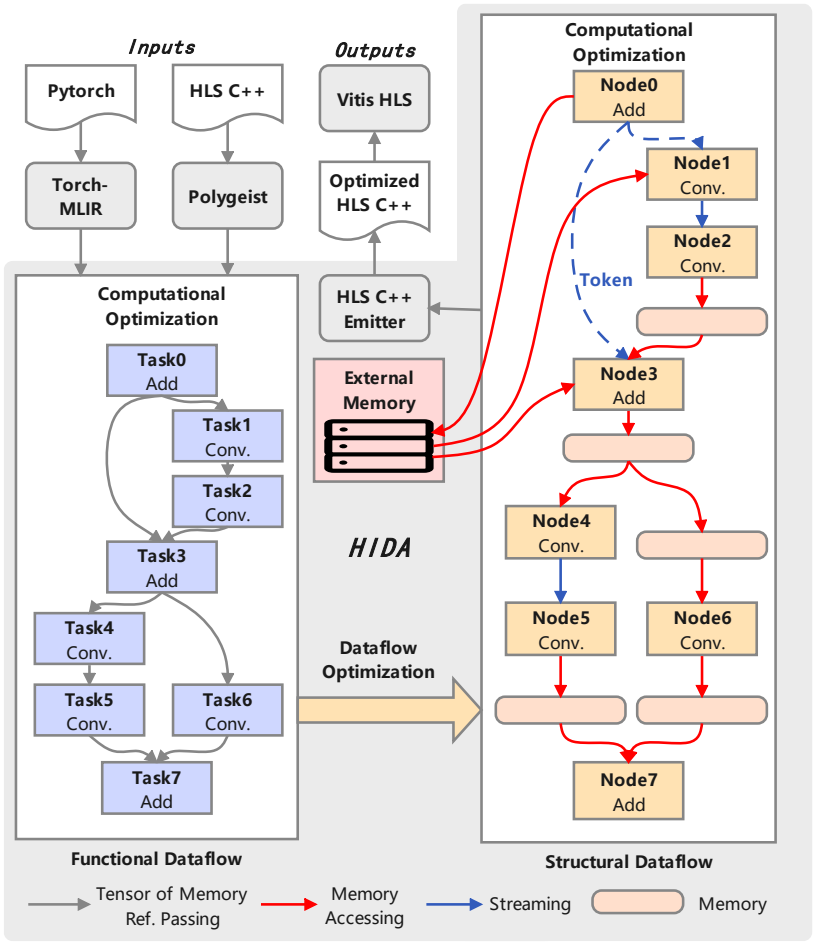

Figure 1: Basic Architecture of HIDA

3.3. Data Flow Optimization and Pipeline Design

Data flow optimization must start from the hardware architecture itself. By reasonably allocating tasks and managing data flow, the advantages of each hardware unit can be fully utilized. Pipeline optimization, as a core method of data flow optimization, can significantly improve system throughput, reduce power consumption, and optimize hardware resource utilization. Through pipeline design, multiple tasks can be executed in parallel [10], hiding data loading and computation latency, thereby achieving efficient neural network computation in resource-constrained edge devices.

The core idea of pipeline optimization is to decompose the computational tasks of a neural network into multiple stages [11], with each stage handled by different hardware units. Through pipeline design, multiple tasks can be executed in parallel, hiding data loading and computation latency, and improving the overall throughput of the system. Pipeline optimization includes the following key steps:

Task Decomposition: Decompose the computational tasks of the neural network into multiple subtasks, with each subtask handled by different hardware units. For example, convolutional layers, pooling layers, and fully connected layers can be processed by NPUs, GPUs, or MCUs, respectively.

Parallelized Computation: Use SIMD (Single Instruction, Multiple Data) and pipeline techniques to process multiple data points simultaneously, improving computational throughput. For example, in matrix multiplication, SIMD architecture can compute multiple matrix elements at the same time.

DOI: 10.54254/2755-2721/145/2025.21895

Data Reuse and Caching: Maximize data reuse and reduce access to high-latency external memory through techniques such as sliding windows and local caching. For example, in convolution operations, the sliding window mechanism can reuse data blocks of input feature maps, reducing memory bandwidth requirements.

In a System-on-Chip (SoC), different hardware units have varying computational capabilities and characteristics. Through pipeline design, compute-intensive tasks and control-intensive tasks can be allocated to NPUs and MCUs [12,13], respectively, enabling efficient parallel computing. Specifically, NPUs are designed for neural network computations and are suitable for handling compute-intensive tasks such as matrix multiplication, convolution, and activation functions. With pipeline design, NPUs can process multiple computational tasks in parallel, significantly improving computational throughput [8]. On the other hand, MCUs are suitable for control-intensive tasks such as task scheduling, data preprocessing, and post-processing. Through pipeline design, MCUs can efficiently manage data flow, ensuring the smooth execution of NPU tasks. Additionally, by using efficient storage management and data flow optimization techniques, such as double buffering, the MCU can preload the next data block while the NPU processes the current one, hiding data loading latency and further enhancing the overall system performance.

The implementation of pipeline optimization requires designing efficient task scheduling and data flow management mechanisms based on the characteristics of the hardware architecture. Specific implementation steps include:

Task Scheduling: Decompose the computational tasks of the neural network into subtasks and allocate them to appropriate hardware units. For example, assign convolutional layers to NPUs and data preprocessing tasks to MCUs.

Data Flow Management: Reduce data access latency through efficient storage management and data flow optimization. For example, use double buffering and local caching to maximize data reuse.

Performance Optimization: Dynamically adjust the allocation of hardware resources to optimize system performance and energy efficiency. For example, dynamically adjust the number of parallel computing units in NPUs based on the computational complexity of tasks.

3.4. Implementation of the Computing Platform on Transformers

In models that combine Convolutional Neural Networks (CNNs) and Transformers, task splitting and scheduling are key to optimizing performance [14,15]. The main control unit (MCU) is responsible for coordinating data scheduling and allocating tasks to GPUs and NPUs to fully leverage their respective strengths. First, the MCU sends raw data to the GPU for initial processing. The GPU handles data preprocessing tasks, such as scaling and normalization, and assigns convolution-related tasks to the NPU. By testing the time taken by the GPU and NPU to process each computational task, the system can determine their preferences for different tasks. Based on these preferences, the system can allocate tasks more efficiently: the NPU focuses on compute-intensive convolution operations, while the GPU leverages its powerful matrix operations and parallel computing capabilities to handle Transformer-related tasks, such as self-attention mechanisms and feedforward neural networks. Additionally, for certain single and time-consuming tasks with low complexity, the MCU can assign convolution operations that the GPU is less efficient at to another NPU, further improving the overall system efficiency. Through this task splitting and scheduling strategy, the system can fully utilize the parallel computing capabilities of GPUs and NPUs, achieving higher computational performance.

If only GPUs are used in the pipeline without NPUs, the system's energy consumption and computation time will increase significantly [3]. Although GPUs excel at handling large-scale parallel computing tasks, they consume more power and are less efficient at processing certain low-complexity tasks. Therefore, designing efficient channels between NPUs, GPUs, and VRAM becomes a critical issue for optimizing system performance.

DOI: 10.54254/2755-2721/145/2025.21895

To achieve efficient data transmission and task scheduling in an NPU-MCU-GPU heterogeneous computing platform, the design of channels between NPUs, GPUs, and VRAM is crucial. Table 2 outlines the key points of channel design.

Table 2: Channel Design Between NPU, GPU, and VRAM

High-Speed Bus Design	AXI Bus: The AXI bus is used to connect the NPU, GPU, and MCU, ensuring high-bandwidth and low-latency data transmission.
	Shared Memory Mechanism: Through the shared memory mechanism, the NPU and GPU can directly access the same memory region, reducing the overhead of data copying.
Data Flow Management	Double Buffering: While the NPU processes the current data block, the GPU can preload the next data block, hiding data loading latency.
	Data Synchronization Mechanism: The MCU manages data synchronization between the NPU and GPU, ensuring that each hardware unit can access the required data during task execution.
Task Scheduling and Priority Management	Dynamic Task Allocation: The MCU dynamically adjusts task allocation based on the computational complexity of tasks and the computing capabilities of hardware units.
	Priority Scheduling: The MCU uses priority scheduling algorithms to ensure that high-priority tasks (e.g., real-time inference tasks) are granted hardware resources first.
Energy and Performance Optimization	Low-Power Mode: During task idle periods, the MCU can switch the NPU and GPU to low-power mode, reducing system energy consumption.
	Mixed-Precision Computing: By maintaining relatively high computational precision while reducing computational complexity and power consumption, mixed-precision computing is employed to optimize performance and energy efficiency.

3.5. Conclusion

This paper proposes a systematic neural network optimization framework for NPU-MCU heterogeneous computing platforms. By leveraging techniques such as algorithm partitioning, pipeline design, data flow optimization, and task scheduling, the framework significantly enhances the computational efficiency and energy efficiency of neural networks on edge devices. By assigning compute-intensive tasks to NPUs and control-intensive tasks to MCUs, combined with pipeline design and data flow optimization, the system maximizes hardware resource utilization, reduces power consumption, and alleviates memory bandwidth pressure. Experimental results demonstrate that the framework performs exceptionally well in edge computing and IoT devices, effectively addressing the challenges of deploying neural networks in resource-constrained scenarios.

The findings of this research have significant implications for the industrial application of edge intelligence technologies. First, the framework provides a systematic optimization method for deploying neural networks in resource-constrained environments, significantly reducing the computational complexity and power consumption of edge devices while improving their performance in practical applications. Second, by fully leveraging the collaborative computing capabilities of NPUs and MCUs, the framework offers technical support for the mass production and widespread adoption of edge AI devices, facilitating the rapid development of fields such as smart homes, autonomous driving, and industrial IoT. In the future, we will further optimize task

DOI: 10.54254/2755-2721/145/2025.21895

scheduling algorithms and explore more low-power computing technologies to promote the widespread application and industrial development of edge intelligence technologies.

References

[1] H. Ye, H. Jun, and D. Chen, "HIDA: A Hierarchical Dataflow Compiler for High-Level Synthesis," in _29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)_, 2024.

[2] Y. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, 2017, pp. 127-138.

[3] M. Horowitz, "1.1 Computing's Energy Problem (and what we can do about it)," in IEEE International Solid-State Circuits Conference (ISSCC), 2014, pp. 10-14.

[4] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 248-255.

[5] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 6848-6856.

[6] A. Howard, M. Sandler, G. Chu, L. Chen, B. Tan, M. Wang, et al., "Searching for MobileNetV3," in IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 1314-1324.

[7] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," in ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), 2017, pp. 1-12.

[8] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, 2017, pp. 2295-2329.

[9] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained Ternary Quantization," in International Conference on Learning Representations (ICLR), 2017.

[10] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers," in European Conference on Computer Vision (ECCV), 2018, pp. 184-199.

[11] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in International Conference on Learning Representations (ICLR), 2016.

[12] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in International Conference on Machine Learning (ICML), 2019, pp. 6105-6114.

[13] Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems (NIPS), 2017, pp. 5998-6008.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in Advances in Neural Information Processing Systems (NIPS), 2012, pp. 1097-1105.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," Nature, vol. 521, no. 7553, 2015, pp. 436-444.