POLITECNICO DI TORINO

MASTER's Degree in ELECTRONIC ENGINEERING

MASTER's Degree Thesis

A Survey: Hardware Neural Architecture Search On FPGA/ASIC

Supervisor

Professor

Mario R. CASU

Candidate

Shulin DENG

JULY 2024

Summary

Deep Learning (DL) technology plays a critical role in various fields. This development is driven by big data, high-performance computing capabilities, and innovative algorithms. Deploying DL models in edge computing scenarios requires efficient hardware platforms such as Field-Programmable Gate Arrays (FPGA) and Application-Specific Integrated Circuits (ASIC), which are essential for meeting the computational demands of modern DL tasks.

This survey aims to provide a comprehensive overview of HW-NAS methodologies, challenges, and trends, with a particular focus on implementations on FPGA and ASIC. The survey explores HW-NAS from three aspects: search space, search algorithms, and evaluation methods.

We hope this survey will serve as a valuable resource for those interested in the latest advancements in HW-NAS on FPGA/ASIC.

Acknowledgements

I am grateful to my supervisor, Professor Mario Roberto Casu for his guidance, to Luca Urbinati for his collaboration.

Special thanks to my family for their love. My parents, Wei Chen and Xiaojun Deng, have always supported me during those challenging times.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep learning (DL) systems are revolutionizing technology across various fields. These breakthroughs are driven by the availability of big data, tremendous growth in computational power, advancements in hardware acceleration, and recent algorithmic innovations. The rapid development of deep learning has spurred demand for efficient hardware implementations capable of handling complex neural network architectures. Due to their flexibility and performance efficiency, Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs) have become key platforms for accelerating deep learning tasks. To fully harness the potential of these hardware platforms, researchers have turned to Neural Architecture Search (NAS), a promising paradigm that automates the design of optimal neural network architectures optimized for specific hardware constraints.

Recently, integrating hardware awareness into the search loop (i.e., HW-NAS) has attracted many researchers and opened up exciting new research directions. Some efforts in HW-NAS have already demonstrated state-of-the-art results, achieving a balance between accuracy and hardware efficiency. Neural architecture search on FPGAs and ASICs involves exploring a wide design space to discover architectures that maximize performance metrics such as accuracy, speed, and energy efficiency. This survey comprehensively examines methodologies, challenges, and trends in hardware-aware NAS, focusing particularly on implementations on FPGAs and ASICs.

HW-NAS consists of three key components[1]: first, the search space defines the types and structures of DL architectures to form effective networks. Second, search algorithms employ multi-objective optimization strategies such as evolutionary algorithms or reinforcement learning to sample network architectures. Finally, evaluation methods calculate DL performance and efficiency metrics, such as accuracy and hardware specifications on target platforms. HW-NAS addresses multi-objective optimization problems aimed at balancing accuracy, inference latency, and energy consumption constraints. Evaluation results guide search

strategies towards promising architectures within the search space. HW-NAS can provide a range of advantageous solutions rather than a singular optimal solution, making multi-objective optimization strategies particularly valuable in edge computing scenarios.

This survey primarily reviews literature on hardware-aware NAS algorithms targeting ASICs and FPGAs. HW-NAS requires interdisciplinary knowledge including device-specific compilation, hardware microarchitecture, neural network design, and efficient NAS algorithms. While many AutoML and NAS review papers focus on theoretical concepts of architecture search, few extensively discuss hardware-based approaches. This survey covers research reported in literature from 2019 to 2024, providing a concise overview of HW-NAS.

Our review is divided into several sections. In Section III, we focus on relevant HW-NAS search spaces. The search space consists of neural network architectures that determine how neural network operators are connected to form effective networks and which operators are allowed. In Section IV, we briefly outline common search strategies and how they explore the search space by sampling candidate neural network architectures. In Section V, we discuss how evaluation methods assess the performance of architectures across various metrics. We hope this review will be helpful to those interested in the latest advancements in HW-NAS on FPGA/ASIC.

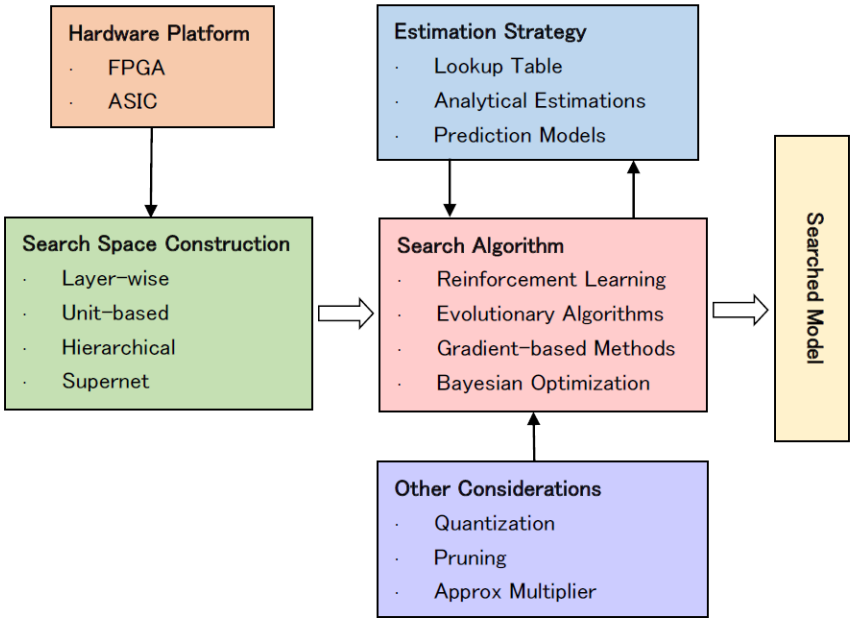

Figure 1.1: Overview of HW-NAS techniques

Chapter 2

Related Work

2.1 Deep Learning

Deep Learning (DL), as a powerful machine learning technique, has made significant strides in recent years, profoundly impacting technology and societal life. Its success stems from the extensive research and understanding of neural network models, especially with the support of large-scale data and high-performance hardware, enabling revolutionary advancements across nearly every scientific domain.

Deep Neural Networks (DNNs), as core tools of deep learning, have been widely applied in tasks such as object classification, detection, and recognition. These models achieve superior performance by extracting high-level features through hierarchical abstraction and learning processes.

In recent years, deep learning systems have garnered widespread research interest and expanded beyond academic labs into industrial and real-world applications. Deployment of most deep learning models has become feasible through cloud and on-premise data centers. However, with the proliferation of Internet of Things (IoT) devices and increasing demands for real-time responses, edge computing is emerging as a new focus and challenge.

Edge devices face constraints in energy and computational power. For instance, autonomous vehicles require real-time object detection and cannot tolerate delays caused by data transmission to and from cloud processing. Simultaneously, there exists a significant mismatch between edge computing capabilities and the complexity of deep learning models, prompting researchers to seek innovative methods to reduce model size, lower floating-point operation requirements, and minimize inference latency.

2.2 NAS

Neural Architecture Search (NAS) is a critical research direction within automated deep learning network design. With the rapid automation of deep learning, the importance of designing network architectures has been increasingly recognized. While modifying architectures can significantly improve the performance of deep learning methods, finding the right architecture itself is a time-consuming, complex, and error-prone task.

Over the years, handcrafted models have excelled in accuracy. However, as the scale and complexity of deep learning models increase, so does the complexity of handcrafted models, such as the number of parameters. To reduce the complexity and subjectivity of manual design, and to improve model performance and design efficiency, the machine learning community has actively explored methods for automating the design of powerful deep convolutional networks. Automated Machine Learning (AutoML)[2], as an approach for automated machine learning model network design, has gained widespread attention. NAS, as a significant branch of AutoML, aims to automatically explore predefined network structure search spaces to find optimal network configurations.

Neural Architecture Search (NAS), as a significant research direction within AutoML, marks the transition of neural network architecture design from manual to automated. In 2016, pioneering NAS work proposed using a recurrent neural network (RNN)-based controller to generate sub-networks and train them using reinforcement learning (RL)[3]. Once trained, this controller can automatically explore predefined search spaces to find optimal network configurations. Experimental results on datasets like CIFAR-10 demonstrate that NAS can generate network structures competitive with manually designed convolutional networks for tasks such as image classification, semantic segmentation, and machine translation.

Classic NAS methods use an RNN as a controller to generate sub-networks, which are then trained and evaluated for their network performance, after which the controller parameters are updated. Traditional NAS methods are simple and have achieved significant results. However, to fully exploit the potential of each sub-network, every time a sub-network is sampled, the controller must initialize and train network weights from scratch. Therefore, traditional NAS methods have a fatal flaw: they are extremely computationally intensive. For example, searching on a small dataset like CIFAR-10 can take 800 GPUs for three to four weeks. Such enormous computational demands are daunting. To address this issue, Efficient Neural Architecture Search (ENAS) proposed the weight-sharing paradigm, significantly improving NAS search efficiency. The weight-sharing NAS paradigm can also be used for evolutionary architecture search, establishing an evolutionary search engine on a pretrained over-parameterized supernet to find optimal architecture candidates.

In 2018, researchers from CMU and Google proposed Differentiable Architecture Search (DARTS), a method that leads the latest advancements in NAS with its powerful search efficiency and reliable search performance. DARTS is a gradient-based neural architecture search method. Its main idea is to introduce a set of architectural parameters, relaxing the discrete search space into a continuous space, thereby allowing simultaneous optimization of network weights and architectural parameters using stochastic gradient descent (SGD). Once differentiable optimization converges, we can derive optimal architectural candidates based on the learned architectural parameters. The success of DARTS further catalyzed extensive subsequent differentiable NAS work, continuously advancing search performance.

2.3 HW-NAS

Hardware-Aware Neural Architecture Search (HW-NAS) is an essential technique within the field of Automated Machine Learning (AutoML), particularly making significant strides in popularizing artificial intelligence (AI) and edge computing.

With the increasing diversity of hardware, designing efficient neural networks for different platforms becomes crucial. Traditional NAS methods, while pursuing model accuracy, often neglect practical hardware constraints, resulting in models that are challenging to deploy on edge devices. Thus, Hardware-Aware Neural Architecture Search (HW-NAS) emerged to incorporate characteristics of target hardware platforms into the NAS process. Common hardware cost metrics include inference latency, power consumption, energy consumption, model size, and memory bandwidth. This approach helps build AI models deployable on resource-constrained edge devices such as IoT, mobile devices, and embedded systems.

Initial HW-NAS methods focused primarily on modifying existing NAS algorithms to consider hardware characteristics. Researchers explored how to incorporate hardware constraints into the NAS search space to optimize model inference latency, energy consumption, and memory usage metrics. With the development of computer hardware and algorithmic technologies, HW-NAS methods have gradually evolved from simple hardware-constrained optimizations to more complex multi-objective optimization approaches. For example, introducing evolutionary algorithms, multi-objective optimization strategies, or reinforcement learning techniques to balance and optimize across multiple performance metrics, thereby obtaining efficient neural network structures applicable to various hardware platforms.

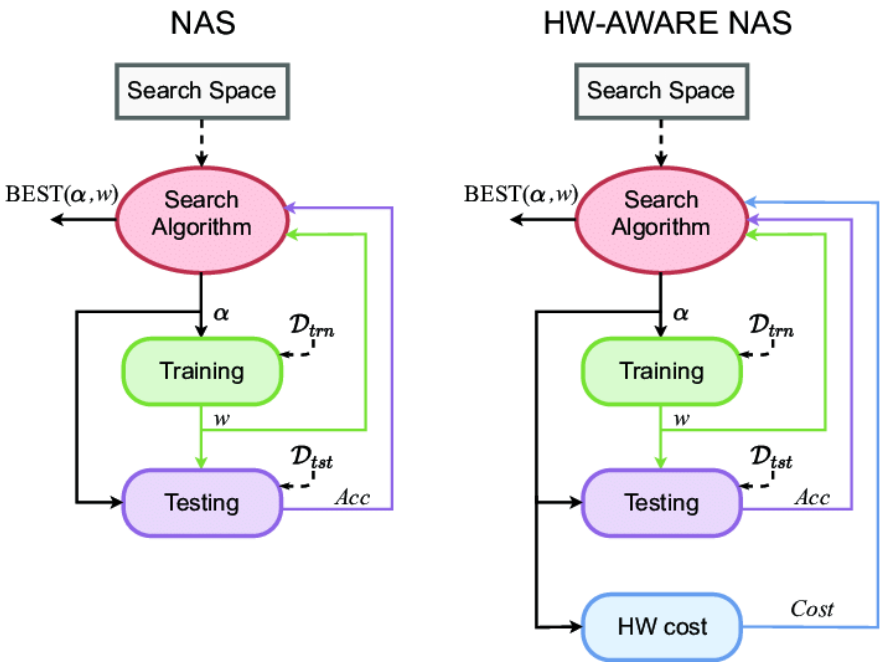

Figure 2.1: Single-objective NAS (left) and HW-NAS (right) [4]

2.4 FPGA and ASIC

FPGA (Field-Programmable Gate Array) and ASIC (Application-Specific Integrated Circuit) are two common integrated circuit technologies in hardware design. FPGA is a programmable logic device that allows defining its functionality and connectivity through programming. In contrast, ASIC is an integrated circuit specifically designed for a particular application, typically featuring customized hardware functions and optimized performance.

FPGA and ASIC each have their strengths and weaknesses in hardware design. FPGA is known for its flexibility and programmability, making it suitable for rapid prototyping and applications that require frequent updates. ASIC excels in power efficiency and performance optimization due to its customized design and specialized capabilities, particularly well-suited for highly optimized specific application domains.

Traditional NAS methods often overlook specific hardware platform constraints, resulting in models that are difficult to effectively deploy on resource-constrained hardware such as FPGA and ASIC. HW-NAS addresses this by incorporating hardware constraints during the search process, optimizing neural network structures to better fit the specific resources and performance requirements of FPGA and ASIC. This optimization significantly enhances model inference efficiency, reduces

	Microprocessor	FPGA	ASIC	GPU
Example	ARM Cortex-A9	Virtex Ultrascale 440	Bitfury 16nm	Nvidia Titan X
Flexibility during development	Medium	High	Very high	Low
Flexibility after development[1]	High	High	Low	High
Parallelism	Low	High	High	Medium
Performance[2]	Low	Medium	High	Medium
Power consumption	High	Medium	Low	High
Development cost	Low	Medium	High	Low
Production setup cost[3]	None	None	High	None
Unit cost[4]	Medium	High	Low	High
Time-to-market	Low	Medium	High	Medium

[1]E.g. to fix bugs, add new functionality when already in production

[2]For a sufficiently parallel application

[3]Cost of producing the first chip

Figure 2.2: Comparison of typical microprocessor, FPGA, ASIC and GPU [5]

power consumption, and minimizes memory usage, thereby maximizing hardware resource utilization.

The flexibility of FPGA is a major advantage, allowing developers to optimize and adjust neural network architectures according to specific requirements without the need for hardware redesign. FPGA can provide substantial computational performance at relatively low power consumption, particularly suitable for edge computing and embedded systems like smart cameras and sensors. ASIC, on the other hand, is a dedicated integrated circuit designed for specific applications, capable of achieving highly customized neural network acceleration through hardware optimization. This customized design typically offers significant advantages in power efficiency and performance.

Although ASIC design and production costs are typically higher, and once designed, ASICs cannot be altered, they offer substantial cost-effectiveness and performance advantages in large-scale production. This makes ASIC particularly suitable for high-performance applications requiring long-term deployment.

Chapter 3

Search Space

In this review, we use the term "search space" to refer to the space containing all hyperparameters of the neural architecture model. Common parameters to consider include layer types, layer connections, activation functions, kernel sizes, and number of kernels, among others, with the aim of finding high-performance architectures. The design of the search space is a fundamental component of Hardware-NAS (HW-NAS), as its performance heavily relies on the quality of the search space. The search space typically serves as the initial step in setting up HW-NAS, defining a set of basic network operators and their connectivity to construct the model's computation graph. It predefines the scope of the search, indirectly determining the performance ceiling of NAS algorithms.

Traditionally, researchers have followed heuristic methods for NAS search space design. In HW-NAS, the design of the search space often represents a crucial trade-off between human biases and search efficiency. When the search space is large, the search complexity increases with the number of elements, requiring longer algorithm runtime but also potentially discovering novel architectures.

3.1 Architecture Search Space

The architecture search space is extensive. When designing accelerators for a given application, numerous hyperparameters need to be selected for their values to identify high-performing architectures. In general, it defines a set of basic network operators and how these operators connect to build the model's computation graph. We distinguish between two approaches to design architecture search spaces [6]:

1) Fixed Architecture Hyperparameter Optimization : In this approach, researchers define a fixed neural architecture in advance. The goal is solely to optimize hyperparameters of the architecture's network structure and each operation's hyperparameters (e.g., number of layers, convolution types, pooling types,

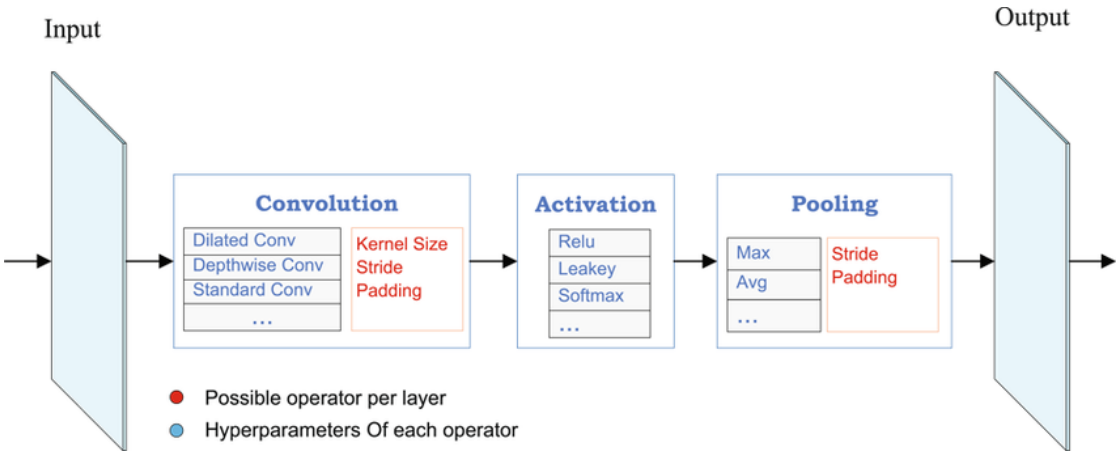

Figure 3.1: An example for generic CNN architecture for each layer [7]

activation functions), without considering the hardware involved. This approach reduces search time and computational resource consumption but may introduce strong human biases.

2) Real Architecture Search Space: In contrast, the real architecture search space is more flexible. The search space allows the optimizer to choose connections between operations and can change the types of operations within each layer. The actual architecture search space is vast. In AnaCoNGA [8], their Quantization Strategy Search (QSS) also includes the quantization strategies for CNNs. This method enables the optimizer to explore various possible connections and operation types freely within the specified search space. However, the increase in search space also implies significantly higher time and computational resource requirements for the search process.

In summary, fixed architecture parameter optimization is suitable for scenarios where there is already a basic structure in place, focusing primarily on optimizing hyperparameters of predefined architecture. On the other hand, the real architecture search space offers greater flexibility, allowing the optimizer to explore a wider range of network structures and operation choices to achieve superior neural network architecture designs.

Typically, the latter approach is further categorized into three types:

- Layer-wise Search Space: In layer-based architectures, each layer comprises a set of selectable operations or layer types, serving as the unit of selection that can only connect to its immediate layers. The entire model is generated from an operator pool. For example, the FBNet search space includes a layer-wise search space with a fixed macro architecture of 22 layers and fixed dimensions for each layer, reducing the search space by limiting options and fixing layer types to shorten

search time.

There are many search spaces of this type. For example, in EdgeTran [9] framework, they designed the FlexiBERT 2.0 architecture as an extension of the design space for Transformer models. FlexiBERT 2.0 is an expansion and optimization of the traditional BERT model, which includes the introduction of various attention mechanisms and operation types, such as weighted multiplicative attention, linear transformation attention, and dynamic span convolutions. These extensions increase the model's flexibility and expressive power, allowing the use of different types of attention operations within a single encoder layer.

The search space of the HALOC [10] framework is achieved by constructing an over-parameterized network, which utilizes a NAS-inspired automatic rank selection method, employing iterative sampling and evaluating different candidate rank settings to differentially learn the most suitable rank. They built an over-parameterized network containing various rank combinations through Tucker-2 decomposition, and these candidate rank combinations form a very large search space.

- Unit-based Search Space: Advanced models like CNNs, designed manually, are often composed of stacked repeating units to form larger and deeper architectures. Thus, rather than searching for entire network architectures from scratch, Zoph et al. proposed searching for relatively small units/modules and stacking the discovered units multiple times to form the final overall architecture. A unit typically represents a small acyclic graph that represents a feature transformation, where each edge in the unit is an operation from a predefined search space. This search space is suitable for tasks that require efficient and rapid design and optimization of network structures, such as when resources are limited or demands change frequently, thus improving efficiency and performance through unit reuse.

FGNAS's [11] search space falls into the category of unit-based search space. FGNAS defines a fundamental operational process for graph neural networks, where each layer consists of three independent stages, which can be viewed as repeated fixed architectural patterns or units. Specifically, each unit includes embedding linear transformation, message weighting, and feature aggregation operations, which are reused to construct larger and deeper graph neural network architectures. In Haiyan's work [12], they used VGG blocks and RepVGG blocks as basic units to construct the entire network architecture and defined the configurations of these basic units at different stages.

- Hierarchical Search Space: Focuses on structural organization between layers and overall design. It involves designing patterns at different levels, where each higher-level pattern is often represented as a DAG of lower-level patterns. Hierarchical search space is typically divided into three steps: defining units, building larger blocks containing a certain number of units, and finally, designing the entire model using generated units.

For example, in HAO [13], they use subgraphs to construct neural network architectures. Here, subgraphs refer to combinations of operations. This search space views the network structure as composed of multiple modules, providing higher flexibility and customizability. It allows mixing precision quantization with different bit widths at different layers, significantly increasing the search space.

In the work of Huizheng and Lingli [14], they proposed a tree structure modeling method to handle the vast design space of HLS (High-Level Synthesis) instruction configurations. The tree structure design space represents the relationships between parameters at different levels. This method can automatically eliminate invalid configurations. The tree structure indicates that certain sub-parameters are only valid when their parent parameters are set to specific values, thus only the valid parameters in the tree structure need to be evaluated, significantly reducing the scale of the search space. They then used a surrogate model, the Multi-Objective Tree-structured Parzen Estimator (MOTPE), for efficient design space search and to model the complex nonlinear relationships between instructions and design objectives. To uniformly represent instruction parameters in the design space, they also adopted float encoding. Through scaling and rounding down, these float values can be converted into corresponding instruction descriptions, simplifying parameter handling during the optimization process.

- Supernet Search Space: The concept of a supernet is to create a large possible architecture space that can be efficiently explored to find the best network for a given task. In a supernet, the weights of subnetworks are not fixed but learned as hyperparameters during training[15]. This allows for a more flexible search space that can better adapt to specific task requirements.

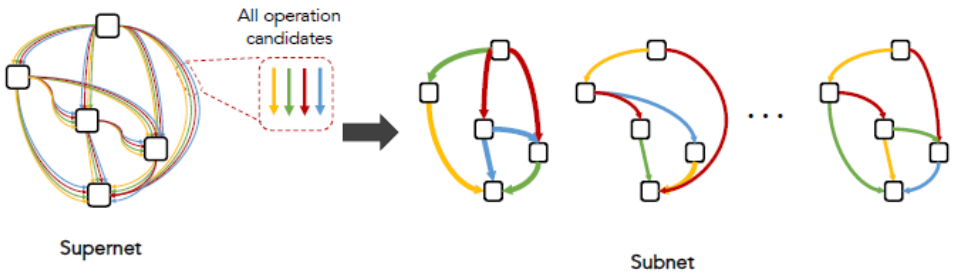

Figure 3.2: Supernet and Subnet Visualization in Neural Architecture Search [3]

For example, in Taehee and Elliott's work[16], they constructed a search space based on a supernet. In the search space, they optimized the number of layers per block and the number of channels per layer while considering the given input image size. The size of the search space is very huge. They built a supernet based on VGG-19, replacing the flattening layer in the original VGGNet with global average pooling

to meet the memory bandwidth requirements of resource-constrained devices. In DeepBurning-MixQ[17], they defined the range of quantization bit widths for each layer model and created a supernet that includes all possible quantization branches. These branches are adjusted by weights, using backpropagation to optimize the quantization settings of MPNN.

In the work of Mohamed et al.[18], they designed Codesign-NAS, which can combine multiple search space structures depending on optimization goals. For part of the CNN model, Codesign-NAS can automatically insert operations, such as adding or connecting operations according to rules, to optimize model performance. It can start searching for the overall network structure from a hierarchical structure and then optimize local details at the unit level, finally conducting a comprehensive search and evaluation through the supernet.

The S3NAS[19] method proposed a novel supernet structure design. In this supernet structure, the number of blocks at each stage can vary to accommodate the latency-accuracy trade-off requirements of different hardware platforms. Within each block, there can be parallel depthwise convolution layers with different kernel sizes (MixConv) to enhance the network's flexibility and adaptability.

ETNAS[20] utilizes an improved search space design based on the supernet architecture of the Differentiable Architecture Search (DARTS) method. The search space of ETNAS includes an operation set, lightweight candidate cells, fixed blocks, and trainable blocks. These improvements in cell design make the network more hierarchical and flexible in structure, allowing it to better adapt to different computational resources and task requirements.

3.2 Hardware Search Space (HSS)

Due to differences in underlying hardware, operations within the search space need to adapt to specific hardware platforms. The latency of the same operation can vary significantly across different devices, even within the same type of hardware. Resource-constrained devices may also fail to meet certain operation requirements. Therefore, many HW-NAS methods include a Hardware Search Space (HSS) component. This component transforms and optimizes hardware specifications through various algorithmic transformations before evaluating the model, tailored to fit hardware designs. While co-exploration is effective, it significantly increases the time complexity of the search space. For example, in the case of FPGAs, their design space may include IP instance types, IP reuse strategies, quantization schemes, parallel factors, data transfer behaviors, slice parameters, and buffer sizes. Considering all these options as part of the search space might be impractical due to increased computational costs. Hence, many existing strategies are limited to only a few options.

The Hardware Search Space (HSS) can be further classified into the following categories[6]:

- Parameters-based: The search space is formalized by a set of different parameter configurations. In hardware search spaces involving FPGA and ASIC, the hardware architecture is defined and optimized through numerous adjustable parameters, such as on-chip memory size (memfp), the number of compute resources (compfp, such as DSPs), and the bandwidth between off-chip and on-chip memory (BWfp), among others.

The design of hardware search spaces is diverse. For example, the search space in HAO[13] includes considerations for different hardware configurations, such as the number of layers, channel sizes, and input resolution parameters. In the work of Nilotpal et al[21]., they used Floating Point Operations (FLOPs) as a hardware cost metric to compare with existing HW-NAS methods. In the work of Mohamed et al [18], they obtained FPGA accelerator parameters from the CHaiDNN library, configuring parameters such as the number of convolution engines, buffer depth, external memory interface width, and the parallelism of filter and pixel dimensions. Considering the design of hardware accelerators, appropriate hardware optimization options can be chosen based on the resource constraints and performance requirements of the target FPGA, such as the utilization of DSP blocks, BRAM, and CLB. They also introduced the ratio conv engines parameter to adjust the DSP allocation ratio among different convolution engines. In FGNAS[11], they further chose the size of feature groups as a key parameter to extend the hardware.

In the work of Haiyan et al.[12], their adjustable parameters included compute elements (CE), processing elements (PE), the number of SIMD units, and row buffer length, among others. By adjusting these parameters, they defined different hardware architectures and designed a reconfigurable streaming-based neural network accelerator. They also used a DMA engine to exchange data between off-chip and on-chip memory, leveraging FIFO to achieve data stream processing. In EdgeTran framework[9], they designed a flexible BERT accelerator framework called ELECTOR. The ELECTOR framework includes multiple hardware modules, forming a vast design space through different combinations. The ELECTOR design space supports various hardware configuration options, including Batch Tile Size, Memory Types, Number of MAC Lanes Per PE, etc. In the work of Grace et al.[22], their search space included various mapping parameters for DNN accelerators, such as tile sizes, loop orderings, and spatio-temporal mappings. These parameters are directly related to how the hardware accelerator executes specific algorithms.

In the work of Panjie et al.[23], their search space also covered the model compression ratio (sparsity ratio). The paper defined the search space by considering a combination of model compression ratios and hardware resource utilization through the proposed algorithm-hardware co-design framework. Specifically, the

search space encompassed different combinations of model compression ratios (such as sparsity) and hardware acceleration schemes (such as FPGA). The sparsity ratio determines the pruning level of the model, affecting the model's size and computational complexity. Meanwhile, hardware parameters such as parallelism factors and the number of digital signal processors (DSPs) impact the execution efficiency and resource utilization of the model on different hardware devices. These combinations can select the optimal device configuration under specific latency and accuracy constraints while maintaining inference accuracy.

- Template-based: In this scenario, the search space is defined as a set of pre-configured templates. Templates are first defined, then adjustable parameters within the templates are determined, and finally, search algorithms are used to explore the parameter space. NASAIC integrates NAS with Application-Specific Integrated Circuits (ASICs). Their hardware search space includes several successfully designed templates. The objective is to find the best model across all templates and different possible parallelization. In current GNN accelerator designs, they also define a template and explore optimal configurations by adjusting template parameters. Template-based hardware search space is an efficient hardware design approach, significantly reducing design complexity and improving optimization efficiency. Below are examples of template-based search spaces.

HotNAS[24] performs NAS based on a selected model, ensuring it meets given time constraints and maintains high accuracy. Its sub-tool iSpace provides a search space that includes filter patterns, channel pruning, quantization, filter expansion, and hardware design.

In a conference paper at ICLR 2022[25], to ensure efficient operation on the target hardware platform, the search space excludes some operators that perform poorly or are unsupported on the target hardware, such as batch normalization, depthwise convolution, squeeze-and-excitation, and Swish activation. They form larger architectures by stacking predefined blocks and optimize these blocks to meet the performance requirements of specific hardware.

In ASICNAS framework[26], they emphasize the importance of heterogeneous accelerator design. Their hardware search space is primarily based on a set of successful existing ASIC design templates, each representing a specific dataflow style. Each template corresponds to a specific dataflow style and defines specific hardware structures and dataflow methods. For instance, Shidiannao is suitable for high-resolution activation channels, the NVDLA style is suited for low-resolution activation channels and large numbers of activation channels, and the Eyeriss style performs well in various dataflow scenarios.

Chapter 4

Search Algorithm

The search strategy defines how the search space is explored. Its primary goal is to construct the model that best fits the given dataset from the vast number of models available in the search space. In this section, we will discuss all the search strategies used in hardware-aware NAS.

4.1 Reinforcement Learning

Many HW-NAS methods use reinforcement learning to search for optimal architectures because the NAS problem can be easily modeled as a Markov decision process. In reinforcement learning, a Recurrent Neural Network (RNN) controller is first used to generate a description of the neural network. This RNN is trained via an RL algorithm to maximize the performance of the generated architecture. The RL controller samples an architecture from the search space and receives rewards based on its accuracy and hardware cost. The agent then adjusts its weights to generate better models. Different works vary in how they represent the agent's policy and how they optimize it. This process is repeated over many iterations.

In HotNAS framework[24], they use an RNN-based reinforcement learning optimizer. A design space is created based on their iSpace tool, and an RNN controller is designed accordingly. Specifically, the controller consists of a softmax classifier that predicts the hyperparameters for each search space in iSpace. The predicted hyperparameters determine specific neural networks and hardware designs, which can be rewarded based on accuracy and latency. The search process optimizes the controller by adjusting its parameters to maximize the expected reward. The parameters are updated using the policy gradient method to predict better architectures over a series of episodes.

In the work of Panjie et al.[23], their RL controller is based on a Recurrent Neural Network (RNN). In each search cycle (episode), the controller predicts and selects

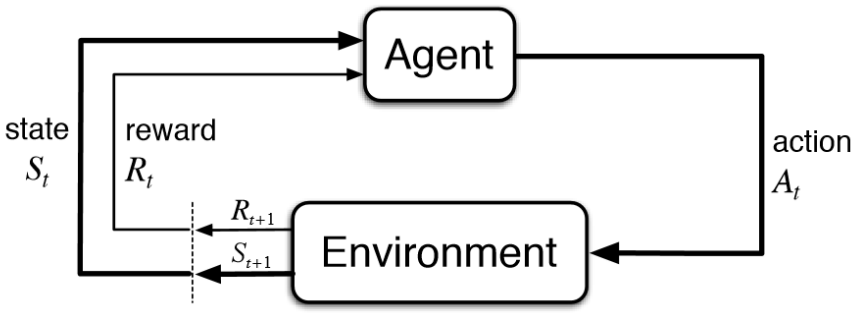

Figure 4.1: Basic Terminologies of Reinforcement Learning [27]

a set of parameters (such as model compression ratios and hardware configurations) based on the evaluation results from the environment. The controller's policy is then updated based on these evaluation results to guide the next round of searches towards better configurations.

Codesign-NAS[18] uses reinforcement learning as the primary search algorithm to guide the joint optimization process of CNN models and hardware accelerators. They found that gradually increasing the threshold makes it easier for the RL controller to learn high-accuracy CNN structures. They evaluated three RL-based search strategies, assessing their proximity to the Pareto optimal points and search convergence speed. The most efficient strategy is phase search, which divides the search process into different phases, such as first fixing the CNN structure and then optimizing the hardware accelerator design, or vice versa. Each phase can use different policy networks to explore the search space more effectively and find local optima.

FGNAS[11] is a graph neural network-based NAS method whose framework consists of three main components: the controller, the FPGA model builder, and the GNN model trainer. For each layer of the subnetwork, the controller generates three types of parameters that define the network topology, hardware implementation, and accuracy. For each sample from the controller, the hardware model is first built and evaluated against predefined constraints. Since most samples may be infeasible, their training is skipped, and their reward is set to zero; otherwise, the network is built, trained, and validated. This approach bypasses the training process as much as possible, making the search faster than pure NAS. Finally, the controller's parameters are updated once after evaluating a small batch of samples. The process terminates after a predefined number of rounds.

In the framework proposed from Weiwen et al.[28], a two-stage (fast and slow) exploration strategy is adopted. The fast exploration stage accelerates the NAS process by efficiently fine-tuning hyperparameters and pruning unqualified hardware

specifications. In the slow exploration stage (SE), candidate architectures are trained on the validation set, and their accuracy and hardware efficiency are calculated to generate rewards and update the controller's parameters.

Reinforcement learning learns the optimal strategy through a trial-and-error process and does not rely on a predefined search space. However, this algorithm has a long training time and consumes a significant amount of computational resources. It may get trapped in local optima, making it challenging to guarantee a globally optimal solution.

4.2 Evolutionary Algorithms

Another popular strategy in traditional NAS is using evolutionary algorithms. These are a class of optimizers inspired by principles of biological evolution and genetics. Typically, neuro-evolution NAS generates an initial population of architectures through random sampling, applies selection techniques to sample some models for generating the next generation, evaluates the fitness of the offspring, and selects the best models to evolve into the population to update the next generation. For the search strategy, evolutionary algorithms transform one neural architecture into another through mutation and crossover operations. When it comes to integrating hardware constraints into NAS algorithms, some research works have utilized evolutionary algorithms.

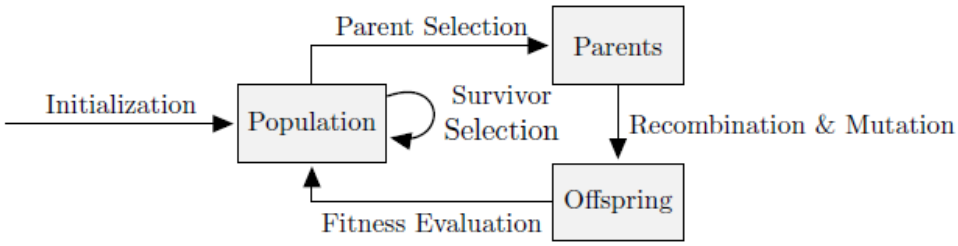

Figure 4.2: Basic Terminologies of Evolutionary Algorithms [29]

GAMMA (Genetic Algorithm for Mapping Machine Learning Algorithms)[30] is a mapping tool based on genetic algorithms, specifically designed to optimize the mapping strategies of convolutional neural networks (CNNs) on hardware accelerators. After generating the initial population, selection is performed based on a fitness function. The performance metrics for GAMMA's fitness function are derived from latency, energy consumption, and computation time obtained through hardware simulators or performance predictors.

Xel-FPGAs[31] is an end-to-end automated framework designed for architectural

space exploration of approximate accelerators in FPGA systems. They chose an exploration evolutionary strategy (ES) + better suited for identifying a wide range of design trade-offs relevant to the target application. This strategy iteratively identifies a set of 200 Pareto-optimal approximate accelerators after 1000 generations. Unlike traditional genetic algorithms, ES relies solely on small random variations within the best individuals rather than complex crossover mechanisms. They iterate over each given Pareto frontier within time constraints to obtain new designs that can be used to extract a new Pareto frontier. These new frontier designs serve as the basis for the next generation of designs.

SkyNet[32] uses a population-based Particle Swarm Optimization (PSO) algorithm to evolve network candidates, enhancing their accuracy and efficiency. PSO works by mimicking the behavior of bird flocks or fish schools searching for food. It explores network configurations such as convolutional layer channel expansions and pooling positions, evaluating and selecting the candidates with the highest fitness scores.

HW-EvRSNAS[21] combines genetic algorithms and random search as its search algorithm. Random search is used to generate an initial diverse set of candidate solutions, and the evolutionary algorithm improves these candidates based on the initial population through crossover, mutation, and selection operations. These operations help gradually optimize the solution quality, with high-performing solutions under evaluation criteria being retained and further optimized.

In a conference paper at ICLR 2022[25], they use an evolutionary algorithm (EA) as their search strategy. For the architecture selection process within the evolutionary algorithm, they propose a simple and efficient method based on the epsilon constraint method to select an elite population. The Upfront algorithm divides the current population into small cells based on the hardware constraint threshold delta T grid. If there is only one hardware constraint, the grid can be one-dimensional; if there are two hardware constraints, it can be two-dimensional. Upfront then selects the architecture with the highest accuracy within each cell. The selected architectures show great potential in terms of accuracy and hardware constraints.

MO-HDNAS[33] employs the metaheuristic optimization technique of genetic algorithms, specifically using NSGA-II. MO-HDNAS conducts architecture search by optimizing three objectives: maximizing representation similarity metrics, minimizing hardware cost, and maximizing hardware cost diversity. The hardware cost diversity objective helps better explore the architecture search space. By introducing the hardware cost diversity objective, the architectures within the population exhibit a broader range of hardware costs, thus increasing the likelihood of discovering high-performance architectures.

In the work of Lotte et al.[34], they utilized the genetic algorithm NSGA-II in conjunction with the ZigZag DSE framework. The ZigZag framework can simulate

various types of neural network accelerators, providing detailed energy consumption and latency analysis, thereby enabling rapid and accurate estimation of hardware performance.

In the work of Halima et al.[35], they also adopted NSGA-II as the search algorithm. However, during the initialization process, they used Latin Hypercube Sampling (LHS) to ensure the breadth of the search. Latin Hypercube Sampling is a statistical method used to generate samples from a multidimensional uniform distribution. When there are many parameters and extensive simulations are needed, it can achieve better coverage of the parameter space more rapidly than simple random sampling or regular grid sampling.

EDD (Efficient Differential DNN Architecture Search) is an optimization method for neural network architecture search (NAS). The core idea of EDD is to search and optimize DNN architectures through Differential Evolution. In the search for DNN architectures, Differential Evolution iterates to generate new architectures by combining features of existing architectures, such as layer types, number of layers, and connectivity patterns, and testing their performance. The EDD method allows for the flexible definition of multiple optimization objectives during the search process, such as improving accuracy and reducing energy consumption. By adjusting the relative importance of these objectives, architectures can be customized according to the specific requirements of the application.

AnaCoNGA[8] uses genetic algorithms (GA) as its core search mechanism. Unlike previous studies, it employs two nested genetic algorithms. The Quantization Strategy Search (QSS) is responsible for finding the optimal quantization strategies, and once a quantization strategy is proposed, the Hardware Architecture Search (HAS) concurrently searches for the optimal hardware configuration for it. If no solution within the hardware Pareto front of a quantization strategy meets the established hardware constraints, QSS receives a signal to avoid further resource wastage on that strategy. This nested approach allows hardware designs to adapt in real-time to the changing needs of CNNs.

LEMONADE[36] applies Lamarckian genetic principles, which assume that individuals can pass on traits acquired during their lifetime to their offspring. In LEMONADE, this means that modifications to the network structure can be directly inherited by its child networks, eliminating the need to train each network from scratch. This approach significantly speeds up the search process, as it allows the network to retain beneficial characteristics already acquired. LEMONADE utilizes network morphisms technology, which permits small, reversible changes to the network structure without altering its functionality. This ensures that adjustments made during the optimization process do not affect its performance, allowing for a safer exploration of the design space without compromising the advantages already achieved.

Evolutionary search has a strong global search capability and can explore a wide

search space. The drawback of this search algorithm is its slow convergence speed, which may require a long time to find high-quality solutions. Additionally, since fitness evaluation requires a significant amount of computational resources, the computational cost of evolutionary algorithms is also high.

4.3 Gradient-based Methods

Gradient-based optimizer algorithms rely on the derivatives or gradients of the objective function to iteratively update parameters/hyperparameters in the direction of their maxima or minima. Among these search strategies, gradient-based methods show promising results. These methods are increasingly used in hardware-aware NAS. However, due to the discrete nature of the search space, running the search and evaluation separately requires significant time and computation. This problem was first addressed by DARTS. DARTS uses a weight-sharing technique that significantly reduces search time. Gradient-based methods train a supernet while obtaining both architecture parameters and weights. However, DARTS can be unstable, and the search process may lead to suboptimal solutions.

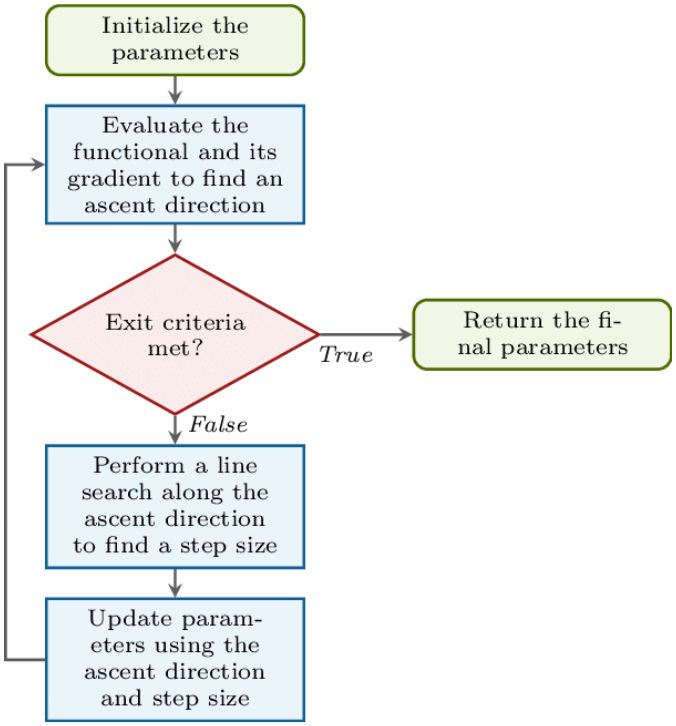

Figure 4.3: Basic Terminologies of Gradient-based Methods [37]

In XNAS, unlike traditional gradient descent, they use the Exponentiated-Gradient algorithm to avoid the decay of architecture weights. This algorithm more effectively facilitates the selection of arbitrary architectures.

BOSHCODE[38] co-design technique extends the BOSHNAS technique, which combines the GOBI algorithm for gradient optimization. By maximizing the upper confidence bound (UCB) estimate of model performance, the co-design process of the model and device is further optimized.

Loong[39] adopts a layer-by-layer training approach to design and train neural networks incrementally, avoiding the high computational cost of searching the entire layer structure at once. They also employ dual-layer gradient optimization, updating parameters and importance weights for each layer's operations through two-step gradient descent. The first step computes cross-entropy loss on the validation dataset to update weights, while the second step updates parameters on the training dataset. They introduce hardware objectives such as memory and latency as regularization terms, and use a lookup table to quickly compute the hardware costs of candidate operations, guiding operation selection.

ETNAS[20] uses a search algorithm based on DARTS, which simultaneously optimizes weights and architecture parameters through gradient descent, addressing memory-intensive issues. They incorporated soft losses into the weight optimization loss function, enabling the search for low-power network architectures by applying segmented penalties for power consumption. ETNAS continuous the search space through Softmax operations, allowing the weights of candidate operations to be optimized during training, ultimately selecting the operation with the highest probability as the final structure.

SurgeNAS[40] employs a single-level optimization approach, optimizing the network structure through accurate gradient estimation to ensure stability and efficiency in the search process. To reduce memory consumption, SurgeNAS introduces an efficient ordered differentiable sampling technique, reducing memory requirements to a single-path level while ensuring fairness throughout the search process.

Gradient-based methods have a fast convergence speed and an efficient optimization process, making them suitable for large-scale neural network architecture searches. However, this type of search algorithm requires precise gradient information and performs poorly in non-smooth or discrete search spaces. It is prone to getting stuck in local optima, making it difficult to find a globally optimal solution.

4.4 Random Search and Bayesian Optimization

Exploring hardware design spaces can be seen as a black-box optimization problem. Firstly, evaluating the objective function is expensive. Secondly, for most hardware

design variables, derivatives do not exist or are difficult to compute, making the properties of the objective function, such as derivatives and convexity, unknown. Bayesian optimization (BO) is a powerful tool and a widely-used sequential model-based optimization framework for solving global optimization problems. In Bayesian optimization, the goal is to find a surrogate function fs that closely mimics the behavior of the desired objective function. It is important to note that in Bayesian optimization, fs is a stochastic process. In the optimization process, BO initially uses random sampling based on previously observed samples to acquire an initial dataset for finding extrema of the hard-to-evaluate objective function, constructing a surrogate model to approximate the objective function. Subsequently, the BO algorithm iteratively samples a new configuration from the design space for evaluation, updating the surrogate model guided by acquisition functions. The acquisition function should balance exploitation of sampled configurations and exploration of unsampled configurations in the design space. Finally, the Pareto set explored by BO in the multi-objective optimization process represents optimal configurations.

In the work of Alireza and Yvon[41], they compare three design space exploration methods: Grey Wolf Optimization (GWO), Bayesian Optimization (BO), and a hybrid GWO-BO approach. GWO, as a metaheuristic method, performs well in design space searches but lacks a systematic surrogate model. On the other hand, BO provides a surrogate model capability but performs less favorably in some benchmark tests compared to GWO. Therefore, the paper proposes an innovative hybrid GWO-BO method that combines BO's surrogate modeling capability with GWO's superior search performance. The hybrid GWO-BO method proposed in this paper not only effectively optimizes FPGA-targeted OpenCL kernels but also achieves efficient exploration of the design space through surrogate modeling.

Automated framework Prospector[42] uses multidimensional Bayesian techniques to optimize synthesized instructions, reducing execution latency and resource usage on Field-Programmable Gate Arrays (FPGAs). The framework coordinates instruction placement and configuration, aiming for low execution time and efficient resource utilization. Prospector improves heterogeneous performance by finding more accurate Pareto-optimal design variants. It achieves these goals in two ways: first, by encoding the design space for better capturing accelerator performance and FPGA costs (e.g., flip-flops, lookup tables, block RAM, and DSPs); second, by sampling the design space to effectively reveal optimal designs, as HLS measurement costs are high.

In the work of Alireza et al.[43], they integrate model-based active learning with Bayesian transfer learning to provide accurate Bayesian models for design space exploration. This method employs Bayesian models to characterize various aspects of hardware performance and combines transfer learning, Gaussian process-based bootstrapping techniques, and active learning to create more accurate models.

Compared to traditional methods, this approach significantly reduces the number of samples required to build performance models while maintaining predictive capability. Once a Bayesian model is constructed, it can be updated using feedback mechanisms provided by real applications.

To further reduce the number of samples required for hardware performance prediction, they use Gaussian process bootstrapping to generate new data for various regression analyses of hardware models, demonstrating that results based on bootstrap samples closely reflect analyses performed on real data. Additionally, they employ Bayesian transfer learning and show that performance of parameter searches is significantly improved by merging information from other related hardware design tasks. Experiments show that their approach can detect optimal design parameters for processor microarchitecture with fewer than 50 samples in most cases. The method demonstrates a significant reduction of 65% in the number of samples needed to create performance models while maintaining their predictive capability.

CODEBench[44] is a unified benchmarking framework used to simulate and model CNN-accelerator pairs and their performance metrics. CODEBench introduces the collaborative design method BOSHCODE, employing second-order gradients and heteroscedastic surrogate models for Bayesian optimization of CNN and accelerator co-design. BOSHNAS operates gradient-based optimization on a single lightweight NN model, using Gradient-Boosted Input to Output (GOBI) to predict model performance and deterministic and stochastic uncertainties.

In the work of Huizhen and Lingli[14], they model design space exploration as a multi-objective black-box optimization problem based on Bayesian optimization, using floating-point encoding to explore the Pareto frontier of HLS instruction design. They develop a general method to model the design space in a tree structure, where the tree structure indicates some sub-parameters are active only when their parent parameters are set to specific values. This method automatically avoids invalid configurations. They adopt the Multi-objective Tree-structured Parzen Estimator (MOTPE) as a surrogate model to flexibly construct HLS design spaces, efficiently searching the tree-structured design space while using Expected Hypervolume Improvement (EHVI) as an acquisition function to guide the optimization process.

In the work of Grace et al.[22], they demonstrate the feasibility of using Bayesian optimization for mapping space search with few samples (less than 100), further reducing the sample count through transfer learning from data collected for other hardware configurations. The key to their approach lies in constructing an encoding scheme that ensures every point in the search space given to the Bayesian algorithm corresponds to a valid mapping. They represent these variables as continuous variables in the optimization program and round them to find actual mapping parameters. Experimental results show that this rounding method optimizes Bayesian optimization of complex functions with a faster convergence rate and capability to handle multiple parameters, matching or surpassing methods based

on discrete alternatives.

For achieving efficient device-side training, they propose a low-overhead dynamic inference and training scheme DynaProp, which dynamically prunes weights, activations, and gradients to skip invalid MAC operations, accelerating Transformer training/inference. DynaProp introduces sparsity in Transformer training and inference using dedicated low-overhead hardware modules. DynaProp achieves 90% sparsity in gradient matrices with negligible accuracy loss and improves training throughput 2.3 times compared to traditional training.

Random Search and Bayesian Optimization do not rely on the characteristics of the search space, making them suitable for multi-objective search spaces and resource-constrained environments. However, Random Search is inefficient and requires a large number of samples to find high-quality solutions. Bayesian Optimization has a high computational complexity, and the process of model training and updating is slow.

Some studies do not use the traditional search algorithms mentioned above. For instance, in Reg-TuneV2[45], they perform continuous optimization of model parameters using polynomial regression and metric learning techniques. This approach is more efficient than traditional search algorithms as it explores and evaluates a large number of model configurations with fewer computational expenses. In HAO, they do not apply heuristic pruning spaces or use reinforcement learning or gradient-based search algorithms; instead, they formulate neural architecture design, quantization, and hardware design as an integer programming problem, enabling the use of efficient optimization algorithms to reduce computational costs.

Chapter 5

Estimation Strategy

In Hardware Neural Architecture Search (HW-NAS), performance estimation is a crucial component. In HW-NAS, an estimation strategy refers to the methods and techniques used to evaluate and predict DL performance and hardware costs (such as latency, power consumption, memory usage, etc.). To apply NAS to specific hardware, hardware constraints must be considered. For example, NAS[46] uses a set of constraints such as peak memory usage, model size, and latency as hardware metrics. These strategies are critical for optimizing and selecting the best hardware architecture.

Typically, researchers build accuracy predictors for architectures in the search space and latency estimators for the target hardware. The goal of accuracy predictors is to quickly and accurately predict the accuracy of neural network architectures without requiring a full training process. Latency estimators aim to quickly assess the inference latency of neural network architectures on specific hardware platforms.

To guide the search strategy, evaluation methods are needed to assess the accuracy of candidate architectures. Evaluating accuracy requires training the architectures to convergence, which demands substantial computational resources, often taking hours or even days of GPU time. One alternative method to accelerate the evaluation process is to estimate accuracy without training the sample architectures. HW-NAS aims to achieve multi-objective optimization, such as high accuracy and low latency. Typically, we cannot optimize conflicting metrics, such as accuracy and latency, simultaneously. Generally, hardware constraint methods or weighted product methods are used to achieve multi-objective optimization goals in NAS.

HW-NAS is formulated as a multi-objective optimization problem, aiming to optimize two or more conflicting objectives, usually requiring improvements in DL performance while reducing hardware costs. In HW-NAS, the results obtained by the search algorithm are usually not a single solution but a set of architectures

with optimal trade-offs between objectives. These sets of optimal solutions are called Pareto frontiers.

There are four common estimation strategies in HW-NAS[6].

5.1 Real-world measurements

This method measures performance metrics by running models on actual hardware platforms. It provides the highest accuracy as it directly reflects the performance in real hardware environments.

However, collecting performance data by running models on actual hardware usually requires more time. If multiple hardware configurations or a large number of different model architectures need to be evaluated, the scalability of real-world testing may be limited. Each hardware or configuration test needs to be conducted independently, which restricts the ability to quickly evaluate a large number of options.

In summary, although real-world testing provides the most reliable data, its high cost, time consumption, and resource demands may make it unsuitable for all scenarios, especially in early exploratory research and resource-constrained applications. In these cases, researchers start exploring other estimation strategies to achieve better efficiency and cost-effectiveness.

5.2 Lookup table

This method involves pre-collecting and storing performance data for different operations on specific hardware platforms. During architecture search, these stored data can be directly used to estimate the hardware cost of the model without the need to actually run the network on the hardware. By avoiding repeated hardware execution and measurement, this estimation strategy is very fast. Additionally, since the lookup table is based on actual hardware measurement data, it can provide accurate performance information.

For example, in the work of Mohamed et al.[18], they measured the latency of each operation (such as convolution, pooling, and element-wise operations) on an FPGA accelerator and stored it in a lookup table. Then, they used a scheduler to allocate operations to parallel computing units based on the latency information in the lookup table and calculated the total latency of the entire CNN model.

However, the flexibility of lookup tables is limited, and they require maintenance costs. Once the hardware is updated or new operation types are added, the lookup table also needs to be updated. Additionally, the lookup table method relies on the assumption of the independence of operations, which may not accurately reflect

performance changes due to interactions between operations (such as memory access conflicts).

5.3 Analytical estimation

This method estimates the performance of deep learning models on specific hardware by analyzing the model's structure and operations, such as the number of multiply-accumulate operations and the latency of each operation. Analytical estimation usually relies on a deep understanding of neural network architecture and hardware performance characteristics, using mathematical models and formulas to predict the execution efficiency of neural networks on hardware, such as computation latency, power consumption, and resource utilization.

Analytical estimation first requires establishing a mathematical model that includes hardware and network architecture parameters. This model typically considers factors such as processor type, memory bandwidth, network layer type, and size. For example, in ASICNAS framework[26], a tool called MAESTRO (Modeling Accelerator Efficiency via Spatio-Temporal Reuse and Occupancy) is mentioned for evaluating and optimizing ASIC (Application-Specific Integrated Circuit) design. It primarily analyzes data reuse patterns to assess the total execution time and the performance and efficiency of DNN (Deep Neural Network) inference in accelerator design. Based on the analysis results, MAESTRO provides specific optimization suggestions and design schemes, such as adjusting the computation graph topology, optimizing memory access patterns, and improving data reuse strategies to enhance hardware design performance and efficiency. This estimation method is particularly suitable for the early stages of hardware design when actual hardware prototypes are not yet available, allowing for performance prediction and design decision-making.

Latency is the most evaluated hardware metric in NAS. There are two main types of latency prediction methods: layer-by-layer prediction and end-to-end prediction. In the work of Bingqian et al.[47], they use the Spearman rank correlation coefficient (SRCC) to quantify the degree of latency monotonicity in practice. Latency monotonicity indicates that a neural architecture that performs well on one device will also perform well on another device. If latency monotonicity holds between two devices, the architecture search results on one device can be generalized to another device. This means that hardware-aware neural architecture search needs to be conducted on only one device to obtain optimal architectures that are equally effective on other devices, avoiding the high costs of building separate latency predictors for each device.

In the work of Panjie et al.[23], a software and hardware parameter-based FPGA performance predictor is proposed for efficiently deploying Transformer

models on different hardware devices. The evaluation method mainly includes latency constraints (LC), accuracy constraints (AC), resource utilization (RU), and reward functions. This predictor can estimate the required clock cycles, memory usage (BRAM), and DSP resource utilization to guide further model and hardware optimization. Specifically, the estimation strategy is based on the PE (Processing Element) size implemented by Vivado HLS, reducing computation latency through simulation and tuning to achieve optimal hardware resource allocation and inference performance.

In the work of Mohamed et al.[18], they created a hardware area model to estimate the resource utilization of hardware accelerators. They decompose the accelerator into components such as convolution engines, buffers, pooling engines, and storage interfaces. Then, using a parameterized approach, they establish a parameterized area model for each component. This model allows for the quick evaluation of hardware resource utilization when jointly optimizing CNN architectures and hardware accelerators, thereby improving design efficiency and performance.

However, analytical estimation also has limitations and is not suitable for all scenarios. The accuracy of analytical estimation highly depends on the model's accuracy and complexity. If the model is overly simplified or ignores some key factors, it may lead to inaccurate results. As the complexity of hardware and model architectures increases, maintaining and updating performance models becomes more challenging, making it difficult to ensure the accuracy and applicability of predictions.

5.4 Prediction models

Accuracy predictors typically use smaller models, such as RNN models, to predict accuracy based on architecture specifications. However, training such models is not straightforward because it requires collecting a dataset of architecture specifications along with their corresponding accuracy on a given task, which increases computational demand. In [30], to accelerate the evaluation of neural network accuracy and latency within the search space, they constructed an accuracy predictor for given neural networks and a latency estimator for specific hardware platforms. Their accuracy predictor uses a data-driven approach aimed at quickly assessing the performance of candidate neural network architectures without full training. They generated over 2400 neural networks from a VGGNet-based search space and trained them using a supernetwork's weights. The collected (architecture, accuracy) pairs were used to train their accuracy predictor. Combining the accuracy predictor and latency estimator not only speeds up the evaluation process but also optimizes neural network architectures by allowing for multi-objective optimization

and search strategies.

In the work of Xiangzhong et al.[48], they proposed an efficient and computationally effective proxy method called Trained Batchwise Estimation (TBE) to reliably estimate the performance of different architectures and integrate it into the hardware-aware NAS scenario. TBE uses early training batch statistics to predict the potential accuracy of architectures, employing a pre-trained multilayer perceptron (MLP) model. This method requires only a portion of the training dataset to achieve efficient estimation.

BRP-NAS[49] introduced a novel prediction algorithm based on Graph Convolutional Networks (GCN), combined with a binary relation predictor and iterative data selection strategy. This significantly enhanced the performance of hardware-aware NAS. By learning graph-structured data to predict end-to-end latency, this graph-based approach can comprehensively consider the overall characteristics of the architecture, providing more accurate results compared to traditional layer-by-layer prediction methods. BRP-NAS demonstrated outstanding performance in the large DARTS search space, finding better models with a smaller search budget.

MAPLE-Edge[50] uses hardware-aware regression models to estimate the inference latency of neural network architectures on previously unseen embedded target devices. MAPLE-Edge employs an automated pipeline to convert models from NAS-Bench-201 into architecture codes and hardware descriptions, which are then input into the regression model to output the inference latency. They also divide each performance counter by the latency of the respective operation, providing a more effective representation and significantly improving the accuracy of latency prediction for optimized graphs on embedded devices.

In the work of Yong et al.[51], they proposed a latency prediction method based on active learning. This method uses a multilayer perceptron (MLP) as the latency prediction model and iteratively selects architectures and collects their latency data through active learning to train the latency prediction model, thus reducing the high cost of collecting latency data. The Kendall correlation coefficient is used to evaluate the prediction accuracy of the latency prediction model, with dataset cost measured by the number of samples.

In the work of Kurt et al.[52], they use Block-Level Surrogate Models (BLSMs) to estimate inference time. This method involves randomly sampling from predefined network blocks and then measuring the inference time of these blocks on a specific hardware platform. The configuration of each block and its corresponding inference time form the training dataset. Then, a BLSM is independently trained for each type of block and hardware platform. Finally, by aggregating the inference time predicted by each BLSM for the blocks, the total inference time of the candidate network can be estimated.

In hardware-aware neural architecture search (HW-NAS), multi-objective estimation strategies are key methods designed to balance and optimize multiple

competing objective metrics, such as latency, power consumption, and accuracy. These strategies consider different hardware platform characteristics and application requirements to design neural network architectures that perform well under various constraints. In the work of Yue et al.[53], they proposed a novel multi-objective predictor framework. This framework consists of a multi-head encoder and a decoder, capable of extracting feature representations from different perspectives (e.g., accuracy and latency). These representations are then concatenated and passed to the decoder to output evaluation scores. They used Higher-performance Architecture Sampling (HPAS) to build the predictor training dataset, selectively collecting training samples from the search space instead of simple random sampling.

In the work of Haiyan et al.[12], the key requirement for efficient hardware-aware neural architecture search (HW-NAS) is the rapid evaluation of accuracy and latency to rank different candidate solutions. To accelerate the search process, they designed a network performance estimator called Eacc to predict accuracy, hardware resource costs, and latency. This performance estimator comprises three parts: an accuracy predictor, a hardware resource estimator, and a hardware latency predictor. They used a neural network with four fully connected layers, with input features generated by an encoder. Mean Squared Error (MSE) was chosen as the loss function, and they optimized the accuracy predictor by minimizing MSE loss. The hardware latency predictor estimates latency through fine-grained analysis of the computation process. Since each convolutional computation unit is independently deployed, hardware latency can be estimated layer by layer.

Inspired by research on pruning at initialization, zero-cost proxies attempt to quantify the trainability and expressiveness of neural network architectures without training them. Multi-Predict[54] is a few-shot predictor method that proposes two new NN encodings (ZCP and HWL), enabling accuracy and latency prediction with as few as 10 samples. These encodings are not dependent on the search space but are based on zero-cost proxies or hardware latency measurements, allowing efficient prediction across multiple search spaces and tasks. ZCP quantifies the trainability and expressiveness of neural network architectures, generating useful metrics without requiring full network training. This greatly reduces the number of samples and computational resources needed. Additionally, ZCP encodings generate metrics independent of specific tasks and search spaces, enabling effective knowledge transfer for predictors across different NAS search spaces and tasks. They also proposed a simple transfer learning strategy for multi-device latency prediction, demonstrating effectiveness in tasks on FPGA and ASIC platforms.

In HAO's work[13], to reduce the computational load required for evaluating different designs, they developed subgraph-level hardware latency models and accuracy predictors for neural network architectures. This model considers the impact of neural network architectures and quantization settings on hardware latency and uses integer programming algorithms to find the optimal configuration

that meets latency constraints. To compare different neural network architectures, they used predictors to estimate the accuracy of pre-trained models under given architectures and ranked candidates that meet latency constraints. In HAO, they stacked the architecture parameters of each layer as input vectors and applied support vector regression (SVR) models to predict accuracy. SVR models typically require less data to train, simplifying and improving sample efficiency.

Reg-TuneV2[45] uses polynomial regression to estimate model accuracy, dynamic power consumption, and inference latency based on existing training data points. This estimation strategy allows researchers to predict the performance of new model configurations on specific hardware platforms without actual deployment and testing. This strategy enables researchers to make more precise, data-driven decisions during model optimization.

In the work of Grace et al.[22], they designed the GPTune tool based on multi-task learning and transfer learning algorithms for automatic tuning using Bayesian optimization. GPTune can share performance sample data between different tasks to improve tuning results. By training surrogate models on various hardware configurations, GPTune can perform rapid tuning on new hardware configurations, significantly reducing the number of samples needed.

In the work of Huizhen and Lingli[14], they used ADRS (Average Distance from Reference Set) to measure the distance between the obtained Pareto front and the reference Pareto front. This metric calculates the normalized distance of each solution from the obtained set to the nearest solution in the reference set and averages these distances to reflect the overall quality of the solution set. By comparing the results of different algorithms, the optimization effectiveness can be evaluated. The smaller the ADRS value, the better the performance of the optimization algorithm.

A3C-S[55] is an automated agent-accelerator joint search framework that optimizes both agent models and hardware accelerators in the field of deep reinforcement learning (DRL). A3C-S uses innovative differentiable neural architecture search (DNAS) and a novel distillation mechanism. This AC distillation mechanism effectively reduces the variance of gradient estimation, improving training stability and convergence speed. The distillation process considers both policy and value functions, further enhancing training results.

Some studies have also evaluated less common metrics. For example, HW-EvRSNAS's[33] evaluation method is mainly based on measuring the similarity of deep neural network representations, specifically using Representation Mutual Information (RMI) as a performance estimation metric. RMI calculates the similarity of hidden layer representations between two architectures. RMI compares the hidden layer representations of the current architecture with those of a reference model and calculates their RMI values. It can compute performance evaluation using only a single training batch, significantly saving computational resources compared to

training each architecture from scratch. This method not only improves search efficiency but also effectively addresses architecture optimization needs under different hardware constraints. RMI can effectively capture the similarity between architectures, making the evaluation results provide reliable architecture choices under actual hardware constraints.

In recent years, researchers have proposed machine learning algorithm-based proxy-assisted evaluation methods to quickly estimate the performance of sampled architectures without training them. In existing HW-NAS methods, each objective is often assigned an independently trained proxy model. These proxy models use different evaluation methods; however, since each proxy model introduces its error, it can lead to low search efficiency and local optima.

In the work of Hadjer et al.[56], they proposed a unified proxy model, hardware-aware Pareto ranking NAS (HW-PR-NAS). They used learning-to-rank theory because they observed that when the proxy model preserves rankings, the time complexity of NAS is significantly reduced, and the exploration path is enhanced. This ranking method is more important than calculating accuracy. HW-PR-NAS ranks by Pareto scores, reducing search time and avoiding potential conflicts between proxy models.

In the work of TransCODE[38], within the BOSHCODE and GPTran frameworks, the authors proposed different estimation strategies to optimize the design and performance of Transformer architectures. BOSHCODE uses trained surrogate models and uncertainty estimation to further optimize model and device embedding choices, including modeling epistemic and aleatoric uncertainty. The GPTran framework further adjusts optimized models through rapid block-level growth and magnitude-based pruning techniques to improve performance and address issues that may arise from inaccurate surrogate model training.

In Sherlock[57], this DSE framework uses various regression models as proxy models, including random forests, Gaussian processes, and kernel-based interpolation methods (such as RBF). Each model has its applicable design space type and characteristics. Proxy models not only provide predictions of output values but also estimate the uncertainty of each prediction. Sherlock guides the next sample selection by calculating the estimated Pareto front and its uncertainty, aiming to more accurately optimize solutions within the design space.

With the rapid development of machine learning and artificial intelligence technologies, predictive model strategies may gradually become mainstream, especially in the early design stages and large-scale searches. Simultaneously, with the rapid iteration and diversification of hardware platforms, lookup table and analytical estimation strategies need continuous updating and optimization to adapt to new hardware characteristics and more complex network architectures. Real-world measurements, as the "gold standard" for final performance verification, will continue to play an irreplaceable role in the final product confirmation stage.

These estimation strategies provide a diverse set of tools and methods for the research and application of HW-NAS, enabling researchers and engineers to choose the most appropriate strategy based on specific needs and resources, thereby effectively promoting the co-optimization of neural network architectures and hardware.

Chapter 6

Other Considerations for HW-NAS

In hardware-aware neural architecture search (HW-NAS), compression techniques are crucial technological means for reducing the size and complexity of deep learning models to meet the demands of various hardware platforms. Quantization and Pruning are two common compression methods, which we will introduce in this section.

6.1 Quantization

In addition to HW-NAS architecture design, quantization is another method to achieve efficient inference and enrich the efficient DNN design space for hardware. It directly impacts model quality by using low-bit representations for weights and activations in a given neural network model. However, simply quantizing DNNs into low-bit representations often leads to poorer inference accuracy, making quantization impractical regardless of hardware efficiency. Mixed precision quantization allows different layers in neural networks to have varying bit-widths, leading to an exponentially growing search space to find optimal bit-width configurations.

In A3[58], researchers found that most neural network tasks can inherently tolerate a certain degree of error, allowing operations to be conducted with lower bit-width representations while maintaining high precision representations. They leverage lower bit-width expansions to design accelerators, avoiding linear or quadratic growth in hardware energy consumption as bit-width representation increases. Additionally, due to the significantly higher energy consumption of floating-point operations compared to fixed-point operations, specialized accelerators typically employ fixed-point representation. Their model first quantizes provided floating-point inputs into i integer bits and f fractional bits (plus one

sign bit), then uses different bit-widths at each stage of the pipeline to maintain precision, avoid overflow, and minimize energy consumption.

FracBNN[32] is a solution focused on low-bit-width DNN design, offering both precise and hardware-friendly network architectures through the use of fractional activation values and double precision activation schemes. It employs novel network building modules and thermometer encoding to optimize binary handling at the input layer, retaining the efficiency of traditional BNNs and achieving high-precision inference results under low-precision conditions.

In the work of Reg-TuneV2[45], they proposed a mixed-precision neural network (MPNN) accelerator design framework specifically for FPGA. The framework achieves efficient execution of neural network models on FPGA by optimizing quantization, hardware implementation, and resource allocation. In [31], they adopted 8-bit fixed-point quantization to optimize weights, input-output of convolutional layers, and fully connected layers. This quantization scheme enhances model inference efficiency by reducing storage and computational resource usage while maintaining accuracy.

Auto-ViT-Acc[59] is an FPGA-based automatic visual Transformer (ViTs) acceleration framework. This framework employs a hybrid quantization scheme, utilizing fixed-point and Power of Two (PoT) quantization within each layer simultaneously. Fixed-point quantization offers high accuracy performance and efficiently utilizes DSP resources on FPGA. PoT quantization, on the other hand, is an efficient quantization scheme that replaces multiplications with simple shift operations, making it suitable for leveraging LUT resources on FPGA.

In the work of Javier et al.[60], they propose an end-to-end collaborative design method that combines Hessian-aware quantization techniques to optimize the deployment of neural networks on both FPGA and ASIC. They introduce new quantization strategies that integrate Hessian information into the optimization process for finer parameter tuning.

In the work of Christoph et al.[36], they compared two fixed-point quantization methods, MaxRange and MinPQE. The MaxRange method determines the step size based on the maximum range of the data distribution, which is suitable for bounded activation functions (such as sigmoid) but may not be suitable for unbounded activation functions (such as ReLU). Moreover, MaxRange does not consider the propagation of quantization errors within the network. The MinPQE method aims to minimize the propagation of quantization errors. It ensures that the difference between the outputs before and after quantization is minimized. This method optimizes the step size for the inputs, weights, and biases of each layer separately, minimizing the difference between the quantized and original neuron outputs, thereby better preserving the network's performance.

LSFQ[61], short for Learnable Parameter Soft Clipping Full Integer Quantization, is an efficient new quantization method. This approach introduces learnable soft

clipping quantization to optimize neural network performance under low-precision integer operations. Employing linear symmetric quantization, it truncates the dense part of weight distribution for quantization to reduce overall quantization error. They further introduce a soft clipping function where parameters are learned via backpropagation to minimize quantization error.

LW-GCN[62] is a lightweight FPGA-based graph convolutional network accelerator. They propose the Packet-level Column-Only Coordinate List (PCOO) format to compress sparse matrix non-zero elements column-wise, reducing storage and computational requirements. They apply 16-bit and 4-bit fixed-point quantization methods to further compress matrix data, reducing memory consumption and computational complexity. They also utilize data blocking techniques to partition large matrices into smaller blocks for processing, adapting to limited on-chip storage resources on FPGA.

In the work of Mohamed et al.[63], to efficiently implement GoogleNet on FPGA, they explore offloading the CNN's compute-intensive parts to FPGA using Xilinx Vitis tools, while running the remaining parts on embedded CPUs. They then apply Post-Training Quantization (PTQ) technology to convert floating-point computations into fixed-point computations. PTQ is a method of quantizing the model after training to reduce hardware resource requirements while minimizing impact on model accuracy.

CoGNN[63] introduces a hardware collaborative design method for small-batch GNN inference, significantly reducing computation and memory access by reusing perception sampling and parallel perception quantization technologies. Parallel perception quantization adjusts quantization parameters dynamically, enabling each parallel computing unit to work under optimal quantization parameters during GNN inference. By employing block-level parallel quantization methods, it drastically reduces memory access and computational complexity while maintaining computational accuracy, enhancing overall performance and energy efficiency of hardware accelerators.

In the work of Xiaofan et al.[64], two quantization techniques tailored for hardware platforms are discussed. ELB-NN (Extremely Low Bitwidth Neural Network) enhances energy efficiency when running image classification on embedded FPGAs. It supports arbitrary DNN quantization through a hybrid low-bitwidth design that improves energy efficiency. ELB-NN balances inference accuracy and computational demands by adjusting the full precision representation of weights and applying binary, ternary, or higher fixed-point representations. The VecQ quantization training framework is based on vector loss. This framework supports flexible bitwidth settings and uses Euclidean distance (L2 distance) to measure and minimize quantization errors. During the quantization process, the VecQ framework does not consider individual parameters but views them as vectors, taking into account the interactions and overall distribution among parameters. By

adjusting the direction and size of the quantized vectors, VecQ precisely controls the quantization process to optimize overall network performance.

6.2 Pruning

Regarding Pruning techniques, it reduces the number of parameters in a network by identifying and removing unimportant neurons or connections to improve efficiency. Previous research has primarily focused on training-phase pruning techniques, where these training-based methods impose sparse constraints on specific parameters (e.g., L1 regularization) during training to identify and prune unimportant architectures. However, these methods are costly in terms of resource and time consumption and are constrained by various conditions, such as parameters through L1 regularization not always achieving sufficient sparsity.

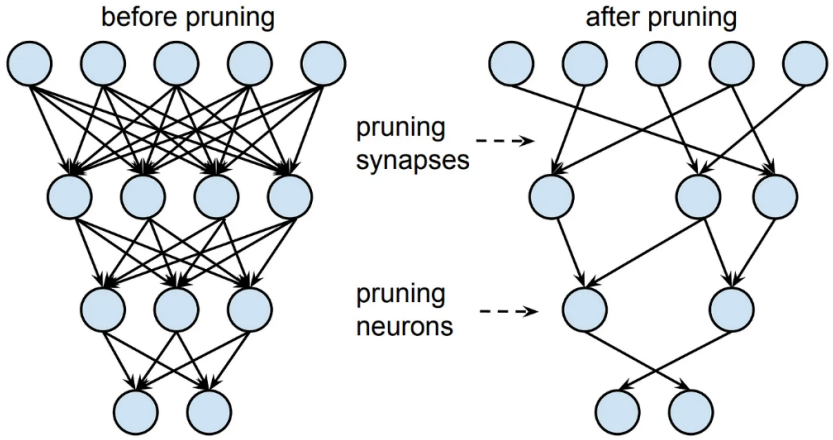

Figure 6.1: Synapses and neurons before and after pruning [65]

In the work of Jan Pieter et al.[66], feasibility and effectiveness in accelerating deep learning model execution on FPGA were demonstrated. The study compared residual connection pruning with traditional L2 norm pruning methods, showing an additional 30% inference speed improvement. The study also explored hardware-aware pruning (HAP), adapting pruning strategies based on FPGA systolic array size to minimize computing and memory access requirements.

In the work of Hengyi et al.[67], they proposed an efficient layer-wise fine pruning method leveraging properties of Batch Normalization (BN) layers and input sparsity at the convolutional layer level. This method based on convolutional layer-level input sparsity and channel-level importance indices compresses and optimizes

DNNs. The pruning method is highly efficient with minimal accuracy loss and can effectively extend to all other neural networks using batch normalization operations.

The HW-Flow[68] framework introduces a hardware-aware pruning method, achieving effective compression of CNN models by employing precise hardware estimates during the pruning process while minimizing prediction accuracy loss. This method performs better than traditional pruning methods significantly.

In the work of Han et al.[2], they propose a Hardware-Aware Automated Quantization (HAQ) model based on AutoML to optimize the latency and accuracy of the network. The HAQ model frames the quantization task as a reinforcement learning problem. They employ an actor-critic model to generate quantization policies. In their approach, they utilize FPGA, which supports mixed precision well, to provide energy and latency costs.

Jef and Toon[66] introduced Hardware-Aware Pruning (HAP). Based on the processing characteristics of FPGA deep learning accelerators, particularly the nearbAI engine, they adjusted pruning strategies to significantly shorten model compression and inference times. They utilized normalized L2 norm to compute the importance of different filters, then iteratively pruned X% of the least important channels. HAP maximized pruning space by combining residual connection pruning and hardware-aware pruning. By simultaneously removing multiple relevant layers, it further enhanced model compression rates and inference speeds. This method effectively expanded pruning space, making the pruning process more efficient.

In HotNAS[24], they linked compression techniques with NAS and hardware design using the iSpace sub-tool. They employed the iDirect performance bottleneck detector to analyze pattern pruning, channel pruning, quantization, and filter expansion in the search space. By reordering IFMs, they ensured effective reduction in computational latency with pruning, effectively addressing the inefficiencies of starting from scratch in search.

In the work of Panjie[23], they combined existing pruning techniques BP and VW to propose a new two-level pruning technique HP. To maintain hardware friendliness, they initially employed BP for the first-level pruning of the model. Then, based on BP, they further pruned using VW. HP technology integrates coarse-grained pruning (BP) and fine-grained pruning (VW), achieving higher sparsity rates while ensuring minimal precision loss and remaining hardware-friendly.

Within the joint design framework of TransCODE[38], they used the DynaProp module to introduce sparsity by dynamically pruning weights, activations, and gradients to skip ineffective MAC operations, speeding up Transformer training/inference processes. DynaProp leverages dedicated low-overhead hardware modules to introduce sparsity in Transformer training and inference. DynaProp achieved 90% sparsity in gradient matrices with negligible accuracy loss, while improving training throughput by 2.3 times compared to traditional training.

GNN has achieved great success in applications such as recommendation systems,

molecular property prediction, and traffic forecasting. HP-GNN[69] is a general framework for generating high-throughput Graph Neural Network (GNN) training implementations on CPU-FPGA heterogeneous platforms. They evaluated the main metric of mini-batch GNN training throughput, defined as vertices traversed per second (NVTPS). They also established a resource utilization model focusing on the usage of DSPs and LUTs.

SpAtten[70] adopted an optimization algorithm combining cascaded pruning and progressive quantization. This pruning technique targets input tokens rather than network weights, dynamically pruning based on attention probabilities across different layers. Pruning decisions are dynamic and depend on input instances, reducing computational burden while ensuring model accuracy, effectively optimizing the computational efficiency and memory access of attention mechanisms.

HALP[71] (Hardware-Aware Latency Pruning) is a latency-driven structured pruning algorithm. HALP models structured pruning as a global resource allocation optimization problem, evaluating filter importance using latency lookup tables and global significance scores. With an enhanced Knapsack Algorithm, HALP surpasses previous work in pruning efficiency and accuracy-efficiency trade-offs.

In the work of Krishna et al.[72], a generalized search space pruning algorithm was proposed. Search space pruning includes four main steps: Enumerate Choices, Clustering, Prune Choices, and Check Optimal Latency. By removing redundant weight and activation precision choices, significant reduction in search time was achieved.

In the work of Mengshu et al.[73], a method was proposed to accelerate 3D Convolutional Neural Networks (3D CNNs) on FPGA using hardware-aware pruning. By combining block-level pruning and an ADMM-based solution framework, significant acceleration was achieved without significant loss of accuracy. Post-ADMM pruning, the pruned model was retrained, updating only non-zero weights to further enhance post-pruning model accuracy.

Krishna introduces various pruning methods for HW-NAS in his paper[74]. HDAP (Hardware Dimension Aware Pruning) performs intelligent pruning on network weights and structures, removing parts that contribute less to the final inference result or are inefficient on specific hardware. Unlike general network pruning methods, HDAP can dynamically adjust pruning strategies based on different hardware platforms. HDAP enhances the scalability and transferability of neural network models across different hardware platforms. Models processed by HDAP are easier to transfer to new hardware environments while maintaining or even improving performance. Another specialized neural network pruning technique is FPAP (Fault and Array Size Based Pruning). This method optimizes network architecture and weight configuration by considering the physical characteristics of the hardware and potential fault patterns, thereby maintaining efficient inference performance even in the presence of hardware defects.

6.3 Approximate Multipliers

Approximate multipliers are an important research direction in the design of FPGA accelerators. The programmability and parallel processing capabilities of FPGAs make them an ideal platform for implementing approximate multipliers. In deep learning models, multiplication is one of the most computationally intensive operations, especially in convolutional layers and fully connected layers. Approximate multipliers simplify multiplication operations by introducing controllable errors, thereby reducing computational complexity and energy consumption. Common approximation methods include truncation, rounding, and logic simplification. Approximate multipliers can significantly reduce the power consumption and latency of DNN accelerators, especially in multiply-accumulate (MAC) operations. Using approximate multipliers in convolutional layers can achieve significant energy efficiency improvements with minimal precision loss.

In the work of Vojtech et al,[75], researchers proposed a technique to extend the original library using an automated method to construct a comprehensive library of approximate circuits useful for real-world applications. They employed single-objective Cartesian Genetic Programming (CGP) and selected WCE as the error metric. The generated approximate circuit library (EvoApproxLib) synthesized hardware models using general design tools and determined all relevant quality parameters through software models. These approximate circuits demonstrated significant performance improvements and energy efficiency optimizations in CNN hardware accelerators.

Another common open-source library is TFApprox4IL, which extends TensorFlow and Keras APIs to support approximate multipliers in convolutional layers. This library uses lookup tables (LUTs) to emulate approximate multiplication operations, optimizing computational efficiency on GPUs.

SMApproxLib is an open-source library specifically designed for FPGAs, containing approximate multipliers with different bit-widths, output precision, and performance gains. The multipliers in this library utilize the efficient use of LUTs and carry chains to optimize FPGA performance, ensuring high computational efficiency while reducing computational complexity.

The Xel-FPGAs[31] framework incorporates statistical learning models to quickly estimate the performance and hardware requirements of different designs during the architectural exploration phase. This approach avoids the time-consuming synthesis and simulation of each design, significantly improving exploration efficiency. Additionally, the Xel-FPGAs framework integrates the ABC tool to estimate the number of LUTs, power consumption, and latency on FPGAs. This integration further reduces design evaluation time, making Xel-FPGAs advantageous in approximate accelerator design and optimization.

AMG[76] (Automated Efficient Approximate Multiplier Generator) is a framework specifically designed to automatically generate efficient approximate multipliers for FPGA systems. The AMG framework employs Bayesian optimization to achieve the automatic generation of efficient approximate multipliers in FPGA systems. This framework automates the generation of HA arrays and supports different bit-widths, promoting the application of approximate computing technology in hardware accelerators.

Overall, quantization and pruning techniques play a crucial role in HW-NAS by optimizing models and hardware implementations, thereby enhancing the inference efficiency and energy efficiency of deep learning models. Approximate multipliers are also an important research direction in accelerator design, successfully reducing computational complexity and energy consumption by simplifying multiplication operations. Future research should further explore the adaptability and optimization potential of these techniques across different hardware platforms to drive the development of more efficient and energy-saving deep learning applications.

Chapter 7

Other Applications

Although most HW-NAS (Hardware-aware Neural Architecture Search) studies focus on conventional applications in image processing and natural language processing, there are also many studies that focus on their successful application in other settings, such as Graph Neural Networks (GNNs) and Transformer networks. These areas, which have gained popularity in recent years, have unique processing requirements that demand different hardware optimizations.

7.1 HW-NAS of GNN

Graph Neural Networks (GNNs) are deep learning networks designed to handle graph-structured data, and are applicable to fields such as social network analysis and protein structure prediction. Although GNNs perform well in various tasks, they require high computational complexity and memory, especially when dealing with large-scale graph data. To face these challenges, some research has designed accelerators specifically for GNN tasks.

The HP-GNN framework aims to provide a high-throughput, efficient mapping of GNN training for CPU-FPGA platforms, offering an excellent solution in the field of hardware-accelerated GNN training. This framework provides optimized templates that support various GNN models, allowing for easy adaptation to different GNN structures and requirements. Additionally, the framework reorders and structures data to optimize memory traffic and random access patterns, minimizing latency and maximizing throughput on FPGAs.

SurgeNAS accelerates the implementation and performance of GNNs through several key technologies. They have integrated a GNN-based latency predictor to avoid cumbersome latency measurements on actual devices. They also resolved issues of search collapse through an effective identity mapping mechanism, enhancing the stability and accuracy of the model during the search process.

LW-GCN is a lightweight FPGA-based GCN accelerator. It decomposes the main operations of GCN into Sparse Matrix-Matrix Multiplication (SpMM) and Matrix-Matrix Multiplication (MM), introducing a novel compression format to balance the workload across Processing Elements (PEs) and prevent data conflicts. The accelerator employs data quantization and workload tiling strategies, mapping both SpMM and MM operations onto a unified architecture on resource-constrained hardware.

In the CoGNN work, they specifically emphasize the use of minibatch sampling to accelerate the inference process of GNNs. Their accelerator supports reuse-aware sampling and parallelism-aware quantization, significantly increases data reusability and reduces data access conflicts.

7.2 HW-NAS of Transformers

Transformer networks, particularly models designed for processing sequential data such as BERT and GPT, have achieved significant success in the field of natural language processing (NLP). However, these models typically require substantial computational resources, which poses a significant challenge on resource-constrained hardware. The application of Hardware-Aware Neural Architecture Search (HW-NAS) in this area aims to optimize the energy efficiency and computational efficiency of these networks, enabling their deployment on mobile devices and embedded systems.

In the work of Hongwu et al.[77], they proposed an FPGA-based transformer model acceleration scheme. The article introduces a hardware-friendly sparse attention operator that reduces the computational complexity of the attention mechanism by decreasing the precision of the data processed. For the issue of inconsistent input sequence lengths in natural language processing tasks, they proposed a length-adaptive hardware resource scheduling algorithm. The accelerator design they proposed leverages the reconfigurable nature of FPGAs, allowing dynamic adjustment of hardware configurations for different tasks.

In the work of Panjie et al.[23], they propose an acceleration framework that co-optimizes hardware design and model compression for transformer models. The article introduces an algorithm-hardware closed-loop acceleration framework. This framework can select the optimal device based on user inputs such as dataset, model, latency, and accuracy constraints. At the same time, to reduce memory usage on FPGAs, they also optimized the sparse matrix storage format based on the HP sparsity pattern.

In EdgeTran[9], an efficient inference solution for transformer models on mobile edge platforms is provided. They introduce an automated design framework named ProTran. This framework evaluates the hardware performance of transformer

architectures across various edge devices, aiding in the identification of the best-performing models with high accuracy for given tasks while minimizing latency, energy consumption, and peak power draw. Utilizing data evaluated by ProTran, the co-design technique within the EdgeTran framework can find the optimal model-device combination. Finally, they employ GPTran for a pruning post-processing step, further enhancing accuracy in a hardware-aware manner.

Energon[78] uses dynamic sparse attention mechanisms to effectively accelerate transformer models. They proposed a multi-precision, multi-round filtering (MP-MRF) algorithm that dynamically identifies a few important query-key pairs. By employing low-bitwidth operations in each filtering round and using high-precision tensors only during the attention phase, they significantly reduce the overall computational complexity.

In SpAtten[70], they prune unimportant tokens and heads through cascading, and then introduce progressive quantization techniques to reduce memory access. SpAtten features a novel high-parallelism top-k engine that quickly identifies the most important tokens and heads. Experiments show that SpAtten can reduce DRAM access by an average of 10 times without loss of accuracy.

Auto-ViT-Acc[59] is a framework for mixed-scheme quantization acceleration designed for Vision Transformers (ViT). The computational engine of this framework utilizes loop tiling techniques to partition the input, weight, and output data of ViT layers, decomposing large matrices into smaller submatrices to conserve FPGA resources. They also employ Pipelining and Unrolling techniques, which significantly enhance the throughput of multi-head attention processing.

HW-NAS has shown its extensive potential across multiple domains, including GNNs and Transformers. Future research will continue to expand the application scope of HW-NAS on FPGA/ASiC and explore new optimization strategies to meet the growing computational demands and diverse hardware environments.

Chapter 8

Conclusion

This survey explores the development trends of Hardware Neural Architecture Search (HW-NAS) on FPGA and ASIC platforms. Deep learning has revolutionized various fields such as image recognition, natural language processing, and autonomous driving. HW-NAS automates the design of neural networks, optimizing their performance tailored to specific tasks and hardware constraints. Integrating NAS with hardware platforms like FPGA and ASIC holds significant potential for deploying efficient and specialized deep learning models.

Beginning with fundamental concepts of deep learning, this survey introduces the role of FPGA/ASIC in accelerating neural network computations. We emphasize the importance of defining efficient search spaces (Chapter 3) and highlight various search algorithms (Chapter 4) for effective exploration. Discussion on evaluation strategies (Chapter 5) is crucial for predicting hardware costs such as latency and energy consumption, essential for deploying models on specific hardware architectures.

Furthermore, we delve into other considerations of HW-NAS, particularly quantization and pruning techniques (Chapter 6). Quantization optimizes model size and complexity by reducing the bit-width of weights and activations, thus enhancing hardware efficiency without compromising inference accuracy. On the other hand, pruning reduces network parameters by identifying and eliminating redundant connections or neurons, creating more efficient model architectures suitable for FPGA and ASIC implementations.

Advancements discussed in each chapter underscore the rapid development in the HW-NAS field, driven by innovations in search algorithms, efficient evaluation strategies, and novel compression techniques. Challenges remain in balancing model accuracy with hardware constraints and managing the computational costs associated with NAS.

Looking ahead, future research should focus on integrating emerging technologies and exploring more sophisticated NAS algorithms tailored for FPGA and ASIC

architectures. Addressing environmental impacts and energy efficiency in deploying deep learning models via HW-NAS will be crucial for sustainable AI applications.

In summary, HW-NAS based on FPGA/ASIC offers a promising pathway to optimize deep learning models, providing scalable solutions across multiple application domains through hardware-specific optimizations. In the era of intelligent computing, continuous collaboration among researchers, industry practitioners, and hardware developers will be key to unlocking the full potential of HW-NAS.

Appendix A

HW-NAS Methods Summary

Table A.1: HW-NAS on FPGA/ASIC Summary 01

Method	Year	Search Space	Objectives	Search Algorithm	Estimation Method	Target Device	Other Consideration
HW-EvRSNAS	2024	supernet	lat, energy	Evolutionary Algorithm	predictive modeling	ASIC/FPGA	no
MO-HDNAS	2024	supernet	lat, energy, size	Evolutionary Algorithm	analytical estimation	not specified	no
MLP	2024	not mentioned	lat, energy, resources	other	predictive modeling	FPGA	no
unnamed	2024	hierarchical	lat, energy, mem	Evolutionary Algorithm	predictive modeling	not specified	quantization
HAO	2023	hierarchical	lat, Size, peak power draw	other	LUT	FPGA	pruning
EdgeTran	2023	hierarchical	lat, energy,	Bayesian Optimization	predictive modeling	ASIC/FPGA	quantization
HALOC	2023	hierarchical	lat, energy	Bayesian Optimization	predictive modeling	ASIC	low-rank compression
unnamed	2023	hierarchical	lat, BRAM,DSPs,FFs,LUTs	Evolutionary Algorithm	predictive modeling	FPGA	quantization
unnamed	2023	hierarchical	power,lat,PPA	Bayesian Optimization	predictive modeling	ASIC/FPGA	no
DeepBurning-MixQ	2023	hierarchical	lat,energy,DSP,resource	Bayesian Optimization	predictive modeling	FPGA	quantization
ETNAS	2023	hierarchical	lat,energy,mem	Gradient-based Methods	analytical estimation	not specified	no
Mapspace	2023	hierarchical	lat, energy,cycle count	Bayesian Optimization	analytical estimation	not specified	no
Xel-FPGAs	2023	hierarchical	lat,power,area,resource	Evolutionary Algorithm	predictive modeling	FPGA	no
unnamed	2023	unit-based	lat,energy	Evolutionary Algorithm	analytical estimation	not specified	no
unnamed	2023	hierarchical	lat, energy, bandwidth	Evolutionary Algorithm	analytical estimation	not specified	quantization
TransCODE	2023	supernet	lat, energy, area	Evolutionary Algorithm	analytical estimation	not specified	pruning
Loong	2023	hierarchical	lat, mem	Evolutionary Algorithm	analytical estimation	not specified	no
SurgeNAS	2023	supernet	lat, energy, FLOPs	Evolutionary Algorithm	predictive modeling	not specified	no
LightNAS	2023	supernet	lat, energy, mem, FLOPs	Evolutionary Algorithm	predictive modeling	not specified	no
unnamed	2023	hierarchical	lat, mem, FFs	Bayesian Optimization	predictive modeling	not specified	no
CODEBench	2023	hierarchical	lat, energy, area	Bayesian Optimization	predictive modeling	ASIC	no
Reg-TuneV2	2023	hierarchical	lat, accuracy, power	other	predictive modeling	FPGA	no
unnamed	2023	hierarchical	inference time, energy, mem	Gradient-based Methods	predictive modeling	not specified	no
Multi-Predict	2023	not mentioned	lat, energy, size	other	predictive modeling	FPGA	no
HW-PR-NAS	2023	not mentioned	lat, energy	Evolutionary Algorithm	analytical estimation	ASIC/FPGA/GPU	no
unnamed	2023	hierarchical	lat, energy, mem	Gradient-based Methods	analytical estimation	ASIC/FPGA	quantization
CoGNN	2023	hierarchical	lat,energy,on-chip data reuse	other	analytical estimation	ASIC/FPGA	quantization
unnamed	2023	layer-wise	lat	not mentioned	analytical estimation	FPGA	pruning
unnamed	2023	hierarchical	lat, array size	Gradient-based Methods	analytical estimation	not specified	quantization
AMG	2023	hierarchical	lat, energy, hw utilization	Bayesian Optimization	analytical estimation	FPGA	approximate multiplier
Energon	2023	not mentioned	lat, energy	other	analytical estimation	not specified	no
AnaCoNGA	2022	hierarchical	lat, DRAM,BRAM	nested genetic algorithms	analytical estimation	not specified	quantization
FGNAS	2022	hierarchical	lat, FFs, DSP blocks	Reinforcement Learning	analytical estimation	FPGA	no

Table A.2: HW-NAS on FPGA/ASIC Summary 02

Method	Year	Search Space	Objectives	Search Algorithm	Estimation Method	Target Device	Other Consideration
unnamed	2022	supernet	lat, energy, size, mem	Evolutionary Algorithm	predictive modeling	FPGA	no
S3NAS	2022	supernet	latency,FLOPs	Gradient-based Methods	predictive modeling	NPU, GPU, CPU	quantization/ pruning
unnamed	2022	supernet	lat, energy, mem, FLOPs	Bayesian Optimization	LUT	FPGA/CPU	no
EDD	2022	hierarchical	lat,throughput,energy,resource	Gradient-based Methods	analytical estimation	FPGA/GPU	quantization
SkyNet	2022	hierarchical	lat, accuracy	Evolutionary Algorithm	analytical estimation	FPGA	quantization
FracBNN	2022	hierarchical	lat,resource,energy	not mentioned	analytical estimation	FPGA	quantization
	2022	supernet	lat, energy, resources	other	predictive modeling	not specified	no
TBE	2022	unit-based	lat, energy	other	predictive modeling	not specified	no
MAPLE-Edge	2022	hierarchical	lat, energy, size, mem, FLOPs	other	predictive modeling	ASIC/FPGA	no
unnamed	2022	not mentioned	lat, energy	other	predictive modeling	not specified	no
Sherlock	2022	hierarchical	throughput, resource	other	analytical + predictive	FPGA	no
Auto-Vit-Acc	2022	hierarchical	lat, energy, DSPs, LUT	other	predictive modeling	FPGA	quantization
LW-GCN	2022	hierarchical	lat, energy, mem	other	analytical estimation	FPGA	quantization
unnamed	2022	layer-wise	lat, FLOPs, resource	not mentioned	analytical estimation	FPGA	pruning
HW-Flow	2022	hierarchical	lat, Ops, DRAM	Reinforcement Learning	analytical estimation	not specified	pruning
HP-GNN	2022	hierarchical	mem	other	analytical estimation	FPGA/CPU	no
unnamed	2022	hierarchical	lat, energy, efficiency, mem	other	analytical estimation	FPGA	quantization/pruning
unnamed	2022	not mentioned	mem, computational resource	other	real-world measurement	FPGA	no
unnamed	2021	hierarchical	latency, energy, mem	Reinforcement Learning	real-world measurement	ASIC/FPGA	pruning
unnamed	2021	hierarchical	mem, parallel data path	Bayesian Optimization	analytical estimation	FPGA	no
A3C-S	2021	hierarchical	lat, energy, size, mem	Gradient-based Methods	predictive modeling	not specified	no
LSFQ	2021	hierarchical	lat, energy, mem	Gradient-based Methods	analytical estimation	FPGA	quantization
SpAtten	2021	not mentioned	mem, computational resource	other	analytical + predictive	not specified	quantization/pruning
HALP	2021	layer-wise	lat,energy	other	LUT	not specified	pruning
Codesign-NAS	2020	hierarchical	area,latency,power	Reinforcement Learning	analytical estimation	FPGA	no
HotNAS	2020	hierarchical	lat,energy,resources	Reinforcement Learning	predictive modeling	FPGA	quantization/ pruning
ASICNAS	2020	hierarchical	lat,energy,area,NoC bandwidth	Reinforcement Learning	predictive modeling	ASIC	no
unnamed	2020	hierarchical	throughput,area,mem,energy	Reinforcement Learning	predictive modeling	FPGA	no
GAMMA	2020	hierarchical	lat, energy, tile sizes	Evolutionary Algorithm	analytical estimation	not specified	no
Prospector	2020	hierarchical	FFs, BRAM, LUT	Bayesian Optimization	analytical estimation	FPGA	no
BRP-NAS	2020	not mentioned	lat, energy, mem, FLOPs	other	predictive modeling	not specified	no
A3	2020	not mentioned	lat, energy	other	predictive modeling	ASIC	no
unnamed	2020	layer-wise	mem, computational resource	Gradient-based Methods	not mentioned	FPGA	pruning

Bibliography

[1] Sasan Salmani Pour Avval, Vahid Yaghoubi, Nathan D Eskue, and Roger M Groves. «Systematic Review on Neural Architecture Search». In: (2024) (cit. on p. 1).

[2] Han Cai, Ji Lin, Yujun Lin, Zhijian Liu, Kuan Wang, Tianzhe Wang, Ligeng Zhu, and Song Han. «Automl for architecting efficient and specialized neural networks». In: *IEEE Micro* 40.1 (2019), pp. 75–82 (cit. on pp. 4, 38).

[3] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. «Neural architecture search: Insights from 1000 papers». In: *arXiv preprint arXiv:2301.08727* (2023) (cit. on pp. 4, 11).

[4] Lukas Sekanina. «Neural architecture search and hardware accelerator co-search: A survey». In: *IEEE access* 9 (2021), pp. 151337–151362 (cit. on p. 6).

[5] Elias Vansteenkiste. «New FPGA Design Tools and Architectures». PhD thesis. Dec. 2016. DOI: 10.13140/RG.2.2.36199.24480 (cit. on p. 7).

[6] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. «A comprehensive survey on hardware-aware neural architecture search». In: *arXiv preprint arXiv:2101.09336* (2021) (cit. on pp. 8, 13, 26).

[7] Yongjia Yang, Jinyu Zhan, Wei Jiang, Yucheng Jiang, and Antai Yu. «Neural architecture search for resource constrained hardware devices: A survey». In: *IET Cyber-Physical Systems: Theory Applications* 8 (July 2023), n/a–n/a. DOI: 10.1049/cps2.12058 (cit. on p. 9).

[8] Nael Fasfous et al. «Anaconga: Analytical hw-cnn co-design using nested genetic algorithms». In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 238–243 (cit. on pp. 9, 19).

[9] Shikhar Tuli and Niraj K Jha. «EdgeTran: Co-designing transformers for efficient inference on mobile edge platforms». In: *arXiv preprint arXiv:2303.13745* (2023) (cit. on pp. 10, 13, 43).

[10] Jinqi Xiao, Chengming Zhang, Yu Gong, Miao Yin, Yang Sui, Lizhi Xiang, Dingwen Tao, and Bo Yuan. «HALOC: hardware-aware automatic low-rank compression for compact neural networks». In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 9. 2023, pp. 10464–10472 (cit. on p. 10).

[11] Qing Lu, Weiwen Jiang, Meng Jiang, Jingtong Hu, and Yiyu Shi. «Hardware/-Software Co-Exploration for Graph Neural Architectures on FPGAs». In: *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2022, pp. 358–362 (cit. on pp. 10, 13, 16).

[12] Haiyan Qin, Yejun Zeng, Jinyu Bai, and Wang Kang. «Searching Tiny Neural Networks for Deployment on Embedded FPGA». In: *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE. 2023, pp. 1–5 (cit. on pp. 10, 13, 30).

[13] Zhen Dong, Yizhao Gao, Qijing Huang, John Wawrzynek, Hayden KH So, and Kurt Keutzer. «Hao: Hardware-aware neural architecture optimization for efficient inference». In: *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2021, pp. 50–59 (cit. on pp. 11, 13, 30).

[14] Huizhen Kuang and Lingli Wang. «Multi-objective Design Space Exploration for High-Level Synthesis via Bayesian Optimization». In: *2023 International Symposium of Electronics Design Automation (ISEDA)*. IEEE. 2023, pp. 150–155 (cit. on pp. 11, 23, 31).

[15] Stephen Cha, Taehyeon Kim, Hayeon Lee, and Se-Young Yun. «A survey of supernet optimization and its applications: Spatial and temporal optimization for neural architecture search». In: *arXiv preprint arXiv:2204.03916* (2022) (cit. on p. 11).

[16] Taehee Jeong and Elliott Delaye. «Hardware-aware neural architecture search with segmentation-based selection». In: *2022 5th International Conference on Information and Computer Technologies (ICICT)*. IEEE. 2022, pp. 121–127 (cit. on p. 11).

[17] Erjing Luo, Haitong Huang, Cheng Liu, Guoyu Li, Bing Yang, Ying Wang, Huawei Li, and Xiaowei Li. «DeepBurning-MixQ: An open source mixed-precision neural network accelerator design framework for FPGAs». In: *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE. 2023, pp. 1–9 (cit. on p. 12).

[18] Mohamed S Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D Lane. «Best of both worlds: Automl codesign of a cnn and its hardware accelerator». In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6 (cit. on pp. 12, 13, 16, 26, 28).

[19] Jaeseong Lee, Jungsub Rhim, Duseok Kang, and Soonhoi Ha. «SNAS: Fast hardware-aware neural architecture search methodology». In: *IEEE transactions on computer-aided design of integrated circuits and systems* 41.11 (2021), pp. 4826–4836 (cit. on p. 12).

[20] Dong Dong, Hongxu Jiang, Xuekai Wei, Yanfei Song, Xu Zhuang, and Jason Wang. «ETNAS: An energy consumption task-driven neural architecture search». In: *Sustainable Computing: Informatics and Systems* 40 (2023), p. 100926 (cit. on pp. 12, 21).

[21] Nilotpal Sinha, Abd El Rahman Shabayek, Anis Kacem, Peyman Rostami, Carl Shneider, and Djamila Aouada. «Hardware aware evolutionary neural architecture search using representation similarity metric». In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2024, pp. 2628–2637 (cit. on pp. 13, 18).

[22] Grace Dinh, Iniyaal Kannan Jegadesan Valsala, Hengrui Luo, Charles Hong, Younghyun Cho, James Demmel, Sherry Li, and Yang Liu. «Sample-Efficient Mapspace Optimization for DNN Accelerators with Bayesian Learning». In: *Architecture and System Support for Transformer Models (ASSYST@ ISCA 2023)*. 2023 (cit. on pp. 13, 23, 31).

[23] Panjie Qi, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Hongwu Peng, Shaoyi Huang, Zhenglun Kong, Yuhong Song, and Bingbing Li. «Accelerating framework of transformer by hardware design and model compression co-optimization». In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2021, pp. 1–9 (cit. on pp. 13, 15, 27, 38, 43).

[24] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. «Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 4154–4165 (cit. on pp. 14, 15, 38).

[25] Taehee Jeong and Elliott Delaye. «Multi-objective optimization for Hardware-aware Neural Architecture Search». In: () (cit. on pp. 14, 18).

[26] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. «Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks». In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6 (cit. on pp. 14, 27).

[27] Guilherme Lopes, Murillo Ferreira, Alexandre Simões, and Esther Colombini. «Intelligent Control of a Quadrotor with Proximal Policy Optimization Reinforcement Learning». In: Nov. 2018, pp. 503–508. DOI: 10.1109/LARS/SBR/WRE.2018.00094 (cit. on p. 16).

[28] Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. «Hardware/software co-exploration of neural architectures». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.12 (2020), pp. 4805–4815 (cit. on p. 16).

[29] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. «A survey on neural architecture search». In: *arXiv preprint arXiv:1905.01392* (2019) (cit. on p. 17).

[30] Sheng-Chun Kao and Tushar Krishna. «Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm». In: *Proceedings of the 39th International Conference on Computer-Aided Design*. 2020, pp. 1–9 (cit. on p. 17).

[31] Bharath Srinivas Prabakaran, Vojtech Mrazek, Zdenek Vasicek, Lukas Sekanina, and Muhammad Shafique. «Xel-FPGAs: An End-to-End Automated Exploration Framework for Approximate Accelerators in FPGA-Based Systems». In: *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE. 2023, pp. 1–9 (cit. on pp. 17, 40).

[32] Xiaofan Zhang, Yuhong Li, Junhao Pan, and Deming Chen. «Algorithm/Accelerator co-design and co-search for edge AI». In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 69.7 (2022), pp. 3064–3070 (cit. on pp. 18, 35).

[33] Nilotpal Sinha, Peyman Rostami, Abd El Rahman Shabayek, Anis Kacem, and Djamila Aouada. «Multi-Objective Hardware Aware Neural Architecture Search using Hardware Cost Diversity». In: *arXiv preprint arXiv:2404.12403* (2024) (cit. on pp. 18, 31).

[34] Lotte Hendrickx, Arne Symons, Wiebe Van Ranst, Marian Verhelst, and Toon Goedemé. «Hardware-aware NAS by Genetic Optimisation with a Design Space Exploration Simulator». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 2275–2283 (cit. on p. 18).

[35] Halima Bouzidi, Hamza Ouarnoughi, El-Ghazali Talbi, Abdessamad Ait El Cadi, and Smail Niar. «Evolutionary-based co-optimization of dnn and hardware configurations on edge gpu». In: *International Conference on Optimization and Learning*. Springer. 2022, pp. 3–12 (cit. on p. 19).

[36] Christoph Schorn, Thomas Elsken, Sebastian Vogel, Armin Runge, Andre Guntoro, and Gerd Ascheid. «Automated design of error-resilient and hardware-efficient deep neural networks». In: *Neural Computing and Applications* 32 (2020), pp. 18327–18345 (cit. on pp. 19, 35).

[37] George Barnett, Simon Funke, and Matthew Piggott. «Hybrid global-local optimisation algorithms for the layout design of tidal turbine arrays». In: (Oct. 2014) (cit. on p. 20).

[38] Shikhar Tuli and Niraj K Jha. «TransCODE: Co-design of transformers and accelerators for efficient training and inference». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.12 (2023), pp. 4817–4830 (cit. on pp. 21, 32, 38).

[39] Guorui Xie, Qing Li, Zhenning Shi, Hanbin Fang, Shengpeng Ji, Yong Jiang, Zhenhui Yuan, Lianbo Ma, and Mingwei Xu. «Generating Neural Networks for Diverse Networking Classification Tasks via Hardware-Aware Neural Architecture Search». In: *IEEE Transactions on Computers* (2023) (cit. on p. 21).

[40] Xiangzhong Luo. «Hardware-aware neural architecture search and compression towards embedded intelligence». In: (2023) (cit. on p. 21).

[41] Alireza Ghaffari and Yvon Savaria. «Efficient design space exploration of OpenCL kernels for FPGA targets using black box optimization». In: *IEEE Access* 9 (2021), pp. 136819–136830 (cit. on p. 22).

[42] Atefeh Mehrabi, Aninda Manocha, Benjamin C Lee, and Daniel J Sorin. «Bayesian optimization for efficient accelerator synthesis». In: *ACM Transactions on Architecture and Code Optimization (TACO)* 18.1 (2020), pp. 1–25 (cit. on p. 22).

[43] Alireza Ghaffari, Masoud Asgharian, and Yvon Savaria. «Statistical Hardware Design With Multi-model Active Learning». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023) (cit. on p. 22).

[44] Shikhar Tuli, Chia-Hao Li, Ritvik Sharma, and Niraj K Jha. «CODEBench: A neural architecture and hardware accelerator co-design framework». In: *ACM Transactions on Embedded Computing Systems* 22.3 (2023), pp. 1–30 (cit. on p. 23).

[45] Arnab Neelim Mazumder and Tinoosh Mohsenin. «Reg-TuneV2: Hardware-Aware and Multi-Objective Regression-Based Fine-Tuning Approach for DNNs on Embedded Platforms». In: *IEEE Micro* (2023) (cit. on pp. 24, 31, 35).

[46] Edgar Liberis, Łukasz Dudziak, and Nicholas D Lane. «μnas: Constrained neural architecture search for microcontrollers». In: *Proceedings of the 1st Workshop on Machine Learning and Systems*. 2021, pp. 70–79 (cit. on p. 25).

[47] Bingqian Lu, Jianyi Yang, Weiwen Jiang, Yiyu Shi, and Shaolei Ren. «One proxy device is enough for hardware-aware neural architecture search». In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5.3 (2021), pp. 1–34 (cit. on p. 27).

[48] Xiangzhong Luo, Di Liu, Hao Kong, Shuo Huai, Hui Chen, and Weichen Liu. «Work-in-Progress: What to Expect of Early Training Statistics? An Investigation on Hardware-Aware Neural Architecture Search». In: *2022 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE. 2022, pp. 1–2 (cit. on p. 29).

[49] Sewon Min et al. «Neurips 2020 efficientqa competition: Systems, analyses and lessons learned». In: *NeurIPS 2020 Competition and Demonstration Track*. PMLR. 2021, pp. 86–111 (cit. on p. 29).

[50] Saeejith Nair, Saad Abbasi, Alexander Wong, and Mohammad Javad Shafiee. «Maple-edge: A runtime latency predictor for edge devices». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 3660–3668 (cit. on p. 29).

[51] Yong Ji, Daikun Wei, Leiyang Shi, Peng Liu, Cheng Li, Jun Yu, and Xu Cai. «An Active Learning based Latency Prediction Approach for Neural Network Architecture». In: *2024 4th International Conference on Neural Networks, Information and Communication (NNICE)*. IEEE. 2024, pp. 967–971 (cit. on p. 29).

[52] Kurt Stolle, Sebastian Vogel, Fons van der Sommen, and Willem Sanberg. «Block-Level Surrogate Models for Inference Time Estimation in Hardware-Aware Neural Architecture Search». In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2022, pp. 463–479 (cit. on p. 29).

[53] Yue Hu, Chongfei Shen, Lixin Yang, Zhipeng Wu, and Yu Liu. «A novel predictor with optimized sampling method for hardware-aware NAS». In: *2022 26th International Conference on Pattern Recognition (ICPR)*. IEEE. 2022, pp. 2114–2120 (cit. on p. 30).

[54] Yash Akhauri and Mohamed S Abdelfattah. «Multi-predict: few shot predictors for efficient neural architecture search». In: *arXiv preprint arXiv:2306.02459* (2023) (cit. on p. 30).

[55] Yonggan Fu, Yongan Zhang, Chaojian Li, Zhongzhi Yu, and Yingyan Lin. «A3C-S: Automated agent accelerator co-search towards efficient deep reinforcement learning». In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2021, pp. 13–18 (cit. on p. 31).

[56] Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smail Niar. «Multi-objective hardware-aware neural architecture search with Pareto rank-preserving surrogate models». In: *ACM Transactions on Architecture and Code Optimization* 20.2 (2023), pp. 1–21 (cit. on p. 32).

[57] Quentin Gautier, Alric Althoff, Christopher L Crutchfield, and Ryan Kastner. «Sherlock: A multi-objective design space exploration framework». In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 27.4 (2022), pp. 1–20 (cit. on p. 32).

[58] Tae Jun Ham et al. «Aˆ 3: Accelerating attention mechanisms in neural networks with approximation». In: *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2020, pp. 328–341 (cit. on p. 34).

[59] Zhengang Li et al. «Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization». In: *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE. 2022, pp. 109–116 (cit. on pp. 35, 44).

[60] Javier Campos, Jovan Mitrevski, Nhan Tran, Zhen Dong, Amir Gholaminejad, Michael W Mahoney, and Javier Duarte. «End-to-end codesign of Hessian-aware quantized neural networks for FPGAs». In: *ACM Transactions on Reconfigurable Technology and Systems* () (cit. on p. 35).

[61] Zhenshan Bao, Kang Zhan, Wenbo Zhang, and Junnan Guo. «LSFQ: A low precision full integer quantization for high-performance FPGA-based CNN acceleration». In: *2021 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE. 2021, pp. 1–6 (cit. on p. 35).

[62] Zhuofu Tao, Chen Wu, Yuan Liang, Kun Wang, and Lei He. «Lw-gcn: A lightweight fpga-based graph convolutional network accelerator». In: *ACM Transactions on Reconfigurable Technology and Systems* 16.1 (2022), pp. 1–19 (cit. on p. 36).

[63] Kai Zhong, Shulin Zeng, Wentao Hou, Guohao Dai, Zhenhua Zhu, Xuecang Zhang, Shihai Xiao, Huazhong Yang, and Yu Wang. «CoGNN: An algorithm-hardware co-design approach to accelerate GNN inference with minibatch sampling». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.12 (2023), pp. 4883–4896 (cit. on p. 36).

[64] Xiaofan Zhang, Yao Chen, Cong Hao, Sitao Huang, Yuhong Li, and Deming Chen. «Compilation and Optimizations for Efficient Machine Learning on Embedded Systems». In: *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Software Optimizations and Hardware/Software Codesign*. Springer, 2023, pp. 37–74 (cit. on p. 36).

[65] Song Han, Jeff Pool, John Tran, and William Dally. «Learning both weights and connections for efficient neural network». In: *Advances in neural information processing systems* 28 (2015) (cit. on p. 37).

[66] Jef Plochaet and Toon Goedemé. «Hardware-aware pruning for fpga deep learning accelerators». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 4482–4490 (cit. on pp. 37, 38).

[67] Hengyi Li, Xuebin Yue, Zhichen Wang, Zhilei Chai, Wenwen Wang, Hiroyuki Tomiyama, and Lin Meng. «Optimizing the Deep Neural Networks by Layer-Wise Refined Pruning and the Acceleration on FPGA». In: *Computational Intelligence and Neuroscience* 2022.1 (2022), p. 8039281 (cit. on p. 37).

[68] Manoj-Rohit Vemparala et al. «Hw-flow: A multi-abstraction level hw-cnn codesign pruning methodology». In: (2022) (cit. on p. 38).

[69] Yi-Chien Lin, Bingyi Zhang, and Viktor Prasanna. «Hp-gnn: Generating high throughput gnn training implementation on cpu-fpga heterogeneous platform». In: *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2022, pp. 123–133 (cit. on p. 39).

[70] Hanrui Wang, Zhekai Zhang, and Song Han. «Spatten: Efficient sparse attention architecture with cascade token and head pruning». In: *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE. 2021, pp. 97–110 (cit. on pp. 39, 44).

[71] Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose M Alvarez. «Halp: hardware-aware latency pruning». In: *arXiv preprint arXiv:2110.10811* (2021) (cit. on p. 39).

[72] Krishna Teja Chitty-Venkata, Yiming Bian, Murali Emani, Venkatram Vishwanath, and Arun K Somani. «Differentiable Neural Architecture, Mixed Precision and Accelerator Co-Search». In: *IEEE Access* (2023) (cit. on p. 39).

[73] Mengshu Sun, Pu Zhao, Mehmet Gungor, Massoud Pedram, Miriam Leeser, and Xue Lin. «3D CNN acceleration on FPGA using hardware-aware pruning». In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2020, pp. 1–6 (cit. on p. 39).

[74] Sai Subra Chitty-Venkata. «Hardware-aware design, search, and optimization of deep neural networks». PhD thesis. Iowa State University, 2023 (cit. on p. 39).

[75] Vojtech Mrazek, Lukas Sekanina, and Zdenek Vasicek. «Libraries of approximate circuits: Automated design and application in CNN accelerators». In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10.4 (2020), pp. 406–418 (cit. on p. 40).

[76] Zhen Li, Hao Zhou, Lingli Wang, and Xuegong Zhou. «AMG: Automated Efficient Approximate Multiplier Generator for FPGAs via Bayesian Optimization». In: *2023 International Conference on Field Programmable Technology (ICFPT)*. IEEE. 2023, pp. 294–295 (cit. on p. 41).

[77] Hongwu Peng et al. «A length adaptive algorithm-hardware co-design of transformer on fpga through sparse attention and dynamic pipelining». In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, pp. 1135–1140 (cit. on p. 43).

[78] Zhe Zhou, Junlin Liu, Zhenyu Gu, and Guangyu Sun. «Energon: Toward efficient acceleration of transformers using dynamic sparse attention». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.1 (2022), pp. 136–149 (cit. on p. 44).