Deep Learning Acceleration and Beyond

Wenqian Zhao

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

in

Computer Science and Engineering

The Chinese University of Hong Kong

August 2024

Thesis Assessment Committee

Professor KING Kuo Chin Irwin (Chair)

Professor YU Bei (Thesis Supervisor)

Professor YANG Ming-chang (Committee Member)

Professor PAN David Z.(External Examiner)

Abstract

The rapid advancement of deep learning techniques has revolutionized various domains, including artificial intelligence (AI) algorithm optimization/acceleration and hardware design reformation. My PhD thesis explores the intersection of these two fields, focusing on hardware-aware AI acceleration and following AI-aided Design Automation acceleration. The objective is to develop novel approaches and methodologies that fully explore potential acceleration opportunities to optimize and accelerate AI computation from algorithm-level optimization and hardware-level optimization and following hardware design automation stage. These processes are mutually coupled and mutually reinforcing, propelling and enhance each other's progression.

The first part of the thesis focuses on Hardware-aware AI acceleration. Given domain specific and resource-limited hardware, efficient deployment of continuously-updated AI models requires both algorithm-level and system-level optimization.

The first work focuses on algorithm-level domain specific inference optimization, presenting a research study on super-resolution (SR) processing, focusing on addressing the challenge of achieving real-time inference in SR models. I develop a full-stack SR acceleration framework specifically designed for embedded GPU devices. The study analyzes and enhances the special dictionary learning algorithm used in SR models through a novel dictionary selective strategy, aiming to compress the model with tailored structural pruning. Additionally, we investigate the hardware programming architecture and model structure to guide the optimal design of computation kernels, minimizing inference latency within resource constraints.

Achieving optimal inference efficiency on hardware requires a combination of algorithm-level model compression techniques, such as model quantization, and system-level optimization, such

as operation reconfiguration and scheduling. The second work introduces a unified deployment framework called BAQE, which aims to bridge the gap between algorithm-level and backend-level optimization. BAQE constructs a global search space to simultaneously optimize model quantization settings and backend configuration parameters. To expedite the process, I propose a searching strategy based on multi-objective Bayesian optimization (BO) using a multi-task Gaussian model as the surrogate model. Importantly, the framework easily adapts to different backends with varying hardware resources due to its awareness of genuine hardware capabilities in each step. The proposed approach evaluates all accuracy/latency metrics and historic knowledge/feedback directly on the device within each iteration, thus accelerating the optimization process.

The second part of the thesis addresses AI acceleration of downstream task with example on VLSI design automation. As the scale and complexity of integrated circuits are increasing very fast, integrating deep learning into the design automation workflow may enhance the efficiency and accuracy and handle Design For Manufactory (DFM) problems. As very large-scale integration (VLSI) technology node continues to shrink, lithography proximity effects have become significant, affecting manufacturability. To address this, resolution enhancement techniques (RETs) like Optical Proximity Correction (OPC) are used. OPC optimization has gained attention in academia and industry, with a focus on numerical optimization and machine learning. Our research identifies pattern complexity variations and repetitive patterns in design layouts. Leveraging these findings, we propose a self-adaptive OPC framework. It selectively employs different OPC solvers for patterns of varying complexity and reuses optimized masks for repeated patterns, resulting in improved efficiency.

Keywords: Deep learning, AI acceleration, Hardware-aware optimization, Domain-specific acceleration, Performance optimization.

摘要

深度學習技術的快速發展已經徹底改變了多個領域,包括人工智能(AI)算法優化/加速和硬體設計變革。我的博士論文探討了這兩個領域的交叉點,重點關注硬體感知的AI加速以及後續的AI輔助設計自動化加速。研究目標是開發新的方法和方法論,充分探索潛在的加速機會,從算法層面和硬體層面的優化以及後續的硬體設計自動化階段來優化和加速AI計算。這些過程相互耦合、相互促進,推動和增强彼此的進展。

論文的第一部分重點介紹硬體感知的AI加速。考慮到特定領域和資源受限的硬體,對不斷更新的AI模型進行高效部署需要同時進行算法層面和系統層面的優化。

第一項工作側重於算法層面的特定領域推理優化,展示了對超分辨率(SR)處理的研究,重點是解決SR模型實現即時推理的挑戰。我專門為嵌入式GPU設備開發了一個完整的SR加速框架。該研究通過一種新穎的字典選擇策略分析並增强了SR模型中使用的特殊字典學習算法,旨在通過定制的結構化剪枝來壓縮模型。此外,我們還研究了硬體編程架構和模型結構,以指導計算核心的最優設計,在資源約束下最小化推理延遲。

要在硬體上實現最佳推理效率,需要結合算法層面的模型壓縮技術(如模型量化)和系統層面的優化(如操作重構和調度)。第二項工作介紹了一個名為BAQE的統一部署框架,旨在彌合算法層面和後端層面優化之間的差距。BAQE構建了一個全局搜索空間,同時優化模型量化設置和後端配置參數。為了加快這個過程,我提出了一種基於多目標貝葉斯優化(BO)的搜索策略,使用多任務高斯模型作為代理模型。重要的是,由於框架在每一步中都能意識到真實的硬體能力,因此它很容易適應具有不同硬體資源的不同後端。所提出的方法在每次迭代中直接在設備上評估所有準確性/延遲指標和歷史知識/反饋,從而加速了優化過程。

論文的第二部分探討了VLSI設計自動化領域的AI加速。隨著集成電路的規模和複雜性急劇增加,將深度學習集成到設計自動化工作流程中可以提高效率和準確性,並處理可製造性設計(DFM)問

題。隨著超大規模集成電路(VLSI)技術節點的不斷縮小,光刻接近效應變得非常顯著,影響了可製造性。為了解決這個問題,使用了分辨率增强技術(RET),如光學接近修正(OPC)。OPC優化已經引起學術界和工業界的關注,重點是數值優化和機器學習。我們的研究識別了設計佈局中的圖案複雜度變化和重複圖案。利用這些發現,我們提出了一個自適應OPC框架。它有選擇地為不同複雜度的圖案採用不同的OPC求解器,並為重複圖案重複使用優化後的掩模,從而提高了效率。

關鍵字:深度學習,AI加速,硬體感知優化,特定領域加速,性能優化

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Artificial Intelligence (AI) technology has thrived in the past decade with leaping advancements that surprised the world. On the other hand, the remarkable achievements in AI algorithms and system developments bring more pressure on the advancement of hardware platforms, which emerge from the water and attract more attention and challenges to meet the massive computation workload requirement. Although many new hardware architectures and devices at different scales are proposed to improve performance, they still need to catch up with the increase in AI algorithms' scaling model size and complexity.

Orthogonal to accuracy performance, speed is another dimension of model. AI model complexity and capacity is also another concern. Given the rocket-growth of both AI performance and model size, the mass data processing feature of latest machine learning algorithms requires latest design with numerous data processing units and high-parallelism property. Latest bulky models have gigantic parameter number which highly requires distributed computation and fast high-bandwidth communication among devices clusters. Many domain specific tasks such as super-resolution are highly suffer pressure from computation complexity cost. Some domains have special requirements on the model inference time, such as video data analysis that strictly requires real-time inference.

Nowadays, software hardware co-design is the new trend for acceleration in the new AI era. As the parameter-based AI models increase exponentially, the model architecture also evolves to adapt to the computation paradigm of highly parallel extensive matrix/tensor processing. Some

new hardware appears to support sparse computation, low-bit computation, and high-bandwidth data movement. With GPU evolving as the mainstream computation hardware for AI, many other domain-specific hardware emerge, such as TPU, special FPGA accelerators, etc. At the same time, the AI algorithms and models adapt as well; model pruning has been proposed to make the models sparse and, therefore, hold fewer parameters and smaller sizes. Quantization techniques are also involved in converting the model parameters and computation from full precision (32-bit floating point) to lower-bit data format. To adapt to different hardware resources, AI frameworks need compilation optimization to adapt to different backend architectures, such as loop-tiling and computation kernel fusion.

Another acceleration corner is the hardware design itself. To mitigate the gap between software and hardware development, new manufacturing technology nodes have appeared with a shrinking gate size such that more transistors can be put into the same area and power consumption can be reduced. However, the advancement in lithography technology has brought more challenges, such that the gate size approaches the wavelength of incident light, and the lithography proximity effect will affect the final pattern printed on the chip. These advancements pose additional challenges for the physical design stage of VLSI design flow. Here, the foundaries play a pivotal role in determining various design rules. These rules, such as the component distance limit or routing line distance, are crucial in ensuring the manufacturing yield. On the other hand, resolution enhancement techniques (RETs) have been developed to improve the printability of the lithography process. Optical Proximity Correction (OPC) is one of the widely used RETs to optimize mask printability by compensating for the diffraction effect in the lithography process. Indeed, all these techniques are still limited to the massive design scale and bounded by time, budget, and computation limitations.

1.1 Challenges

Computational Complexity limit. Same as the AI for mask optimization that treats mask generation as conditional image generation task. Another vision task super-resolution also suffers extreme computation cost as image resolution gets higher to 4K. Despite the effectiveness and usability of these vendor-provided commercial deployment tools, SR algorithms still face some complex

and thorny problems which hinder the models from traditional Deep learning optimization strategies. To realize the objective of real-time inference (*i.e.*, equal or more than 25 frames per second), some particular properties of SR algorithms need to be considered.

Hardware-unaware inefficiency. As for AI compression on specific hardware such as quantization, the process can theoretically reduce bit-level operation number. However, selecting bit-width and tuning the model is not the final stage in the real scenario of quantization deployment. A complete deployment flow includes the important stage of backend configuration optimization on specific backends. This is a crucial step to fully utilize the hardware resources on the device and to explore for higher inference efficiency. Hence, it is necessary to propose a compilation method to map the quantized DNN model into a series of backend-level kernels to launch, which remains not well solved.

Robusness of End-to-end AI. Although AI has shown much potential to promote higher efficiency in Design Automation flow, there are many inherent challenges that may possibly hazard the design result. Firstly, for AI-aided mask optimization specifically, the robustness is a huge problem. To achieve efficient and desired OPC results on real designs, it is necessary to conduct a systematic analysis of pattern distribution and complexity. Through careful examination of a real design, we have identified several properties that can be utilized to aid in this analysis. One such property is the presence of varying pattern densities, indicating that certain regions exhibit high density while others are more sparse. This *diversity*, suggests the need for different types of OPC solutions in different regions.

The lethal drawback is that machine learning model is a data-driven black box based on probabilities. Such methods are not guaranteed to work for some critical patterns. We cannot solely rely on AI for end-to-end solution without further checking.

1.2 Thesis Overview

. The thesis revolves around hardware-aware AI acceleration and AI acceleration for hardware design automation. Many stages of optimization are covered with cross-level optimization.

Chapter 2 is the literature review section with background and previous works. This section also briefly introduces the problem definition.

Chapter 3 focuses on Hardware-aware AI acceleration with algorithm-level domain specific inference optimization, presenting a research study on super-resolution (SR) processing, focusing on addressing the challenge of achieving real-time inference in SR models. A full-stack SR acceleration framework specifically designed for embedded GPU devices is proposed.

Chapter 4 discusses combination of algorithm-level model compression techniques, such as model quantization, and system-level optimization, such as operation reconfiguration and scheduling. A unified deployment framework called BAQE, which aims to bridge the gap between algorithm-level and backend-level optimization is introduced.

Chapter 5 mainly discusses AI-aided Design Automation acceleration in terms of Design For Manufactory (DFM). This chapter proposes a self-adaptive OPC framework to replace conventional lithography mask optimization flow with superior performance and efficiency. A customized dynamic pattern library is proposed to reuse repetitive patterns, using a hierarchical graph with online update along with a greedy graph-based nearest neighbor search for fast matching.

Chapter 2

Literature Review

2.1 Super-Resolution Algorithm Acceleration

Super-resolution (SR) techniques are crucial graphical processing methods that hold significant relevance in the digital image era. They play a vital role in the generation or restoration of high-resolution (HR) video frames from low-resolution (LR) frames. Out of all the available methods, the basic approach is to interpolate the low-resolution (LR) image using bilinear or bicubic RGB values obtained from nearby pixels that are spatially invariant. The significant advancements in deep learning within computer vision have paved the way for a range of robust super-resolution (SR) techniques with impressive performance. Over the past decade, various methods have emerged, ranging from conventional convolutional neural networks (Dong et al., 2015) to innovative generative adversarial networks (Ledig et al., 2017a; Kim et al., 2020). In recent developments, the incorporation of dictionary learning techniques alongside pixel-level local feature fusion operations (Li et al., 2020b; Wenbo et al., 2020) has led to further enhancements in the image quality of generated high-resolution images or videos. These advancements have resulted in the recovery of richer color and texture details, thanks to the utilization of dictionary learning and pixel-level local feature fusion operations.

As these algorithms continue to improve in performance, there is a growing focus on the efficient and optimized deployment of deep learning-based super-resolution (SR) methods on hardware. This aspect has garnered increased attention within the field. Over the past few years, Single-image-super

resolution and related super-sampling techniques have found applications in real-time upscaling of content up to 4K resolution and even higher resolution (Xiao et al., 2020) for the vast application in upcoming 4K displays, laptops, and TVs. The process of AI-based content upscaling often involves computationally intensive operations, primarily due to the large-scale convolution operations in neural networks. The complexity of these computations, measured in terms of Multiply-Accumulate (MAC) operations, increases significantly with precision PSNR/SSIM. To address this issue, several lightweight methods have been proposed, aiming to reduce model size and complexity (Dong et al., 2016; Ahn et al., 2018). However, even the lightest method, FSRCNN (Dong et al., 2016), may not meet the real-time requirement of 33 frames per second (fps) without specific customization or hardware resource. More flexible SR compression algorithms appeared to satisfy various requirements. SESR (Bhardwaj et al., 2022) proposes flexible model size to meet different size requirement. (Lee et al., 2020) adopt Neural Architecture Search (NAS) to automatically search for the most efficient model architecture to achieve extreme parameter efficiency.

Several previous methods have been proposed to facilitate the domain-specific deployment of different these algorithms on various hardware platforms (Hao et al., 2019; Guo et al., 2020; Li et al., 2019a; Sun et al., 2021). These models are widely employed for tasks such as object classification (Wei et al., 2017), detection (Pinkham et al., 2020; Bai and Wang, 2019), neural language processing (NLP) (Cao et al., 2019), and more. Despite their diverse application scenarios and variations in task data format, these models share similar deep learning operators in their implementations, eliminating the need for explicit special techniques. The most commonly used operators include convolution, pooling, softmax, fully connected operations, and others. Due to the widespread use of these operators, vendor-provided commercial tools often apply customization to achieve state-of-the-art performance. These tools utilize fixed, manually written hardware codes tailored to specific operators. For instance, TensorRT (GPU, [n. d.]) outperforms other tools on NVIDIA GPUs, while Intel MKL-DNN (CPU, [n. d.]) boasts dominant inference latency on Intel CPUs.

2.2 Learning Model Compression And Quantization

In recent times, deep neural networks (DNNs) have emerged as highly successful in various visual/language tasks. Nonetheless, the computational and memory requirements of current DNN models present challenges for their utilization in devices with limited memory resources or applications with stringent latency demands. Consequently, there is a natural inclination towards compressing and accelerating deep networks while preserving model performance to a significant extent. Over the past five years, substantial advancements have been achieved in this field, showcasing remarkable progress. Previous studies have demonstrated the effectiveness of network pruning and quantization in reducing network complexity and mitigating the issue of overfitting.

The main objective of pruning is to prune redundant, non-informative weights in a pre-trained DNN model. (Srinivas and Babu, 2015) firstly investigated the redundancy within neurons in neural networks and introduced a data-free pruning method to eliminate redundant neurons. Han (Han et al., 2015b) started to treat the network as a whole pruning candidate and proposed a method to reduce the overall number of parameters and operations in the network with a train-prune-retrain 3-step strategy. (Chen et al., 2015) developed the HashedNets model, which employed a low-cost hash function to group weights into hash buckets for parameter sharing. The deep compression approach described in [10] eliminated redundant connections, quantized weights, and then utilized Huffman coding to encode the quantized weights. In (Ullrich et al., 2017), a simple regularization method based on soft weight-sharing was proposed, which encompassed both quantization and pruning within a single re-training procedure. The aforementioned pruning strategies primarily focus on connections pruning in DNNs.

On the other hand, quantization has emerged as a highly effective strategy to accelerate DNN inference by utilizing lower numerical precision. In practical deep learning scenarios such as autonomous driving and VR/AR technology, quantization enables deployment without necessitating modifications to the original architecture. The fundamental concept behind quantization involves replacing the FP32 numerical representation with half-precision FP16 or even Int8/Int4 formats. This process is commonly expressed as:

$$\text{quantize}(r) = \text{round}(r/S) - Z, \tag{2.1}$$

Here, the operation quantize$(\cdot)$ represents the quantization process. The original weight/activation r is initially scaled by a real value factor S in FP32 precision. Subsequently, it is rounded to the nearest integer value (with truncation). The integer value Z serves as the zero point, which calibrates the mapping value shift from the floating-point range to the integer range.

Numerous prior studies, including those mentioned in (Rastegari et al., 2016; Zhou et al., 2017; Jacob et al., 2018; Zhao et al., 2023; Nagel et al., 2020; Yao et al., 2021; Li et al., [n. d.]), have extensively utilized weight or activation quantization techniques with low-bit precision. Among these approaches, integer quantization has demonstrated remarkable effectiveness in real-world deployment scenarios due to its compatibility with hardware constraints. In general, fixed-point arithmetic operations, such as multiplication and addition, are significantly simpler and faster when using integer quantization. These operations involve only shift and add operations, eliminating the need for time-consuming steps like normalization, de-normalization, or exponent alignment commonly found in floating-point arithmetic.

The process of quantization can potentially result in a reduction in accuracy, particularly for layers that are sensitive to precision deduction. To address this issue, mixed-precision approaches have been proposed, as mentioned in (Dong et al., 2019; Habi et al., 2020; Hu et al., 2021; Qu et al., 2020; Wang et al., 2020b; Zhao et al., 2021b; Zhou et al., 2018). These approaches aim to mitigate accuracy degradation by assigning different bit precisions to each layer. However, this fine-grained distribution of bit widths poses a precision assignment problem, with the number of choices exponentially increasing as the network depth grows. In the work of (Wu et al., 2018a), the Differentiable NAS (DNAS) method was employed to efficiently select the bit candidates. On the other hand, (Dong et al., 2019) determined the bit width based on the trace of the Hessian matrix. In the study by (Yao et al., 2021), integer linear programming was utilized to optimize the bit width selection process.

Compared to other DNN model compression techniques like model pruning or knowledge distillation, quantization heavily relies on hardware support. The availability of hardware support for quantized data types significantly impacts the options for quantized bit-width and the resulting acceleration ratio. Notably, NVIDIA's Ampere GPU architecture with 3rd generation Tensor Core offers unprecedented acceleration for tensor operations at Int1/4/8, bfloat16, and FP16 precision.

Similarly, ARM Neon, with its Single Instruction Multiple Data (SIMD) architecture, can handle 8, 16, 32, and 64-bit vector operations. It is important to note that the effectiveness of quantization for acceleration may vary based on the available hardware resources. With the versatile bit-width support provided by advanced hardware, algorithm designers have greater flexibility to apply mixed-precision quantization across different layers. Numerous previous works (Dong et al., 2019; Habi et al., 2020; Hu et al., 2021; Qu et al., 2020; Wang et al., 2020b; Zhao et al., 2021b; Zhou et al., 2018) have explored this direction to determine the optimal bit-width configuration for each layer of a DNN model.

2.3 Lithography Mask Optimization

The continuous reduction in VLSI technology nodes has had a significant impact on the manufacturability of integrated circuits, primarily due to the notable lithography proximity effect (Pan et al., 2013). This effect can lead to complications during the printing process. To overcome this challenge and enhance the printability of lithography, resolution enhancement techniques (RETs) are employed.

Among the various RETs, optical proximity correction (OPC) is widely utilized. OPC optimizes the printability of masks by compensating for the diffraction effect that occurs during lithography. OPC methods can be classified into the following categories:

- Rule-based OPC (Park et al., 2000),
- Model-based OPC (Kuang et al., 2015; Su et al., 2016; Matsunawa et al., 2015),
- Inverse lithography technique (ILT)-based OPC (Poonawala and Milanfar, 2007; Ma et al., 2020; Yu et al., 2021), and
- Machine learning (ML)-based OPC (Yang et al., 2019; Jiang et al., 2019; Geng et al., 2019; Chen et al., 2021).

Rule-based techniques present a heuristic approach to tackle the issue, providing a straightforward and efficient solution. However, their suitability is limited to less complex designs. In contrast, model-based OPC techniques utilize mathematical models of the lithography process to precisely

adjust the mask's edge fractures, ensuring a high level of accuracy. Nonetheless, these methods encounter limitations in dealing with advanced technology nodes due to constraints within the solution space. Conversely, ILT-based methods tackle the OPC problem by solving the inverse problem of the imaging system, employing an optimized objective function. This approach represents the most effective analytical method for addressing the OPC challenge. In recent years, the rapid progress of machine learning algorithms and hardware has led to remarkable advancements in ML-based OPCs, resulting in accelerated workflows and gaining recognition in the academic field of design for manufacturing (DFM)(Wang et al., 2023; Pei et al., 2023b; Zhao et al., 2023). Notably, studies like(Yang et al., 2019) have utilized deep learning models to generate initial masks, reducing the required number of iterations for ILT. Additionally, (Jiang et al., 2020) employed a deep learning model to simulate the conventional ILT correction process.

In general, the machine learning approaches in lithography mask optimization can be categorized into two trends based on task definition: discriminative approaches and generative approaches. For example, (Jiang et al., 2020) is the one of the discriminative methods that utilize machine learning model to predict the mask printability, accelerating the OPC process with EPE prediction and edge movement guidance. Another example of generative approaches is (Yang et al., 2019) that leverages generative adversarial networks (GANs) to generate the desired mask for a given target pattern. A proper initial mask pattern is generated to for OPC engine to avoid redundant iterations of modification. Typically, learning-based approaches rely on a labeled dataset and undergo supervised training using a loss function. However, it is important to acknowledge that machine learning models have a significant drawback of being opaque, relying solely on data and lacking interpretability. Consequently, these methods do not guarantee effectiveness for critical patterns. In conclusion, there is no flawless or universally superior approach. Different approaches are necessary for patterns of varying complexities.

As for lithography simulation, machine learning-based techniques also have been suggested for simulating lithography. For example, (Watanabe et al., 2017) utilized a CNN to establish the function model for resist model simulation. Another study by (Ye et al., 2019) presented LithoGAN, a GAN-based architecture that aims to transform input masks into output resist patterns. Furthermore, (Shao et al., 2020) introduced a two-stage DNN-based approach that treats the mask-to-SEM

prediction as a domain transfer problem, using CycleGAN (Zhu et al., 2017) to model the transfer process.

Chapter 3

Domain Specific Sparse Super-Resolution Acceleration

Domain Specific AI acceleration is one of the critical obstacles for AI application in Design Automation flow. Similar to Mask Optimization, Super-resolution (SR) techniques plays an important role in the digital image era treats image as 2-D matrix, same way as OPC. The SR task aims at generating or recovering high-resolution (HR) video frames given frames with low-resolution (LR).

This section proposes several specific techniques to tackle those mentioned challenges. Firstly, we bring in an agile yet robust SR model compression strategy to reduce the size of large models with dense parameters and heavy computation. Structured pruning was utilized to delicately select and slim down the SR dictionaries. Meanwhile, the strategy needs to reserve the most important dictionaries and remarkably accelerate some serial computation iterations with no accuracy degradation.

Secondly, we also strive to achieve optimal hardware-level implementation of tensor operations given the SR models and specific hardware resources. The GPU architecture is discussed in details. Both memory/cache resources and computing power are considered constraints of inference efficiency. Performance and number of feasible hardware implementations are restricted by these restrictions. Some invalid and inefficient designs are dropped and a novel design space searching algorithm based

on Bayesian optimization is proposed focusing on efficiently finding the optimal design parameters in regard of these hardware limitations. As a result, we manage to reduce the communication cost and further improve bandwidth usage. Last but not least, we re-organize the original large task into many smaller sub-tasks and run these sub-tasks in parallel.

The main contributions of this section are listed as follows:

- We build a specifically designed engine to remarkably accelerate dictionary learning, especially on extremely large data for the first time.
- We propose a model slimming strategy for SR dictionary queries, which greatly reduces computation and communication workloads from the large data frames and heavy dictionary-related computation.
- We propose a targeted 8-bit adaptive quantization approach to further compress the SR model and realize further acceleration.
- We analyze both resources- and workloads-aware constraints dedicated for GPUs to guide the search for optimal hardware implementations.
- Our method achieves faster and real-time SR processing on edge embedded GPU NVIDIA Jetson Xavier NX and server-level 2080Ti, in comparison with TensorRT.

3.1 Preliminaries

The primary goal of the Super-resolution algorithm is to generate a high-resolution (HR) image by restoring the lost details from a low-resolution (LR) input. Over the past few decades, numerous approaches have been put forward because the SR algorithm finds extensive utility in diverse scenarios.

For a given pair of high-resolution and low-resolution images, the transformation process, as depicted in Equation 3.1, establishes the corresponding relationship. The high-resolution image, represented as a vector $\boldsymbol{y} \in \mathbb{R}^{HWs^2}$, undergoes a down-sampling operation of scale s and is blurred using a filter to yield the low-resolution counterpart $\boldsymbol{x} \in \mathbb{R}^{HW}$. Here, W and H denote the width

and height of the image, respectively.

$$\boldsymbol{x} = \boldsymbol{SHy}, \tag{3.1}$$

Here, $\boldsymbol{H} \in \mathbb{R}^{HWs^2}$ represents a blurring operation (such as Gaussian Blurring), and $\boldsymbol{S} \in \mathbb{R}^{H}W \times HWs^2$ denotes the down-sampling operation. The objective of super-resolution is essentially the inverse of this process: given a low-resolution $\boldsymbol{x}$, the goal is to upscale and deblur it in order to restore $\boldsymbol{y}$. However, a major challenge arises due to the ill-posed nature of the transformation described in Equation 3.1. This implies that a single low-resolution $\boldsymbol{x}$ may not correspond to a unique $\boldsymbol{y}$, potentially having multiple valid solutions. Consequently, solving this problem becomes notably difficult as a straightforward inverse transformation cannot be learned. Moreover, in certain real-world scenarios, accessing high-resolution $\boldsymbol{y}$ data is not feasible. For instance, the low-resolution $\boldsymbol{x}$ may be directly captured by a digital camera, or the high-resolution information could be permanently lost during data communication. In such cases, deriving the reverse transformation becomes even more challenging.

To tackle these challenges, certain earlier approaches have employed basic linear interpolation methods such as bilinear and bicubic interpolations. While these methods appear simple and straightforward, they inevitably overlook certain variations in content and local structures. Subsequently, dictionary learning algorithms have been proposed to bridge the gap in mapping, effectively capturing the numerical relationships between the high-resolution (HR) and low-resolution (LR) spaces. Through training on the embedding and query process, the model acquires the ability to map LR patches to HR patches. In this regard, HR patches are intuitively considered as spatial combinations of LR patches, and the learning objective becomes the generation of combination coefficients. In recent times, the performance of deep learning models has witnessed remarkable advancements, leading to the emergence of various new methods. These methods excel in learning dictionaries and combination coefficients, resulting in significant improvements in HR quality (Dai et al., 2019; Huang et al., 2020; Wenbo et al., 2020).

The general processing flow of the deep dictionary learning-based SR model is presented as follows: Firstly, the initial step involves vectorizing the input LR image, represented by $\boldsymbol{x} \in \mathbb{R}^{HW}$, and segmenting it into patches of size k^2. Next, an upsampled matrix denoted as $\boldsymbol{B} \in \mathbb{R}^{HWs^2 \times k^2}$ is constructed, comprising HWs^2 upsampled LR patches, each with a size of k^2. Subsequently, a

series of transformation operations are performed to convert the LR batches into HR batches. To determine the i-th pixel, denoted as $\boldsymbol{y}_i$, within the HR image vector $\boldsymbol{y} \in \mathbb{R}^{HWs^2}$, we employ a process of integrating the neighboring pixels from the batch $\boldsymbol{B}_i$ (i.e., the i-th row of $\boldsymbol{B}$), centered around the coordinate of $\boldsymbol{y}_i$. This pixel-level operation can be mathematically expressed as shown in Equation 3.2.

$$\boldsymbol{y}_i = \boldsymbol{F}_i \boldsymbol{B}_i^\top, \text{ with } \boldsymbol{F}_i = \boldsymbol{\Phi}_i \boldsymbol{D}, \tag{3.2}$$

Here, the integration coefficient vector $\boldsymbol{F}_i \in \mathbb{R}^{1\times k^2}$, also referred to as a filter, plays a significant role. Additionally, the filter $\boldsymbol{F}_i$ and the combination coefficient vector $\boldsymbol{\Phi}_i \in \mathbb{R}^{1\times L}$ can be collectively considered as a linear combination of the fixed pre-defined dictionary $\boldsymbol{D} \in \mathbb{R}^{L\times k^2}$. While the dictionary $\boldsymbol{D}$ remains unchanged during model inference, the primary objective is to compute the coefficients $\boldsymbol{\Phi}_i$ in real-time, satisfying the given requirements. As Equation (3.2) formulates the pixel-level operation, image-level transformation is represented accordingly in Equation (3.3):

$$\boldsymbol{y} = \boldsymbol{F} \boldsymbol{B}^\top, \text{ with } \boldsymbol{F} = \boldsymbol{\Phi} \boldsymbol{D}, \tag{3.3}$$

In this context, we introduce $\boldsymbol{F} \in \mathbb{R}^{HWs^2\times k^2}$ and $\boldsymbol{\Phi} \in \mathbb{R}^{HWs^2\times L}$. Previous works (Yang et al., 2012; Romano et al., 2016; Getreuer et al., 2018) have explored the learning of the coefficient matrix $\boldsymbol{\Phi}$ and the dictionary $\boldsymbol{D}$. To simplify the learning process, LAPAR (Wenbo et al., 2020) employs a pre-defined $\boldsymbol{D}$ comprising a set of Gaussian (G) filters and difference of Gaussian (DoG) filters. This choice of $\boldsymbol{D}$ helps expedite the learning process. The coefficient matrix $\boldsymbol{\Phi}$ is obtained as the output of a residual network (more details about the network will be discussed in Section 3.1.1).

Considering the communication patterns of Equation (3.3), $\boldsymbol{\Phi}$ and $\boldsymbol{B}$ usually occupy much more bandwidth than $\boldsymbol{D}$, *i.e.*,

$$HWs^2 \times L + HWs^2 \times k^2 \gg L \times k^2. \tag{3.4}$$

The computation patterns are strongly influenced by the dictionary $\boldsymbol{D}$, which holds significant importance. It determines the conditions under which computations are performed on the given data in $\boldsymbol{\Phi}$ and $\boldsymbol{B}$. The dictionary serves as a crucial link and translator connecting $\boldsymbol{\Phi}$ and $\boldsymbol{B}$. Based on the characteristics of $\boldsymbol{D}$, certain unnecessary data, if they do not impact performance, can be skipped from being loaded into the on-chip cache, leading to skipped computations. The unique role

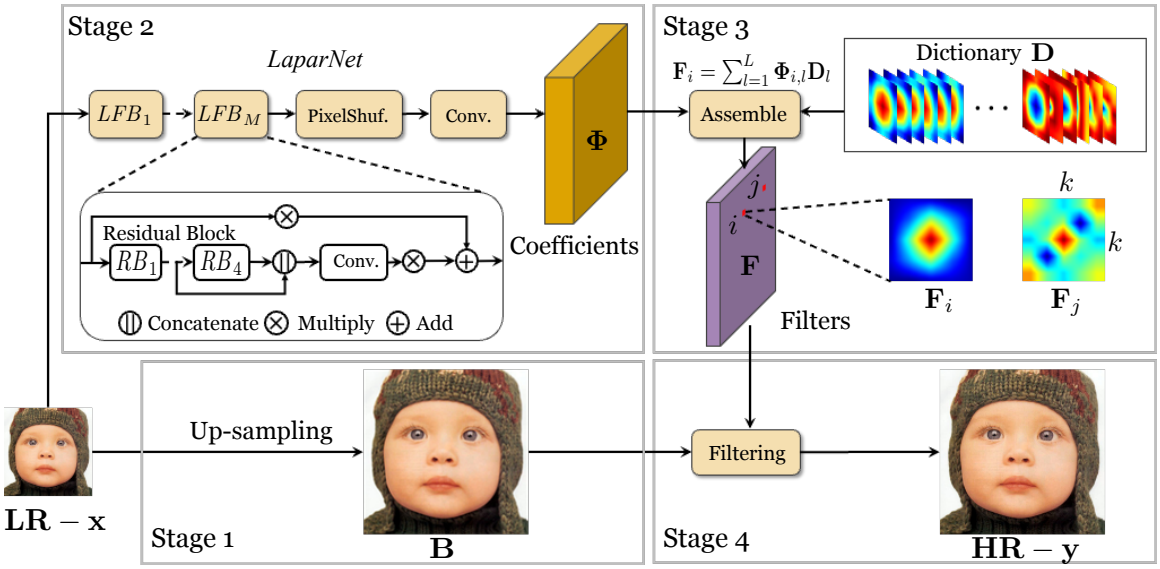


Figure 3.1: The architecture of linearly-assembled pixel-adaptive regression network (LAPAR) (Wenbo et al., 2020)

of $\boldsymbol{D}$ in dictionary learning sets it apart from conventional deep learning algorithms, as it considers more than just weights and features. Once the dictionary is optimized, it enables simultaneous resolution of both communication and computation bottlenecks.

3.1.1 SR Model Architecture

By convention, dictionary learning-based models comprise layers of residual blocks, followed by convolutions, pixel-shuffle operations, and the most important dictionary assembling, and *etc.*

To illustrate the model structure and inference flow, we use LAPAR (Wenbo et al., 2020), the state-of-the-art SR model, as an example. The inference flow consists of four stages, as depicted in Figure 3.1. In the initial stage, the input image $\boldsymbol{x}$ undergoes bicubic interpolation to upscale it, resulting in the patch matrix $\boldsymbol{B}$. Next, the *LaparNet* model takes $\boldsymbol{x}$ as input and produces the coefficient matrix $\mathbf{\Phi}$ as output. The *LaparNet* model comprises several local fusion blocks (LFBs) (Wang et al., 2019a), pixel-shuffle layers, and convolution layers. Each LFB consists of residual blocks, followed by concatenations, multiplications, and short-cut additions. In the third stage, the dictionary assembling is performed to obtain the transformation matrix $\boldsymbol{F}$. This is achieved by querying the pre-defined dictionary $\boldsymbol{D}$ with $\mathbf{\Phi}$. Finally, in the last stage, the HR image $\boldsymbol{y}$ is obtained

by filtering the up-sampled $\boldsymbol{B}$ with $\boldsymbol{F}$, *i.e.*, $\boldsymbol{y} = \boldsymbol{F}\boldsymbol{B}^\top$. To efficiently deploy SR models on GPUs, it is crucial to analyze the dictionary learning module in detail, which has been overlooked in previous research.

3.1.2 GPU Programming Architecture

NVIDIA provides a well-designed high-level abstraction of their GPU architecture as shown in Figure 3.2, which enables the software engineers or algorithm designers to access hardware resources and write easy and convenient low-level hardware implementations. Streaming multiprocessors (SMs) are key computation modules within GPU hardware architecture. Each SM has independent shared memory units, control logic, several processing blocks, and *etc.* Within each SM, each single processing block is composed of a batch of computation cores (CUDA cores, Tensor Cores, and *etc.*), register files, load/store units, and *etc.*

NVIDIA provides CUDA programming model (Cheng et al., 2014) in order to help implement computation tasks with parallelism on GPU. The programming model is designed as follows: A host device (CPU) is included to control the data movement or execution of CUDA kernels. Kernels will be launched and run on a device (GPU) to realize a parallel computation, as shown in Figure 3.2. Each kernel will be launched with a computation grid, and multiple blocks will be assigned to each cell of the grid. Following the single instruction multiple threads (SIMT) mechanism, each block is further partitioned into a group of threads. Each thread runs the same piece of code with different data synchronously. Each kernel will launch all threads to execute the same code piece at once after the program is compiled. Meanwhile, different thread blocks may execute in order, given hardware resource constraints.

3.1.3 Low Precision Inference

Currently, the dominant data format in most Deep Learning applications is 32-bit single-precision floating-point. To mitigate model inference latency while maintaining high throughput, quantization is employed. This technique allows for the utilization of integer instructions or lower bit data formats, resulting in reduced inference time without substantial accuracy loss. A study conducted by (Wu et al., 2020) demonstrated that on NVIDIA GPUs, tensor operators that involve intensive

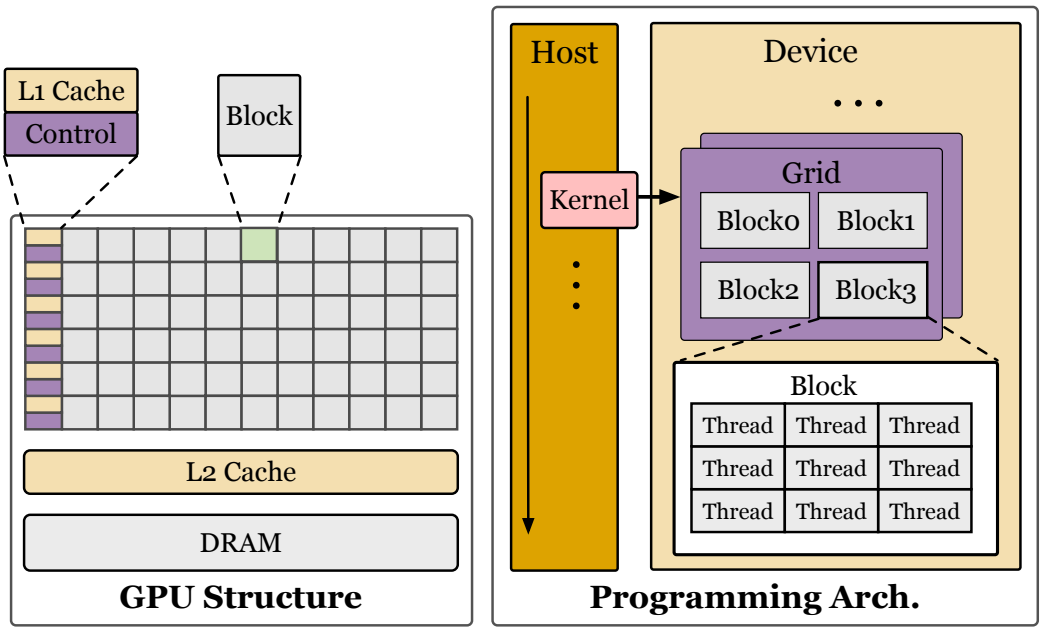


Figure 3.2: GPU memory hierarchy and communication mode.

mathematical computations can achieve a speed-up of 16 times when using 8-bit signed integer data format instead of FP32. Similarly, tensor operators that primarily rely on memory operations can experience a speed-up of up to 4 times.

Several post-training quantization (PTQ) methods have been introduced to safely quantize models to 8-bit without requiring retraining. In a data-free approach, (Nagel et al., 2019) successfully implemented 8-bit quantization. To address the quantization challenge, (Nagel et al., 2020) proposed a layer-wise calibration strategy that minimizes the Hessian of the task loss. An analytical solution for quantization clipping range selection was put forth by (Banner et al., 2019). In addition, previous studies have pushed the boundaries of quantization to extremely low-bit formats, such as ternary (2-bit) or even binary (1-bit) data formats, as explored by researchers (Courbariaux et al., 2014; Han et al., 2015a; Mellempudi et al., 2017; Courbariaux et al., 2015; Hubara et al., 2016).

Scale quantization, also known as symmetric quantization, is an efficient approach with good support of GPU hardware support. Each floating-point parameter is transformed into an 8-bit integer with a range mapping:

$$s = \frac{\epsilon}{2^{b-1} - 1}, \tag{3.5}$$

$$\hat{x} = \text{clip}(\text{round}(\frac{x}{s}), -2^{b-1} + 1, 2^{b-1} - 1). \tag{3.6}$$

Let us consider the variables x as the original floating-point number and $\hat{x}$ as the quantized integer. The quantization process, as depicted in Equation 3.6, can be understood as a sequence of three steps: scaling, rounding, and clipping. Initially, the floating-point value x is multiplied by a scaling factor s and then rounded to the nearest neighboring integer. Subsequently, this integer is clipped within the range $[-2^{b-1} + 1, 2^{b-1} - 1]$ to ensure it falls within the representative range of an 8-bit value. In the case of 8-bit signed integer, the range is $[-127, 127]$. The corresponding clipping range in floating point value is $[-\epsilon, \epsilon]$ and scaling factor s is calculated by Equation (3.5). For multiplication of 2 tensors $\boldsymbol{A}$ and $\boldsymbol{B}$ with scale factors $s_{\boldsymbol{A}}$ and $s_{\boldsymbol{B}}$, the quantized computation can be simply conducted as:

$$\boldsymbol{A} \cdot \boldsymbol{B} = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{B}} \cdot s_{\boldsymbol{A}} \cdot s_{\boldsymbol{B}} \tag{3.7}$$

3.1.4 Dictionary Slimming

The LAPAR algorithm, known for its exceptional performance in lightweight super-resolution (SR) tasks, has demonstrated state-of-the-art capabilities even with a compact model size (number of parameters ¡1M) (Wenbo et al., 2020). Despite its lightweight architecture, the LAPAR algorithm still falls short of meeting the real-time inference requirements, even when leveraging the powerful NVIDIA TensorRT (GPU, [n. d.]). In the context of SR tasks, the inference speed is primarily influenced by the large spatial scale of feature maps rather than the number of parameters. This is illustrated in the breakdown of running time presented in Figure **??**. Figure **??** clearly highlights that the dictionary learning process serves as the bottleneck, accounting for the largest percentage of time cost. This can be attributed to the lack of customized support in existing commercial tools for certain specialized operations or extreme-scale feature maps. These tools typically offer efficient and reliable implementations for common deep neural network (DNN) layers such as convolution, rectified linear unit (ReLU), and batch normalization (BN). As a result, certain computation graphs may experience significant time costs due to the absence of tailored optimization for these operations.

Slimming and reducing the size of the dictionary not only helps in saving computation costs but also alleviates the communication burden on hardware resources. To compress the dictionary, we

propose a dictionary slimming strategy. Ideally, the dictionary $\boldsymbol{D}$ should be informative enough to provide ample embedding data for restoring image details. However, we also aim to avoid excessive bulkiness and redundant information in the dictionary $\boldsymbol{D}$. By slimming the dictionary, we can achieve the desired outcome of performing inference under stringent speed requirements without significant accuracy degradation. To control the slimming process, we introduce a sparsity threshold $\alpha \in (0, 1)$ that reflects the desired sparsity level of the dictionary $\boldsymbol{D} \in \mathbb{R}^{L \times k^2}$. Following slimming, only the most essential $\alpha \cdot L$ items out of the original L will be retained. It is worth noting that aggressive slimming of the dictionary to a specific ratio α can be challenging and may not yield optimal results. To simplify the problem, we adopt an iterative approach where we gradually reduce the sparsity from 1 to α. During each iteration t with sparsity α_t, where $\alpha_t < \alpha_{t-1}$, we retain the most important $\alpha_t L$ items from the current dictionary and discard the rest. In the next iteration, the sparsity goal is updated as $\alpha_{t+1} = \alpha_t - \Delta_\alpha$ to further prune additional items.

After removing redundant items through pruning, we proceed to fine-tune the *LaparNet* to adapt to the newly slimmed dictionary. This is accomplished by minimizing the reconstruction error. The problem can be formulated as follows:

$$\begin{aligned} \boldsymbol{\beta}, \boldsymbol{W} = \quad & \arg\min_{\boldsymbol{\beta}, \boldsymbol{W}} \frac{1}{N} \left\| \boldsymbol{Y} - \sum_{i=0}^{L} \beta_i \boldsymbol{\Phi} \boldsymbol{D} \right\|_2^2, \\ \text{s.t.} \quad & \boldsymbol{\Phi} = \textit{LaparNet}(\boldsymbol{X}, \boldsymbol{W}), \\ & \|\boldsymbol{\beta}\|_0 \leq \alpha L, \end{aligned} \tag{3.8}$$

where N is the batch size of the input images. $\boldsymbol{W}$ denotes the parameters in *LaparNet*, whose output is the coefficient vector $\boldsymbol{\Phi}$. $\boldsymbol{Y}$ is the output tensor after querying the original unpruned dictionary with no fine-tuning. Selector $\boldsymbol{\beta}$ determines which item of $\boldsymbol{D}$ is pruned. i-th item of $\boldsymbol{D}$ will be neglected if $\beta_i = 0$.

Furthermore, we extend the modifications made to Equation 3.8 by incorporating the evaluation of the final image in the super-resolution (SR) pipeline. To capture the discrepancy between the image generated by the compressed model and the ground truth high-resolution (HR) image $\boldsymbol{H}_{gt}$,

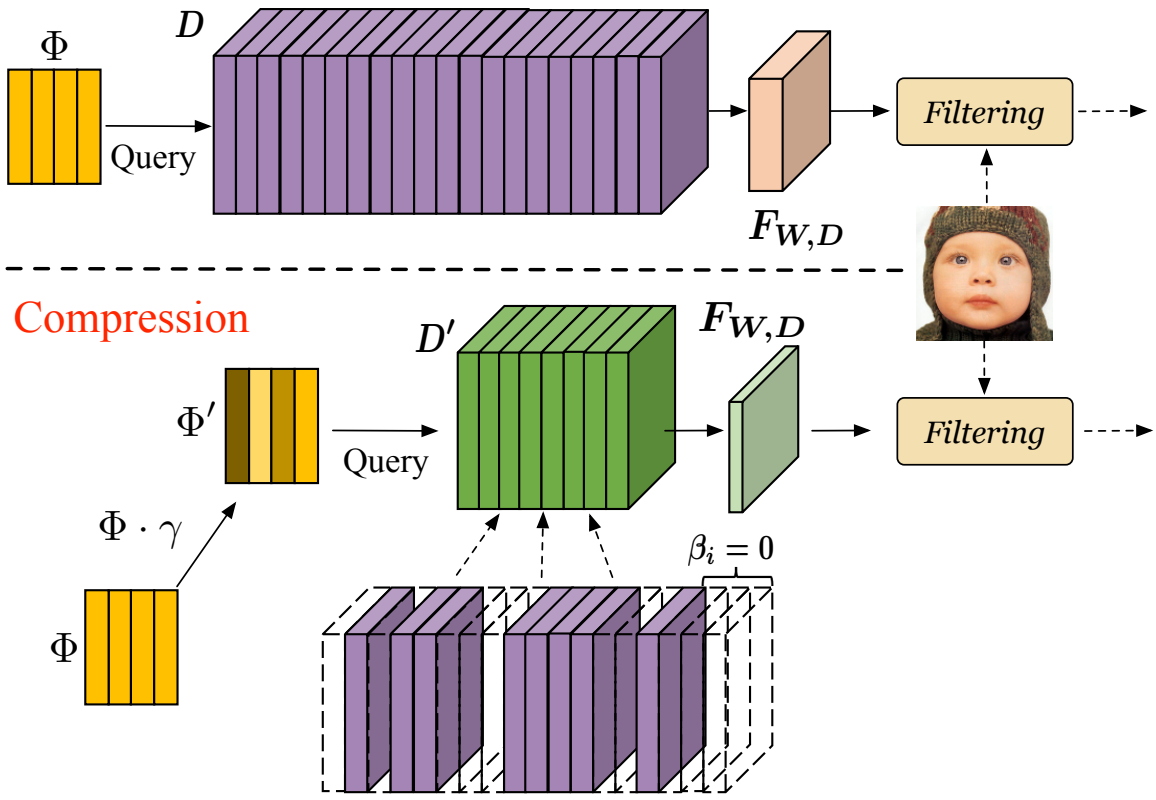


Figure 3.3: Visual illustration of dictionary slimming, the upper flow represents original dictionary query and filtering, namely stage 3 + stage 4 in Figure 3.1, The flow below demonstrates the slimming process of the dictionary query.

we define the reconstruction error using Equation 3.9:

$$\begin{aligned} \boldsymbol{\beta}, \boldsymbol{W} = \quad & \arg\min_{\boldsymbol{\beta},\boldsymbol{W}} \tfrac{1}{N} \left\| \boldsymbol{H}_{gt} - \boldsymbol{F}_{W,\beta} \boldsymbol{B}^{\top} \right\|_2^2, \\ \text{s.t.} \quad & \boldsymbol{F}_{W,\beta} = \textstyle\sum_{i=0}^{L} \beta_i \boldsymbol{\Phi} \boldsymbol{D}, \\ & \boldsymbol{\Phi} = \mathit{LaparNet}(\boldsymbol{X}, \boldsymbol{W}), \\ & \|\boldsymbol{\beta}\|_0 \leq \alpha L. \end{aligned} \tag{3.9}$$

Both $\boldsymbol{\beta}$ and $\boldsymbol{W}$ serve as optimization objectives in our approach. To simplify the problem and improve efficiency, we employ an alternating optimization strategy, wherein each objective is optimized in turn. In the first step, we focus on finding the optimal selector β to meet the sparsity requirement α_t, while keeping the parameters of the *LaparNet* fixed. In the second step, we fix β

and fine-tune the parameters $\boldsymbol{W}$ to minimize the reconstruction error as defined in Equation 3.9. Directly optimizing the selector β with an ℓ_0 norm constraint is known to be NP-hard. However, we address this challenge by employing LASSO regression. By introducing an ℓ_1 regularization term (He et al., 2017) into the original loss function, we can effectively optimize the sparsity. This is illustrated in Equation 3.10.

$$\begin{aligned} \boldsymbol{\beta} = \quad & \arg\min_{\boldsymbol{\beta}} \frac{1}{N} \left\| \boldsymbol{H}_{gt} - \boldsymbol{F}_{W,\beta} \boldsymbol{B}^{\top} \right\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1, \\ & \text{s.t.} \|\boldsymbol{\beta}\|_0 \leq \alpha L. \end{aligned} \tag{3.10}$$

The complete selection strategy is illustrated in Algorithm 1.

Algorithm 1 Dictionary Selection Strategy

Input: $\boldsymbol{D} \in \mathbb{R}^{L \times k^2}$, small λ_0, target α, tolerance ϵ;
Input: pre-trained $\boldsymbol{W}_0$, coefficient matrix $\boldsymbol{\Phi}$;
$t \leftarrow 0$, $\alpha_0 \leftarrow 1.0$, $\beta_0 \leftarrow \mathbf{1} \in \mathbb{R}^L$, $\gamma_0 \leftarrow \mathbf{1} \in \mathbb{R}^L$;
$\mathscr{L} \leftarrow$ reconstruction error ▷ Equation (3.9)
repeat
    $\alpha_{t+1} \leftarrow \alpha_t - \Delta\alpha$;
    $\lambda_{t+1} \leftarrow \lambda_t$;
    while $|\beta_{t+1}|_0 > \alpha_{t+1} \cdot L$ **do**
        Fix $\boldsymbol{W}_t$, update $\beta_{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \mathscr{L}(\boldsymbol{W}_t, \boldsymbol{\beta D}) + \lambda_{t+1} |\boldsymbol{\beta}|$; ▷ Equation (3.10)
        $\lambda_{t+1} \leftarrow 2 \cdot \lambda_{t+1}$
    end while
    $\lambda_{left} \leftarrow 0.5\lambda_{t+1}$, $\lambda_{right} \leftarrow \lambda_{t+1}$;
    while $\left|\alpha_{t+1} \cdot L - |\beta_{t+1}|_0\right| > \epsilon \cdot L$ **do**
        $\lambda_{t+1} = 1/2(\lambda_{left} + \lambda_{right})$;
        Fix $\boldsymbol{W}_t$, update $\beta_{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \mathscr{L}(\boldsymbol{W}_t, \boldsymbol{\beta D}) + \lambda_{t+1} |\boldsymbol{\beta}|$;
        if $|\beta_{t+1}|_0 < \alpha_{t+1} \cdot L$ **then**
            $\lambda_{left} \leftarrow \lambda_{t+1}$;
        else if $|\beta_{t+1}|_0 > \alpha_{t+1} \cdot L$ **then**
            $\lambda_{right} \leftarrow \lambda_{t+1}$;
        end if
    end while
    Fix β_{t+1}, update $\boldsymbol{W}_{t+1} \leftarrow \arg\min_{\boldsymbol{W}} \mathscr{L}(\boldsymbol{W}, \beta_{t+1}\boldsymbol{D})$; ▷ Equation (3.11)
    $t = t + 1$;
until $\alpha_t \leq \alpha$

Prior to channel selection, it is necessary to gather calibration data, which includes the output feature maps of the *LaparNet* before querying the dictionary, as well as the corresponding high-resolution ground-truth images. To control the sparsity, we carefully adjust the regularization weight

λ in Equation 3.10. We adopt a greedy approach to search for the optimal values of β. Starting with a small λ, we iteratively increase its value, doubling it at each iteration. This progressively strengthens the sparsity regulation until the desired level α is achieved. Towards the end of the slimming process, if the step size of λ becomes excessively large due to the exponential update, we employ a binary search within the range $[\lambda_t, \lambda_{t+1}]$ to finely adjust the slimming ratio, approaching α_{t+1} with precision. This adjustment is depicted in lines 12–20 of Algorithm 1.

After the selector β optimization, we need to fine-tune the parameters $\boldsymbol{W}$ accordingly, as shown in Algorithm 1.

$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}} \frac{1}{N} \left\| \boldsymbol{H}_{gt} - \boldsymbol{F}_{W,D'} \boldsymbol{B}^{\top} \right\|_2^2 . \tag{3.11}$$

However, tuning all parameters in the *LaparNet* at each iteration can be computationally expensive and time-consuming. In Equation 3.11, we introduce $\boldsymbol{D'}$ as the compressed dictionary, obtained by neglecting layers during the previous LASSO step. To efficiently adjust the output of the *LaparNet* to align with the newly compressed dictionary $\boldsymbol{D'}$, we focus on reconstructing the parameters of the last layer, just before the dictionary query. This approach allows for faster tuning, and we achieve it by utilizing linear regression to learn a channel-wise factor for the original parameters, resulting in improved efficiency. The parameters in the last layer, denoted as $\boldsymbol{W}D'$, are multiplied by a regression coefficient $\boldsymbol{\gamma}$ specific to each channel. By doing so, we can reformulate the parameter-tuning step from Equation 3.11 to Equation 3.12. Here, $\boldsymbol{\gamma}$ represents the channel-wise coefficient used to scale the parameters of the updated $\boldsymbol{W}D'^{new}$ on each channel.

After the tuning, a new coefficient matrix $\boldsymbol{\Phi'}$ will be generated to query the slimmed dictionary $\boldsymbol{D'}$. The visualization of the complete dictionary query and filtering flow after slimming is shown in Figure 3.3.

$$\begin{aligned} \boldsymbol{\gamma} = \arg\min_{\boldsymbol{\gamma}} \quad & \tfrac{1}{N} \left\| \boldsymbol{H}_{gt} - \textstyle\sum_{i=0}^{L} \gamma_i \boldsymbol{F}_{W,D'} \boldsymbol{B}^{\top} \right\|_2^2 , \\ & \boldsymbol{W}_{D'}^{new} = \boldsymbol{\gamma} \boldsymbol{W}_{D'} . \end{aligned} \tag{3.12}$$

According to the results presented in Figure 3.4, slimming the dictionary does not impact the performance of the original super-resolution (SR) model. The information preserved in the dictionary exhibits sufficient sparsity, and a well-trained model is capable of capturing valuable information even when certain items are set to zero. In our experiments, we demonstrate that the dictionary can be

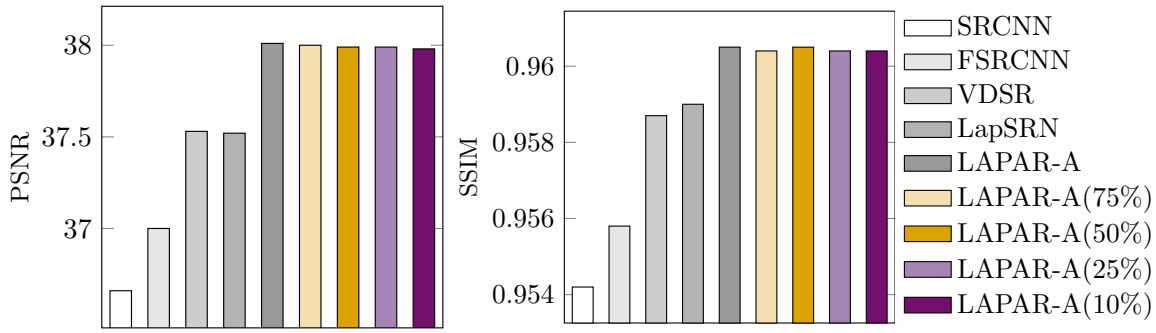


Figure 3.4: Single image super-resolution (SISR) performance of our model with different dictionary compression ratios in comparison with other SR methods. LAPAR-A (Per.%) represents our model with dictionary size shrunk to Per.%. PSNR means peak signal-to-noise ratio. SSIM means structural similarity index measure. PSNR and SSIM are two common metrics to measure the quality of images. The higher, the better.

slimmed down to only 10% of its original size without any noticeable loss in accuracy. Furthermore, for a fair comparison, we highlight that the compressed model outperforms other widely-used SR models, such as those mentioned in (Dong et al., 2016; Kim et al., 2016).

3.1.5 Constraint-Based Optimization of Deployments On GPU

While reducing the size of the dictionary can contribute to speeding up the dictionary query process, the subsequent filtering operation remains a bottleneck in terms of inference latency. This filtering operation involves performing a Hadamard product between two tensors and then applying a reduce-sum operation along the channel dimension to convert the tensor into a 2-D image. Despite the significance of such computations in super-resolution (SR) tasks, they are often overlooked by current mainstream deployment tools. In this section, we introduce a domain-specific low-level design that leverages the parallelization capabilities of GPUs to enhance computation throughput from a hardware perspective. Figure 3.5 illustrates an example of the proposed computation engine, which embodies the aforementioned design principles.

To implement the Hadamard product and reduce-sum in a parallel manner within the existing inference flow, we follow a specific approach. During the inference stage, all tensor data, including images and filters, is stored consecutively in the (N, C, H, W) format, representing the (`batch size, channel, height, width`) dimensions of each tensor. This linear memory arrange-

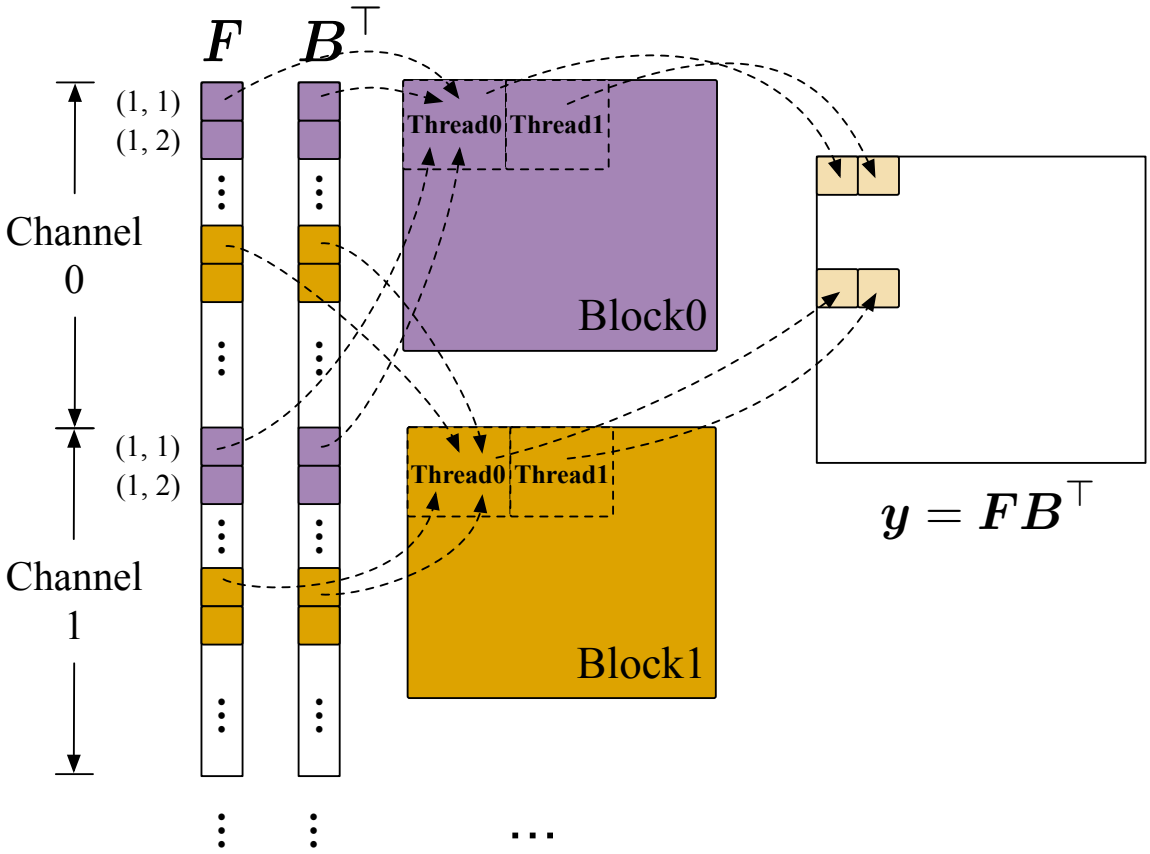


Figure 3.5: An example of the proposed computation engine for image filtering operation.

ment enables efficient parallel processing. Since both computation operations are spatially independent, we can assign the calculation of values at each 2-D location on the $H \times W$ map to different computation units, as depicted in Figure 3.5. Each block represents a distinct computation unit, and the color of the data indicates which block it is assigned to. For example, purple data is assigned to block 0, while orange data is assigned to block 1. To optimize thread assignment, we strive to assign consecutive data to consecutive threads as much as possible. Simultaneously, data with the same index but from different channels are assigned to the same thread. For instance, data at location $(1, 1)$ and $(1, 2)$ are respectively assigned to threads 0 and 1 in block 0 for all channels. The Hadamard product begins by performing element-wise multiplication between $\boldsymbol{F}$ and $\boldsymbol{B}$. Subsequently, the products for each channel are accumulated to produce the final high-resolution images. These steps can be computed in parallel for each 2-D location index. In essence, the computation assigned to

each thread corresponds to multiplying two vectors, where each pair of vectors consists of channel data from $\boldsymbol{F}$ and $\boldsymbol{B}$ at the same 2-D location. In our implementation, each thread performs addition and multiplication simultaneously by adding the intermediate product of each channel to the final result. All threads execute the same code in parallel, carefully avoiding thread divergence issues (Cheng et al., 2014). Furthermore, we consider the cache-memory mechanism and aim to minimize frequent interactions in our design. As illustrated in Figure 3.2, each Streaming Multiprocessor (SM) has an exclusive shared memory/L1 cache for all blocks within it. We take advantage of this architecture to optimize memory access and reduce latency.

However, it is important to note that parallelism cannot be endlessly extended. We must consider complex limitations at both the hardware and programming model levels, as these factors have a significant impact on the performance of parallel implementations. To begin with, the computation patterns are determined by the number of assigned threads, blocks, and other parameters. However, the computing capabilities of different GPU devices can vary, leading to varying limits on thread/block configurations. To provide an example, let's consider the edge device NVIDIA Jetson Xavier NX, which incorporates a Volta microarchitecture GPU. At the hardware level, each NX device consists of six streaming multiprocessors (SMs). Each SM has a fixed shared memory size (on-chip memory) of 96KB and contains four physical processing blocks. These processing blocks house 16 FP32 cores, 8 FP64 cores, 16 INT32 cores, 2 Tensor cores, and a 64KB shared register file. At the programming model level, the computation kernel is launched as a computation grid, where each cell represents a thread block. It's important to note that the concept of a thread block here is virtual and differs from the previous processing block. Each thread block is assigned to a single SM. When discussing hardware resource limits, it is necessary to introduce the concept of a warp. A warp represents the basic execution unit in an NVIDIA GPU, comprising 32 consecutive threads. In the current SIMT architecture, each thread block is further divided and assigned to multiple warps after being scheduled to an SM. Warp scheduling in the GPU is unordered within each thread block. The only limitation is the number of active warps with regard to the available SM resources. When a warp becomes idle due to race conditions, the SM can schedule other available warps to maximize

utilization. The number of warps for a thread block can be determined as follows:

$$\text{Warps Per Block} = \left\lceil \frac{\text{Threads Per Block}}{\text{Warp Size}} \right\rceil, \tag{3.13}$$

The programming model also imposes limitations on the number of warps within a thread block to accommodate the sizes of warp schedulers, instruction registers, and other factors. Furthermore, we need to consider memory constraints. There are two levels of data sharing among parallel executions: 1. Sharing data in the shared register files among parallel threads within the same processing block. 2. Sharing data among processing blocks within the same SM. Both scenarios can potentially lead to race conditions where multiple threads access the same data in memory simultaneously.

We need to balance the contradiction between parallelism and congestion by carefully selecting an appropriate block size. The size of a block is restricted by both the size of input data and available on-chip resources. Meanwhile, once the resources are available, the tasks will be assigned to occupy the resources to accelerate the computations as much as possible. In other words, the parallelism is maximized so as to reach the upper-bound value of resource utilization. We denote the size of input data to be $D = H \times W \times C$, which is three-dimensional. The threads blocks can also be regarded as three-dimensional with size (n_x, n_y, n_z). Based on the detailed analysis above, we can formulate a series of constraints to this optimization problem. We assume each GPU has S SMs inside, where each SM holds P processing blocks. We assume that each processing block has R register files, and the maximum threads number is each warp in WS. The CUDA programming model also sets a warp number limit to each block T_{sm}. T_r denotes input data assigned to each SM (evenly). Each processing block will manage and schedule the computational resources inside implicitly, and the computational resources constraints T_{sm} is prefixed at a constant value given fixed compute capability. The number of warps T in each processing block is upper-bounded by both T_r and T_{sm}. Another constraint comes from the input data size. Therefore, these constraints

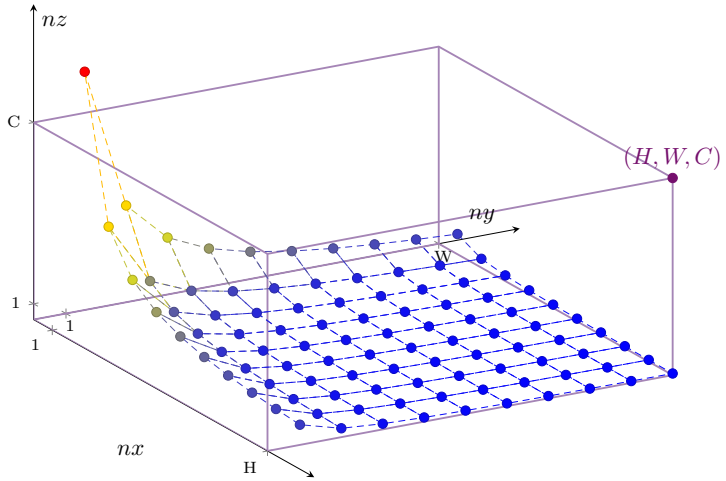


Figure 3.6: The visualized solution space. The solution points below the dotted points are legal configurations.

can be formulated as:

$$
\begin{aligned}
&T_r = (H \times W \times C)/(S \times P \times R),\\
&T \leq \min(T_r, T_{sm}),\\
&n_x \times n_y \times n_z \leq WS \times P \times T,\\
&1 \leq n_x \leq H,\\
&1 \leq n_y \leq W,\\
&1 \leq n_z \leq C.
\end{aligned}
\tag{3.14}
$$

By applying these constraints, we can reduce the search space by ignoring wasteful choices and therefore saving optimization workloads. For example, for $T \in [T_r, T_{sm}]$, these T values are legal while on-chip resources are not fully utilized, and the system parallelism can be further improved. The search space regarding these constraints is visualized in Figure 3.6. To the best of our knowledge, we are the first to take these constraints for deployments of DNN models on GPUs into consideration, as compared with *e.g.*, (Chen et al., 2018b).

Despite the constraints mentioned above, the optimization process may still suffer from slow performance due to the time-consuming on-board compilation and execution of each candidate configuration. To address this issue, we employ Bayesian Optimization as an alternative to grid search or manual tuning (Gardner et al., 2018; Bai et al., 2021). Bayesian Optimization offers superior

efficiency by leveraging the comprehensive information obtained from past experiments to better sample candidate values for nx, ny, and nz. The key components of Bayesian optimization consist of a probabilistic surrogate model, denoted as $S(\cdot)$, which enables fast evaluation of inference latency, and an acquisition function, denoted as $A(\cdot)$, which selects the most informative candidates based on the largest upper confidence bound (UCB) (Srinivas et al., 2009). Our implementation utilizes a Gaussian process (GP) model as the surrogate model. The complete searching process is outlined in Algorithm 2. Initially, we randomly sample a small batch of configurations $\hat{N}$ from the search space $\mathcal{N}$. These configurations are used to initialize the surrogate model, which is a Gaussian process model in our implementation. Several rounds of sampling and updating the surrogate model are performed. Finally, we select the configuration with the best inference speed as the optimal choice.

Algorithm 2 Configuration Search Process

Input: search space $\mathcal{N}$, round T, surrogate model $S(\cdot)$, acquisition function $A(\cdot)$;
Get initial samples: $\hat{N} \leftarrow Sample(\mathcal{N})$;
Get sample performance: $\hat{P} \leftarrow Eval(\hat{N})$; ▷ On-board test
Get Optimal in sample: $(n_{x,y,z})^*, p^* \leftarrow argmax_{n_{x,y,z} \in \hat{N}} \hat{P}$;
for $i \leftarrow range(|\hat{N}|, T)$ **do**
    $S(y|(n_x, n_y, n_z), \hat{N}) \leftarrow Fit(\hat{P}, \hat{N})$; ▷ Fit GP Model
    $(n_{x,y,z})_i \leftarrow \text{argmax}_{(n_{x,y,z}) \in \mathcal{N}} A(S(y|(n_{x,y,z}), \hat{N}), n_{x,y,z})$;
    $p_i \leftarrow Eval((n_x, n_y, n_z)_i)$; ▷ On-board test
    $\hat{P} \leftarrow \hat{P} \cup p_i, \hat{N} \leftarrow \hat{N} \cup (n_{x,y,z})_i$; ▷ Add to samples
    if $p_i > p^*$ **then**
        $(n_x, n_y, n_z)^* \leftarrow (n_{x,y,z})_i$;
        $p^* \leftarrow p_i$;
    end if
end for
Output: $(n_x, n_y, n_z)^*$;

3.2 Adaptive 8-bit Quantization

As shown in **??**, The inference latency of the dictionary query and filtering step was significantly reduced by dictionary compression and hardware constraint-aware optimization. After these steps, typical deep learning operators such as convolution and ReLU in *LaparNet* become the most time-consuming stage, occupying up to 70% of inference time.

Different from other computer vision tasks such as object detection or classification, super-resolution is a fine-grained task where the RGB value of high-resolution images is recovered. We

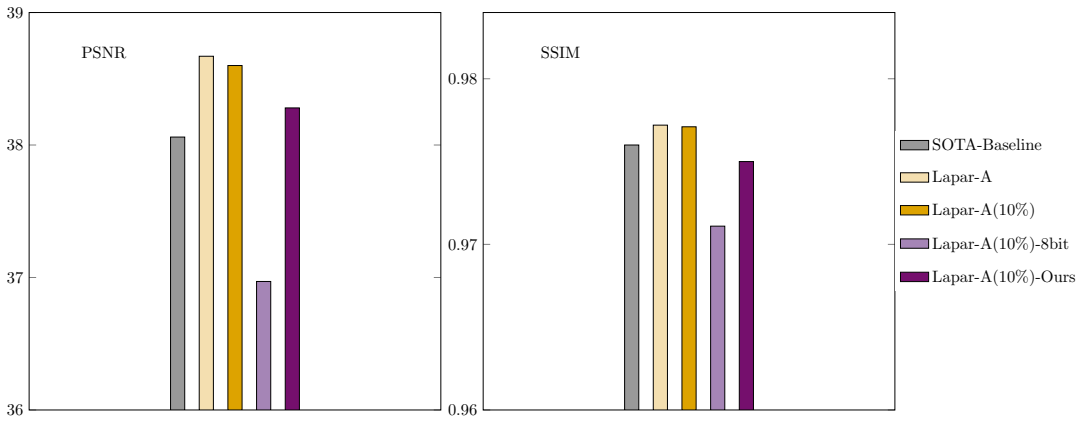


Figure 3.7: SR performance of 8-bit inference in comparison with SOTA baseline SR methods and original Lapar-A model. Lapar-A(10%) represents the model with the dictionary shrunk to 10% size. Lapar-A(10%)-8bit represents the model with naive 8-bit scale quantization. Lapar-A(10%)-ours is our adaptive 8bit approach.

adopt the 8-bit post-training quantization (PTQ) technique to further accelerate the inference by fully increasing the math throughput of hardware using 8-bit data type for both weight and activations of *LaparNet*. The reason why we choose the simple 8-bit quantization instead of some other SOTA quantization methods is threefold: Firstly, the RGB value of each pixel ranges from 0 to 255, which can be represented using no less than 8 bits. In this way, 8-bit is the lower bound bit-width to avoid significant information loss. Secondly, the priority of maintaining accuracy is higher than the model compression, so we do not necessarily need to quantize the model to lower bits aggressively. Last but not least, 8-bit integer computations are well supported by the current mainstream accelerator with mature hardware and software support

By convention, the rounding step in Equation (3.7) is a simple yet fixed rounding-to-nearest action, which is intuitive. And the naive approach for clipping range selection is to minimize the KL-divergence of activations at each layer:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} D_{KL}(\boldsymbol{act}_{8-bit} || \boldsymbol{act}_{FP32}), \tag{3.15}$$

where $\boldsymbol{act}$ denotes the activation value distribution, which can be derived from a calibration set. KL-divergence is capable of measuring the information loss of clipping and rounding of quantization by calculating the entropy of quantized and unquantized data distribution.

However, despite the effectiveness of KL-divergence-guided clipping scale in minimizing layer-wise information loss from quantization, it is unable to prevent significant performance degradation. When we apply naive scale quantization, both PSNR and SSIM experience a significant drop, even lower than baseline methods. This is illustrated in Figure 3.7. In their work, (Nagel et al., 2020) raised concerns regarding the commonly used rounding-to-nearest step in integer quantization. They pointed out that quantization clipping and rounding can be viewed as a shift in weight values from the original full-precision network. This shift leads to differences in the coefficient vector $\mathbf{\Phi}$ extracted from *LaparNet*, resulting in performance degradation:

$$\Psi(\boldsymbol{X}, \boldsymbol{W}, \Delta\boldsymbol{W}) = \text{loss}(\boldsymbol{X}, \boldsymbol{W} + \Delta\boldsymbol{W}) - \text{loss}(\boldsymbol{X}, \boldsymbol{W}), \tag{3.16}$$

where $\boldsymbol{W}$ is the weight and $\boldsymbol{X}$ is the input. $\Delta\boldsymbol{w}$ is the weight value shift from rounding and clipping and, degradation Ψ is evaluated by the task loss change from quantization. However, direct optimization of such value differences is not easy. As quantization aims to minimize the disturbance on the final result, we can simplify the optimization goal with an equivalent objective: minimizing SR task loss difference. Then we can expand the equation by second-order Taylor expansion, as indicated by (Nagel et al., 2020). The objective of minimizing the accuracy loss can be derived:

$$\begin{aligned} &\underset{\Delta\boldsymbol{W}}{\text{argmin}}\, \mathbb{E}\left[\text{loss}(\boldsymbol{X}, \boldsymbol{W} + \Delta\boldsymbol{W}) - \text{loss}(\boldsymbol{X}, \boldsymbol{W})\right] \\ \approx\ &\underset{\Delta\boldsymbol{W}}{\text{argmin}}\, \mathbb{E}\left[\Delta\boldsymbol{W}^\top \nabla_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W})\right. \\ &\qquad \left. + \Delta\boldsymbol{W}^\top \nabla^2_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W}) \Delta\boldsymbol{W}\right]. \end{aligned} \tag{3.17}$$

Instead of focusing solely on minimizing the weight shift $\boldsymbol{\Delta W}$ through the rounding-to-nearest step, it may not always be the optimal choice. In Equation (3.17), the first term becomes negligible due to the convergence of training, causing the first-order gradient $\nabla_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W})$ to approach zero. Consequently, the objective of quantization relies on the second term, where the Hessian matrix $\nabla^2_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W})$ becomes crucial. (Li et al., 2020a) has provided mathematical proof that this second-order error optimization can be transformed into Equation (3.18), where $\Delta\mathbf{\Phi}$ represents the difference in the coefficient vector $\mathbf{\Phi}$ before and after quantization, and $\nabla^2_{\mathbf{\Phi}} \text{loss}(\boldsymbol{X}, \boldsymbol{W})$ denotes the Hessian matrix associated with the coefficient vector $\mathbf{\Phi}$. This transformation allows for optimization by leveraging the coefficient vector difference $\Delta\mathbf{\Phi}$, which is more readily obtainable during the

forwarding inference step.

$$\begin{aligned} &\underset{\Delta \boldsymbol{W}}{\operatorname{argmin}} \mathbb{E}\left[\Delta \boldsymbol{W}^{\top} \nabla_{\boldsymbol{W}}^{2} \operatorname{loss}(\boldsymbol{X}, \boldsymbol{W}) \Delta \boldsymbol{W}\right] \\ \approx &\underset{\Delta \boldsymbol{W}}{\operatorname{argmin}} \mathbb{E}\left[\Delta \boldsymbol{\Phi}^{\top} \nabla_{\boldsymbol{\Phi}}^{2} \operatorname{loss}(\boldsymbol{X}, \boldsymbol{W}) \Delta \boldsymbol{\Phi}\right]. \end{aligned} \tag{3.18}$$

To adjust the value of $\Delta \boldsymbol{W}$, the original scale quantization is modified with fixed rounding-down and a controllable adaptive term $\boldsymbol{V}$ added before clipping:

$$\boldsymbol{W} + \Delta \boldsymbol{W} = \operatorname{clip}(\operatorname{round}(\frac{W}{s}) + \sigma(\boldsymbol{V}), -\epsilon, \epsilon), \tag{3.19}$$

where $\boldsymbol{V}$ is a continuous variable in real number, which can be regarded as a tunable parameter and updated through gradient descent during optimization. The rectified sigmoid function $\sigma(\cdot)$ (Louizos et al., 2018) is to force the adaptive value within range $[0, 1]$ and has non-vanishing gradient around 0 or 1. The overall optimization objective is:

$$\underset{\boldsymbol{V}}{\operatorname{argmin}} \mathbb{E}[\Delta \boldsymbol{\Phi}^{\top} \nabla_{\boldsymbol{\Phi}}^{2} \operatorname{loss}(\boldsymbol{X}, \boldsymbol{W}) \Delta \boldsymbol{\Phi}] + \lambda \sum_{i} (1 - |2\sigma(\boldsymbol{V}_i) - 1|^{\tau}). \tag{3.20}$$

The second regularization term is to encourage $\sigma(\boldsymbol{V}_i)$ to converge to value 0/1 with an appropriately annealed hyper-parameter τ.

While we have the formulation presented in Equation 3.20, optimizing all $\boldsymbol{V}$ for the entire network remains challenging due to the model's size. To address this complexity, we adopt a finer granularity approach by dividing the network into a series of quantization submodules and iteratively tuning the $\boldsymbol{V}$ values for each submodule. This allows us to reduce the complexity associated with optimizing $\boldsymbol{V}$ and focus more on quantization error at the submodule level. By shrinking the size of each submodule, we decrease the computational complexity involved in optimizing $\boldsymbol{V}$ for individual submodules. However, finer-grained submodules may potentially lead to local optima for each submodule, deviating from the global optimal solution. In order to achieve the highest restored accuracy, careful selection of submodule sizes is crucial, as depicted in Figure 3.8. Considering the network structure of *LaparNet*, we treat each individual residual block as a separate submodule and perform quantization on submodules one by one using the first term in Equation 3.20. To approximate the first term of the objective in Equation 3.20, we utilize the $l2$-norm of the difference between the quantized and unquantized output tensors, denoted as $|\boldsymbol{\Phi W} + \Delta \boldsymbol{W} - \boldsymbol{\Phi W}|_F^2$, at each

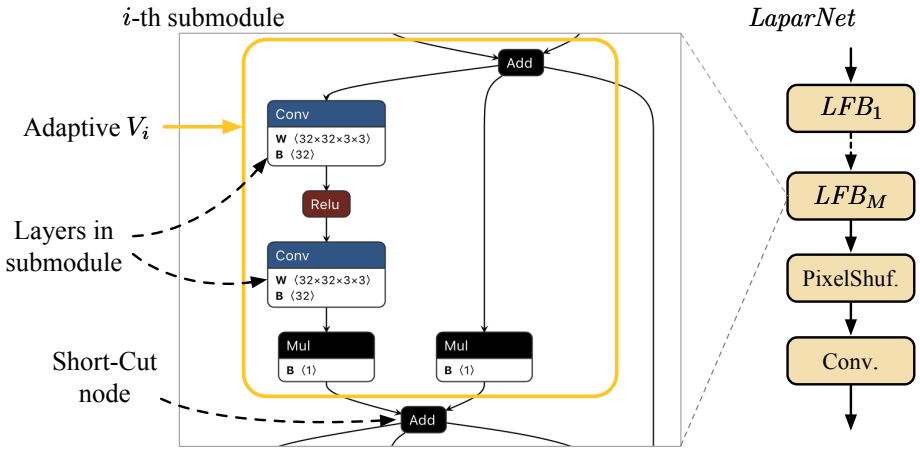


Figure 3.8: Visualization of quantization submodule for $\boldsymbol{V}$ optimization. We divide *LaparNet* with each submodule either stacked with less than 4 layers or ended with an Short-Cut node. In practice, most submodules share the same structure with a residual block in *LaparNet*.

submodule.

3.3 Experimental Results

Hardware Implementation: We validate our high-performance accelerator on NVIDIA Jetson Xavier NX, an edge device with embedded GPU. Metrics, including acceleration ratio and Super-resolution quality, are all compared with the state-of-the-art tool NVIDIA TensorRT to show the performance. NX integrates an ARM v8.2 64-bit CPU processor and a 384-core NVIDIA Volta GPU with 48 Tensor Cores. For a fair comparison, we choose 15W of power as the experimental setting, where it delivers up to 21 TOPS computing power. The clock frequency of the ARM processor is 2-core 1900MHz, and 4/6 core 1400MHz. The clock frequency of the GPU processor is 1100MHz. The accuracy comparison is evaluated on NVIDIA GeForce RTX 2080 Ti with 4352 FP32 FPUs (CUDA cores) and 544 Tensor cores for accuracy evaluation via PyTorch.

Software Implementation: The experimental environment is CUDA 11.0 and TensorRT 7.1.3. We use 32-bit floating point precision data type for full-precision evaluation and 8-bit integer data type for quantized model evaluation. The model-level training and accuracy evaluation are based on the official LAPAR code repository(SR-, [n. d.]).

Table 3.1: Inference Time (ms) and Acceleration ratios.

Input size	Scale	NVIDIA GeForce RTX 2080 Ti					NVIDIA Jetson Xavier NX		
		PyTorch	TensorRT	Ours	Acc. (PyTorch)	Acc. (TensorRT)	TensorRT	Ours	Acc. (TensorRT)
64 × 64	×2	6.94	1.30	1.02	×680.39%	×127.45%	12.37	9.04	×136.84%
	×3	8.26	1.94	1.40	×590.00%	×138.57%	22.62	14.28	×158.40%
	×4	9.86	2.79	1.88	×524.46%	×148.40%	35.83	20.54	×174.44%
128 × 128	×2	8.74	3.59	2.66	×328.57%	×134.96%	52.12	37.25	×139.92%
	×3	13.04	6.19	4.16	×313.46%	×148.80%	90.33	54.26	×166.48%
	×4	18.07	9.71	6.13	×294.78%	×158.40%	144.34	81.29	×177.56%
180 × 320	×2	17.12	12.40	9.25	×185.08%	×134.05%	177.57	124.12	×143.06%
	×3	30.83	21.66	14.63	×210.73%	×148.05%	325.07	200.02	×162.52%
	×4	44.69	34.69	22.12	×202.03%	×156.82%	534.99	318.60	×167.92%
360 × 640	×2	67.36	50.26	37.47	×179.77%	×134.13%	748.72	530.23	×141.21%
	×3	105.32	88.45	59.20	×177.90%	×149.41%	1466.91	973.25	×150.72%
	×4	406.93	141.08	91.09	×540.02%	×154.88%	-	-	-
Average	-	61.43	31.17	**20.91**	**×352.27%**	**×144.49%**	328.26	**214.81**	**×156.28%**

Inference time on NVIDIA Jetson Xavier NX with input size 360 × 640 and scale 4 is not available due to the memory limit of the edge device.

Dataset: The proposed accelerator is evaluated on common single image super-resolution (SISR) Set5(Bevilacqua et al., 2012), Set14(Ledig et al., 2017b), B100(Martin et al., 2001), Urban100(Ledig et al., 2017c), Manga109(Lai et al., 2017a) dataset.

3.3.1 Performance Evaluation

In order to demonstrate the effectiveness of our design in achieving speed-up, we conducted comparisons with the original model on both NVIDIA Jetson Xavier NX and RTX 2080 Ti devices. To showcase the generalization ability and robustness of our approach, we evaluated the inference performance across various input frame sizes and scale ratios. The obtained results are based on 32-bit floating point precision, and the corresponding running times are presented in Table 3.1. We successfully achieved real-time inference with SR output of 540P quality. Notably, our design outperforms PyTorch, delivering a 352.27% faster inference on the RTX 2080 Ti. Overall, our design surpasses TensorRT with a 144.49% faster inference on the RTX 2080 Ti and a 156.28% improvement on the Jetson Xavier NX, on average. Moreover, when compared to TensorRT, our accelerator

achieves a remarkable speed-up ranging from +27.45% to +77.56%. It is interesting to observe that Jetson Xavier NX exhibits more pronounced acceleration compared to the RTX 2080 Ti. This observation implicitly verifies that our design is particularly effective for embedded GPUs that have limited computation and communication resources.

In Section 3.3.1, we provide a comparison of the SR results' quality between our design and other well-known models, which serve as baseline methods. The performance evaluation is based on two widely used metrics for measuring the quality of restored high-resolution frames: PSNR (peak signal-to-noise ratio) and SSIM (structural similarity index measure). Higher values for these metrics indicate better performance. Remarkably, despite our model being a compressed version with 90% of the dictionary slimmed out, while the other baseline models are not subject to such compression, our design still outperforms most of the baseline models in terms of both PSNR and SSIM.

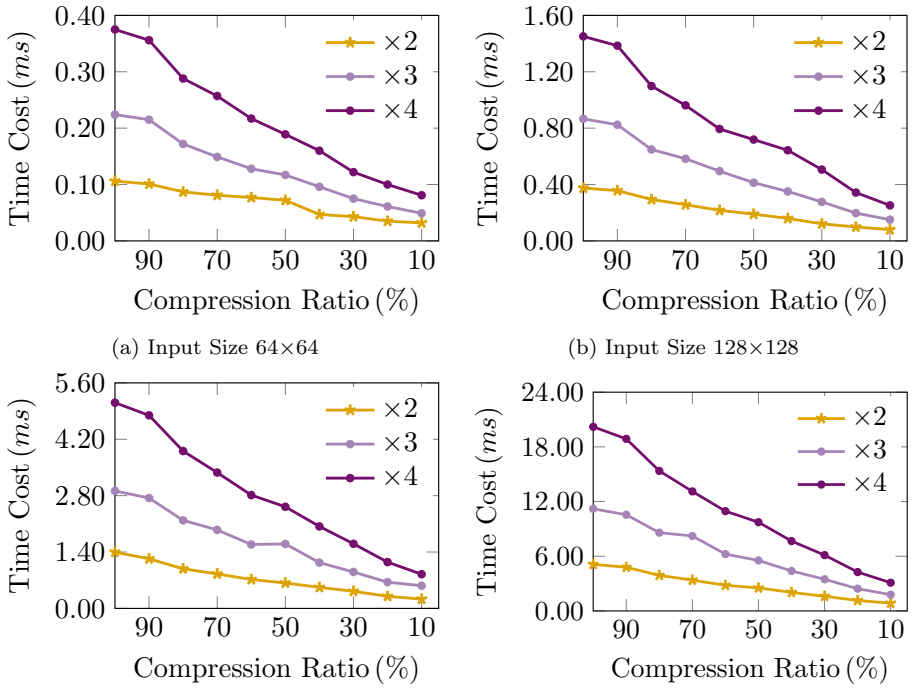
In order to validate the sparsity within the dictionary and its potential for acceleration, we conducted an ablation study on the slimming ratio as depicted in Figure 3.9. In this study, a slimming ratio of 100% represents the original uncompressed dictionary. Notably, the results demonstrate a linear decrease in time costs with respect to the compression ratio for various scales. The query and filtering processes for the dictionary exhibit a significant acceleration potential, with speeds reaching approximately 20 times faster than the original uncompressed version.

3.3.2 Compressed 8-bit Analysis

We show the practicability of our adaptive 8-bit post-training quantization by comparing it with other baseline methods on all five benchmarks and different upscaling factors. During the implementation process, we find out the tensor multiplication operation is sensitive to low-bit quantization and strongly affects the SR task accuracy. One possible reason is that large tensor multiplication may cause a wide activation value distribution, which may lead to information loss after clipping on the range during quantization. Therefore, we manually tick off the quantization node for the "mul" operation and analyze the effectiveness of other steps in our quantization flow. As shown in Table 3.2 that even compressed with quantized 8-bit inference, our approach still achieves comparable or even better performance with other SOTA baseline methods implemented in full precision.

Scale	Method	Params	MAC	Set5	Set14	B100	Urban100	Manga109
×2	SRCNN(Dong et al., 2014)	57K	53G	36.66/0.9542	32.42/0.9063	31.36/0.8879	29.50/0.8946	35.74/0.9661
	FSRCNN(Dong et al., 2016)	12K	6G	37.00/0.9558	32.63/0.9088	31.53/0.8920	29.88/0.9020	36.67/0.9694
	VDSR(Kim et al., 2016)	665K	613G	37.53/0.9587	33.03/0.9124	31.90/0.8960	30.76/0.9140	37.22/0.9729
	DRRN(Tai et al., 2017)	297K	6,797G	37.74/0.9591	33.23/0.9136	32.05/0.8973	31.23/0.9188	37.92/0.9760
	LapSRN(Lai et al., 2017b)	813K	30G	37.52/0.9590	33.08/0.9130	31.80/0.8950	30.41/0.9100	37.27/0.9740
	SRFBN-S(Li et al., 2019b)	282K	680G	37.78/0.9597	33.35/0.9156	32.00/0.8970	31.41/0.9207	38.06/0.9757
	FALSR-A(Chu et al., 2021)	1,021K	235G	37.82/0.9595	33.55/0.9168	32.12/0.8987	31.93/0.9256	-
	SRMDNF(Zhang et al., 2018)	1,513K	348G	37.79/0.9600	33.32/0.9150	32.05/0.8980	31.33/0.9200	-
	TPSR(Lee et al., 2020)	60K	14G	37.38/0.9583	33.00/0.9123	31.75/0.8942	30.61/0.9119	-
	SESR-M11(Bhardwaj et al., 2022)	27K	6.3G	37.58/0.9593	33.03/0.9128	31.85/0.8956	30.72/0.9136	37.40/0.9746
	Ours	528K	153G	**37.98/0.9604**	**33.59/0.9181**	**32.19/0.8999**	**32.09/0.9281**	**38.60/0.9771**
×3	SRCNN(Dong et al., 2014)	57K	53G	32.75/0.9090	29.28/0.8209	28.41/0.7863	26.24/0.7989	30.59/0.9107
	FSRCNN(Dong et al., 2016)	12K	5G	33.16/0.9140	29.43/0.8242	28.53/0.7910	26.43/0.8080	30.98/0.9212
	VDSR(Kim et al., 2016)	665K	613G	33.66/0.9213	29.77/0.8314	28.82/0.7976	27.14/0.8279	32.01/0.9310
	DRRN(Tai et al., 2017)	297K	6,797G	34.03/0.9244	29.96/0.8349	28.95/0.8004	27.53/0.8378	32.74/0.9390
	SelNet(Choi and Kim, 2017)	1,159K	120G	34.27/0.9257	30.30/0.8399	28.97/0.8025	-	-
	CARN(Ahn et al., 2018)	1,592K	119G	34.29/0.9255	30.29/0.8407	29.06/0.8034	28.06/0.8493	-
	SRFBN-S(Li et al., 2019b)	376K	832G	34.20/0.9255	30.10/0.8372	28.96/0.8010	27.66/0.8415	33.02/0.9404
	Ours	575K	96G	**34.35/0.9267**	**30.33/0.8420**	**29.11/0.8054**	**28.12/0.8523**	**33.48/0.9439**
×4	SRCNN(Dong et al., 2014)	57K	53G	30.48/0.8628	27.49/0.7503	26.90/0.7101	24.52/0.7221	27.66/0.8505
	FSRCNN(Dong et al., 2016)	12K	5G	30.71/0.8657	27.59/0.7535	26.98/0.7150	24.62/0.7280	27.90/0.8517
	VDSR(Kim et al., 2016)	665K	613G	31.35/0.8838	28.01/0.7674	27.29/0.7251	25.18/0.7524	28.83/0.8809
	DRRN(Tai et al., 2017)	297K	6,797G	31.68/0.8888	28.21/0.7720	27.38/0.7284	25.44/0.7638	29.46/0.8960
	LapSRN(Lai et al., 2017b)	813K	149G	31.54/0.8850	28.19/0.7720	27.32/0.7280	25.21/0.7560	29.09/0.8845
	CARN(Ahn et al., 2018)	1,592K	91G	32.13/0.8937	28.60/0.7806	27.58/0.7349	26.07/0.7837	-
	SRFBN-S(Li et al., 2019b)	483K	1,037G	31.98/0.8923	28.45/0.7779	27.44/0.7313	25.71/0.7719	29.91/0.9008
	TPSR(Lee et al., 2020)	61K	4G	31.10/0.8779	27.95/0.7663	27.15/0.7214	24.97/0.7456	-
	SplitSR (HI=2)(Liu et al., 2021)	94k	99G	31.53/**0.8950**	28.18/**0.7887**	27.28/**0.7458**	25.20/0.7704	-
	SESR-M11(Bhardwaj et al., 2022)	32.14K	1.85G	31.27/0.8810	27.94/0.7660	27.20/0.7225	25.00/0.7466	28.73/0.8815
	Ours	640K	76G	**32.15**/0.8944	**28.61**/0.7817	**27.59**/0.7366	**26.14/0.7873**	**30.39/0.9072**

We also analyze the inference speed of our implemented 8-bit inference. As shown in Table 3.3, we achieve significant speed-up even in comparison with our previous ICCAD2021 work. In general, our quantized implementation is 49.25% faster. More specifically, the acceleration ratio increases as the input size gets higher. The inference flow switches from compute-bound to memory-bound when the model and tensor size are bigger. Therefore, in this case, the low-bit inference is more effective. As for the full-precision "mul" operation, such kernels are already well optimized for GPU devices. As we profile the time consumption, all "mul" kernels combined only hold $< 2\%$ of total inference time, including the data transformation. We show the time consumption of "mul" kernels at different scale and input size in Table 3.3. This way, such a full-precision kernel will not introduce



(a) Input Size 64×64

(b) Input Size 128×128

(c) Input Size 180×320

(d) Input Size 360×640

Figure 3.9: Time consumption of the dictionary query and filtering with different compression ratios. Different input image sizes and scaling factors (from 2 to 4) are evaluated.

much overhead to the processing speed.

We also conduct a detailed ablation study to verify the effectiveness of each method applied in our quantization flow As shown in Table 3.4. For data calibration and tuning of the adaptive variable $\boldsymbol{V}$, we use Manga109 as the validation and test dataset.

3.3.3 Discussions

We present the impressive performance of our domain-specific high-performance SR accelerator through comprehensive experimental results, showcasing its effectiveness on the NVIDIA Jetson Xavier NX, which is equipped with limited power and hardware resources. Our approach addresses the challenges posed by dictionary learning algorithms used in the SR task, which exhibit specific memory and computation patterns that are not feasible for existing deployment toolkits. Furthermore, we face the challenge of dealing with large input frames and intermediate feature maps, which

Table 3.2: Performance evaluation of our fully-compressed Lapar-A (10%) at 8-bit inference on all benchmarks with other unquantized full-precision (FP32) state-of-the-art baseline methods.

Benchmark	Scale	Baseline (SOTA)	Ours (8-bit)
Set5	×2	37.82/0.9595	**37.87/0.9597**
	×3	34.29/0.9255	**34.30/0.9262**
	×4	**32.13/0.8937**	32.07/0.8926
Set14	×2	**33.55**/0.9168	33.47/**0.9169**
	×3	**30.29**/0.8407	30.27/**0.8410**
	×4	**28.60/0.7806**	28.50/0.7798
B100	×2	**32.12/0.8987**	32.07/0.8983
	×3	**29.06**/0.8034	29.02/**0.8035**
	×4	**27.58/0.7349**	27.45/0.7334
Urban100	×2	31.93/0.9256	**32.00/0.9268**
	×3	28.06/0.8493	**28.10/0.8514**
	×4	26.07/0.7837	**26.07/0.7857**
Manga109	×2	38.06/**0.9757**	**38.28**/0.9750
	×3	33.02/0.9404	**33.41/0.9429**
	×4	29.91/0.9008	**30.24/0.9047**

Table 3.3: Quantized 8-bit acceleration analysis in comparison with full-precision ICCAD21 (Zhao et al., 2021a). "mul" denotes the time consumption of "mul" kernel and corresponding data transformation.

Input size	Scale	ICCAD21 (mul) (Zhao et al., 2021a)	Ours (8-bit)	Acc.
64 × 64	×2	1.02 (0.04)	1.02	×100.00%
	×3	1.40 (0.10)	1.36	×102.94%
	×4	1.88 (0.09)	1.91	×98.43%
128 × 128	×2	2.66 (0.18)	2.09	×127.27%
	×3	4.16 (0.19)	3.40	×122.35%
	×4	6.13 (0.20)	5.13	×119.49%
180 × 320	×2	9.25 (0.43)	5.85	×158.12%
	×3	14.63 (0.44)	10.33	×141.63%
	×4	22.12 (0.44)	16.55	×133.66%
360 × 640	×2	37.47 (0.98)	20.72	×180.84%
	×3	59.20 (1.01)	36.49	×162.24%
	×4	91.09 (1.02)	63.33	×143.83%
Average	-	20.92 (0.42)	**14.02**	×**149.25**%

Table 3.4: Ablation Study on the performance influence of each quantization option. Performance metrics are PSNR/SSIM. The left-most column in **bold** is original full-precision FP32 Lapar-A for comparison with no quantization applied.

Methods	Original	(a)	(b)	(c)
Quantized		✓	✓	✓
Naive-8bit		✓	✓	
Exclude mul			✓	✓
Adaptive-8bit				✓
×2	**38.65/0.9772**	33.97/0.8977	37.74/0.9711	38.50/0.9762
×3	**33.51/0.9441**	31.30/0.8626	33.06/0.9415	33.45/0.9437
×4	**30.38/0.9073**	29.26/0.8381	30.02/0.9036	30.32/0.9065

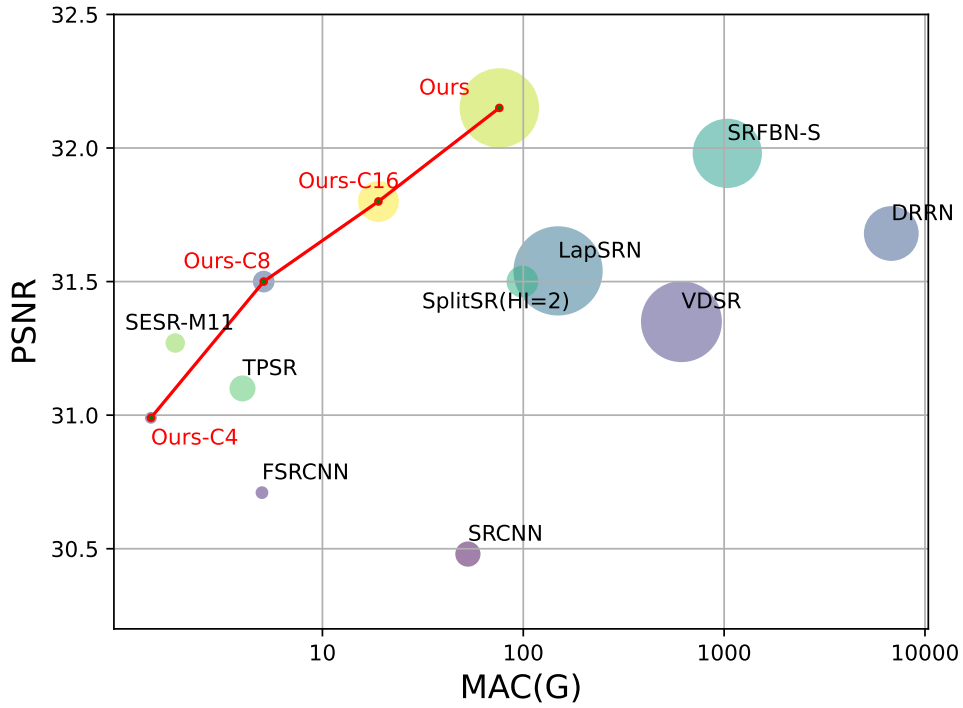


Figure 3.10: Comparison between our proposed model after compression and other lightweight methods (<1M parameters) on Set5 for ×4 setting. Circle sizes are set proportional to the numbers of parameters. "Ours" denotes our full-size LAPAR model and "Ours-C16, 8, 4" denote our LAPAR models with embedding channel number reduced from 32 to 16, 8 and 4.

impose significant memory pressure on the hardware.

In Figure 3.10, we observe a clear trade-off between performance and scale in the SR task.

Different models need to be carefully selected based on specific hardware resources and scenarios. While some models are compact with fast inference and support large frames, they may sacrifice accuracy and fail to restore sufficient texture details in high-resolution frames. For a fair comparison of scaling up the SR task to 720P with a 4x ratio under our consistent GPU hardware setting, we find that super-lightweight state-of-the-art (SOTA) models like SESR-M11 (Bhardwaj et al., 2022) or TPSR (Lee et al., 2020) can achieve inference times under 3ms per frame but may not meet the required accuracy, as depicted in Figure 3.10. On the other hand, the SOTA baseline model SplitSR (Liu et al., 2021) outperforms the former two baselines in terms of performance but exhibits a significantly longer inference time of 19ms.

In contrast, our dictionary-based approach consistently outperforms all other baselines with an inference time of 16ms per frame. Additionally, we introduce a Pareto curve in Figure 3.10 to demonstrate the flexibility of our deployed dictionary-based algorithm. To create the curve, we scale down our model to 1.56% of its full size, which is similar to the size of the smallest baseline model. We achieve this by halving the embedding channel number of all residual blocks at each point, starting from 32 and progressing to 16, 8, and 4. As shown in Figure 3.10, our deployed algorithm achieves the highest performance-scale efficiency at channel sizes of 32, 16, and 8, except for the extremely compressed 4-channel case, which exhibits slightly lower PSNR than SESR-M11 (Bhardwaj et al., 2022).

To the best of our knowledge, our proposed accelerator represents the first successful achievement of superior performance in SR applications on edge embedded GPUs.

3.4 Summary

This Chapter presents hardware-aware AI optimization with the development of a domain-specific high-performance acceleration for deploying super-resolution models based on the LAPAR framework. In our framework design, we introduce a strategy for dictionary slimming, aimed at extracting the most informative dictionary items to enhance the efficiency of inference. Additionally, we have designed a hardware-aware acceleration engine, which effectively utilizes the limited hardware resources to optimize the inference process. Furthermore, we conducted experiments involving low-bit

inference, employing an adaptive 8-bit quantization strategy to further enhance the acceleration. Through various evaluations, our system demonstrates superior performance compared to the state-of-the-art tools TensorRT and PyTorch, specifically on edge embedded GPUs such as the NVIDIA Jetson NX and 2080 Ti, without any compromise in terms of quality.

Chapter 4

Hardware-Aware Quantization Acceleration With Bayesian Optimization

Quantization has emerged as a highly effective strategy for accelerating AI inference using lower numerical precision. In practical deep learning scenarios, such as autonomous driving and VR/AR technology, quantization enables deployment without requiring changes to the original architecture.

Although some researchers (Dong et al., 2019; Hawks et al., 2021; He et al., 2018; Wang et al., 2020a; Yao et al., 2021) have tried to introduce hardware information into the quantization stage, these hardware-aware approaches still mainly focus on the quantization stage with only bit-width selection as the objective. Some works simply insert indirect constraints/limitations to their quantization objective, such as the number of operations, number of memory references, etc. The study (Liu et al., 2022) has shown that computation amount (FLOPS/BOPs), parameter numbers, or memory accesses of the model may not be good proxies for inference latency. For example, quantization can affect the memory alignment, which relates to the inference latency but is hard to analyze statically through the proxies. On the other hand, backend configuration search space is related to and varies

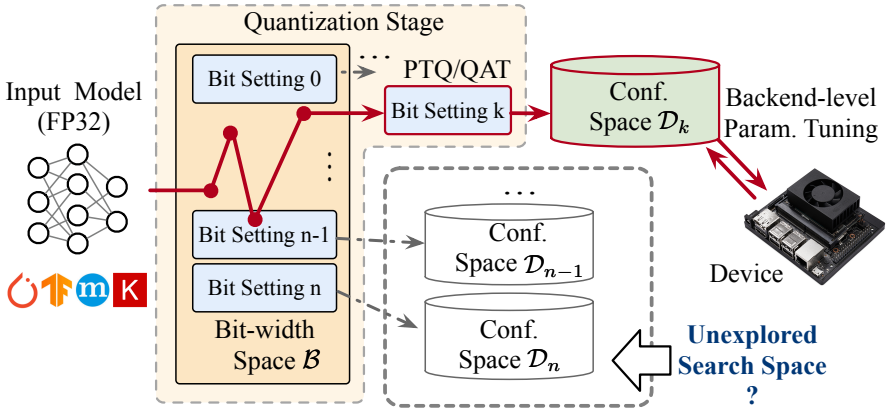
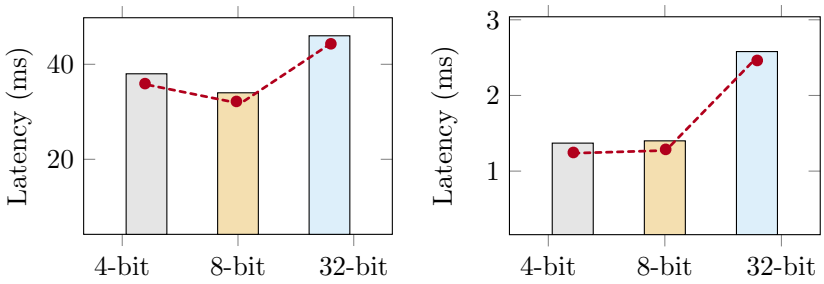


Figure 4.1: Current quantization model deployment is a 2-stage flow composed of quantization at the model level and compilation optimization at the backend level. PTQ/QAT denote post-training quantization and quantization aware training. Decoupling these two steps may lead to potential unexplored search space. Each "Bit setting" denotes a bit-width assignment for layers in the model.

along with the pre-determined model bit-width setting. As visualized in the lower area in Figure 4.1, decoupling these two stages may result in inadequate search, where some subspace on backend and quantization bit-setting combination are unexplored, which may possibly result in sub-optimal deployment results. Another challenge of existing quantization methods is transferability. The same quantization scheme may show different speed-up ratios on different hardware backends, as shown in Figure 4.2, which indicates potential bias when transferring quantization configuration to different backends. Most previous works leave the necessity to genuinely evaluate the real performance of the quantized model on different hardware backends. HAQ (Wang et al., 2019b) relies on a simulator to retrieve the power/latency of an FPGA-based accelerator. HAWQ-V3 (Yao et al., 2021)'s ILP solver requires pre-collected latency, bit operations count, as well as the Hessian score of each kernel on specific hardware. They avoid it because exploration is time-consuming, considering the large search space and huge time cost per evaluation. Firstly, mixed precision search space has complexity $\mathcal{O}(k^n)$, where k is the number of bit candidates supported by the backend and n is the number of layers in the model. Moreover, for each bit-width configuration, the deployment flow requires backend-level compilation of the loop scheduling parameters and data layout, which is also a time-consuming process.

Given these challenges, we propose **BAQE** framework to reconstruct the quantization deploy-



(a) Jetson Orin NX (b) RTX 3070

Figure 4.2: Inference latency comparison with example model Resnet-50 and data type in Int4, Int8 and Int32. Red dotted line indicates the speed-up ratio from quantization, which differs for Jetson Orin NX and RTX 3070. Speed-up from quantization on NX is marginal, and Int8 inference is even slightly faster than Int4. In comparison, the RTX3070 speed-up is more noticeable.

ment flow to simultaneously search for quantization bit-width settings and backend configuration. The contribution of this section is listed as follows:

- We discuss the backend adaption challenge for DNN quantization deployment and propose **BAQE** to bridge the gap between algorithm-level and backend-level optimization.
- A unified search space is built to synchronously optimize both the model quantization bit-width setting and backend configuration parameters together.
- A two-stage searching strategy is proposed to efficiently reach the optimal solution in the unified search space without prior backend information.
- Experiments show that our approach achieves superior inference time and accuracy trade-off and quickly reaches Pareto optimality.

4.1 Preliminaries

4.1.1 Hardware-Aware Quantization.

One of the primary objectives of DNN quantization is to enhance inference latency, which is highly dependent on the underlying hardware. Several methods, such as those discussed in (Dong et al., 2019; Hawks et al., 2021; He et al., 2018; Wang et al., 2020a; Yao et al., 2021), have considered the

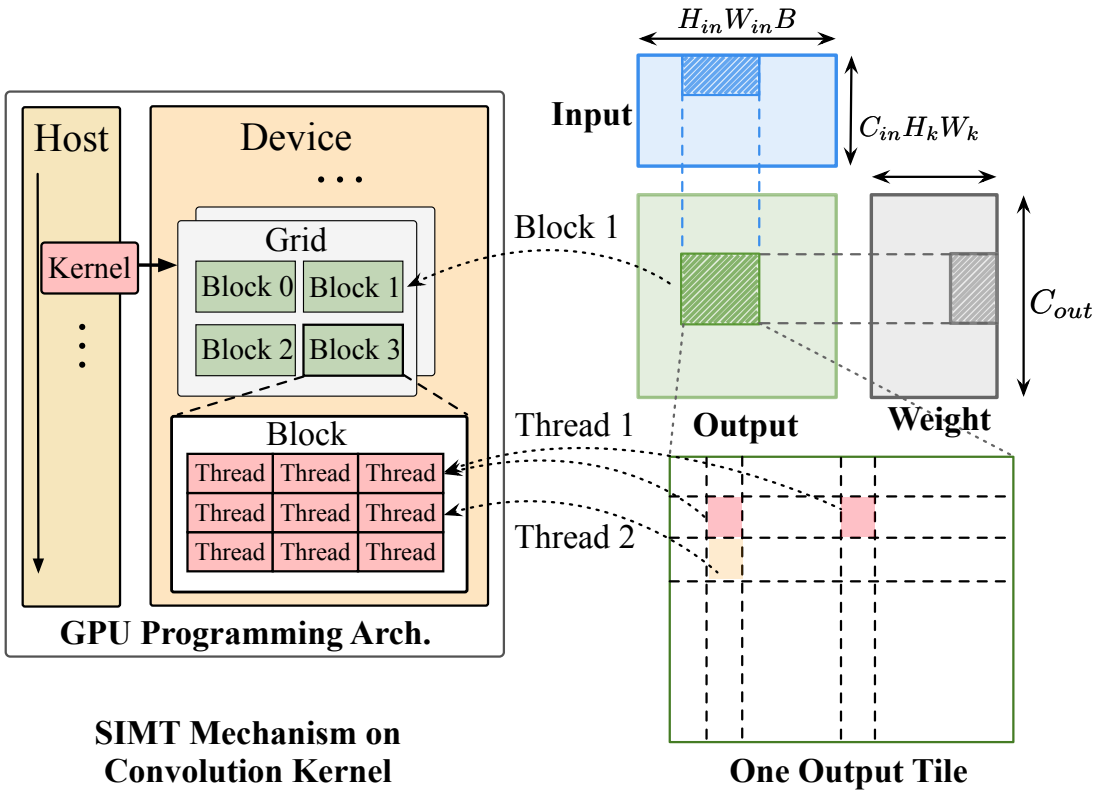


Figure 4.3: Example of flatten 2-D convolution workload partition on CUDA programming architecture with hierarchical parallelism. 2-D matrix operations are split into tiles, which are assigned to multiple thread blocks in GPU device.

limitations of hardware resources and formulated the problem as a constrained optimization task. The goal is to minimize information loss or accuracy degradation while simultaneously adhering to memory/speed metrics such as model size or GFLOPs. In the work of (Wang et al., 2019b), reinforcement learning (RL) was utilized to determine the optimal bit-width configuration. This involved mapping to a simulator to obtain feedback on energy consumption and latency. Furthermore, (Yao et al., 2021) extended this awareness by directly compiling all layers at different bit-widths on the target hardware to measure the actual latency before making precise bit-width selections.

4.1.2 Hardware Backend Deployment Optimization.

In the present era, most deep learning layers involve dense tensor operations that can be decomposed into a multi-level representation of for-loops. However, the original execution order of these loops

may not fully exploit the computational units or memory bandwidth available. Various approaches, such as loop reordering, loop unrolling, and loop tiling, have been developed to enhance execution efficiency and reduce memory cache misses. Similarly, transforming the data layout to align with the loop order can yield the same beneficial effects. In the case of the GPU Backend, the selection of the "tile size" refers to the number of parallel threads assigned to each CUDA block for execution. It is crucial to choose an appropriate tile size that strikes a balance between communication and parallelism.

By re-scheduling the aforementioned loops along with the operations, it becomes possible to explore the optimal configuration of the backend by reorganizing the low-level implementation. In the field of DNN compilation, several approaches, as mentioned in (Chen et al., 2018a,c; Mu et al., 2020; Zheng et al., 2020), have proposed techniques for automatically tuning these backend parameters. The work of (Chen et al., 2018a,c) initially employed a simulated annealing algorithm as a search strategy to optimize the backend configuration. On the other hand, both (Mu et al., 2020) and (Zheng et al., 2020) utilized guided genetic algorithms and evolutionary search methods to explore the search space of backend configurations.

4.2 Methodology

4.2.1 Overview of BAQE Framework

Figure 4.4 visually describes the framework of BAQE. In the first step, we establish a comprehensive and extensive search space encompassing both model-level and backend-level parameters, as detailed in Section 4.2.2. Subsequently, in Section 4.2.3, we employ a TED-based method as an initial step to generate diverse samples across the search space for the bit-width $\boldsymbol{b}$ and backend configuration parameters $\boldsymbol{hp}$. This step ensures a broad exploration of the parameter space. Following that, in Section 4.2.4, BAQE leverages a multi-objective Bayesian optimization (BO) algorithm to search for the optimal values of $\boldsymbol{b}$ and $\boldsymbol{hp}$. The performance of each configuration is evaluated on the target device, and automatic deployment and tuning take place at each iteration, as described in Section 4.2.6.

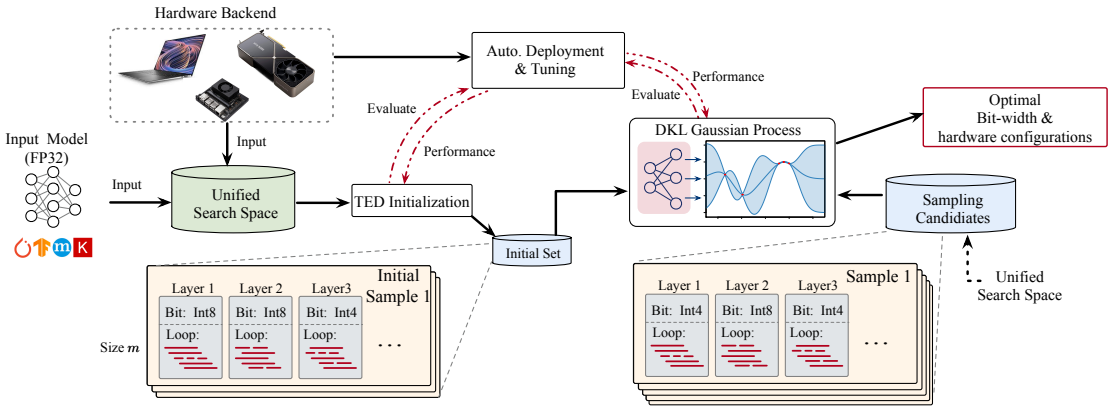


Figure 4.4: Overview of the framework of BAQE. Our framework searches for bit-width settings $\boldsymbol{b}$ and backend configuration parameters $\boldsymbol{hp}$ (loop scheduling parameters) simultaneously. Quantization scaling factor $\boldsymbol{s}$ and network parameters are tuned on the device at each searching step.

4.2.2 The Unified Global Search Space

The unified global search space $\mathcal{S}$ is expanded from the original mixed-precision space to:

$$\begin{aligned}\mathcal{S} = (&b_1, b_2, ..., b_l, w_1, w_2, ..., w_l, \\ &s_1, s_2, ..., s_l, hp_1, hp_2, .., hp_j),\end{aligned} \tag{4.1}$$

where $\boldsymbol{b}$: $[b_1, b_2, ..., b_l]^\top$ are the bit-width of each layer l to quantize, $\boldsymbol{s}$: $[s_1, s_2, ..., s_l]^\top$ are the corresponding scaling factors. $\boldsymbol{w}$: $[w_1, w_2, ..., w_l]^\top$ are the model weights. $\boldsymbol{hp}$: $[hp_1, hp_2, .., hp_j]^\top$ denotes the backend configuration parameters, including loop orders, tiling size, and data layout. This new search space is more comprehensive with extended variables, whereas also more complicated. Here the $b_1, b_2, ..., b_l$ and $hp_1, hp_2, .., hp_j$ are discrete variables while $s_1, s_2, ..., s_l$ are continuous. In correspondence, our framework chose both auto-tuning and discrete sampling strategies.

In order to provide a clear understanding of the backend configuration parameters $\boldsymbol{hp}$, we present a visualization of a 2-D convolution in Figure 4.5. As illustrated in Figure 4.5, rearranging the loop topology enables the inner loops to access and process contiguous data from memory, while preserving the output values. This reordering effectively reduces memory pressure by minimizing cache miss rates. Furthermore, using a data layout with dimensions "nchw" proves to be more efficient for memory retrieval when the loop on H_{in}/W_{in} is nested inside the loop C_{in}. Loops can also be unrolled into tiles of multi-threads and mapped to hardware thread blocks for parallel execution, as

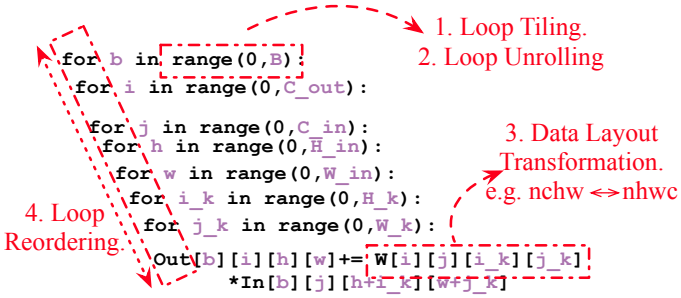


Figure 4.5: Example of pseudo implementation of the 2-D convolution kernel. Annotations in red color denote backend-level optimization methods on this multi-loop code piece, which can increase parallelism or optimize memory read/write efficiency.

shown in Figure 4.3.

4.2.3 TED-based Initial Sampling

As mentioned earlier, BAQE serves as a backend-adaptive framework that operates without prior knowledge of the backend domain. In situations where initial knowledge of the specific hardware backend is limited, it becomes necessary to collect actual latency/accuracy data through online tuning and evaluation. This process is crucial for accurate optimization, as it allows for real-time adjustments based on the observed performance. Consequently, the optimization flow heavily relies on efficient sampling and searching techniques to address the challenge of acceleration while ensuring thorough exploration of the parameter space.

Transductive experimental design (TED) is a sampling strategy that effectively enhances the quality of samples for regression tasks, even in the absence of labels or prediction values for the sample data. This aligns well with our problem context, as the hardware platform is essentially a black box where prior knowledge is lacking. The primary objective of experimental design is to carefully select a set of candidates, denoted as $(\boldsymbol{b}, \boldsymbol{hp})0, (\boldsymbol{b}, \boldsymbol{hp})1, ...$, that offer maximum information content. In this context, we represent each candidate as $\boldsymbol{x}$, and it can be defined as follows:

$$\boldsymbol{x}_i = [\boldsymbol{b}^\top, \boldsymbol{hp}^\top]_i = [b_1, b_2, .., b_l, hp_1, hp_2, ..., hp_j]. \tag{4.2}$$

Suppose we are trying to build a linear regression of $[\boldsymbol{b}, \boldsymbol{hp}]$ on performance y, which can be formu-

lated as:

$$\min_{\boldsymbol{w}} \Phi(\boldsymbol{w}) = \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i - y_i)^2 + \mu \|\boldsymbol{w}\|^2. \tag{4.3}$$

In this case, m represents the number of samples, $\boldsymbol{w}$ denotes the regression weight, and μ signifies the regularization coefficient. The maximum likelihood estimate, denoted as $\hat{\boldsymbol{w}}$, is obtained by minimizing the function $\Phi(\boldsymbol{w})$. The estimation error, $\boldsymbol{w} - \hat{\boldsymbol{w}}$, possesses a mean of 0 and a covariance of $\sigma^2 \mathbf{C_w}$. In this context, σ is a constant, and $\mathbf{C_w}$ refers to the inverse Hessian of $\Phi(\boldsymbol{w})$.

$$\begin{aligned} \mathbf{C_w} &= ([\boldsymbol{B}, \boldsymbol{HP}]^\top [\boldsymbol{B}, \boldsymbol{HP}] + \mu \mathbf{I})^{-1} \\ &= (\boldsymbol{X}^\top \boldsymbol{X} + \mu \mathbf{I})^{-1}, \end{aligned} \tag{4.4}$$

Here the set of all m sampled bit-width and backend parameters can be represented as $\boldsymbol{X} = [\boldsymbol{x}1, \boldsymbol{x}2, ..., \boldsymbol{x}m]$. It is important to note that the covariance reflects the level of confidence in the estimation. A higher degree of confidence indicates greater informativeness of the sampled data. In essence, maximizing the trace $Tr(\mathbf{Cw})$ aims to enhance the level of informativeness. However, a potential drawback is that $\mathbf{C_w}$ may not align with the quality of all other data points in the search, introducing a limitation to consider.

When sampling m candidates $(\boldsymbol{b}, \boldsymbol{hp})$ from the search space $\mathcal{S}$, we denote the set of all data points to search as $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, ...]^\top$, where each $\boldsymbol{v}_i$ represents a combination of bit-width and backend configuration $(\boldsymbol{b}, \boldsymbol{hp})$. The regression error associated with predicting the performance on $\boldsymbol{V}$ is characterized by $\sigma \mathbf{C_V}$, where:

$$\begin{aligned} \mathbf{C_V} &= \boldsymbol{V} \mathbf{C_w} \boldsymbol{V}^\top \\ &= \boldsymbol{V} ([\boldsymbol{B}, \boldsymbol{HP}]^\top [\boldsymbol{B}, \boldsymbol{HP}] + \mu \mathbf{I})^{-1} \boldsymbol{V}^\top \\ &= \boldsymbol{V} (\mathbf{X}^\top \mathbf{X} + \mu \mathbf{I})^{-1} \boldsymbol{V}^\top. \end{aligned} \tag{4.5}$$

With the original goal of optimizing sample informativeness, we replace the $\mathbf{C_w}$ and switch to minimize trace of $\mathbf{C_V}$. In addition, we try to add non-linearity with a kernel function k: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}$, replacing original sample data matrix with element distance. In the covariance matrix, all data points $\boldsymbol{V}$ is replaced with $\mathbf{K_{VX}}$ that $(\mathbf{K_{VX}})_{ij} = k(\boldsymbol{v}_i, \boldsymbol{x}_j)$. The sample product $\boldsymbol{X}^\top \boldsymbol{X}$ is

replaced with $\mathbf{K}_{\boldsymbol{XX}}$ where $(\mathbf{K}_{\boldsymbol{XX}})_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Then our initial sampling objective becomes:

$$\begin{aligned} \max_{\boldsymbol{X}} \quad & Tr(\mathbf{K}_{\boldsymbol{VX}}(\mathbf{K}_{\boldsymbol{XX}} + \mu\mathbf{I})^{-1}\mathbf{K}_{\boldsymbol{XV}}) \\ \text{s.t.} \quad & \boldsymbol{X} \subset V, |\boldsymbol{X}| = m. \end{aligned} \tag{4.6}$$

After deriving the optimization objective, the initial sampling strategy of BAQE is constructed and listed in Algorithm 3. Note that here the kernel function $k(\cdot, \cdot)$ is replaceable with any non-linear distance function. The initial sampling of BAQE iteratively chooses the most informative bit-width/backend parameters to sample $\boldsymbol{x}^*$ while dynamically updating $\mathbf{K}$ for each iteration.

Algorithm 3 TED-Based Initial Sampling

Input: Bit/backend parameters space $\mathcal{S}$, regularization coefficient μ, sample size m.
Output: Sample $\boldsymbol{X}$ with $|\boldsymbol{X}| = m$.
$\boldsymbol{X} \leftarrow \emptyset, \boldsymbol{V} \leftarrow \mathcal{S}$;
$\boldsymbol{K} \leftarrow (\boldsymbol{K})_{ij} = k(\boldsymbol{v}_i, \boldsymbol{v}_j), \forall \boldsymbol{v}_i, \boldsymbol{v}_j \in \mathbf{V}$;
for i in range$(0, m)$ **do**
    $\boldsymbol{x}^* \leftarrow \text{argmax}_{\boldsymbol{x} \in \boldsymbol{V}} Tr(\mathbf{K}_{\boldsymbol{Vx}}(\mathbf{K}_{\boldsymbol{xx}} + \mu\mathbf{I})^{-1}\mathbf{K}_{\boldsymbol{xV}})$;
        ▷ $\mathbf{K}_{\boldsymbol{Vx}}, \mathbf{K}_{\boldsymbol{xx}}, \mathbf{K}_{\boldsymbol{xV}}$ are from rows/columns of $\mathbf{K}$
    $\boldsymbol{X} \leftarrow \boldsymbol{X} \cup \boldsymbol{x}^*$, $\boldsymbol{V} \leftarrow \boldsymbol{V} \backslash \boldsymbol{x}^*$;
    $\boldsymbol{K} \leftarrow \boldsymbol{K} - Tr(\mathbf{K}_{\boldsymbol{Vx}^*}(\mathbf{K}_{\boldsymbol{x}^*\boldsymbol{x}^*} + \mu\mathbf{I})^{-1}\mathbf{K}_{\boldsymbol{x}^*\boldsymbol{V}})$;
end for
Return $\boldsymbol{X}$;

4.2.4 Multi-objective Exploration

After gathering the initial dataset of $(\boldsymbol{b}, \boldsymbol{hp})$, our next step is to navigate through the search space. Determining the optimal quantized bit widths and backend configuration parameters that strike a balance between on-device inference time, accuracy, and model size within a limited time budget is a challenging task. This challenge arises due to two primary reasons. Firstly, the relationship f between a post-quantized DNN model and its on-device inference time and model accuracy is intricate and unknown. Secondly, the process of tuning and evaluating the accuracy of a DNN model is time-consuming. To address this problem, we propose a multi-objective exploration approach based on Bayesian optimization (BO). This methodology aims to find effective solutions that optimize the trade-offs between the aforementioned objectives.

Bayesian optimization (BO) encompasses two key components, namely the *surrogate model* and

the *acquisition function*. To construct the surrogate model, we utilize the initial dataset generated by the algorithm proposed in Section 4.2.3. This surrogate model is often implemented using a Gaussian process, acting as a representation of the underlying relationship f discussed earlier. The acquisition function plays a crucial role in guiding the direction of the multi-objective exploration. It determines the selection of candidate solutions, encompassing bit widths and loop scheduling parameters, based on predictions obtained from the surrogate model. By evaluating the quantized DNN model's on-device inference time, accuracy, and other relevant factors, we optimize the overall exploration runtime. BO proceeds iteratively, selecting new candidate solutions at each iteration. In the following sections, we explain in detail the structure of our proposed surrogate model and the acquisition function for multi-objective exploration.

Surrogate model. Regarding the exploration focusing on multi-objectives, *i.e.*, on-device inference time, accuracy, and model size, we propose the Gaussian process with deep kernel learning (DKL-GP) as the surrogate model. Without loss of generality, suppose a set of candidates $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$, where $\boldsymbol{x}_i = (\boldsymbol{b}, \boldsymbol{hp})_i$. Each candidate $\boldsymbol{x}_i$ is applied to the DNN model, and the corresponding metrics, such as on-device inference time, model accuracy, and size, are defined as:

$$\boldsymbol{Y} = \Big(\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_n\Big)^\top = \begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ \vdots & \vdots & \vdots \\ y_{n1} & y_{n2} & y_{n3} \end{pmatrix}, \tag{4.7}$$

where y_{il} refers to the value of the i-th input $\boldsymbol{x}_i$ for l-th metric. DKL-GP places a Gaussian process (GP) prior f_l for these metrics, respectively, as shown in Equation (4.8).

$$y_{il} \sim \mathcal{N}(f_l(\boldsymbol{x}_i), \sigma_l^2), \tag{4.8}$$

where the variance for the l-th metric is denoted as σ_l. We introduce a positive semi-definite matrix $\boldsymbol{K}^f$ to represent the inter-objective similarities, while $\boldsymbol{K}^x$ captures the covariance function over the input $\boldsymbol{x}_i$. The inter-objective similarity illustrates how an observation of one metric can influence the predictions of another metric. For instance, a larger DNN model size often results in increased on-device inference time due to the higher computational requirements. To model the correlations

between metrics, we utilize the DKL-GP technique and employ Equation 4.9.

$$\text{cov}(f_l(\boldsymbol{x}), f_k(\boldsymbol{x}')) = \boldsymbol{K}^f_{lk}\boldsymbol{K}^x(\boldsymbol{x}, \boldsymbol{x}'), \tag{4.9}$$

The inter-objective similarities between metrics l and k are represented by $\boldsymbol{K}^f_{lk}$. By incorporating Equation 4.9 into the DKL-GP approach, we can effectively capture correlations among the metrics (Bonilla et al., 2007). These correlations play a vital role in balancing the trade-offs between on-device inference time, model accuracy, and size during the exploration procedure. For instance, when provided with a new input $\boldsymbol{x}^*$, DKL-GP predicts its metric mean values with:

$$\begin{aligned} f_l(\boldsymbol{x}^*) &= (\boldsymbol{K}^f_{\cdot l} \otimes \boldsymbol{K}^x(\boldsymbol{x}^*, \boldsymbol{X}))^\top \Sigma^{-1}\boldsymbol{Y}; \\ \Sigma &= \boldsymbol{K}^f \otimes \boldsymbol{K}^x(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{D} \otimes \boldsymbol{I}, \end{aligned} \tag{4.10}$$

The l-th column of $\boldsymbol{K}^f$ is denoted as $\boldsymbol{K}^f_{\cdot l}$. The Kronecker product is represented by $\otimes$. We use $\boldsymbol{D}$ to indicate the diagonal matrix, where the diagonal elements are σ_l^2, and $\boldsymbol{I}$ refers to the identity matrix. To parameterize the kernels for $\boldsymbol{K}^f$ and $\boldsymbol{K}^x$, we employ a stacked multilayer perceptron (MLP) approach. This allows us to model the deep kernels, which offer improved robustness compared to previous kernel formulations such as automatic relevance determinant (ARD). Equation (4.11) shows the example deep kernels,

$$\begin{aligned} \text{cov}(\boldsymbol{x}, \boldsymbol{x}') &= \sigma^2 \exp\left(-\boldsymbol{\beta}^\top \Sigma^{-1}\boldsymbol{\beta}\right); \\ \boldsymbol{\beta} &= (\phi(\boldsymbol{x}, \boldsymbol{w}) - \phi(\boldsymbol{x}', \boldsymbol{w})), \end{aligned} \tag{4.11}$$

where $\boldsymbol{w}$ denotes weights of multilayer perceptions, and ϕ are non-linear transformations.

Acquisition function. BAQE utilizes the improvement of Pareto hypervolume (EPVI) as the chosen acquisition function to direct the multi-objective search. Throughout the Bayesian optimization (BO) process, the evaluation of performance on bit-width $\boldsymbol{b}$ and backend configuration parameters $\boldsymbol{hp}$ encompasses on-device inference time, accuracy, and model size. However, the Gaussian Process, employed for modeling the surrogate function, can only handle a single performance metric for evaluation. Given this limitation, we aim to formulate a new objective that adequately captures all three dimensions of the objective. This task presents a challenge since these three factors exhibit negative correlations. For instance, compressing the model size or reducing the bit-width may result

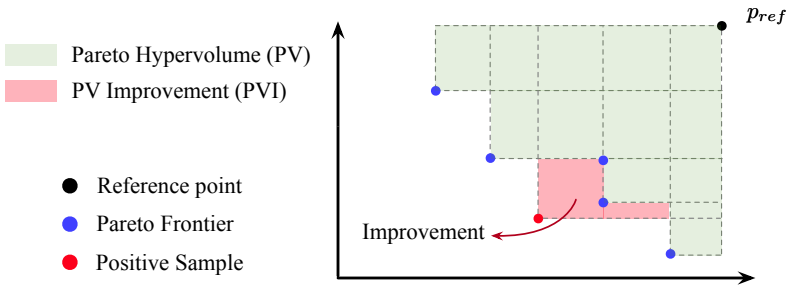


Figure 4.6: 2-D visualization of the idea of Pareto Hypervolume (PV) and Expected Improvement of Pareto Hypervolume (EPVI).

in a loss of accuracy. Similarly, increasing accuracy through higher precision incurs higher inference latency. In light of this, our optimization objective shifts from co-optimization to identifying a favorable trade-off point that achieves overall optimality. Within BAQE, we adopt the Expected Improvement of Pareto Hypervolume (EPVI) as the acquisition function. This choice enables efficient guidance of the optimization process towards the Pareto frontier of the three objectives.

In Figure 4.6, we illustrate the concept of Pareto Hypervolume and EPVI using a 2-D visualization. In the 3-D space of (accuracy, latency, model size) represented as $\Omega(x, y, z)$, we consider a reference point p_{ref}. The Pareto Hypervolume refers to the "half-arch" space bounded by p_{ref} and the optimal points (depicted as blue dots in Figure 4.6) on the Pareto frontier $\mathcal{P}$. This volume represents a Lebesgue measure of the Pareto optimality across the three dimensions. Its formulation is as follows:

$$PV_{\Omega}(\mathcal{P}, p_{ref}) = \sum_{p \in \Omega} [\mathbb{1}(p \succcurlyeq p_{ref})[1 - \prod_{p' \in \mathcal{P}} \mathbb{1}(p \not\preccurlyeq p')]], \tag{4.12}$$

where $\mathbb{1}(\cdot)$ is 1 if the statement is true and 0 otherwise. "$\succcurlyeq$" denotes "better or equal" at all three dimensions. The PV bound by p_{ref} is the green area in Figure 4.6. p_{ref} is manually selected on space Ω.

With the definition of PV, the improvement of Pareto hypervolume (PVI) is rather clear; namely, the optimality/PV increased from a new positive sample point p^+ in Ω that surpasses the old frontier (red area in Figure 4.6):

$$PVI_{\Omega}(\mathcal{P}, p_{ref}, p^+) = PV_{\Omega}(\mathcal{P} \cup p^+, p_{ref}) - PV_{\Omega}(\mathcal{P}, p_{ref}), \tag{4.13}$$

while the expected PV improvement (EPVI) is:

$$\begin{aligned} EPVI_{\Omega}(\mathcal{P}, p_{ref}) &= \mathbb{E}_{p^+|\Omega}[PVI_{\Omega}(\mathcal{P}, p_{ref}, p^+)] \\ &= \sum_{p^+ \in \Omega} Prob(p^+|\Omega) \cdot PVI_{\Omega}(\mathcal{P}, p_{ref}, p^+). \end{aligned} \tag{4.14}$$

Our approach constructs the acquisition function using Equation 4.14 to identify the point p^+ that maximizes the expected improvement. The probability $Prob(p^+|\Omega)$ is modeled using a multi-objective Gaussian Process (GP) for the dimensions of accuracy, latency, and model size. We optimizes the process by iteratively selecting $\boldsymbol{x}^*$ from a set of candidates $\boldsymbol{X}$ in a manner that maximally increases the likelihood of the DKL-GP function $f(\cdot)$:

$$\boldsymbol{x}^* = \underset{f(\boldsymbol{x})=p^+}{\operatorname{argmax}} EPVI_{\Omega}(\mathcal{P}, p_{ref}). \tag{4.15}$$

4.2.5 Complete Exploration Flow

In addition to providing a visualization and description for each component mentioned earlier, we provide a comprehensive flow description below. The entire process of multi-objective exploration in BAQE is outlined in Algorithm 4.

To begin with, the TED-based initial sampling procedure generates the first batch $\boldsymbol{X}_{init}$, which serves as the training dataset for fitting the DKL-GP surrogate model.

In our experiments, the selection of the initial sample size m is not critical, as any number greater than 10 demonstrates satisfactory convergence. For both the Resnet-18 and Resnet-50 benchmark models, we choose 10 as the value for m. As discussed, the TED-based sampling approach aims to maximize the trace $\mathbf{K}_{\boldsymbol{VX}}(\mathbf{K}_{\boldsymbol{XX}} + \mu\mathbf{I})^{-1}\mathbf{K}_{\boldsymbol{XV}}$, which indirectly minimizes the covariance of the performance regression model. At a high-level, this step reduces the prediction uncertainty of the regression model by employing a more sparse sampling strategy across the search space $\mathcal{S}$.

Subsequently, the Bayesian optimization (BO) process continues by sampling from all candidates $\boldsymbol{V}$ and selecting the combination $(\boldsymbol{b}, \boldsymbol{hp})$ that maximizes the expected improvement in Pareto hypervolume. Each candidate is associated with a multilayer perceptron model denoted as $\phi(\cdot)$. In our experiments, the multilayer perceptron $\phi(\cdot)$ consists of four linear layers, with the first three layers followed by a ReLU layer to introduce non-linearity. The dimensions of the linear layers are set as

Algorithm 4 BAQE Complete Exploration Flow

Input: Bit/backend parameters space $\mathcal{S}$, Stopping iteration number N, initial smaple size m, regularization coefficient μ.
Output: Pareto optimal solution set $\boldsymbol{X}_{optim}$.
$\boldsymbol{X} \leftarrow \emptyset, \boldsymbol{V} \leftarrow \mathcal{S}$;
`//Initial sampling`
$(\boldsymbol{b}, \boldsymbol{hp}) \leftarrow \textbf{TED}(\mathcal{S}, \mu, m)$;
Take $\boldsymbol{X}_{init} \leftarrow (\boldsymbol{b}, \boldsymbol{hp})$ as initial set and deploy on board to evaluate on-device performance $\boldsymbol{Y}_{init}$;
$\boldsymbol{Y} \leftarrow \boldsymbol{Y}_{init}, \boldsymbol{X} \leftarrow \boldsymbol{X}_{init}$;
`//Remove initial samples from candidates`
$\boldsymbol{V} \leftarrow \boldsymbol{V} \backslash \boldsymbol{X}_{init}$;
`//Multi-objective BO iterations`
for $i = 1 \leftarrow N$ **do**
    Fit DKL-GP on $\boldsymbol{Y}$ and $\boldsymbol{X}$;
    $\boldsymbol{x}^* \leftarrow \text{argmax}_{\boldsymbol{x} \in \boldsymbol{V}} EPVI(\boldsymbol{x}|\boldsymbol{V})$;
    on-device performance $\boldsymbol{y}^* \leftarrow$ Auto-deploy&tuning
    `// Add the new sample from candidates`
    $\boldsymbol{X} \leftarrow \boldsymbol{X} \cup \boldsymbol{x}^*, \boldsymbol{Y} \leftarrow \boldsymbol{Y} \cup \boldsymbol{y}^*$;
    $\boldsymbol{V} \leftarrow \boldsymbol{V} \backslash \boldsymbol{x}^*$;
end for
Construct Pareto frontier and select Pareto optimal $\boldsymbol{X}_{optim}$ from $\boldsymbol{X}$;
Return $\boldsymbol{X}_{optim}$;

1000, 500, 50, and 48, respectively. Prior to calculating the covariance in DKL-GP, each candidate $\boldsymbol{x}$ is embedded as a feature using the aforementioned multilayer perceptron model.

4.2.6 Automatic Deployment and Tuning

During each iteration of the Bayesian optimization (BO) process, a data sample $\boldsymbol{x} = (\boldsymbol{b}, \boldsymbol{hp})$ is selected from a large pool of candidate samples. However, the weight $\boldsymbol{w}$ and scaling factor $\boldsymbol{s}$ remain at their initial values. At each iteration, BAQE adjusts the continuous variables $w_0, w_1, ..., w_l$ and $s_0, s_1, ..., s_l$. To obtain accurate performance feedback, BAQE incorporates automatic deployment and tuning at each sampling stage to update $\boldsymbol{w}$ and $\boldsymbol{s}$. We avoid fine-tuning with simulated quantization as it often introduces significant accuracy bias, especially in deeper networks. Instead, BAQE strives for genuine evaluation to ensure reliable performance updates.

Our framework is not limited to Post-Training Quantization (PTQ) or Quantization-Aware Training (QAT). However, for the sake of fair comparison, we adopt the QAT scheme used in the baseline approach. During the tuning process, only a small batch of calibration data, which amounts to 0.01%

of the ImageNet dataset, is sufficient. The fine-tuning stage does not require extensive training. Our main objective is to obtain an indication of on-device performance as feedback, rather than achieving a highly trained model. Furthermore, we have the flexibility to further optimize the backend-level parameters during the on-device compilation and execution. Lastly, in order to streamline the optimization process for a given hardware backend, BAQE excludes certain combinations of $(\boldsymbol{b}, \boldsymbol{s}, \boldsymbol{w}, \boldsymbol{hp})$ that have already been explored.

4.3 Experiments

Platforms. We assess the performance of the benchmark models on three distinct platforms. The first platform is the NVIDIA Jetson Orin NX system-on-module (SoM) edge device, equipped with 16 GB LPDDR5 RAM. This device is based on NVIDIA's Ampere architecture, comprising 1024 CUDA cores and 32 tensor cores. It features an 8-core ARM Cortex-A78E CPU and offers three power modes: 10W, 15W, and 20W. To ensure a fair comparison, all experiments here are conducted using the 15W mode. The second platform is a PC device powered by a 14-core Intel i7-12700H CPU and 64GB RAM. Additionally, the PC is equipped with an NVIDIA RTX3070 GPU drawing 130W of power. This GPU, also built on the Ampere architecture, possesses 5160 CUDA cores and 184 tensor cores, along with 8GB GDDR6 RAM. The final platform is a server platform featuring a 10-core Intel Xeon(R) 4210R CPU and 128GB RAM. It is equipped with an NVIDIA RTX 3090 GPU consuming 350W of power. This GPU, also based on the Ampere architecture, encompasses 10496 CUDA cores and 328 tensor cores, along with 24 GB RAM. In our implementation, we have chosen three platforms with GPUs based on the same architecture. This ensures that they possess the same computational units, such as CUDA cores and tensor cores, while differing in quantity. This approach enables consistent support for the same bit-width types and facilitates handling of identical bit-width computations, effectively eliminating other environmental variances.

Dataset. The ImageNet-1K dataset serves as the dataset for both training and evaluation in the classification task. To ensure a fair evaluation of accuracy, we employ all 50,000 images from the validation set to calculate the Top-1 score. During the optimization process, auto-tuning is

performed on the model weight and scaling factor. For the Resnet-18 model, we utilize only 0.01% of the training set, while for the Resnet-50 model, we use 0.05% of the training set. For the final evaluation of quantized performance, we utilize the entire ImageNet-1K evaluation set.

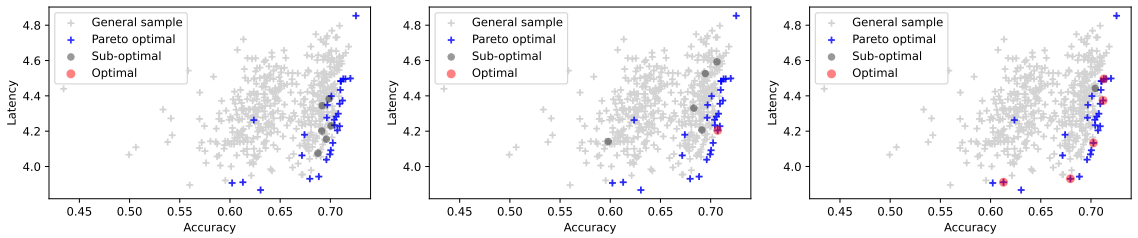
QAT/PTQ. In our practical implementation, we opted for Quantization-Aware Training (QAT) while making only minor adjustments to the model. For the QAT process, we utilized a small subset of the ImageNet dataset, specifically 0.01% for Resnet-18 (and 0.05% for Resnet-50). We trained the models with a batch size of 128 and a learning rate of 0.0001 for a single epoch. It is important to note that the QAT process is decoupled from the exploration stage. The small subset of the dataset used during the QAT process can also serve as a calibration set for Post-Training Quantization (PTQ) if desired.

Implementation Details. To establish our benchmark, we have chosen the widely used Resnet-18 and Resnet-50 models. Our BAQE framework is constructed using BOTorch and TVM (Chen et al., 2018a). We have extended the TVM (Chen et al., 2018a) framework to incorporate support for the third-generation Tensor Cores and a wider range of bit-width options (Int4, Int8, FP32, Int32), enabling practical mixed-precision quantization. In our implementation, the precision of each layer's weight and activation values is consistently maintained to align with the genuine multiplication and addition calculations performed by the underlying hardware.

4.3.1 Search Strategy Analysis

We assess the search efficiency of BAQE in optimizing both bit-width and backend configurations through visual and numerical comparisons. We have recorded the results of the initial thirty sampling trials. Our baseline methods encompass random search within our unified search space and Simulated Annealing (SA) with XGBoost serving as the performance prediction model, which aligns with the strategy employed by TVM (Chen et al., 2018a).

Visual Analysis. To provide a visual demonstration, we have selected the highly representative Jetson Orin NX as the backend and evaluated the search process using Resnet-18. We have recorded the results of the initial thirty sampling trials. For clear visualization, we have utilized a two-dimensional coordinate system with latency and Top-1 score. As depicted in Figure 4.7, our



(a) Random Search (b) SA + XGBoost (c) Ours

Figure 4.7: Comparison of performance of searched samples. Blue dots denote all Pareto optimal points, which is a projection from $(Accuracy, Latency, Size)$ to $(Accuracy, Latency)$. Red dots are found in Pareto points by searching. Gray dots denote searched sub-optimal points that are not on the Pareto frontier. Our BAQE found the most optimal points within the same search time.

BAQE search strategy successfully identifies the most optimal bit-width and backend configurations, delivering a balance between inference speed and accuracy. Moreover, even the remaining sub-optimal points are situated near the frontier, indicating that BAQE exhibits superior search quality.

Numerical Analysis. In order to evaluate both the search quality and efficiency, we have conducted numerical comparisons. To demonstrate the adaptiveness and generality of BAQE, we have assessed all three backends and two benchmark models, as presented in Table 4.1 and Table 4.2. In terms of efficiency, we have collected the normalized search time required to obtain an equivalent number of optimal points. For quality evaluation, we have determined the number of Pareto optimal points within the thirty trials. Additionally, we have calculated the average distance to the reference set (ADRS) as another metric. ADRS provides an indication of the extent to which the sample performance vector $\boldsymbol{Y}$ deviates from the Pareto frontier points $\mathcal{P}$:

$$ADRS(\boldsymbol{Y}, \mathcal{P}) = \frac{1}{|\boldsymbol{Y}|} \sum_{p \in \boldsymbol{Y}} \min_{p' \in \mathcal{P}} D(p, p'), \tag{4.16}$$

where D denotes Euclidean distance in the performance space. Results in Table 4.1 and Table 4.2 show that our method is superior in efficiency and quality.

Search Time Cost. BAQE has a fixed search time budget of 40 trials (10 from inital TED and 30 from BO process). In comparison, conventional baselines takes ILP + collecting real latency of each layer/bit pair, of which time complexity is $\mathcal{O}(k^n)$. BAQE holds the search time advantage.

Table 4.1: Normalized Searching Results on Resnet-18.

Backend	Search strategy	Normalized ADRS	Normalized search time	Optimal points
Jetson NX Orin (Edge device)	Random Search	1.061	2.533	0
	SA+XGBoost	1.038	1.533	1
	BO+DKLGP (Ours)	**1.000**	**1.000**	**5**
RTX 3070 (PC)	Random Search	1.113	1.941	0
	SA+XGBoost	1.197	1.471	3
	BO+DKLGP (Ours)	**1.000**	**1.000**	**6**
RTX 3090 (Server)	Random Search	1.452	2.333	1
	SA+XGBoost	1.806	1.533	1
	BO+DKLGP (Ours)	**1.000**	**1.000**	**4**

Table 4.2: Normalized Searching Results on Resnet-50.

Backend	Search strategy	Normalized ADRS	Normalized Time	Optimal points
Jetson NX Orin (Edge device)	Random Search	1.306	2.214	0
	SA+XGBoost	1.158	2.357	1
	BO+DKLGP (Ours)	**1.000**	**1.000**	**3**
RTX 3070 (PC)	Random Search	1.695	3.250	1
	SA+XGBoost	1.128	3.167	0
	BO+DKLGP (Ours)	**1.000**	**1.000**	**2**
RTX 3090 (Server)	Random Search	1.161	2.143	0
	SA+XGBoost	1.297	2.143	0
	BO+DKLGP (Ours)	**1.000**	**1.000**	**2**

4.3.2 Quantization Performance Analysis

We evaluate the performance of BAQE and the state-of-the-art hardware-aware quantization method, HAWQ-V3 (Yao et al., 2021), by comparing the Pareto points on all three backends in terms of on-device inference speed and accuracy, as shown in Table 4.3. HAWQ-V3 (Yao et al., 2021) utilizes a bit-width setting generated using an ILP solver. It is worth noting that our search strategy generates a group of samples that lie on the Pareto frontier, representing optimal performance trade-offs. For each benchmark and backend, we provide three samples with varying levels of accuracy, speed, and model size, demonstrating the search efficiency and capability of BAQE. To ensure a fair comparison, we assess the hypervolume of our Pareto set against HAWQ-V3 (Yao et al., 2021). In this analysis, we set the reference point as $(0, 0, 0)$ and normalize the final values. The results, as presented in Table 4.4, reveal that BAQE outperforms the baseline, achieving an improvement ranging from 6% to 96%. We shows better top-1 score and shorter inference latency, which indicates BAQE's searching efficiency and ability.

Table 4.3: Performance comparison with SOTA HAWQ-V3 (Yao et al., 2021) in model size (MB), inference latency (ms) and Top-1 score (%).

Benchmark	Method	Jetson Orin NX (Edge)			RTX 3070 (PC)			RTX 3090 (Server)		
		Model	Latency	Top-1	Model	Latency	Top-1	Model	Latency	Top-1
Resnet-18	HAWQ-V3 (Yao et al., 2021)	7.09	**3.81**	70.58	6.51	0.28	70.01	7.08	0.18	70.09
	BAQE (Ours)	**6.40**	3.93	67.96	10.88	0.27	**72.55**	8.61	0.17	**71.07**
		8.01	4.13	70.25	**6.25**	0.26	67.05	**6.62**	**0.16**	68.65
		9.51	4.50	**71.29**	6.62	**0.25**	68.64	9.37	0.17	70.86
Resnet-50	HAWQ-V3 (Yao et al., 2021)	18.81	12.60	75.48	18.54	0.68	76.27	18.57	0.37	76.02
	BAQE (Ours)	19.91	10.78	**76.38**	16.98	**0.66**	**76.45**	18.19	0.37	**76.07**
		16.56	10.73	75.72	16.55	0.70	76.28	**16.43**	0.37	75.98
		15.68	**10.40**	75.27	**15.68**	0.67	75.27	16.80	**0.35**	75.55

Table 4.4: Comparison in normalized hypervolume.

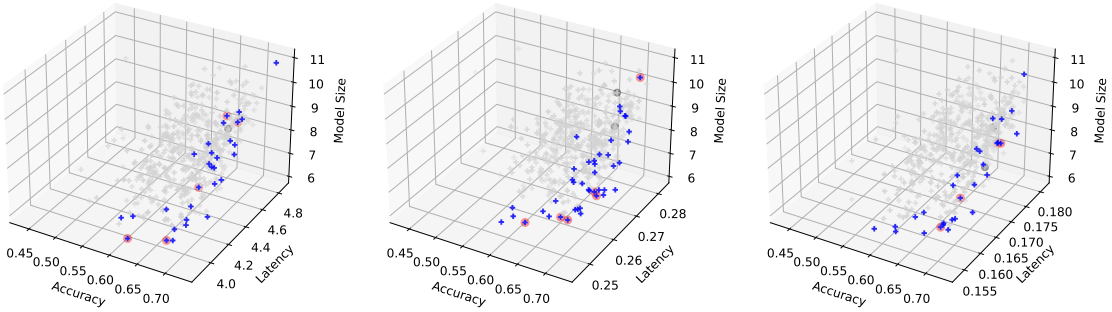
Benchmark	Method	Jetson Orin NX	RTX 3070	RTX 3090
Resnet-18	HAWQ-V3 (Yao et al., 2021)	1	1	1
	BAQE (Ours)	**1.06**	**1.08**	**1.13**
Resnet-50	HAWQ-V3 (Yao et al., 2021)	1	1	1
	BAQE (Ours)	**1.96**	**1.52**	**1.53**

Table 4.5: Example comparison of Latency with/without Loop-level optimization on Orin NX.

Model	w.o. ***hp*** Optim.	w. ***hp*** Optim.	Acce. Ratio
Resnet-18	13.63 ms	4.519 ms	$\times$ 301.6% $\uparrow$

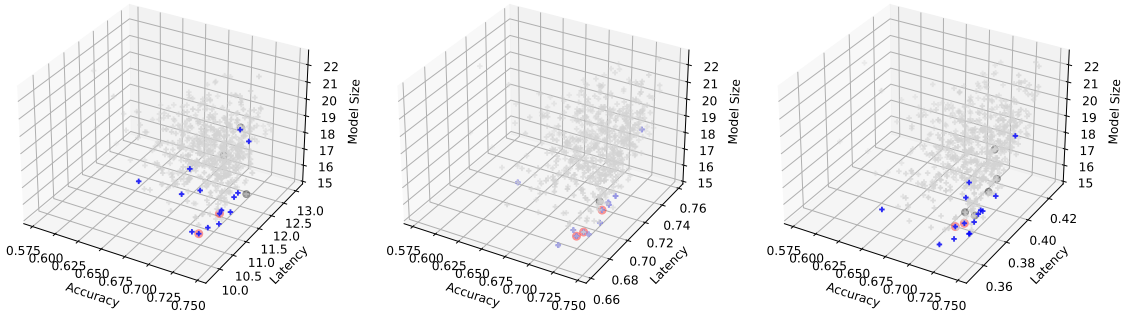
Furthermore, this result table substantiates our assumption that, given different backends, higher latency does not necessarily result in slower inference times. The interaction of backend configurations disrupts the linear relationship between these metrics. Hence, it is evident that indirect metrics such as the number of parameters and FLOPs/BOPs are not reliable indicators. On-device evaluation, which provides ultimate feedback in the form of on-device latency and accuracy, accurately reflects real-world performance.

Ablation on *hp* Optimization. To further validate our assertion that optimizing hardware parameters, in addition to bit-width, is crucial during quantization, we conduct an ablation study with and without loop-level optimization, as presented in Table 5.3. This analysis aims to provide justification for our motivation and demonstrate the significance of hardware parameter optimization.



(a) Jetson Orin NX (b) PC + RTX 2030 (c) Server + RTX 3090

Figure 4.8: 3-D performance distribution of all $(\boldsymbol{b}, \boldsymbol{hp})$ on three different backends with benchmark model Resnet-18.



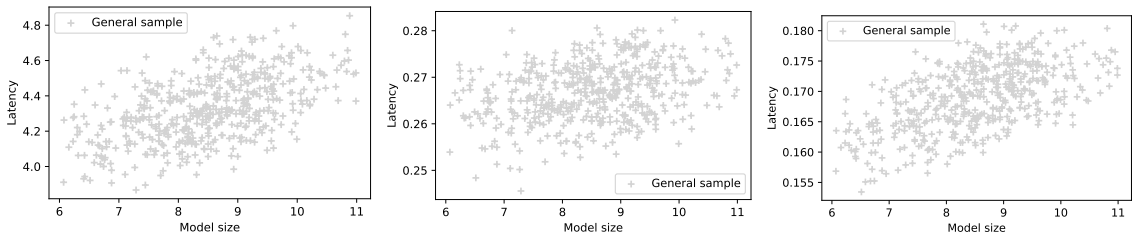
(a) Jetson Orin NX (b) PC + RTX 2030 (c) Server + RTX 3090

Figure 4.9: 3-D performance distribution of all $(\boldsymbol{b}, \boldsymbol{hp})$ on three different backends with benchmark model Resnet-50.

4.3.3 Pareto Optimality Analysis

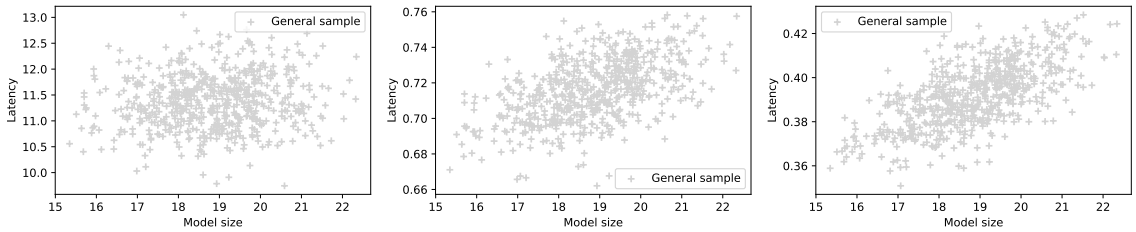
When visualizing the performance in a 2-D projection on the $(Accuracy, Latency)$ space, we can observe the distribution of performance as well as the searched optimal samples with Pareto optimal performance. For a comprehensive understanding, the original 3-D performance distribution in the search space is visualized in Figure 4.8 and Figure 4.9. The notations used in the main paper remain consistent here, where Blue dots represent all Pareto optimal points, Red dots indicate Pareto points discovered through the search process, and Gray dots denote sub-optimal points that were searched but do not lie on the Pareto frontier.

It is important to note that we have emphasized in both the introduction and experiments that



(a) Jetson Orin NX (b) PC + RTX 2030 (c) Server + RTX 3090

Figure 4.10: Relation of latency and model size for all $(\boldsymbol{b}, \boldsymbol{hp})$ on three different backends with benchmark model Resnet-18.



(a) Jetson Orin NX (b) PC + RTX 2030 (c) Server + RTX 3090

Figure 4.11: Relation of latency and model size for all $(\boldsymbol{b}, \boldsymbol{hp})$ on three different backends with benchmark model Resnet-50.

model size alone does not guarantee an accurate indication of performance; it is merely a metric related to memory consumption. On-device performance is influenced by various factors, particularly when hardware resources are limited. In order to illustrate this, we plot the relationship between latency and model size in Figure 4.10 and Figure 4.11. From these plots, we can observe a slight positive correlation between latency and model size, but with significant variance. As we transition from server to edge devices like Jetson NX, this positive correlation diminishes. This finding aligns with our claim that using model size as an indirect indicator of speed or latency is not reliable. On-device evaluation is indispensable for accurate performance assessment.

4.4 Summary

In this work, we discuss the several challenges of the current quantization methodology in a real deployment scenario. Decoupling quantization and deployment may cause some search space unexplored. In addition, we also stress the inevitability of on-device evaluation because of the factor of

backend configuration. To mitigate the problems, we propose a backend-adaptive DNN deployment framework to realize synchronous algorithm-level and backend-level optimization as a thorough solution for quantization deployment. We unify the model-level and backend-level search space and design a multi-objective search strategy to efficiently find the optimal set of bit-width settings and backend configurations. Experiments not only verify our proposition but also demonstrate the efficiency and effectiveness of our framework.

Chapter 5

AI-Accelerated Adaptive Mask Optimization Framework

The continuous shrinking of VLSI technology nodes has led to a significant impact on the manufacturability of integrated circuits. This is due to the non-negligible lithography proximity effect (Pan et al., 2013), which can cause issues during the printing process. Resolution enhancement techniques (RETs) are employed to address this challenge and improve the printability of the lithography process. One of the most widely used RETs is optical proximity correction (OPC), which optimizes mask printability by compensating for the diffraction effect that occurs during the lithography process.

As Figure 5.1 suggests the diverse patterns in real design where machine learning may fail at corner cases. Furthermore, upon conducting a more detailed examination of each sub-region, we have observed a certain degree of similarity in the pattern distribution across different sub-regions. Numerous patterns are recurrently positioned throughout the entire design layer, featuring comparable geometric shapes but varying locations. This pattern repetition allows us to exploit their shared geometric characteristics, leading to the idea that the OPC solution for one pattern can be applied to similar patterns, thereby enhancing efficiency. Inspired by these findings, we introduce a self-adaptive framework called AdaOPC, specifically designed for performing OPC on real designs.

Firstly, AdaOPC incorporates pattern analysis capabilities, enabling the classification of sub-

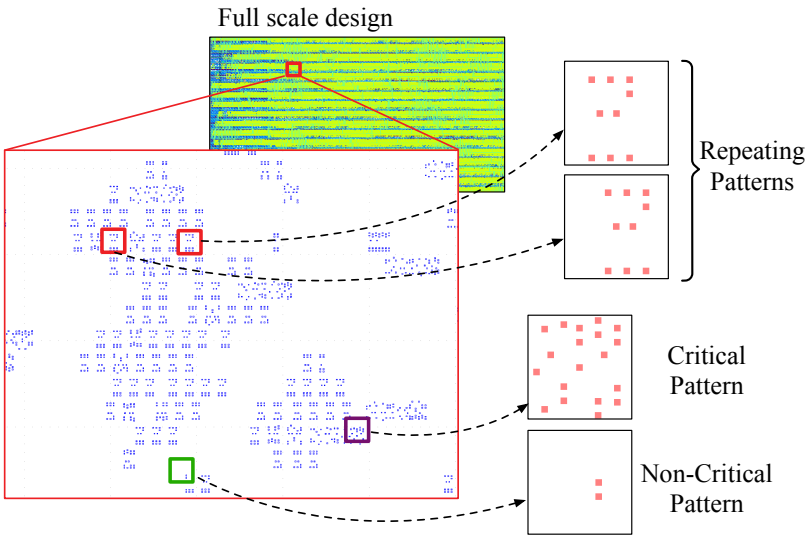


Figure 5.1: Visualization of a real design layer. Two key observations served as the motivation for our OPC framework design. Firstly, patterns were found to be distributed unevenly throughout the design layout, exhibiting varying levels of complexity. We classified intricate patterns as critical, while simple patterns were labeled as non-critical. Secondly, a significant proportion of the patterns displayed a high degree of repetition across the entire layout.

regions as either critical or non-critical. This classification allows for the appropriate selection of the OPC solver. In particular, densely scattered sub-regions exhibit not only the diffraction effect but also the optical interference caused by neighboring components, both of which collectively impact the final printed image. These intricate patterns, requiring robust and rigorous numerical optimization methods, are considered critical and are better suited for achieving higher manufacturability. On the other hand, sub-regions with sparsely scattered patterns are simpler, making them more amenable to mask optimization processes utilizing machine learning models, which offer superior inference speed through learned representations.

Secondly, considering the presence of numerous recurring patterns on the design layer, all sharing the same geometric shape as depicted in Figure 5.1, we explore the possibility of reusing optimized masks for these repetitive patterns to eliminate redundant OPC iterations. However, our idea faces three significant obstacles:

1. Slicing the large design layout into smaller patterns, as illustrated in Figure 5.2, unavoidably introduces a shift in the location of patterns with identical geometric shapes. The question

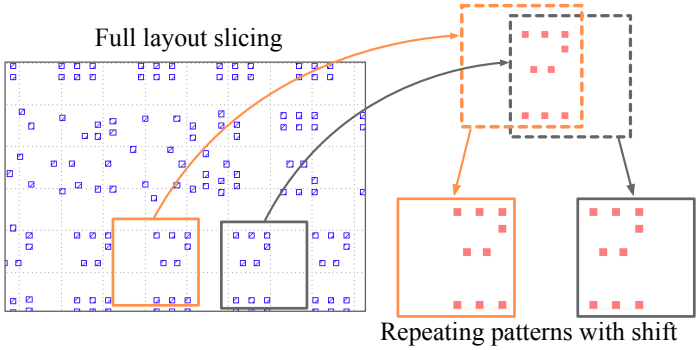


Figure 5.2: Slicing repeating full layout inevitably causes some location shift on repeating patterns.

arises: Can an optimized mask with a location shift be effectively reused? And if so, how?

2. How can we accurately and efficiently match a given query pattern with the corresponding pattern from a vast repository of stored patterns?

3. How to measure the geometric similarity of patterns with location shift?

To address the aforementioned questions, we have developed a **dynamic pattern library** with online updating capabilities, enabling us to store and reuse both repeating patterns and optimized masks. This is achieved through the construction of a dynamic hierarchical graph. Furthermore, we have mathematically demonstrated the shift equivariance property of the lithography process, affirming the feasibility of mask reuse. By accurately calculating the shift of the design pattern and calibrating the mask accordingly, we ensure the effective reuse of masks despite pattern location shifts. We have implemented a graph-based approximation method for efficient pattern matching, enabling us to quickly identify the nearest neighbors within a short query time. We summarize the contributions of this section as follows:

- We propose a self-adaptive workflow that allows for flexible selection of OPC solvers.
- We prove the feasibility of mask resue to speed up the OPC process for real design patterns and provide an efficient mask shift calibration method in practice.
- We generate design patterns embedding by supervised contrastive learning for similarity measurement and pattern matching.

- We construct a dynamic pattern library using a hierarchical graph with online update along with a greedy graph-based nearest neighbor search for fast matching.
- We bring a new weighting strategy during last modification stage to handle the sizing problem.
- With experiments on different pattern cases from a real design layout, we proved our framework can reduce over 90% runtime while still preserving the optimal OPC performance.

5.1 Preliminaries

5.1.1 Lithography Simulation Model

The lithography process begins with projecting an input mask $\mathbf{M} \in \mathbb{R}^{h \times w}$ onto a wafer plane through a series of optical lenses. This projection results in an aerial image, represented as $\mathbf{I} \in \mathbb{R}^{h \times w}$. The aerial image is subsequently utilized to form a coating on the wafer using photoresist, which ultimately creates the desired final pattern $\mathbf{Z} \in \mathbb{R}^{h \times w}$. Traditionally, simulating the lithography process involves two sequential stages: first, the optical projection model, followed by the photoresist model.

The projection process in coherent imaging systems has been widely studied using the Hopkins diffraction model (Hopkins, 1951) for mathematical analysis. Nevertheless, a different method employing singular value decomposition (SVD) has emerged as a favored alternative because of its computational efficiency. Originally introduced by (Cobb, 1998), this SVD-based approximation breaks down the Hopkins diffraction model into a sum of coherent systems via eigenvalue decomposition:

$$\mathbf{I}(x,y) = \sum_{k=1}^{N^2} w_k |\mathbf{M}(x,y) \otimes h_k(x,y)|^2, \quad x,y = 1,2,...N \tag{5.1}$$

In this equation, k represents the kernel index within the coherent system, while h_k denotes the parameters of the k-th kernel itself. The weight assigned to the k-th kernel is symbolized by w_k. An approximation of order K is demonstrated in (Gao et al., 2014) as follows:

$$\mathbf{I}(x,y) \approx \sum_{k=1}^{K} w_k |\mathbf{M}(x,y) \otimes h_k(x,y)|^2, \tag{5.2}$$

For our experiment, we chose $K = 24$ to approximate the model. After the optical simulation,

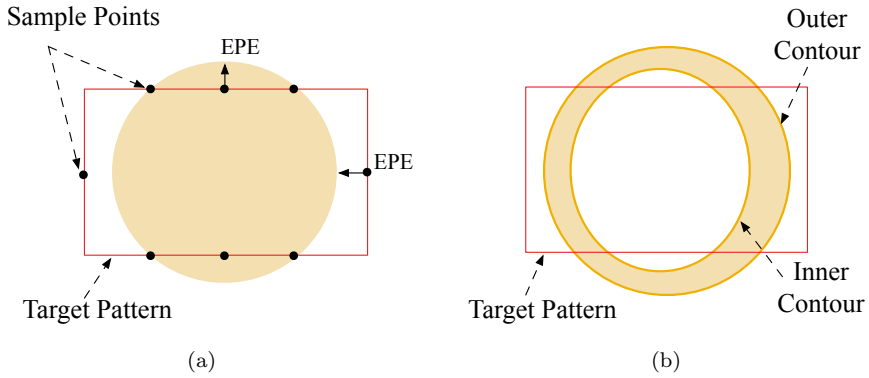


Figure 5.3: OPC evluation creteria: (a) Visualization of EPE measurement (b) Visualization of PVBand.

the resulting lithography intensity $\mathbf{I}$ is input into the photoresist model, which produces the final binary pattern $\mathbf{Z}$ based on an exposure resist threshold I_{th}:

$$\mathbf{Z}(x,y) = \begin{cases} 1, & \text{if} \quad \mathbf{I}(x,y) \geq I_{th}, \\ 0, & \text{if} \quad \mathbf{I}(x,y) < I_{th}, \end{cases} \tag{5.3}$$

Although DNN models frequently provide faster computation, we have chosen to use the Hopkins model because of its analytical properties. By employing a white box model, we can perform mathematical analysis on the pattern shift equivariance property during the lithography process.

5.1.2 OPC Evaluation Criteria

Edge Placement Error (EPE). After the lithographic process, the printed pattern on the wafer may display geometric deviations from the intended design target. These deviations are typically evaluated using the edge placement error (EPE) metric. Figure 5.3a demonstrates the EPE measurement process, where a collection of measuring points is selected along the perimeter of the target design pattern, including both horizontal and vertical edges. At each coordinate (x, y), if the distance $D(\cdot)$ between the printed image and the target surpasses a predetermined threshold th_{EPE} at

a particular measuring point, it is classified as an EPE violation:

$$\text{EPE\_violation}(x, y) = \begin{cases} 1, & D(x, y) \geq th_{EPE}, \\ 0, & D(x, y) < th_{EPE}. \end{cases} \tag{5.4}$$

Process Variation Band (PV Band). In real-world lithography processes, variations in the manufacturing conditions can lead to inconsistencies between the intended and the actual printed patterns. These discrepancies may cause printing errors, as the final printed images can exhibit a range of contour variations depending on factors such as the focus or defocus depth and the intensity of the incident light. To quantify the resilience of the printing process against variations, the concept of the Process Variation Band (PV Band) is introduced. The PV Band represents the region of dissimilarity (XOR) between the innermost and outermost contours of the printed patterns, as illustrated in Figure 5.3b. This metric provides a means to assess the influence of process variations on the quality of the printed output.

$$\text{PVBand} = \sum_{x,y}^{N^2} |\mathbf{Z}_{out} - \mathbf{Z}_{in}|, \tag{5.5}$$

In the equation above, N represents the dimensions of the pattern. The printed pattern corresponding to the outer contour is denoted by $\mathbf{Z}out$, while $\mathbf{Z}in$ represents the inner contour.

5.2 Adaptive Framework

The workflow we propose is illustrated in Figure 5.4. Initially, as described in Section 5.2.1, we present the OPC solver selection module, which is responsible for determining the most suitable OPC solver based on the characteristics of the input patterns. Subsequently, Section 5.3.2 explores the utilization of supervised contrastive learning to transform patterns into high-dimensional vector representations, enabling efficient pattern matching within the library. Lastly, Section 5.4 addresses the practicality of our approach by examining mask reusability and its associated requirements, demonstrating the property of shift equivariance during the lithography process. Furthermore, we propose a solution that employs shift calibration to address this aspect.

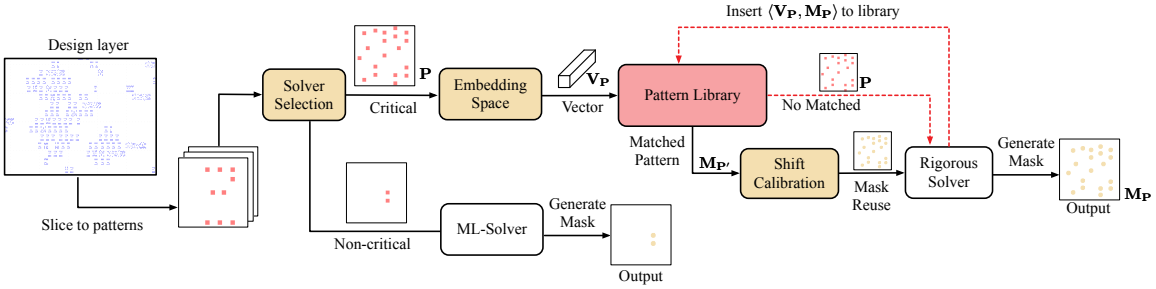


Figure 5.4: The overall workflow of AdaOPC is depicted using colored blocks to represent functional modules. The red dashed lines indicate the flow of library updates within the workflow.

5.2.1 OPC Solver Selection

In order to handle the differing levels of complexity among various patterns, our framework incorporates a versatile solver pool that selects the most appropriate OPC solutions. We divide the sliced design patterns into two distinct categories: critical and non-critical patterns. A solver selector module is employed to determine the most suitable OPC solver for each case. This solver selector can be considered a 2-class classifier developed using a straightforward deep-learning classification model. As the foundational network, we utilize ResNet-18 (He et al., 2016) and train it with the aim of minimizing the cross-entropy loss L:

$$L = -\frac{1}{N}\sum_{i}^{N} y_i \log(p_i) + (1 - y_i) \log(1 - p_i), \tag{5.6}$$

Throughout the training process, each sample i is linked to a binary label y_i, denoting its membership in the critical pattern class (1) or otherwise (0). The classifier model generates a probability p_i for each sample. The primary objective of training is to minimize the discrepancy between the predicted probabilities p_i and their corresponding labels y_i. Despite the apparent simplicity of this approach, the straightforward combination of a basic network architecture and the associated loss function proves to be remarkably effective in delivering rapid and accurate predictions for pattern classification without the need for any additional complexities or embellishments.

When it comes to the ML-Solver for handling non-critical patterns, we employ a generative neural network model that takes inspiration from DAMO-DMG (Chen et al., 2021), which previously held the title of state-of-the-art (SOTA) OPC solver for the via layer. Specifically, we utilize U-Net++

with residual connection as the foundational model architecture. Our training strategy for this generative model closely follows the approach outlined in (Chen et al., 2021). However, there is a notable difference in our training data. Rather than relying on a DNN simulator as done in (Chen et al., 2021), we curate our own training dataset by sourcing patterns from a real full-scale design. Moreover, the masks in our dataset are generated using a robust OPC engine that incorporates a genuine lithography model. This approach to data preparation more accurately reflects the real-world OPC scenario, where the lithography model is the only source of ground truth information.

For critical patterns, we opt for a rigorous optimization technique detailed in (Gao et al., 2014), leveraging GPU acceleration via CUDA and thoroughly optimized memory management. This decision is made despite the remarkable performance demonstrated by deep learning methods in comprehensive assessments on specific pattern test sets. Although data-driven black-box deep learning models exhibit proficiency in replicating and inverting diffraction phenomena, they may face difficulties when encountering optical interference stemming from intricate neighboring elements. In these situations, the rigorous numerical solver provides an analytical solution that is impervious to the geometric intricacy of patterns. Moreover, in a real-world OPC context involving a novel design and possibly a new lithography engine, patterns and optimized masks produced using robust techniques can be employed as a dataset for training the machine learning model to adapt to these fresh circumstances.

It is crucial to recognize that the solver pool is designed with extensibility in mind. This implies that any OPC solution possessing specific strengths for certain patterns has the potential to be integrated as a replacement or complementary candidate within the pool. If the pool contains more than two solvers, the classifier loss can be readily adapted as follows:

$$L = -\frac{1}{N}\sum_{i}^{N}\sum_{c=1}^{C} y_{ic}\log(p_{ic}), \tag{5.7}$$

where C signifies the number of pattern classes, which corresponds to the number of associated OPC solvers. The label y_{ic} indicates whether a pattern belongs to category c (1) or not (0). By employing this methodology, we can effectively transform the problem into a multi-classification scenario, enabling the incorporation of multiple OPC solvers within the solver pool.

5.2.2 Empirical Risk Minimization for Selector

As previously stated, patterns originating from the same design may result in an inadequate training set for robustly training the selection model, considering that the initial training is entirely supervised and the label is binary (one-hot encoded for multiple solvers). The simplistic labeling approach may lead to overfitting on the training patterns, which even exhibit similarities to the evaluation set. This appears to be unavoidable due to the repetitive nature of these patterns. However, expanding the dataset with additional designs is not feasible, necessitating the use of augmentation techniques.

For all pattern and corresponding critical label distributions $(x, y) \sim P$, our objective is to minimize the *expected empirical risk* of the trained model f:

$$R(f) = \frac{1}{n} \sum_{1}^{n} l(f(x_i), y_i), \tag{5.8}$$

It is important to note that the sample $(x_1, y_1), ..., (x_n, y_n)$ serves as an approximation to the actual distribution. Applying Gaussian noise to both the input pattern x and critical label y can help reduce this risk term.

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1 - \lambda) x_j, \\ \tilde{y} &= \lambda y_i + (1 - \lambda) y_j. \end{aligned} \tag{5.9}$$

where λ is selected from a Beta distribution with a value range of $\lambda \in [0, 1]$. Overall, such augmentation techniques can enhance the dataset distribution.

This method significantly aids the pattern selector in precisely distinguishing patterns with densities that are near either critical or non-critical ones. This ability enables the selector to make well-informed decisions based on the specific pattern density. This is a crucial motivation in the AdaOPC workflow. Our motivation extends beyond time efficiency; robustness is enhanced at this initial stage. By accurately identifying ambiguous areas, the risk of erroneously classifying critical patterns is mitigated, further optimizing overall robustness.

5.3 Dynamic Pattern Library

For patterns that are more complex and critical, once the simpler cases have been eliminated, it becomes imperative to employ a robust solver that can handle these intricacies. In order to further

optimize the process and boost efficiency, we introduce a dynamic pattern library that stores coupled patterns: the sliced pattern $\mathbf{P} \in \mathbb{R}^{h \times w}$ alongside its post-OPC mask counterpart $\mathbf{M_P} \in \mathbb{R}^{h \times w}$. By doing so, we enable the reuse of masks for patterns that appear repeatedly, effectively eliminating the need to start the time-intensive OPC iterations from the beginning for each occurrence. The core idea behind this approach is to recognize and identify stored patterns that are recurring before initiating the OPC process. To ensure the library remains up-to-date, we implement an online update mechanism that allows for the seamless insertion of newly encountered patterns and their associated masks.

Drawing inspiration from the work of (Malkov and Yashunin, 2018), we construct the pattern library using a graph-based structure. In this representation, each node within the graph symbolizes a stored pattern, while the edges connecting the nodes signify a strong similarity between the associated patterns. Considering the vast quantity of patterns present in a complete design layout, the graph can grow to a considerable size. Performing a naive search for the shortest path between nodes through exhaustive pairwise distance comparisons would be impractical and computationally prohibitive. To overcome this challenge, we introduce the following optimizations to enhance the efficiency of pattern matching:

- Sparse neighborhood graph structure: By maintaining sparse connections between nodes that are far apart, we effectively reduce the total number of edges in the graph, thereby minimizing computational complexity.

- Graph hierarchy: We organize the graph into hierarchical layers, with each layer imposing restrictions on the degree of nodes. The lower layers accommodate a higher number of edges, facilitating a greedy search approach for identifying the nearest neighbors at each layer.

For a visual representation of the hierarchical sparse graph structure, please refer to Figure 5.5.

5.3.1 Pattern Matching And Online Database Update.

The objective of identifying the pattern with the closest geometric shape is a nearest neighbor search problem (NNS). Following the approach of (Malkov and Yashunin, 2018), we utilize the Hierarchical Navigate Small World (HNSW) algorithm for efficient matching. As depicted in Figure 5.5, the

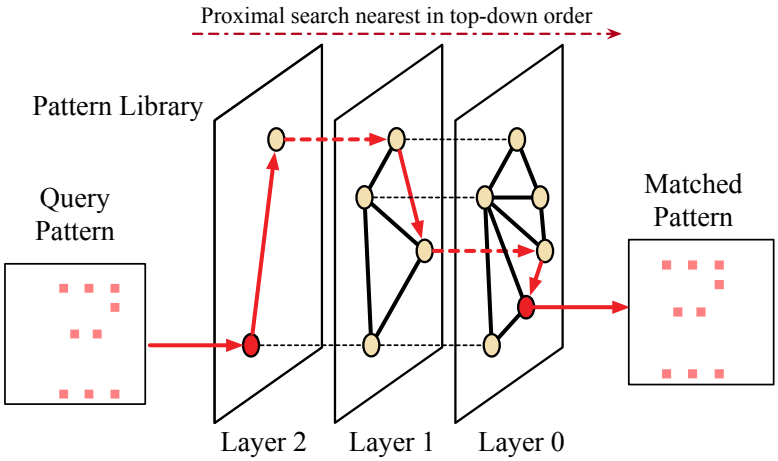


Figure 5.5: The graph-based pattern matching flow is visually represented, illustrating the traversal of the query design pattern $\mathbf{P}$ greedily traversing the hierarchical graph. The nearest node reached at layer 0 corresponds to a match pattern $\mathbf{P}'$, which has the most similar geometric shape with $\mathbf{P}$.

matching process employs a greedy approach, traversing the graph from higher layers down to the bottom layer. During this top-down traversal, a list of potential pattern nodes representing the nearest candidates is maintained. This list gets updated whenever a closer pattern is encountered, surpassing the distance of one of the existing candidates in the list. This matching strategy is based on the concept of proximity graph nearest neighbor search. For a detailed understanding of the pattern-matching search strategy at each hierarchical layer, refer to Algorithm 5. Once we reach the bottom layer, patterns in the candidate list C that have a distance smaller than the threshold σ are considered matches. If the smallest distance in C is still larger than σ, we classify it as a new pattern. This approach maintains its speed and accuracy even as the graph expands with continuous insertions of new patterns into the library.

The pattern library follows an online update method. When encountering a new pattern that has no matches, the mask undergoes OPC iterations starting from scratch for optimization. Subsequently, the library inserts the pattern and its optimized mask as a new node, updating the edge hierarchy of the graph to accommodate the new pattern. The detailed steps of the online update process are outlined in Algorithm 6.

As outlined in Algorithm 6, the insertion of a new pattern into one of the hierarchical layers is determined by a decaying probability. Within the same layer, edges are established between the

Algorithm 5 Graph-Based Pattern Matching Greedy Search

Require: Query pattern $\mathbf{P}$, starting nodes q_s, number of nearest neighbor to return k,layer number l, distance measurement $d(\cdot)$.
Ensure: Nearest pattern candidates C.

$V \leftarrow q_s$; ▷ Visited nodes
$W \leftarrow q_s$; ▷ Waiting list of nodes to visit
$C \leftarrow q_s$;
while $|W| > 0$ **do**
  $q^* \leftarrow$ nearest pattern from W to $\mathbf{P}$;
  $q_f \leftarrow$ furthest pattern from C to $\mathbf{P}$;
  if $d(\mathbf{P}, q^*) > d(\mathbf{P}, q_f)$ **then**
    break;
  end if
  for $e \in$ `neighbor`(q^*) in layer l **do**
    if $e \notin V$ **then**
      $V \leftarrow V \cup \{e\}$;
      $q_f \leftarrow$ furtherest pattern from C to $\mathbf{P}$;
      if $d(\mathbf{P}, e) < d(\mathbf{P}, q_f)$ or $|C| < k$ **then**
        $W \leftarrow W \cup \{e\}$;
        $C \leftarrow C \cup \{e\}$;
        if $|C| > k$ **then**
          Remove furthest pattern from C to $\mathbf{P}$;
        end if
      end if
    end if
  end for
end while

Algorithm 6 New Pattern Insertion and Graph Update

Require: hierarchical graph G, new pattern $\mathbf{P}$, total layer number L, G's starting nodes q_s, max degree M.
Ensure: updated hierarchical graph G.

$l \leftarrow$ `random`$(0, L)$; ▷ exponentially decaying probability
for $l_c \leftarrow L, ..l$ **do**
  $C \leftarrow$ `search`(P, q_s, k, l_c); ▷ Algorithm 5
  $q_s \leftarrow$ nearest pattern of q in C;
end for
for $l_c \leftarrow l, ..0$ **do**
  Insert $\mathbf{P}$ to layer l_c of G; ▷ add $\mathbf{P}$ into graph
  C $\leftarrow$ `search`(P, q_s, k, l_c); ▷ Algorithm 5
  `neighbors`$((P)) \leftarrow$ top M nearest patterns in C;
  for $e \leftarrow$ `neighbors`$(\mathbf{P})$ **do**
    Add edge (P, e);
    if Degree of $e > M$ **then**
      `neighbors`$(e) \leftarrow$ top k nearest patterns to e;
      Remove all edges connecting e;
      Create edges e with each one in `neighbors`(e);
    end if
  end for
end for

newly inserted pattern and the top k patterns that exhibit the closest proximity. When the degree of neighboring nodes surpasses the upper limit k due to the addition of new edges, a process of edge reconnection is triggered. As a result, the degree of each node in the graph is constrained to k. It is crucial to acknowledge that the number of edges has a direct impact on the complexity of the matching process. By leveraging a sparse hierarchical graph structure, efficient searching can be achieved even as the graph grows in size.

To quantify the similarity between vectors, we have introduced several distance metrics. One such metric involves calculating the inner product of two vectors to gauge the difference in their directional orientation:

$$\text{Inner}(\mathbf{VP}1, \mathbf{VP}2) = \mathbf{VP}1 \cdot \mathbf{VP}2 = \sum_{i=0}^{k} V_{\mathbf{P}1,i} V\mathbf{P}2, i, \tag{5.10}$$

In this equation, $\mathbf{VP}1$ and $\mathbf{VP}2$ denote the embedded vectors corresponding to patterns $\mathbf{P}1$ and $\mathbf{P}2$, respectively. $V\mathbf{P}1, i$ represents the i-th element of vector $\mathbf{VP}_1$. The dimension of the embedded vector is set to $k = 256$. It is worth noting that the inner product employed in the equation does not adhere to the positivity property, which stipulates that an element should be closer to itself than to any other element.

In contrast to the inner product, **cosine similarity** upholds the positivity property, making it a suitable measure for assessing the similarity between two vectors within an inner product space:

$$\begin{aligned} d_{\text{Cosine}}(\mathbf{V}_{\mathbf{P}_1}, \mathbf{V}_{\mathbf{P}_2}) &= 1.0 - \frac{\mathbf{V}_{\mathbf{P}_1} \cdot \mathbf{V}_{\mathbf{P}_2}}{\|\mathbf{V}_{\mathbf{P}_1}\| \, \|\mathbf{V}_{\mathbf{P}_2}\|}, \\ &= 1.0 - \frac{\sum_{i=0}^{k} V_{\mathbf{P}_1,i} V_{\mathbf{P}_2,i}}{\sqrt{\sum_{i=0}^{k} {V_{\mathbf{P}_1,i}}^2} \sqrt{\sum_{i=0}^{k} {V_{\mathbf{P}_2,i}}^2}}, \end{aligned} \tag{5.11}$$

An alternative approach is to use **Euclidean distance**, where the embedding metric space is treated as an Euclidean space. Each vector represents a position in Cartesian coordinates, and the similarity between two vectors can be determined by calculating the squared -2 norm of the difference between the coordinates.

$$d_{\text{Euclid}}(\mathbf{V}_{\mathbf{P}_1}, \mathbf{V}_{\mathbf{P}_2}) = \|\mathbf{V}_{\mathbf{P}_1} - \mathbf{V}_{\mathbf{P}_2}\|_2^2 = \sqrt{\sum_{i=0}^{k} (V_{\mathbf{P}_1,i} - V_{\mathbf{P}_2,i})^2}. \tag{5.12}$$

Any metric that follows the principles of nearest neighbor search (NNS) can be used as a feasible similarity measurement metric, allowing for the exploration of different metrics based on various embedding spaces. For our implementation, the embedding space learned with the pattern feature loss mentioned in Equation (5.7), the optimal pattern matching accuracy is achieved by Euclidean distance measurement.

In this section, we highlight the distinct motivations behind the selection and library stages, despite the fact that both stages involve a pattern embedding process. Consequently, it is essential to employ different metrics for the embedding objectives in these two stages. The Cross-entropy loss presented in Equation (5.7) facilitates the classification of all patterns into groups based on explicit pattern density. In contrast, the library stage does not assign any explicit labels to individual patterns; instead, it necessitates a hidden embedding space where the similarity between all pattern pairs can be quantified and compared. We have conducted a comparative analysis of several metrics for pattern-matching in our implementation and have determined that the Euclidean distance measurement Equation (5.12) yields the highest matching accuracy.

5.3.2 Embedding Space Construction

To utilize a stored mask from the library, it is essential to find a matching pattern with the identical geometric shape. However, the task of directly comparing the geometric similarity of two patterns is intricate. To overcome this challenge, we have devised an embedding metric space that captures the geometric properties using a high-dimensional vector representation called **VP**. Instead of storing the original $\langle \mathbf{P}, \mathbf{MP} \rangle$ pair in the library, it is substituted with the $\langle \mathbf{VP}, \mathbf{MP} \rangle$ pair. This approach allows us to determine whether two patterns are the same by employing a similarity metric to compare the embedded vectors.

The construction of the embedding space is accomplished by transforming it into a feature extraction procedure utilizing a deep learning model. Subsequently, the metric space is established through deep metric learning, where the embedded vector serves as the output of an embedding neural network. The embedding process comprises two modules:

- The first module is the Encoder, represented as $Enc(\cdot)$, which encodes each input pattern $\mathbf{P}$ into a feature map $\mathbf{F}_P \in \mathbb{R}^{h \times w \times c}$. The feature map has spatial dimensions of h and w, while

c denotes the number of channels.

- The second module is the Projector, denoted as $Proj(\cdot)$, responsible for embedding the feature map $\mathbf{F}P$ into a representation vector $\mathbf{VP} \in \mathbb{R}^k$. During the training stage, the output $Proj(\mathbf{F}P)$ is normalized to reside on the unit hypersphere in $\mathbb{R}^k$ to facilitate loss calculation.

Consequently, the formulation of the embedding process is as follows:

$$\mathbf{V_P} = Proj(Enc(\mathbf{P})) \in \mathbb{R}^k. \tag{5.13}$$

In prior deep learning-based OPC methods (Yang et al., 2019; Ye et al., 2019; Chen et al., 2021), the backbone structure commonly employed was UNet or its variant UNet++. Nevertheless, the OPC problem entails a stringent condition that necessitates the output mask to maintain the precise resolution of the input design. This constraint considerably limits the available choices for network backbone structures.

By incorporating an embedding process that eliminates these constraints, we gain the flexibility to choose from a range of network structure candidates. In our approach, we intentionally select ResNet-18 (He et al., 2016) as the encoder, which is widely recognized and extensively used. The input pattern $\mathbf{P}$ corresponds to a 2D image with dimensions of 2048×2048. To address the computational burden and minimize time delays, we employ a greedy downsampling strategy, reducing the pattern size to 256x256 before feeding it into the neural network. This downsampling technique has negligible impact on performance and does not noticeably degrade the results.

In addition to the original ResNet-18 structure, we introduce a depthwise convolution layer to decrease the feature channel size from 512 to 256. Towards the end of the neural network, a linear layer is applied to convert the resulting 3D feature into the final 1D embedded vector, denoted as $\mathbf{VP}$. The size of $\mathbf{VP}$ represents a trade-off, where a larger size indicates improved matching accuracy but slower computation and matching speed. Based on extensive experimentation, we have determined that a size of 256 strikes a balance between excellent performance and minimal matching time.

The embedding space $\mathbf{S}$ is specially designed with certain objectives:

- Patterns with the same shape exhibit similar embedded vectors with the shortest distance between them.

- Patterns with different shapes are sparsely clustered in the embedding space and located far apart.

In order to train the embedding space effectively, it is necessary to have an ample amount of data that encompasses various patterns and instances of the same pattern. During the training process, data belonging to the same pattern is considered positive samples, and the embedded vectors are encouraged to be closely aligned, indicating a higher degree of similarity. Conversely, embedded vectors representing different patterns are treated as negative samples and are pushed apart as much as possible. To construct the training dataset, we extract numerous patches of patterns from an actual full-scale design. This dataset offers a wide range of pattern samples that enable effective training of the embedding space.

Data Preparation. The cropping process involves two distinct steps aimed at generating the requisite positive and negative samples for training purposes. In the initial step, anchor points along the design layer are randomly chosen. These anchor points serve as reference points for subsequent operations. In the subsequent step, random shifts are applied around each anchor point, resulting in a set number of patches. These patches exhibit the same pattern within a square region but possess varying relative positions. By labeling the patterns according to their respective anchor points, we ensure that each batch of training data adheres to the criteria of containing both positive and negative samples.

Supervised Contrastive Loss. To generate the necessary positive and negative samples for training, the cropping process incorporates two distinct steps. When training the neural network to acquire the ability to embed patterns into representative vectors, traditional cross-entropy loss might not adequately capture inter-class distances or effectively handle noisy labels. In the domain of self-supervised learning, a family of losses based on metric distance learning exists, as demonstrated by previous works such as (Hadsell et al., 2006; Wu et al., 2018b; Pei et al., 2023a; Hjelm et al., 2018). Among these, Contrastive loss (Chen et al., 2020) has proven to be particularly potent in learning representative embeddings. Motivated by the work of (Khosla et al., 2020), we extend the contrastive loss to a supervised contrastive loss. This extension involves genuinely generating and labeling all positive and negative samples during the data preparation stage, ensuring that the labels

accurately reflect the inherent nature of the samples.

$$\mathbf{z} = \text{normalize}(Proj(Enc(\mathbf{P}))) \in \mathbb{R}^k, \tag{5.14}$$

Then the loss function is formulated as:

$$\mathcal{L}_{supCon} = -\sum_{i \in I} \frac{1}{|J(i)|} \sum_{j \in J(i)} \log \frac{\exp(\mathbf{z}_i \cdot \mathbf{z}_j / \tau)}{\sum_{a \in A(i)} \exp(\mathbf{z}_i \cdot \mathbf{z}_a / \tau)}, \tag{5.15}$$

Within the context of the training batch, we denote i as the anchor index and j as the anchor index of positive samples. The set $A(i) = I \backslash i$ encompasses all anchor indices in this batch, excluding i. Consequently, $A(i) \backslash J(i)$ represents the anchor indices corresponding to negative samples. Here, τ denotes a scalar temperature parameter. The term $\exp(\mathbf{z}i \cdot \mathbf{z}j/\tau)$ in the numerator signifies the similarity between the positive sample pairs $\mathbf{z}i$ and $\mathbf{z}j$. Conversely, the term $\exp(\mathbf{z}i \cdot \mathbf{z}a/\tau)$ in the denominator represents the similarity between all sample pairs, including the negative ones. Through the minimization of the loss, the training process aims to enhance the similarity among positive samples while diminishing the similarity among negative samples.

5.4 Mask Reuse With Shift Calibration

5.4.1 Mask Reusability

We proceed under the assumption that if the query design pattern $\mathbf{P}$ matches a stored design pattern $\mathbf{P}'$ in the library, both exhibiting the same shape, it is possible for repeating patterns to efficiently share masks, leading to enhanced efficiency. However, as demonstrated in Figure 5.2, when the entire design is partitioned into smaller patterns, encountering pattern location shifts $(\Delta x, \Delta y)$ between $\mathbf{P}$ and $\mathbf{P}'$ becomes inevitable. In real lithography and OPC flow scenarios, where no external factors influence the lithography process, the printed wafer image patch should not undergo any geometric distortion. Instead, it will only exhibit an identical shift in comparison to the design pattern, as depicted in Figure Figure 5.6. Thus, to facilitate mask reuse, the primary requirement is to ensure that the location shift during lithography does not result in any geometric distortions.

To demonstrate the feasibility of the mask shift calibration approach, we provide mathematical

proof that the location shift remains unchanged before and after the lithography process. We represent the Hopkins diffraction model during lithography, as shown in Section 5.1.1, as Litho$(\cdot)$. We denote the location shift as $\delta_{\Delta x,\Delta y}(\cdot)$. Through our mathematical analysis, we establish the following result:

Theorem 1 (Shift Equivariance)**.** *Given pattern* $\mathbf{P}$ *and mask* $\mathbf{M_P}$ *where*

$$\mathbf{P} = \mathit{Litho}(\mathbf{M_P}). \tag{5.16}$$

The following statement always holds:

$$\delta_{\Delta x,\Delta y}(\mathbf{P}) = \mathit{Litho}(\delta_{\Delta x,\Delta y}(\mathbf{M_P})). \tag{5.17}$$

Proof. For any position (x,y) on pattern $\mathbf{P}$:

$$\begin{aligned}
&\delta_{\Delta x,\Delta y}(\mathbf{P}(x,y)) = \mathbf{P}(x+\Delta x, y+\Delta y),\\
&= \sum_{k=1}^{N^2} w_k \left| h_k(x+\Delta x, y+\Delta y) \otimes \mathbf{M_P}(x+\Delta x, y+\Delta y)\right|^2,\\
&= \sum_{k=1}^{N^2} w_k | \sum_{i=1}^{N}\sum_{j=1}^{N} h_k(i,j)\mathbf{M_P}(x+\Delta x+i-\frac{N}{2}, y+\Delta y+j-\frac{N}{2})|^2,\\
&= \sum_{k=1}^{N^2} w_k | \sum_{i=1}^{N}\sum_{j=1}^{N} h_k(i,j)\mathbf{M_P}(x+i-\frac{N}{2}+\Delta x, y+j-\frac{N}{2}+\Delta y)|^2,\\
&= \sum_{k=1}^{N^2} w_k | \sum_{i=1}^{N}\sum_{j=1}^{N} h_k(i,j)\delta_{\Delta x,\Delta y}(\mathbf{M_P}(x+i-\frac{N}{2}, y+j-\frac{N}{2}))|^2,\\
&= \sum_{k=1}^{N^2} w_k \left| h_k(x,y) \otimes \delta_{\Delta x,\Delta y}(\mathbf{M_P}(x,y))\right|^2,\\
&= \mathrm{Litho}(\delta_{\Delta x,\Delta y}(\mathbf{M_P}(x,y))).
\end{aligned} \tag{5.18}$$

Therefore Equation (5.17) is proved. □

Considering that mask shift during lithography solely results in a printing shift, it becomes feasible for repeating patterns within a design to share OPC-optimized masks by applying a simple shift correction. To accomplish this, we choose the corresponding mask $\mathbf{MP'}$ from the pattern library and apply a correction of $(-\Delta x, -\Delta y)$ to derive the initial mask $\mathbf{MP}$ for pattern $\mathbf{P}$.

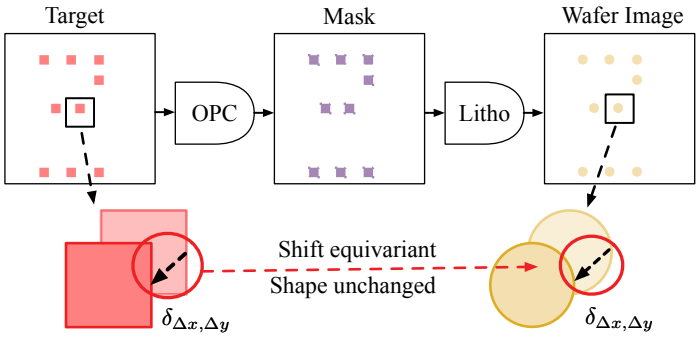


Figure 5.6: The printed wafer image must exhibit an identical location shift to the design pattern without any geometric shape distortion.

5.4.2 Pattern Shift Calibration

To determine the shift between patterns $\mathbf{P}$ and $\mathbf{P}'$, we employ pixel-level similarity analysis. Specifically, we calculate the pixel-wise **cross-correlation** between $\mathbf{P}$ and $\mathbf{P}'$. This measurement of cross-correlation serves as an indicator of the similarity between individual pixels, with the highest response value on the correlation map representing the shift in the position of the center point (x_{ctr}, y_{ctr}). However, it is essential to acknowledge that conducting cross-correlation computation on two large 2D patterns can be time-consuming. The computational process for cross-correlation is analogous to convolving $\mathbf{P}$ with the 180° rotation of $\mathbf{P}'$, denoted as $Rotate(\cdot)$:

$$CrossCorr(\mathbf{P}, \mathbf{P}') = Conv(\mathbf{P}, Rotate(\mathbf{P}')), \tag{5.19}$$

To expedite the computation process, we employ Convolution and leverage the efficiency of Fast Fourier Transform (FFT) (Vasilache et al., 2014). This approach allows us to calculate the pattern shift using the following formula:

$$\begin{aligned} x^*, y^* &= \underset{x,y}{\text{argmax}} \ \ Conv\_FFT(\mathbf{P}, Rotate(\mathbf{P}')), \\ \Delta x = x^* - x_{ctr}&, \ \ \Delta y = y^* - y_{ctr}, \end{aligned} \tag{5.20}$$

And the initial mask is corrected with:

$$\mathbf{M_P} = \delta_{-\Delta x, -\Delta y}(\mathbf{M_{P'}}). \tag{5.21}$$

In practical scenarios, we validate the calibrated mask by inputting it into the lithography model. If needed, we may further iterate once or twice using the Inverse Lithography Technology (ILT) solver to accommodate any potential noise introduced during the shift calibration process. To maintain consistency, we adopt the pattern size described in (Yang et al., 2019), which is 2048×2048. With our implementation, the calculation time for determining the shift is less than 0.25 seconds when executed on a CPU.

5.5 Post-Calibration Enhancement

Once the pre-collected mask has been matched and reused, the initial workflow of AdaOPC (Zhao et al., 2022) proceeds by directly feeding the calibrated mask into the ILT flow for additional optimization, as illustrated in Figure 5.4. This approach ensures the quality of the output mask and serves as a reliable methodology. Additionally, it is worth noting that the final result, measured in terms of EPE/PVBand, remains upper-bounded by the original ILT approach. However, we have delved into the patterns following the initial AdaOPC process and proposed a post-stage for mask correction, aiming to further enhance the quality of the mask.

AdaOPC (Zhao et al., 2022) initially focuses on via layers, which exhibit relatively simple geometric shapes but complex distributions. Through the analysis of printed aerial images and evaluation metrics at each iteration, we observe that the printed aerial images commonly exhibit a "circle" shape with smooth boundaries. In many instances, the size of the printed circle for each target is strongly influenced by neighboring components following the lithography stage. Conventional approaches in ILT, employing either pixel-based methods or level-set-based methods, primarily concentrate on addressing boundary healing to overcome disconnection or boundary merging issues, as illustrated in Figure 5.7. In the case of metal layers, irregular object patterns and geometric shapes may occasionally encounter such problems during lithography. However, it is crucial to address and resolve this sizing problem. Our evaluation of various test cases demonstrates that this is the primary cause of most EPE numbers.

As an additional stage after calibration, we introduce a "Halo-weighting" process following the matching of a pre-collected mask obtained from the pure ILT stage. In this stage, our objective is

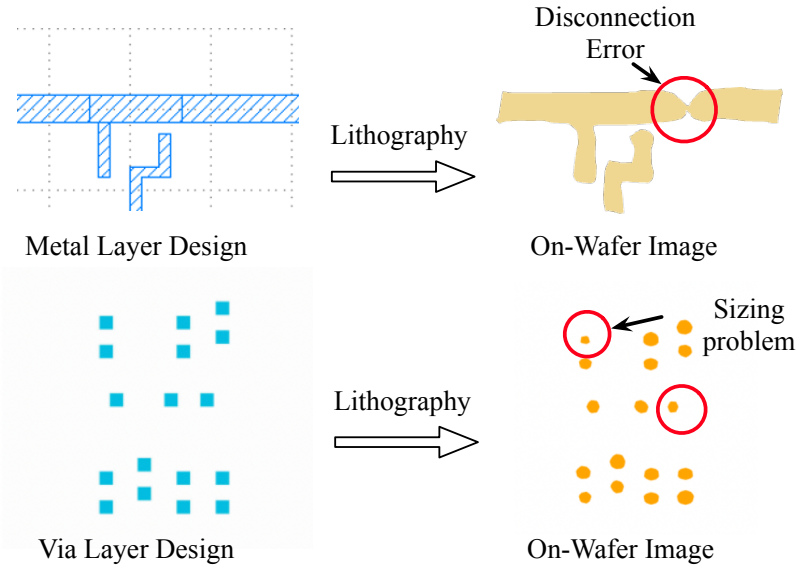


Figure 5.7: Visualization of the sizing problem in Via layer mask optimization problem, in comparison with conventional metal layer. Some printed components are too small or too big, which may cause problem during manufacturing.

to promote balanced object sizes and restore their proximity to the original via employing a filtering step on the loss matrix during the update. When the matched mask is passed to the rigorous solver, we incorporate a weighting filter $\mathbf{H} \in \mathbb{R}^{h \times w}$. This weighting filter aligns with the original design shape and assigns the highest weight to the boundary, gradually decreasing as the distance from the boundary increases. The resulting heatmap resembles a "Halo" that gradually fades away from the boundary of the via in the original design pattern, as shown in Figure 5.8.

Such weight filter is derived with two consecutive steps: the first step is boundary derivation. We apply a morphological operation on the binary design image, calculating the *Morphological Gradient* to retrieve the boundary $\mathbf{B} \in \mathbb{R}^{h \times w}$ matrix of each pattern $\mathbf{P}$:

$$\mathbf{B} = \mathbf{P} \oplus b - \mathbf{P} \ominus b, \tag{5.22}$$

where the $\mathbf{P} \oplus b$ and $\mathbf{P} \ominus b$ denote the "dilation" and "erosion" operations. $\mathbf{b}$ is a grayscale structuring

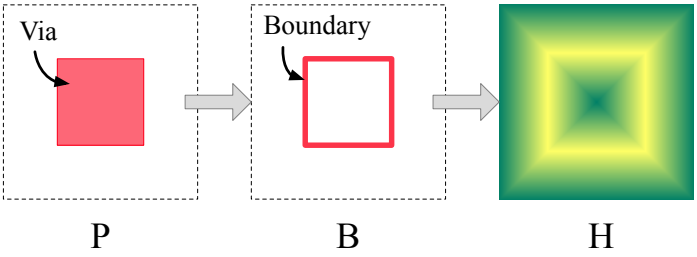


Figure 5.8: Visualized derivation of "halo" weight filter **H**. The square on the left is a patch **B** from original design. The square in the middle is the derived boundary **B** with morphological gradient from Equation (5.22). The heapmap square on the right is the "halo" weight filter with higher value on the brighter area and lower value on the greener area ($0 \sim 1$).

element used in computer graphics (here, the pattern is regarded as a 1-channel picture):

$$b(x) = \begin{cases} 0, & \text{if} \quad \|x\| \leq 1, \\ -\infty, & \text{otherwise.} \end{cases} \tag{5.23}$$

The second step is to blur the derived boundary **B** for a gradually decreasing weight. One off-the-shelf algorithm is distance transformation. The weight along the "halo" is determined by its distance from the boundary of each component derived from Equation (5.22), where the value at (x, y) is the distance to the boundary:

$$\mathbf{H}(x, y) = 1 - \text{Norm}(\text{distance}(x, y, \mathbf{B})), \tag{5.24}$$

It is guaranteed that the weight filter shows a gradual decrease as the position moves farther from the boundary. We normalize the distance to control the value range into [0, 1]. We subtract 1 with this value such that the highest value on the boundary and the value range of **H** is still [0,1]. The distance is calculated using simple Euclidean distance for faster calculation not only to attain a faster calculation but also to bring a smoother yet effective weight distribution:

$$\text{distance}(x, y, \mathbf{B}) = \max_{(x_b, y_b) \in \mathbf{B}} (\|x - x_b\|, \|y - y_b\|). \tag{5.25}$$

In our implementation, we maintain the dilation and erosion size at 1, resulting in a boundary length of 3 ($+1/-1$). The distribution of weights is depicted in Figure 5.8, illustrating a smooth

weight distribution along the "halo" region within and outside the boundary of the via component. By incorporating this filter into the loss matrix during ILT after calibration, the following effects are observed:The restoration of the boundary restricts the area in proximity to the boundary. Conversely, the accumulated loss further away from the boundary is given less weight, considering that this is a post-matched mask. The significant errors have already been addressed prior to their inclusion in the library and are no longer present at this stage.

5.6 Experimental Results.

Our framework is primarily developed using Python, with the machine learning components implemented using the PyTorch library. On the other hand, the lithography and Inverse Lithography Technology (ILT) modules are constructed using C/C++ and leverage the CUDA 11.3 toolkit for optimized performance. Our framework is mainly developed using Python. The machine learning components of our framework are implemented using the PyTorch library. On the other hand, the lithography and Inverse Lithography Technology (ILT) modules are built using C/C++ and utilize the CUDA 11.3 toolkit for optimized performance. To evaluate the performance and speed of our framework, we conducted experiments on a CentOS-7 system equipped with an Intel i7-5930K CPU running at 3.50GHz, along with an Nvidia GTX Titan X GPU. For our experiments, we employed the publicly available lithography engine from the ICCAD 2013 CAD Contest (Banerjee et al., 2013), which includes a set of 24 optical kernels. The photoresist intensity threshold was set to 0.055. We adopted a lithography wavelength of 193nm, with a defocus range spanning ±25nm and a dose range of ±2%. To identify Edge Placement Error (EPE) violations, we set the EPE violation threshold (th_{EPE}) to 15nm. Despite the original implementation of AdaOPC utilizing GPU and CUDA acceleration for the lithography process at each iteration, our analysis in Figure 5.9b reveals that this bottleneck still hampers the efficiency of the optimization. To address this limitation and enhance the ILT process compared to the original AdaOPC, we have re-implemented it to alleviate the memory constraint during inference and gradient back-propagation. Primarily, we store all intermediate tensors/matrices and kernel weights in the High Bandwidth Memory (HBM), which is the GPU memory. Within each update iteration, the matrix calculations are performed directly on the

GPU, while the utilized weights and updated matrices remain stationary in the GPU memory. This approach eliminates unnecessary CPU-GPU data transfers and synchronization. Furthermore, we have optimized our CUDA kernel at the thread level to minimize cache misses, resulting in improved efficiency. In summary, our optimizations yield a speed-up of up to 2.2× for the same amount of ILT calculations.

5.6.1 Data Preparation.

To ensure the validity of our OPC experiments, we utilized real design data extracted from a GDS file generated by the open-source layout generation tool OpenROAD (Ajayi and Blaauw, 2019). Specifically, we sliced patterns of size 2048 × 2048, following the established approach in previous research works (Yang et al., 2019; Yu et al., 2021; Chen et al., 2021). This size aligns with the supported clip size of our Lithography engine, eliminating the need for additional resize operations that may not contribute to pixel-based OPC tasks. The patterns were derived from a comprehensive via layer containing over 1.9×10^6 vias, with each pixel representing an area of $1nm^2$. It is important to note that our proposed workflow is not limited to via layer patterns. The repetition property also applies to metal layer patterns, although they have a continuity requirement. For instance, the mask of a metal component on the stitching area should exhibit continuity and smoothness. If these corner cases are addressed, the adaptive OPC flow can be extended to metal layer OPC.

For the training dataset of our ML-Solver, we randomly extracted 4000 patterns from the design layer. These patterns were accompanied by masks optimized using the ILT Solver. The same set of patterns was used to train the pattern classifier. To assign critical/non-critical labels, we directly applied lithography to these patterns and assigned labels based on the Edge Placement Error (EPE) values. This labeling approach is intuitive as it reflects the level of mask optimization difficulty. Regarding the training data for Metric Space embedding, we followed the steps outlined in Section 5.3.2. We selected 400 random anchor points and generated shifts around each anchor point within a range of ±10% of the pattern width. We ensured that the number of positive samples for each anchor point was equal to or greater than the number of anchors, ensuring a positive-to-negative ratio in each training batch. During the slicing stage, we applied padding to the mask, considering the boundary area (10% width) of the pattern as the padding area. This allowed the boundary to

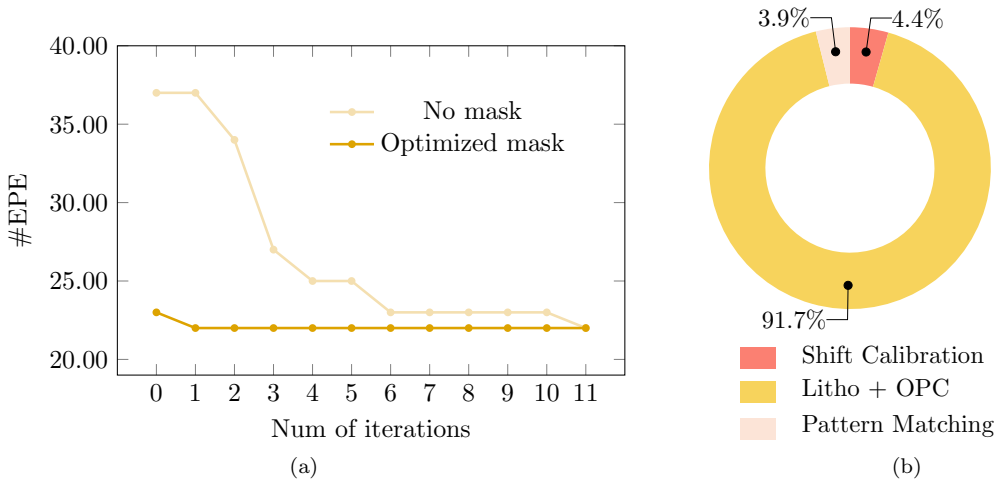


Figure 5.9: (a) EPE convergence comparison (b) Runtime breakdown of AdaOPC on critical patterns.

Library Size	128-D	256-D	512-D
100	0.9ms	1.3ms	1.5ms
500	3.4ms	5.7ms	8.8ms
1000	9.0ms	13.3ms	22.2ms
2000	20.4ms	31.2ms	52.2ms
5000	59.8ms	93.0ms	156.6ms
10000	130.1ms	206.5ms	413.6ms

Table 5.1: Pattern-matching speed Analysis on different embedding dimension.

go through the lithography engine, but only the inner area of the mask clip was updated. This approach helps mitigate proximity effect issues at the edges. During the stitching stage, we needed to overlap the mask clip onto the padding area.

5.6.2 Peformance Analysis

To validate the effectiveness of mask reuse, we initially observed the descending trend of EPE. In this regard, we conducted a demonstration experiment on a specific pattern, where we recorded the EPE descending trend during the iterations of Inverse Lithography Technology (ILT) with a calibrated optimized mask serving as the initial state. We also recorded the trend when starting the ILT process from scratch without an initial mask for comparison purposes. As depicted in Figure 5.9a, when an initial mask is utilized, the EPE number starts at an almost optimal value of 23, and the

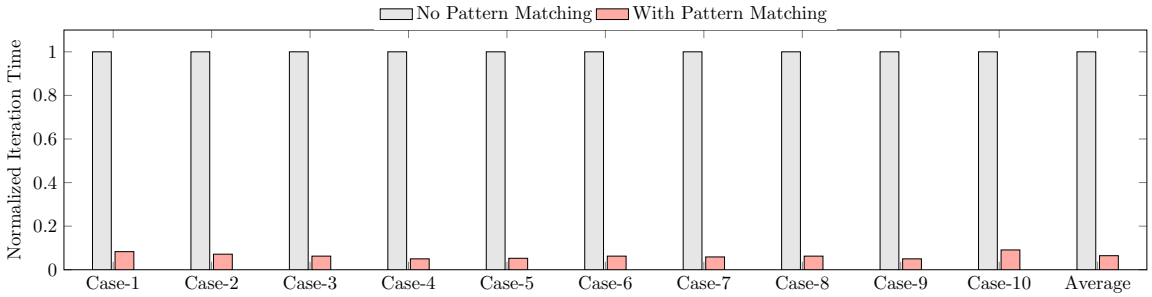


Figure 5.10: Convergence speed of mask optimization with and without Pattern Matching. In order to clearly demonstrate the acceleration ratio, we normalized the time of mask optimization without pattern matching to 1. During optimization without pattern matching, the initial state of the mask was not calibrated. In the case of optimization with pattern matching, we utilized the calibrated mask as the initial state.

Case	DAMO-DGS			ILT-GPU			AdaOPC			AdaOPC+Faiss			Ours		
ID	#EPE	PVB (nm^2)	RT (s)	#EPE	PVB (nm^2)	RT (s)	#EPE	PVB (nm^2)	RT (s)	#EPE	PVB (nm^2)	RT (s)	#EPE	PVB (nm^2)	RT (s)
1	22	23323	5.20	23	23329	41.15	22	23232	5.50	22	23232	5.35	21	23492	2.42
2	26	26729	5.26	25	26762	48.5	24	26580	5.41	24	26580	5.26	22	26670	2.52
3	27	26938	5.22	24	26720	55.92	24	26718	5.37	24	26718	5.22	24	26720	2.61
4	36	27975	5.18	29	28127	70.57	25	27934	5.40	25	27934	5.25	25	27982	2.51
5	35	28805	5.32	30	28925	66.89	30	28927	5.44	30	28927	5.29	27	29128	2.67
6	30	26960	5.31	25	26762	55.81	24	26775	5.38	24	26775	5.23	24	26615	2.75
7	33	26382	5.23	28	26453	59.47	28	26281	5.43	28	26281	5.28	26	26327	2.56
8	32	30646	5.38	25	29450	54.88	27	29341	5.42	27	29341	5.27	27	29108	2.66
9	25	24054	5.25	24	24053	70.62	23	24022	5.43	23	24022	5.28	22	24111	2.63
10	24	21939	5.29	23	21701	37.59	22	21644	5.53	22	21644	5.38	21	21679	2.75
Avg.	29.0	26375	5.26	25.6	26228	56.14	24.9	26145	5.43	24.9	**26145**	5.28	**23.9**	26183	**2.61**
Ratio	1.165	1.009	0.997	1.028	1.003	10.637	1.000	1.000	1.028	1.000	**1.000**	1.000	**0.959**	1.001	**0.495**

Table 5.2: Comparisons of baseline approaches.

descending trend converges after the first iteration. In contrast, when ILT is initiated from scratch without an initial mask, the EPE number begins at 37 and requires six iterations to reach the initial EPE number achieved through mask reuse. Overall, it takes 12 iterations for the ILT process to converge to an EPE of 22.

In order to evaluate the effectiveness of our framework, a series of runtime analysis experiments were conducted. Our framework utilizes highly efficient machine learning-based methods to handle non-critical patterns, while giving primary attention to critical patterns. For a visual breakdown of the time taken by each step in the critical pattern OPC process within AdaOPC, refer to Figure 5.9b. It is evident from the breakdown that lithography and ILT OPC iterations account for 91.7%

Selector Training	Error Rate
w.o augmen.	1.9%
w. augmen.	0.2%

Table 5.3: Ablative study on ERM augmentation.

of the total runtime. On the other hand, pattern matching and shift calibration together contribute only 8.3% to the overall process time. This indicates that the impact of these steps on the entire process is minimal. Moreover, it showcases the adaptability of our framework, allowing seamless integration of new and powerful OPC tools or litho-models to further enhance speed.

Furthermore, we investigated the scenario of an expanding pattern library. Although generating a large number of "ground-truth" masks is constrained by time and computational resources, we tested the speed of pattern matching using a substantial number of synthesized pattern vectors. As shown in Table 5.1, even when the pattern library is expanded to accommodate 10,000 patterns with a dimension of 512, the query and matching process can still be completed within 0.4 seconds. The time overhead incurred is negligible in the overall process.

To assess the convergence speed of mask updates with and without pattern matching, we conducted 10 cases after adding 800 patterns to the pattern library. The results presented in Experiment 5.10 demonstrate a significant reduction of 93.6% in the average number of iterations required for mask convergence when pattern matching is utilized.

The performance gain from the post-calibration stage is listed in the last three columns on the right in Table 5.2. We set our SOTA baseline as AdaOPC (Zhao et al., 2022) with matching algorithm updated to the latest 2024 version of Faiss (Douze et al., 2024). By employing "halo" weight filtering on the loss during the final restoration process, we were able to observe an average EPE error of 23.9. This represents a further 4% reduction compared to the results presented in the conference, where the SOTA baseline was used. An ablative study on the accuracy of solver selection was conducted, with and without data augmentation using the ERM objective. The study, depicted in Table 5.3, revealed an improvement in accuracy from 98.1% to 99.8%. Furthermore, the error rate experienced an 89% reduction. These findings highlight the effectiveness of our approach. With the integration of our newly implemented GPU-efficient lithography system and ILT process, we achieved a 52%

reduction in the overall runtime compared to the conference version. This enhancement significantly enhances the efficiency of the system.

Chapter 6

Conclusion

In this thesis, we have discuss the potential acceleration aspects of AI technology from algorithm, to hardware-level optimization, all the way to AI for hardware design automation. We illustrates the possibility of AI acceleration on Design-For-Manufactory and specific Mask Optimization. On the contrary, we also discuss the Hardware-aware AI acceleration and domain specific AI acceleration.

- In Chapter 3, we design a domain-specific high-performance acceleration framework for super-resolution deployment with a model originating from LAPAR. In our framework design, we propose a dictionary slimming strategy to extract the most informative dictionary items for efficient inference. We also designed a hardware-aware acceleration engine to fully utilize the limited hardware resources for inference optimization. Moreover, we make trials on low-bit inference with an adaptive 8-bit quantization strategy to further accelerate the process. Based on various evaluation results, our system outperforms the state-of-the-art tool TensorRT, and PyTorch on edge embedded GPU NVIDIA Jetson NX and 2080 Ti significantly, without quality degradation.

- In Chapter 4, we discuss the several challenges of the current quantization methodology in a real deployment scenario. Decoupling quantization and deployment may cause some search space unexplored. In addition, we also stress the inevitability of on-device evaluation because of the factor of backend configuration. To mitigate the problems, we propose a backend-

adaptive DNN deployment framework to realize synchronous algorithm-level and backend-level optimization as a thorough solution for quantization deployment. We unify the model-level and backend-level search space and design a multi-objective search strategy to efficiently find the optimal set of bit-width settings and backend configurations. Experiments not only verify our proposition but also demonstrate the efficiency and effectiveness of our framework.

- In Chapter 5, we present a customized self-adaptive framework for Optical Proximity Correction (OPC) that focuses on mask optimization in real designs. Our framework utilizes a comprehensive analysis of the design's characteristics to enhance the optimization process. To intelligently select an appropriate OPC solver based on pattern complexity, we introduce an extensible OPC solver selector. Moreover, we introduce a dynamic pattern library that enables the reuse of optimized masks for repeating patterns with identical geometric shapes. To improve pattern matching efficiency, we employ supervised contrastive learning to embed patterns into vectors. Additionally, we devise a graph-based search strategy to accelerate the pattern matching process. We validate the reusability of masks by demonstrating the pattern shift equivariance property. To address any potential shifts, we introduce a practical shift calibration tool. Through extensive experiments, we demonstrate that our framework achieves a co-optimization of OPC speed and robustness for real design patterns.

References

[n. d.]. Intel MKL-DNN. https://github.com/oneapi-src/oneDNN.

[n. d.]. NVIDIA TensorRT. https://docs.nvidia.com/deeplearning/tensorrt/index.html.

[n. d.]. Simple-SR. https://github.com/dvlab-research/Simple-SR.

Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. 2018. Fast, accurate, and lightweight super-resolution with cascading residual network. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 252–268.

Tutu Ajayi and David Blaauw. 2019. OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain. In *Proceedings of Government Microcircuit Applications and Critical Technology Conference*.

Yang Bai and Weiqiang Wang. 2019. ACPNet: Anchor-Center Based Person Network for Human Pose Estimation and Instance Segmentation. 1072–1077.

Yang Bai, Xufeng Yao, Qi Sun, and Bei Yu. 2021. AutoGTCO: Graph and Tensor Co-Optimize for Image Recognition with Transformers on GPU. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

Shayak Banerjee, Zhuo Li, and Sani R Nassif. 2013. ICCAD-2013 CAD contest in mask optimization and benchmark suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 271–274.

Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems* 32 (2019).

Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. 2012. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. (2012).

Kartikeya Bhardwaj, Milos Milosavljevic, Liam O'Neil, Dibakar Gope, Ramon Matas, Alex Chalfin, Naveen Suda, Lingchuan Meng, and Danny Loh. 2022. Collapsible linear blocks for super-efficient super resolution. *Proceedings of Machine Learning and Systems* 4 (2022), 529–547.

Edwin V Bonilla, Kian Chai, and Christopher Williams. 2007. Multi-task Gaussian Process Prediction. *Annual Conference on Neural Information Processing Systems (NIPS)* 20 (2007).

Shijie Cao, Chen Zhang, Zhuliang Yao, Wencong Xiao, Lanshun Nie, Dechen Zhan, Yunxin Liu, Ming Wu, and Lintao Zhang. 2019. Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity. In *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*. 63–72.

Guojin Chen, Wanli Chen, Qi Sun, Yuzhe Ma, Haoyu Yang, and Bei Yu. 2021. DAMO: Deep agile mask optimization for full chip scale. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2021).

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning (ICML)*. PMLR, 1597–1607.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018a. TVM: an automated end-to-end optimizing compiler for deep learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 579–594.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018b. TVM: An automated end-to-end optimizing compiler for deep learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 578–594.

Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018c. Learning to optimize tensor programs. *Annual Conference on Neural Information Processing Systems (NIPS)* 31 (2018).

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In *International conference on machine learning*. PMLR, 2285–2294.

John Cheng, Max Grossman, and Ty McKercher. 2014. *Professional CUDA C programming*. John Wiley & Sons.

Jae-Seok Choi and Munchurl Kim. 2017. A deep convolutional neural network with selection units for super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 154–160.

Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, and Qingyuan Li. 2021. Fast, accurate and lightweight super-resolution with neural architecture search. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 59–64.

Nicolas Bailey Cobb. 1998. *Fast optical and process proximity correction algorithms for integrated circuit manufacturing*. University of California, Berkeley.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024* (2014).

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems* 28 (2015).

Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. 2019. Second-order attention network for single image super-resolution. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 11065–11074.

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2014. Learning a deep convolutional network for image super-resolution. In *European conference on computer vision*. Springer, 184–199.

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. 2015. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 38, 2 (2015), 295–307.

Chao Dong, Chen Change Loy, and Xiaoou Tang. 2016. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*. Springer, 391–407.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *IEEE International Conference on Computer Vision (ICCV)*. 293–302.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *arXiv preprint arXiv:2401.08281* (2024).

Jhih-Rong Gao, Xiaoqing Xu, Bei Yu, and David Z Pan. 2014. MOSAIC: Mask optimizing solution with process window aware inverse correction. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

Jacob R Gardner, Geoff Pleiss, David Bindel, Kilian Q Weinberger, and Andrew Gordon Wilson. 2018. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. In *Annual Conference on Neural Information Processing Systems (NIPS)*.

Hao Geng, Wei Zhong, Haoyu Yang, Yuzhe Ma, Joydeep Mitra, and Bei Yu. 2019. SRAF insertion via supervised dictionary learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 39, 10 (2019), 2849–2859.

Pascal Getreuer, Ignacio Garcia-Dorado, John Isidoro, Sungjoon Choi, Frank Ong, and Peyman Milanfar. 2018. BLADE: Filter learning for general purpose computational photography. IEEE, 1–11.

Cong Guo, Yangjie Zhou, Jingwen Leng, Yuhao Zhu, Zidong Du, Quan Chen, Chao Li, Bin Yao, and Minyi Guo. 2020. Balancing efficiency and flexibility for DNN acceleration via temporal GPU-systolic array integration. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

Hai Victor Habi, Roy H Jennings, and Arnon Netzer. 2020. Hmq: Hardware friendly mixed precision quantization block for cnns. In *European Conference on Computer Vision (ECCV)*. Springer, 448–463.

Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 2. IEEE, 1735–1742.

Song Han, Huizi Mao, and William J Dally. 2015a. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

Song Han, Jeff Pool, John Tran, and William Dally. 2015b. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015).

Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. 2019. FPGA/DNN Co-Design: An Efficient Design Methodology for 1oT Intelligence on the Edge. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

Benjamin Hawks, Javier Duarte, Nicholas J Fraser, Alessandro Pappalardo, Nhan Tran, and Yaman Umuroglu. 2021. Ps and qs: Quantization-aware pruning for efficient low latency neural network inference. *Frontiers in Artificial Intelligence* 4 (2021), 676564.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *European Conference on Computer Vision (ECCV)*. 784–800.

Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *IEEE International Conference on Computer Vision (ICCV)*.

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. 2018. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670* (2018).

Harold Horace Hopkins. 1951. The concept of partial coherence in optics. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 208, 1093 (1951), 263–277.

Peng Hu, Xi Peng, Hongyuan Zhu, Mohamed M Sabry Aly, and Jie Lin. 2021. Opq: Compressing deep neural networks with one-shot pruning-quantization. In *AAAI Conference on Artificial Intelligence*, Vol. 35. 7780–7788.

Yan Huang, Shang Li, Liang Wang, Tieniu Tan, et al. 2020. Unfolding the Alternating Optimization for Blind Super Resolution. *Annual Conference on Neural Information Processing Systems (NIPS)* 33 (2020).

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *Advances in neural information processing systems* 29 (2016).

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2704–2713.

Bentian Jiang, Lixin Liu, Yuzhe Ma, Hang Zhang, Bei Yu, and Evangeline FY Young. 2020. Neural-ILT: Migrating ILT to neural networks for mask printability and complexity co-optimization. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–9.

Bentian Jiang, Hang Zhang, Jinglei Yang, and Evangeline FY Young. 2019. A fast machine learning-based mask printability predictor for OPC acceleration. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 412–419.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Annual Conference on Neural Information Processing Systems (NeurIPS)* 33 (2020), 18661–18673.

Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. 2016. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1646–1654.

Soo Ye Kim, Jihyong Oh, and Munchurl Kim. 2020. JSI-GAN: GAN-Based Joint Super-Resolution and Inverse Tone-Mapping with Pixel-Wise Task-Specific Filters for UHD HDR Video.. In *AAAI Conference on Artificial Intelligence*. 11287–11295.

Jian Kuang, Wing-Kai Chow, and Evangeline FY Young. 2015. A robust approach for process variation aware mask optimization. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. IEEE, 1591–1594.

Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. 2017a. Deep laplacian pyramid networks for fast and accurate super-resolution. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 624–632.

Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. 2017b. Deep laplacian pyramid networks for fast and accurate super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 624–632.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017a. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4681–4690.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017b. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4681–4690.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. 2017c. Photo-realistic single image super-resolution using a generative adversarial network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4681–4690.

Royson Lee, Łukasz Dudziak, Mohamed Abdelfattah, Stylianos I Venieris, Hyeji Kim, Hongkai Wen, and Nicholas D Lane. 2020. Journey towards tiny perceptual super-resolution. In *European Conference on Computer Vision*. Springer, 85–102.

Haitong Li, Mudit Bhargav, Paul N Whatmough, and H-S Philip Wong. 2019a. On-Chip Memory Technology Design Space Explorations for Mobile Deep Neural Network Accelerators. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

Wenbo Li, Xin Tao, Taian Guo, Lu Qi, Jiangbo Lu, and Jiaya Jia. 2020b. MuCAN: Multi-Correspondence Aggregation Network for Video Super-Resolution. *European Conference on Computer Vision (ECCV)* (2020).

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. [n. d.]. BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction. In *International Conference on Learning Representations (ICLR)*.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. 2020a. BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction. In *International Conference on Learning Representations*.

Zhen Li, Jinglei Yang, Zheng Liu, Xiaomin Yang, Gwanggil Jeon, and Wei Wu. 2019b. Feedback network for image super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3867–3876.

Liang Liu, Mingzhu Shen, Ruihao Gong, Fengwei Yu, and Hailong Yang. 2022. NNLQP: A Multi-Platform Neural Network Latency Query and Prediction System with An Evolving Database. 1–14.

Xin Liu, Yuang Li, Josh Fromm, Yuntao Wang, Ziheng Jiang, Alex Mariakakis, and Shwetak Patel. 2021. Splitsr: An end-to-end approach to super-resolution on mobile devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 1 (2021), 1–20.

Christos Louizos, Max Welling, and Diederik P Kingma. 2018. Learning Sparse Neural Networks through L_0 Regularization. In *International Conference on Learning Representations*.

Yuzhe Ma, Wei Zhong, Shuxiang Hu, Jhih-Rong Gao, Jian Kuang, Jin Miao, and Bei Yu. 2020. A unified framework for simultaneous layout decomposition and mask optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 39, 12 (2020), 5069–5082.

Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 42, 4 (2018), 824–836.

David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. 2001. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *IEEE International Conference on Computer Vision (ICCV)*, Vol. 2. 416–423.

Tetsuaki Matsunawa, Bei Yu, and David Z Pan. 2015. Optical proximity correction with hierarchical bayes model. In *Optical Microlithography XXVIII*, Vol. 9426. International Society for Optics and Photonics, 94260X.

Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey. 2017. Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462* (2017).

Jiandong Mu, Mengdi Wang, Lanbo Li, Jun Yang, Wei Lin, and Wei Zhang. 2020. A history-based auto-tuning framework for fast and high-performance DNN design on GPU. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. 2020. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*. PMLR, 7197–7206.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. 2019. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1325–1334.

David Z Pan, Bei Yu, and Jhih-Rong Gao. 2013. Design for manufacturing with emerging nanolithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 32, 10 (2013), 1453–1472.

Ji-Soong Park, Chul-Hong Park, Sang-Uhk Rhie, Yoo-Hyon Kim, Moon-Hyun Yoo, Jeong-Taek Kong, Hyung-Woo Kim, and Sun-Il Yoo. 2000. An efficient rule-based OPC approach using a DRC tool for 0.18/spl mu/m ASIC. In *IEEE International Symposium on Quality Electronic Design (ISQED)*. 81–85.

Zehua Pei, Xufeng Yao, Wenqian Zhao, and Bei Yu. 2023a. Quantization via Distillation and Contrastive Learning. *IEEE Transactions on Neural Networks and Learning Systems* (2023).

Zehua Pei, Wenqian Zhao, Zhuolun He, and Bei Yu. 2023b. Bit-Level Quantization for Efficient Layout Hotspot Detection. In *International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 465–470.

Reid Pinkham, Shuqing Zeng, and Zhengya Zhang. 2020. QuickNN: Memory and Performance Optimization of k-d Tree Based Nearest Neighbor Search for 3D Point Clouds. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 180–192.

Amyn Poonawala and Peyman Milanfar. 2007. Mask design for optical microlithography—an inverse imaging problem. *IEEE Transactions on Image Processing (TIP)* 16, 3 (2007), 774–788.

Zhongnan Qu, Zimu Zhou, Yun Cheng, and Lothar Thiele. 2020. Adaptive loss-aware quantization for multi-bit networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 7988–7997.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*. Springer, 525–542.

Yaniv Romano, John Isidoro, and Peyman Milanfar. 2016. RAISR: rapid and accurate image super resolution. *IEEE Transactions on Computational Imaging* 3, 1 (2016), 110–125.

Hao-Chiang Shao, Chao-Yi Peng, Jun-Rei Wu, Chia-Wen Lin, Shao-Yun Fang, Pin-Yian Tsai, and Yan-Hsiu Liu. 2020. From IC layout to die photograph: A CNN-based data-driven approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 40, 5 (2020), 957–970.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. 2009. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995* (2009).

Suraj Srinivas and R Venkatesh Babu. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149* (2015).

Yu-Hsuan Su, Yu-Chen Huang, Liang-Chun Tsai, Yao-Wen Chang, and Shayak Banerjee. 2016. Fast lithographic mask optimization considering process variation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 35, 8 (2016), 1345–1357.

Qi Sun, Chen Bai, Hao Geng, and Bei Yu. 2021. Deep Neural Network Hardware Deployment Optimization via Advanced Active Learning. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. 1510–1515.

Ying Tai, Jian Yang, and Xiaoming Liu. 2017. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3147–3155.

Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008* (2017).

Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. 2014. Fast convolutional nets with FBFFT: A GPU performance evaluation. *arXiv preprint arXiv:1412.7580* (2014).

Chaofeng Wang, Zheng Li, and Jun Shi. 2019a. Lightweight image super-resolution with adaptive weighted learning network. *arXiv preprint arXiv:1904.02358* (2019).

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019b. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8612–8620.

Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. 2020b. Apq: Joint search for network architecture, pruning and quantization policy. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2078–2087.

Ying Wang, Yadong Lu, and Tijmen Blankevoort. 2020a. Differentiable joint pruning and quantization for hardware efficiency. In *European Conference on Computer Vision (ECCV)*. Springer, 259–277.

Zixiao Wang, Yunheng Shen, Wenqian Zhao, Yang Bai, Guojin Chen, Farzan Farnia, and Bei Yu. 2023. DiffPattern: Layout Pattern Generation via Discrete Diffusion. In *ACM/IEEE Design Automation Conference (DAC)*.

Yuki Watanabe, Taiki Kimura, Tetsuaki Matsunawa, and Shigeki Nojima. 2017. Accurate lithography simulation model based on convolutional neural networks. In *Optical Microlithography XXX*, Vol. 10147. International Society for Optics and Photonics, 101470K.

Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *ACM/IEEE Design Automation Conference (DAC)*. 29:1–29:6.

Li Wenbo, Zhou Kun, Qi Lu, Jiang Nianjuan, Lu Jiangbo, and Jia Jiaya. 2020. LAPAR: Linearly-Assembled Pixel-Adaptive Regression Network for Single Image Super-resolution and Beyond. In *Annual Conference on Neural Information Processing Systems (NIPS)*.

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018a. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090* (2018).

Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. 2020. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602* (2020).

Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. 2018b. Unsupervised feature learning via non-parametric instance discrimination. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3733–3742.

Lei Xiao, Salah Nouri, Matt Chapman, Alexander Fix, Douglas Lanman, and Anton Kaplanyan. 2020. Neural supersampling for real-time rendering. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 142–1.

Haoyu Yang, Shuhe Li, Zihao Deng, Yuzhe Ma, Bei Yu, and Evangeline FY Young. 2019. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 39, 10 (2019), 2822–2834.

Jianchao Yang, Zhaowen Wang, Zhe Lin, Scott Cohen, and Thomas Huang. 2012. Coupled dictionary training for image super-resolution. *IEEE Transactions on Image Processing (TIP)* 21, 8 (2012), 3467–3478.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. 2021. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning (ICML)*. PMLR, 11875–11886.

Wei Ye, Mohamed Baker Alawieh, Yibo Lin, and David Z Pan. 2019. LithoGAN: End-to-end lithography modeling with generative adversarial networks. In *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

Ziyang Yu, Guojin Chen, Yuzhe Ma, and Bei Yu. 2021. A GPU-enabled level set method for mask optimization. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. IEEE, 1835–1838.

Kai Zhang, Wangmeng Zuo, and Lei Zhang. 2018. Learning a single convolutional super-resolution network for multiple degradations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3262–3271.

Sijie Zhao, Tao Yue, and Xuemei Hu. 2021b. Distribution-aware adaptive multi-bit quantization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 9281–9290.

Wenqian Zhao, Yang Bai, Qi Sun, Wenbo Li, Haisheng Zheng, Nianjuan Jiang, Jiangbo Lu, Bei Yu, and Martin DF Wong. 2023. A High-Performance Accelerator for Super-Resolution Processing on Embedded GPU. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2023).

Wenqian Zhao, Qi Sun, Yang Bai, Wenbo Li, Haisheng Zheng, Bei Yu, and Martin DF Wong. 2021a. A High-Performance Accelerator for Super-Resolution Processing on Embedded GPU. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

Wenqian Zhao, Xufeng Yao, Ziyang Yu, Guojin Chen, Yuzhe Ma, Bei Yu, and Martin DF Wong. 2022. AdaOPC: A self-adaptive mask optimization framework for real design patterns. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–9.

Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, et al. 2020. Ansor: Generating high-performance tensor programs for deep learning. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 863–879.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights. In *International Conference on Learning Representations (ICLR)*.

Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. 2018. Adaptive quantization for deep neural network. In *AAAI Conference on Artificial Intelligence*, Vol. 32.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision (ICCV)*. 2223–2232.