A High-Performance Accelerator for Super-Resolution Processing on Embedded GPU

Wenqian Zhao, Yang Bai, Qi Sun, *Member, IEEE*, Wenbo Li, *Graduate Student Member, IEEE*, Haisheng Zheng, Nianjuan Jiang, *Member, IEEE*, Jiangbo Lu, *Senior Member, IEEE*, Bei Yu, *Senior Member, IEEE*, and Martin D. F. Wong

***Abstract*—Over the past few years, super-resolution (SR) processing has achieved astonishing progress along with the development of deep learning. Nevertheless, the rigorous requirement for real-time inference, especially for video tasks, leaves a harsh challenge for both the model architecture design and the hardware-level implementation. In this article, we propose a hardware-aware acceleration on embedded GPU devices as a full-stack SR deployment framework. The most critical stage with dictionary learning applied in SR flow was analyzed in details and optimized with a tailored dictionary slimming strategy. Moreover, we also delve into the programming architecture of hardware while analyzing the model structure to optimize the computation kernels to reduce inference latency and maximize the throughput given restricted computing power. In addition, we further accelerate the model with 8-bit integer inference by quantizing the weights in the compressed model. An adaptive 8-bit quantization flow for SR task enables the quantized model to achieve a comparable result with the full-precision baselines. With the help of our approaches, the computation and communication bottlenecks in the deep dictionary learning-based SR models can be overcome effectively. The experiments on both edge embedded device NVIDIA NX and 2080Ti prove that our framework exceeds the performance of state-of-the-art NVIDIA TensorRT significantly and can achieve real-time performance.**

***Index Terms*—Edge computing, neural network compression, super-resolution (SR).**

I. Introduction

SUPER-RESOLUTION (SR) is an important class of graphical processing techniques that plays an important role in the digital image era. The SR task aims at generating or recovering high-resolution (HR) video frames given frames with low-resolution (LR). Among all existing approaches, the naive solution is to interpolate the LR image with RGB value collected bilinear or bicubic from spatially invariant nearest-neighbor pixels. Advanced development of deep learning in computer vision has stimulated a group of powerful SR approaches with impressive performance for SR. From conventional convolution neural networks [2] to novel generative adversarial networks [3], [4], various methods have appeared in the last decade. Recently, by introducing dictionary learning methods with pixel-level local feature fusion operations [5], [6], the image quality of generated HR images or videos is further improved with richer color/texture details recovered thanks to the idea of dictionary learning and pixel-level local feature fuse operations. As algorithms get performant, the efficient and optimized deployment of such deep learning-based SR methods on hardware has gradually become the new spot of attention.

A variety of previous methods have been proposed for the domain-specific deployment of different deep learning algorithms on different hardware platforms, [7], [8], [9], [10]. Among which most deployed models are design for object classification [11], detection [12], [13], neural language processing (NLP) [14], etc. Regardless of their wide scenario coverage and different task data format, these models still share similar deep learning operators in their implementation with each other and therefore require no explicit special technique. The most common operators are convolution, pooling, softmax, fully connected operation, etc. In consideration of the popularity of these operators, vendor-provided commercial tools usually apply some customization and achieve state-of-the-art performance on these operators by using fixed, manually written hardware codes. For example, TensorRT [15] exceeds other tools on NVIDIA GPUs and Intel MKL-DNN [16] has the dominating inference latency on Intel CPUs.

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/s), some particular properties of SR algorithms need to be considered. First, deep learning-based SR models hold completely opposite algorithmic processing logic to other mainstream task models. Traditional DNN models spatially scale down the input frame layer-by-layer

Manuscript received 24 July 2022; revised 20 October 2022 and 6 January 2023; accepted 10 January 2023. Date of publication 8 February 2023; date of current version 20 September 2023. This work was supported in part by the Research Grants Council of Hong Kong, SAR, under Grant CUHK24209017; in part by the Innovation and Technology Fund under Grant PRP/065/20FX; and in part by SmartMore. The preliminary version has been presented at the IEEE/ACM International Conference on Computer-Aided Design (ICCAD) in 2021 [1] [DOI: 10.1109/ICCAD51958.2021.9643472]. This article was recommended by Associate Editor T. Mitra. *(Corresponding author: Bei Yu.)*

Wenqian Zhao, Yang Bai, Qi Sun, Wenbo Li, Bei Yu, and Martin D. F. Wong are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Haisheng Zheng, Nianjuan Jiang, and Jiangbo Lu are with the Heterogeneous Computing Center, SmartMore Corporation Limited, Hong Kong.

Digital Object Identifier 10.1109/TCAD.2023.3241110

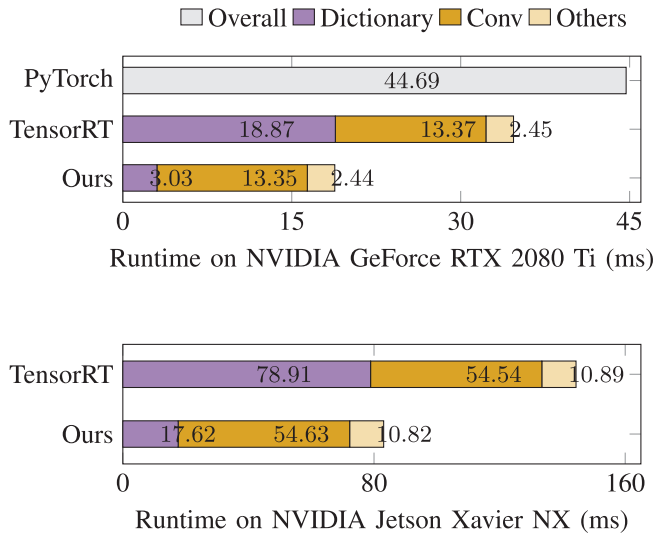

Fig. 1. Time breakdown of the inference flow of state-of-the-art SR model. We divide the inference time into three major categories: 1) dictionary query and filtering step; 2) convolution operation; and 3) data reformatting, concatenation, or other operations.

while embedding and learning the hidden feature, e.g., VGG, GoogleNet, MobileNet, ResNet, Faster R-CNN, etc. Such a down-sampling operation deliberately retains the volume of the feature map as the embedding tensor size increase in depth, to some extent relieving the communication and computation pressures from features and weights. On the other hand, to recover more pixel-wise color/texture details, SR models usually keep or scale up the input frame. In this case, the feature map volume is much larger than the weights, therefore becoming the dominating factor that makes the current off-the-shelf memory optimization techniques ineffective. Jung et al. [17] also discovered similar phenomenons. Second, unlike some traditional widely used operations, e.g., convolution and pooling, some domain-specific tensor operations such as local pixel-shuffle and dictionary learning appear exclusively in SR task, which remains unsolved through various techniques and exacerbate these challenges. In comparison, as shown in Fig. 1, these novel operations in SR are time-consuming and may need special computation reorganizations and parallelisms. Due to these challenges, the existing solutions are nonoptimal, even the state-of-the-art commercial tool, e.g., TensorRT.

This article proposes several specific techniques to tackle those mentioned challenges. First, 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. Second, 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 reorganize the original large task into many smaller subtasks and run these subtasks in parallel.

The main contributions of this article are listed as follows.

1) We build a specifically designed engine to remarkably accelerate dictionary learning, especially on extremely large data for the first time.
2) 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.
3) We propose a targeted 8-bit adaptive quantization approach to further compress the SR model and realize further acceleration.
4) We analyze both resources- and workloads-aware constraints dedicated for GPUs to guide the search for optimal hardware implementations.
5) 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. Runtime breakdowns are visualized in Fig. 1.

The following pages of this article are organized. Section II briefly introduces the deep SR models, dictionary learning, the GPU programming background, and the low-bit scale quantization. Section III demonstrates our acceleration approaches from algorithm level to hardware level. Section V demonstrates the experiments and results. Finally, we conclude this article in Section VI.

II. Preliminaries

A. Super-Resolution Algorithms and Dictionary Learning

The objective of SR algorithm is to reconstruct an HR image with details recovered from an LR input. A variety of methods have been proposed in the past few decades since the SR algorithm can be widely applied in many scenarios.

Given a high/low-resolution image pair, the corresponding relation is a transformation process described in (1). The HR vectorized image $\boldsymbol{y} \in \mathbb{R}^{HWs^2}$ is applied with a down-sampling of scale s and a blurring filter to obtain the LR counterpart $\boldsymbol{x} \in \mathbb{R}^{HW}$, where we denote W and H as the width and height of the image

$$\boldsymbol{x} = \boldsymbol{SHy} \tag{1}$$

where $\boldsymbol{H} \in \mathbb{R}^{HWs^2}$ denotes a blurring operation (e.g., Gaussian Blurring) and $\boldsymbol{S} \in \mathbb{R}^{HW \times HWs^2}$ is the down-sampling operation. The objective of SR is simply reversing such process: given an LR $\boldsymbol{x}$, the task needs to up-scale and deblur to restore $\boldsymbol{y}$. One of the obstacles is that the transformation (1) is ill-posed. That is to say, each LR $\boldsymbol{x}$ may not necessarily correspond to a single unique [$\boldsymbol{y}$] and may possibly have more than one valid solution.

In this way, the problem is notoriously challenging to solve because we cannot learn a naive inverse transformation. On the other hand, in some real scenarios, HR $\boldsymbol{y}$ data is inaccessible. For example, sometimes LR $\boldsymbol{x}$ is directly taken from a digital camera, or the HR information is permanently lost through data communication. In these cases, the reverse transformation is even more difficult to derive.

Some earlier approaches adopt naive basic linear interpolation methods, e.g., bilinear and bicubic interpolations to tackle these challenges. Despite the simplicity and straightforwardness, these methods inevitably neglect some content varieties and local structures. Afterward, some dictionary learning algorithms are proposed to bridge the mapping gap, embedding the mapping relationships between the HR space and LR space numerically. With training on the embedding and query process, the model learns how to map LR patches to HR patches. Intuitively HR patches are regarded as a spatial combination of LR patches, and now the learning objective is to generate combination coefficients. Recently, deep learning model performance has been leaping forward impressively, which stimulates a variety of new methods which can learn dictionaries and combination coefficients better with great performance in HR quality [6], [18], [19].

The general processing flow of the deep dictionary learning-based SR model is illustrated as follows. First, the input LR image is vectorized as $\boldsymbol{x} \in \mathbb{R}^{HW}$ and sliced into patches of k^2. Upsampled matrix $\boldsymbol{B} \in \mathbb{R}^{HWs^2 \times k^2}$ is composed of HWs^2 upsampled LR patches with size k^2. Second, some transformation operations are conducted to transform the LR batches into HR batches. The ith pixel $\boldsymbol{y}_i$ in the HR image vector $\boldsymbol{y} \in \mathbb{R}^{HWs^2}$ is obtained via integrating the neighboring pixels of batch $\boldsymbol{B}_i$ (i.e., the ith row of $\boldsymbol{B}$) centered at the coordinate of $\boldsymbol{y}_i$. This pixel-level operation can be formulated as

$$\boldsymbol{y}_i = \boldsymbol{F}_i \boldsymbol{B}_i^\top, \text{ with } \boldsymbol{F}_i = \boldsymbol{\Phi}_i \boldsymbol{D} \tag{2}$$

where $\boldsymbol{F}_i \in \mathbb{R}^{1 \times k^2}$ denotes the integration coefficient vector (also known as a filter). Furthermore, filter $\boldsymbol{F}_i$ together with combination coefficient vector $\boldsymbol{\Phi}_i \in \mathbb{R}^{1 \times L}$ can be jointly regarded as a linear combination of dictionary $\boldsymbol{D} \in \mathbb{R}^{L \times k^2}$. The predefined dictionary $\boldsymbol{D}$ is fixed during model inference, while the coefficients $\boldsymbol{\Phi}_i$ are the goal to compute under the real-time requirement. As (2) formulates the pixel-level operation, image-level transformation is represented accordingly in

$$\boldsymbol{y} = \boldsymbol{F}\boldsymbol{B}^\top, \text{ with } \boldsymbol{F} = \boldsymbol{\Phi}\boldsymbol{D} \tag{3}$$

with $\boldsymbol{F} \in \mathbb{R}^{HWs^2 \times k^2}$ and $\boldsymbol{\Phi} \in \mathbb{R}^{HWs^2 \times L}$. $\boldsymbol{\Phi}$ [20], [21], [22] have made some attempts to learn the coefficient matrix $\boldsymbol{\Phi}$ and dictionary $\boldsymbol{D}$. To save the effort of learning both objectives, linearly assembled pixel-adaptive regression network (LAPAR) [6] utilizes a predefined $\boldsymbol{D}$ composed of a series of Gaussian (G) filters as well as some difference of Gaussian (DoG) filters to speed up the learning process. The coefficient matrix $\boldsymbol{\Phi}$ is produced as the output of a residual network (details about the network will be covered in Section II-B).

Considering the communication patterns of (3), $\boldsymbol{\Phi}$ and $\boldsymbol{B}$ usually occupy much more bandwidth than $\boldsymbol{D}$, i.e.,

$$HWs^2 \times L + HWs^2 \times k^2 \gg L \times k^2. \tag{4}$$

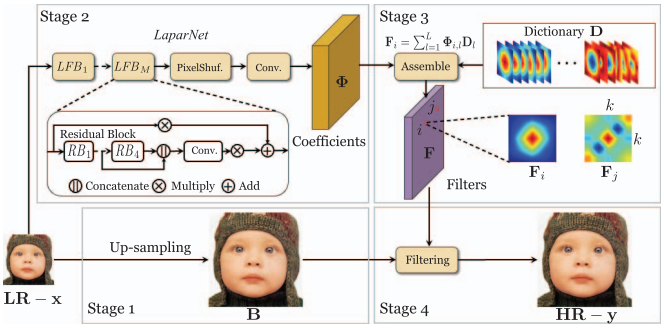

Fig. 2. Architecture of LAPAR [6].

The dictionary $\boldsymbol{D}$ is the key point when considering the computation patterns. It determines whether and how to compute given the data in $\boldsymbol{\Phi}$ and $\boldsymbol{B}$, which plays as a bridge and translator to connect $\boldsymbol{\Phi}$ and $\boldsymbol{B}$. According to $\boldsymbol{D}$, some unnecessary data, if no harm to the performance, can skip being loaded to the on-chip cache, and the corresponding computation can be skipped. The special role of $\boldsymbol{D}$ in dictionary learning distinguishes itself from typical deep learning algorithms by considering more than weights and features. Once the dictionary is optimized, both communication and computation bottlenecks can be simultaneously resolved.

B. 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, etc.

We take LAPAR [6], the state-of-the-art SR model LAPAR [6] as an example to explain the model structure and inference flow. The inference flow can be divided into four stages as shown in Fig. 2. At the first stage, the input image $\boldsymbol{x}$ is up-scaled with bicubic interpolation to generate patch matrix $\boldsymbol{B}$. Second, *LaparNet* also takes $\boldsymbol{x}$ as input and generate the coefficient matrix $\boldsymbol{\Phi}$ as the output. *LaparNet* model include several local fusion blocks (LFBs) [23], pixel-shuffle layers, and some convolution layers, where each LFB is structured by residual blocks and following concatenations, multiplications, and short-cut additions. In the third stage, the dictionary assembling is applied to retrieve the transformation matrix $\boldsymbol{F}$ by querying the predefined dictionary $\boldsymbol{D}$ with $\boldsymbol{\Phi}$. The final stage obtains the HR image $\boldsymbol{y}$ with details restored by filtering the up-sampled $\boldsymbol{B}$ with $\boldsymbol{F}$, i.e., $\boldsymbol{y} = \boldsymbol{F}\boldsymbol{B}^\top$. It is necessary to analyze the dictionary learning module in detail to efficiently deploy the SR models on GPU, which has been ignored in previous work.

C. GPU Programming Architecture

NVIDIA provides a well-designed high-level abstraction of their GPU architecture as shown in Fig. 3, 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,

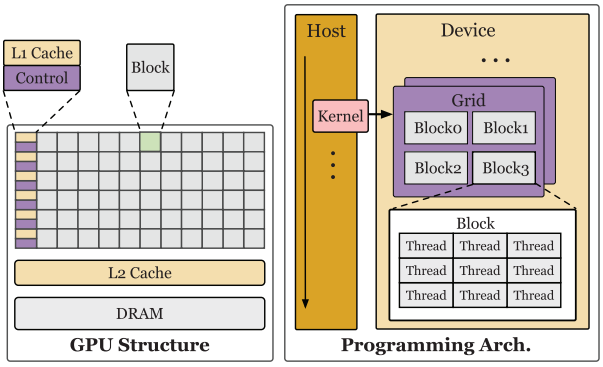

Fig. 3. GPU memory hierarchy and communication mode.

several processing blocks, 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, etc.

NVIDIA provides CUDA programming model [24] 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 Fig. 3. 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 (SIMTs) 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.

D. Low Precision Inference

Nowadays, 32-bit single-precision floating-point is the mainstream data format for most deep learning applications. Quantization is a technique used to reduce model inference latency with high throughput integer instructions or even lower bit data formats without significant accuracy loss. Reference [25] has shown that on NVIDIA GPU, math-intensive tensor operators can reach 16× speed-up with 8-bit signed integer data format in comparison with FP32 while memory-intensive tensor operators reach up to 4× speed-up.

Many post-training quantization (PTQ) methods were proposed to quantize models to 8-bit without retraining safely. Nagel et al. [26] implemented 8-bit quantization in a data-free style. Nagel et al. [27] proposed a layer-wise calibration strategy by minimizing the Hessian of task loss. Banner et al. [28] proposed an analytical solution for quantization clipping range selection. Some previous research even explored very low-bit quantization, all the way to ternary (2-bit) or even binary (1-bit) data format [29], [30], [31], [32], [33].

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

$$s = \frac{\epsilon}{2^{b-1}-1} \tag{5}$$

$$\hat{x} = \text{clip}\left(\text{round}\left(\frac{x}{s}\right), -2^{b-1}+1, 2^{b-1}-1\right) \tag{6}$$

where x is the original floating-point number, $\hat{x}$ is the quantized integer. The quantization process in (6) can be regarded as three consecutive steps: 1) scaling; 2) rounding; and 3) clipping. First, floating-point x is multiplied with a scaling factor s and rounded to the nearest neighboring integer. After that, the integer is clipped by range $[-2^{b-1}+1, 2^{b-1}-1]$ to fit in the representative range of 8-bit. 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 (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

$$\boldsymbol{A} \cdot \boldsymbol{B} = \hat{\boldsymbol{A}} \cdot \hat{\boldsymbol{B}} \cdot s_{\boldsymbol{A}} \cdot s_{\boldsymbol{B}}. \tag{7}$$

III. Optimization of Deployments on GPU

A. Dictionary Slimming

The most performant lightweight SR algorithm LAPAR [6], shows state-of-the-art ability even with a compact model size (num. of params $<$ 1M). However, this lightweight architecture still cannot fulfill the requirement of real-time inference even with the powerful NVIDIA TensorRT [15]. Considering the distinction of the SR task, the large spatial scale of feature maps instead of the number of parameters is the key factor in the inference speed. The running time breakdown is in Fig. 1. As shown in Fig. 1, dictionary learning is the bottleneck and takes the largest percentage of time cost. This can be explained by the fact that existing commercial tools do not provide customized support for certain special operations or extreme-scale feature maps and only have efficient and reliably tailored implementation for most common DNN layers, such as conv, ReLU, and BN, which may result in a large time cost for some computation graphs.

Slimming the dictionary not only can save some computation costs but also ease communication pressure on hardware. We propose a dictionary slimming strategy to compress the dictionary. Ideally, it requires a valid dictionary $\boldsymbol{D}$ to be informative enough to provide sufficient embedding data for the restoration of image details. However, on the other hand, we do not expect the dictionary $\boldsymbol{D}$ to be too bulky with redundant information. By slimming the dictionary, a desired slim dictionary $\boldsymbol{D}$ can perform inference under the harsh speed requirement without significant accuracy degradation. We control the slimming effort with a sparsity threshold $\alpha \in (0, 1)$ which reflects the sparsity of the dictionary $\boldsymbol{D} \in \mathbb{R}^{L \times k^2}$. After slimming, the most essential $\alpha \cdot L$ items from L will be reserved. Note that aggressive slimming of the dictionary to ratio α is difficult and may not lead to the optimum. To simplify the problem, we slim the dictionary iteratively while gradually reducing the sparsity from 1 to α. At each iteration t with the sparsity α_t, with $\alpha_t < \alpha_{t-1}$ set, We retain the most important $\alpha_t L$ items in the current dictionary and regard the others. At

next iteration, the current sparsity goal is updated $\alpha_{t+1} = \alpha_t - \Delta_\alpha$, to further prune more items. After pruning out redundant items, we still need to fine-tune the *LaparNet* to adapt to the newly slimmed dictionary by minimizing the reconstruction error. The problem can be formulated as follows:

$$\begin{aligned} \boldsymbol{\beta}, \boldsymbol{W} = \arg\min_{\boldsymbol{\beta}, \boldsymbol{W}} \frac{1}{N} \left\| \boldsymbol{Y} - \sum_{i=0}^{L} \beta_i \boldsymbol{\Phi} \boldsymbol{D} \right\|_2^2 \\ \text{s.t.} \quad \boldsymbol{\Phi} = \mathit{LaparNet}(\boldsymbol{X}, \boldsymbol{W}) \\ \|\boldsymbol{\beta}\|_0 \leq \alpha L \end{aligned} \tag{8}$$

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. ith item of $\boldsymbol{D}$ will be neglected if $\beta_i = 0$.

In addition, we make further modifications to (8) by taking the final filtering stage of SR flow into consideration by evaluating the final image. We formulate the reconstruction error between the compressed model generated image and ground truth HR image $\boldsymbol{H}_{gt}$ into

$$\begin{aligned} \boldsymbol{\beta}, \boldsymbol{W} = \arg\min_{\boldsymbol{\beta}, \boldsymbol{W}} \frac{1}{N} \left\| \boldsymbol{H}_{gt} - \boldsymbol{F}_{W,\beta} \boldsymbol{B}^\top \right\|_2^2 \\ \text{s.t.} \quad \boldsymbol{F}_{W,\beta} = \sum_{i=0}^{L} \beta_i \boldsymbol{\Phi} \boldsymbol{D} \\ \boldsymbol{\Phi} = \mathit{LaparNet}(\boldsymbol{X}, \boldsymbol{W}) \\ \|\boldsymbol{\beta}\|_0 \leq \alpha L. \end{aligned} \tag{9}$$

Both $\boldsymbol{\beta}$ and $\boldsymbol{W}$ are the optimization objectives. We simplify the problem and solve it efficiently by alternatively optimizing each objective. At first, we search the optimal selector β to fulfill requirement of sparsity α_t with fixed *LaparNet* parameters. At second step, we fix β and tune the parameters $\boldsymbol{W}$ to minimize the reconstruction error in (9). Direct optimization of selector β with l-0 norm constraint is NP-hard. However, we can optimize the sparsity by using LASSO regression with a ℓ_1 regulation term [34] added to the original loss function, as shown in

$$\begin{aligned} \boldsymbol{\beta} = \arg\min_{\boldsymbol{\beta}} \frac{1}{N} \left\| \boldsymbol{H}_{gt} - \boldsymbol{F}_{W,\beta} \boldsymbol{B}^\top \right\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 \\ \text{s.t.} \quad \|\boldsymbol{\beta}\|_0 \leq \alpha L. \end{aligned} \tag{10}$$

The complete selection strategy is illustrated in Algorithm 1.

Before channel selection, we need to prepare a set of calibration data, including output feature maps of *LaparNet* before querying the dictionary as well as corresponding HR ground-truth images. We control the sparsity by carefully adjusting regulation weight λ in (10). We search the β greedily by starting with a small λ. After each iteration, we double the value of λ, forcing a stronger sparsity regulation until the required α is achieved. At the end of slimming, in case of the step size λ gets too large after the exponential update, we apply a binary search within the range $[\lambda_t, \lambda_{t+1}]$ to delicately adjust the slimming ratio close to α_{t+1}, as shown in lines 12–20 in Algorithm 1.

Algorithm 1 Dictionary Selection Strategy

1: **Input:** $\boldsymbol{D} \in \mathbb{R}^{L \times k^2}$, small λ_0, target α, tolerance ϵ;
2: **Input:** pre-trained $\boldsymbol{W}_0$, coefficient matrix $\boldsymbol{\Phi}$;
3: $t \leftarrow 0$, $\alpha_0 \leftarrow 1.0$, $\beta_0 \leftarrow \mathbf{1} \in \mathbb{R}^L$, $\gamma_0 \leftarrow \mathbf{1} \in \mathbb{R}^L$;
4: $\mathscr{L} \leftarrow$ reconstruction error ▷ Equation (9)
5: **repeat**
6: $\quad \alpha_{t+1} \leftarrow \alpha_t - \Delta\alpha$;
7: $\quad \lambda_{t+1} \leftarrow \lambda_t$;
8: $\quad$ **while** $|\beta_{t+1}|_0 > \alpha_{t+1} \cdot L$ **do**
9: $\quad\quad$ Fix $\boldsymbol{W}_t$, update $\beta_{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \mathscr{L}(\boldsymbol{W}_t, \boldsymbol{\beta}\boldsymbol{D}) + \lambda_{t+1}|\boldsymbol{\beta}|$; ▷ Equation (10)
10: $\quad\quad \lambda_{t+1} \leftarrow 2 \cdot \lambda_{t+1}$
11: $\quad$ **end while**
12: $\quad \lambda_{left} \leftarrow 0.5\lambda_{t+1}$, $\lambda_{right} \leftarrow \lambda_{t+1}$;
13: $\quad$ **while** $\left|\alpha_{t+1} \cdot L - \left|\beta_{t+1}\right|_0\right| > \epsilon \cdot L$ **do**
14: $\quad\quad \lambda_{t+1} = 1/2(\lambda_{left} + \lambda_{right})$;
15: $\quad\quad$ Fix $\boldsymbol{W}_t$, update $\beta_{t+1} \leftarrow \arg\min_{\boldsymbol{\beta}} \mathscr{L}(\boldsymbol{W}_t, \boldsymbol{\beta}\boldsymbol{D}) + \lambda_{t+1}|\boldsymbol{\beta}|$;
16: $\quad\quad$ **if** $\left|\beta_{t+1}\right|_0 < \alpha_{t+1} \cdot L$ **then**
17: $\quad\quad\quad \lambda_{left} \leftarrow \lambda_{t+1}$;
18: $\quad\quad$ **else if** $\left|\beta_{t+1}\right|_0 > \alpha_{t+1} \cdot L$ **then**
19: $\quad\quad\quad \lambda_{right} \leftarrow \lambda_{t+1}$;
20: $\quad\quad$ **end if**
21: $\quad$ **end while**
22: $\quad$ Fix β_{t+1}, update $\boldsymbol{W}_{t+1} \leftarrow \arg\min_{\boldsymbol{W}} \mathscr{L}(\boldsymbol{W}, \beta_{t+1}\boldsymbol{D})$; ▷ Equation (11)
23: $\quad t = t + 1$;
24: **until** $\alpha_t \leq \alpha$

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

$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}} \frac{1}{N} \left\| \boldsymbol{H}_{gt} - \boldsymbol{F}_{W,D'} \boldsymbol{B}^\top \right\|_2^2. \tag{11}$$

However, tuning all parameters in *LaparNet* at each iteration may cost too much computation power and time. As shown in (11), The $\boldsymbol{D}'$ is the dictionary which is the compressed dictionary with layers neglected in the previous LASSO step. To efficiently adjust the output of *LaparNet* for the newly compressed dictionary $\boldsymbol{D}'$, we simply reconstruct the parameters of the last layer before the dictionary query instead. To achieve fast tuning, we use linear regression to learn a channel-wise factor for original parameters, which is more efficient. Parameters in the last layer $\boldsymbol{W}_{D'}$ are weighted with a regression coefficient $\boldsymbol{\gamma}$ at each channel. In this way, we can reformulate this parameter-tuning step from (11) to (12). $\boldsymbol{\gamma}$ is a channel-wise coefficient to scale the parameters on each channel of updated parameters $\boldsymbol{W}_{D'}^{\text{new}}$.

After the tuning, a new coefficient matrix Φ' 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 Fig. 4

$$\begin{aligned} \boldsymbol{\gamma} = \arg\min_{\boldsymbol{\gamma}} \frac{1}{N} \left\| \boldsymbol{H}_{gt} - \sum_{i=0}^{L} \gamma_i \boldsymbol{F}_{W,D'} \boldsymbol{B}^\top \right\|_2^2 \\ \boldsymbol{W}_{D'}^{\text{new}} = \boldsymbol{\gamma} \boldsymbol{W}_{D'}. \end{aligned} \tag{12}$$

Slimming the dictionary will not affect the performance of the original SR model according to Fig. 5. Information embedded in the dictionary is sparse enough, and a well-trained

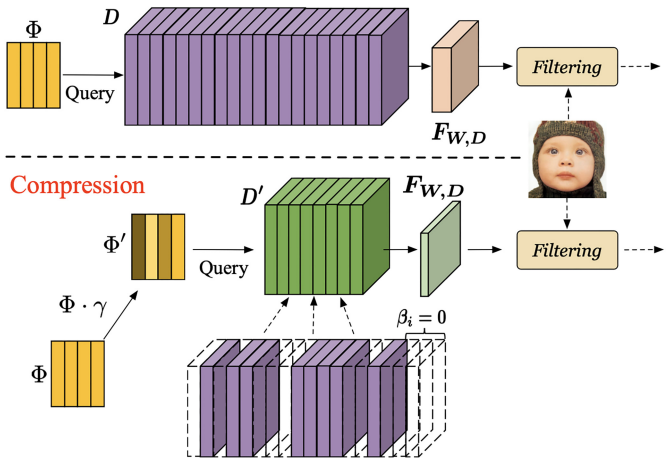

Fig. 4. Visual illustration of dictionary slimming, the upper flow represents original dictionary query and filtering, namely, stage 3 + stage 4 in Fig. 2, The flow below demonstrates the slimming process of the dictionary query.

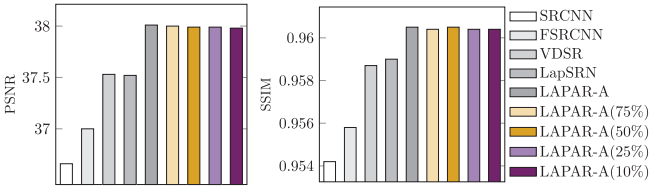

Fig. 5. SISR performance of our model with different dictionary compression ratios in comparison with other SR methods. LAPAR-A (Per.%) represents our model with dictionary size shrunk to Per.%. PSNR means peak signal-to-noise ratio. SSIM means structural similarity index measure. PSNR and SSIM are two common metrics to measure the quality of images. The higher, the better.

model can capture useful information even with some items being zero-out. In experiments, we show the dictionary can be slimmed to 10% of its original size without noticeable accuracy loss. For a fair comparison, the compressed model also outperforms other widely used SR models, e.g., [35] and [36].

B. Constraint-Based Optimization of Deployments on GPU

Although the slimming of the dictionary size may accelerate the dictionary query to some extent, the following filtering operation is still a bottleneck of the inference latency. The filtering operation comprises a Hadamard product of two tensors and a reduce-sum on the channel to flatten the tensor into a 2-D image. Such computations are common in SR tasks but neglected by the current mainstream deployment tools. In this section, we propose a domain-specific low-level design by utilizing the parallelizing mechanism of GPU to improve the computation throughput from a hardware perspective. An example of the proposed computation engine is shown in Fig. 6.

First, we will discuss how to implement the Hadamard product and reduce-sum in a parallel style within the current inference flow. During the inference stage, all tensor data (including images and filters) is always consecutively stored in (N, C, H, W) style in linear memory addresses, which denotes `(batch size, channel, height, width)`

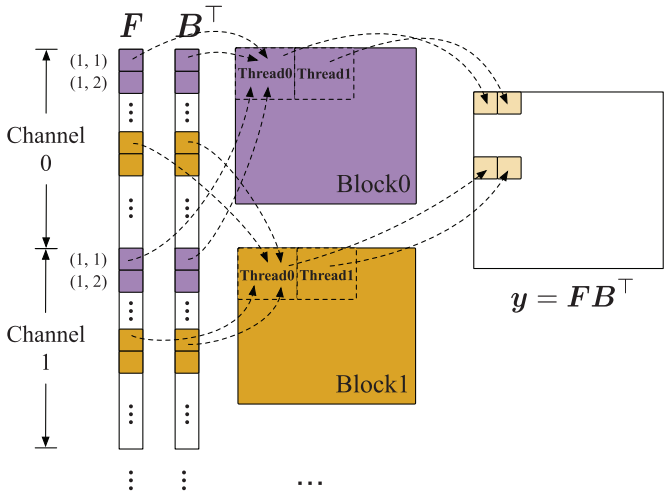

Fig. 6. Example of the proposed computation engine for image filtering operation.

dimensions of each tensor. Given that both computation operations are spatially independent, the calculation of value at each 2-D location index on the map of size $H \times W$ can be assigned to different computation units separately, as visualized in Fig. 6. The color of the data represents which block they are sent to compute. For example, the data in purple and data in orange are separately assigned to block 0 and block 1. We deliberately manipulate the thread assignment to make data at the consecutive 2-D location being assigned to the same block as much as possible. More specifically, we try to assign consecutive data to consecutive threads. Meanwhile, data with the same index but from various channels are assigned to the same thread. For example, the data at location $(1, 1)$ and data at $(1, 2)$ are assigned to thread 0 and thread 1 in block 0 accordingly for all channels. The Hadamard Product starts with element-wise multiplication of $\boldsymbol{F}$ and $\boldsymbol{B}$. Then, the products at each channel are accumulated to render the final HR images. Both steps can be computed for each 2-D location index parallelly. In other words, the computation assigned to each thread is equivalent to the multiplication of two vectors, where each pair of vectors are data along the channel dimension from $\boldsymbol{F}$ and $\boldsymbol{B}$ at the same 2-D location. In our implementation, each thread applies addition and multiplication simultaneously by adding the intermediate product of each channel to the final result. All threads are launched to run the same code piece in parallel delicately to avoid getting stuck in the paradox of thread divergence [24]. Moreover, we also consider the cache-memory mechanism and try to avoid frequent interaction in our design. As shown in Fig. 3, each SM holds an exclusive shared memory/L1 cache for all blocks inside. We manage to minimize the cache miss rate by assigning consecutive data from memory to consecutive blocks of the same SM for each channel.

On the other hand, parallelism cannot be extended infinitely. We need to consider complicated limits from both the hardware level and programming model level, which significantly affect the performance of the parallel implementation. First, the number of assigned threads, blocks, and etc. will determine

the computation patterns. However, since the computing capabilities of different GPU devices can be distinct and various, the corresponding limits on the thread/block configuration vary as well. For further illustration, let us take the edge device NVIDIA Jetson Xavier NX as an example which has a Volta microarchitecture embedded GPU. At the hardware level, each NX device has six SMs in the architecture. Each SM's shared memory size (on-chip memory) is fixed at 96 kB. There are four physical processing blocks inside each SM, where each processing block holds 16 FP32 cores, 8 FP64 cores, 16 INT32 cores, 2 Tensor cores, and a 64 kB shared register file. At the programming model level, the computation kernel is launched as a computation grid where each cell in the grid is a thread block. Note that the concept of thread block here is a virtual concept, which is not the same as the previous processing block. One thread block will be assigned to a single SM. While discussing hardware resource limit, we need to introduce the concept of warp, which is the basic execution unit in NVIDIA GPU that holds 32 consecutive threads. In the current SIMT architecture, each thread block will be further divided and assigned to many warps after being scheduled to an SM. The warp scheduling in GPU is orderless within each thread block. The only restriction is the number of active warps regarding the SM resources. Once a warp idles for the race conditions, the SM is free to schedule other available warps. The number of warps for a thread block can be determined as follows:

$$\text{Warps Per Block} = \left\lceil \frac{\text{Threads Per Block}}{\text{Warp Size}} \right\rceil \tag{13}$$

where Warp Size $= 32$ for mainstream NVIDIA GPUs. The number of warps in a thread block is also constrained by the programming model to fit the sizes of warp schedulers, instruction registers, and etc. Besides, memory bound also needs to be considered. There are two levels of data sharing among parallel executions: 1) sharing data in the shared register files among the parallel threads in the same processing block and 2) sharing data among the processing blocks in the same SM. Both may cause a race condition: multiple threads accessing the same data in the 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 3-D. The threads blocks can also be regarded as 3-D 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 can be formulated as

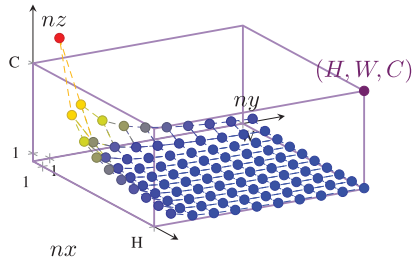

Fig. 7. Visualized solution space. The solution points below the dotted points are legal configurations.

$$\begin{aligned} &T_r = (H \times W \times C)/(S \times P \times R) \\ &T \le \min(T_r, T_{sm}) \\ &n_x \times n_y \times n_z \le WS \times P \times T \\ &1 \le n_x \le H \\ &1 \le n_y \le W \\ &1 \le n_z \le C. \end{aligned} \tag{14}$$

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 Fig. 7. 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., [37].

Although the search space is compressed by the constraints listed above, the optimization process may still not be fast enough because each candidate configuration's on-board compilation and execution is considerably time-consuming. We choose Bayesian Optimization to better sample candidate values nx, ny, and nz than grid search or manual tuning [38], [39], which shows superior efficiency in searching by utilizing the full information gained from past experiments. The core components of Bayesian optimization consist of a probabilistic surrogate model $S(\cdot)$ to fastly evaluate inference latency, and an acquisition function $A(\cdot)$ to select the most informative candidates which hold largest upper confidence bound (UCB) [40]. First, we randomly sample a small batch of configurations $\hat{N}$ from search space $\mathcal{N}$ to initialize the surrogate model, which is a Gaussian process (GP) model in our implementation. The complete searching process is shown in Algorithm 2. After several rounds of sampling and updating the surrogate model, we choose the final configuration with the best inference speed.

IV. Adaptive 8-Bit Quantization

As shown in Fig. 1, The inference latency of the dictionary query and filtering step was significantly reduced

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 Optimimal 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: $(n_x, n_y, n_z)^*$;

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, SR is a fine-grained task where the RGB value of HR images is recovered. We adopt the 8-bit 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: First, 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. Second, 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 (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

$$\alpha^* = \underset{\alpha}{\text{argmin}}\, D_{KL}(\boldsymbol{act}_{\text{8-bit}} || \boldsymbol{act}_{FP32}) \tag{15}$$

where ***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.

Nevertheless, KL-divergence-guided clipping scale cannot avoid apparent performance degradation, despite its effectiveness in minimizing layer-wise information loss from quantization. As we conduct the naive scale quantization, both peak signal-to-noise ratio (PSNR) and structural similarity index measure (SSIM) drop significantly even lower than baseline methods, as shown in Fig. 8. Nagel et al. [27] first raised doubts over the most common rounding-to-nearest step used

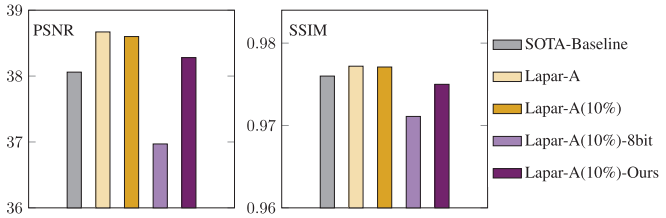

Fig. 8. SR performance of 8-bit inference in comparison with SOTA baseline SR methods and original Lapar-A model. Lapar-A(10%) represents the model with the dictionary shrunk to 10% size. Lapar-A(10%)-8 bit represents the model with naive 8-bit scale quantization. Lapar-A(10%)-ours is our adaptive 8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

in integer quantization. Since the quantization clipping and rounding can be regarded as a weight value shift to the original full-precision network, which leads to a value difference in coefficient vector $\boldsymbol{\Phi}$ extracted from *LaparNet*, resulting in a performance degradation

$$\Psi(\boldsymbol{X}, \boldsymbol{W}, \Delta\boldsymbol{W}) = \text{loss}(\boldsymbol{X}, \boldsymbol{W} + \Delta\boldsymbol{W}) - \text{loss}(\boldsymbol{X}, \boldsymbol{W}) \tag{16}$$

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 [27]. The objective of minimizing the accuracy loss can be derived

$$\begin{aligned}\underset{\Delta\boldsymbol{W}}{\text{argmin}}\ & \mathbb{E}\left[\text{loss}(\boldsymbol{X}, \boldsymbol{W} + \Delta\boldsymbol{W}) - \text{loss}(\boldsymbol{X}, \boldsymbol{W})\right] \\ \approx \underset{\Delta\boldsymbol{W}}{\text{argmin}}\ & \mathbb{E}\Big[\Delta\boldsymbol{W}^\top \nabla_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W}) \\ & + \Delta\boldsymbol{W}^\top \nabla^2_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W}) \Delta\boldsymbol{W}\Big].\end{aligned} \tag{17}$$

The rounding-to-nearest step focuses on minimizing the weight shift $\boldsymbol{\Delta W}$, which may not be the optimal choice. The first term in (17) is negligible as the convergence of training leads the first-order gradient $\nabla_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W})$ to be close to 0. Therefore, the objective of quantization falls to the second term where $\nabla^2_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W})$ is Hessian matrix. Reference [41] has mathematically proved this second-order error optimization can be transformed into (18), where $\Delta\boldsymbol{\Phi}$ denotes the value difference in coefficient vector $\boldsymbol{\Phi}$ before and after the quantization and $\nabla^2_{\boldsymbol{\Phi}} \text{loss}(\boldsymbol{X}, \boldsymbol{W})$ is the Hessian regarding coefficient vector $\boldsymbol{\Phi}$. Such transformation enables the optimization as the coefficient vector difference $\Delta\boldsymbol{\Phi}$ is much easier to acquire during the forwarding inference step

$$\begin{aligned}\underset{\Delta\boldsymbol{W}}{\text{argmin}}\ & \mathbb{E}\Big[\Delta\boldsymbol{W}^\top \nabla^2_{\boldsymbol{W}} \text{loss}(\boldsymbol{X}, \boldsymbol{W}) \Delta\boldsymbol{W}\Big] \\ \approx \underset{\Delta\boldsymbol{W}}{\text{argmin}}\ & \mathbb{E}\Big[\Delta\boldsymbol{\Phi}^\top \nabla^2_{\boldsymbol{\Phi}} \text{loss}(\boldsymbol{X}, \boldsymbol{W}) \Delta\boldsymbol{\Phi}\Big].\end{aligned} \tag{18}$$

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

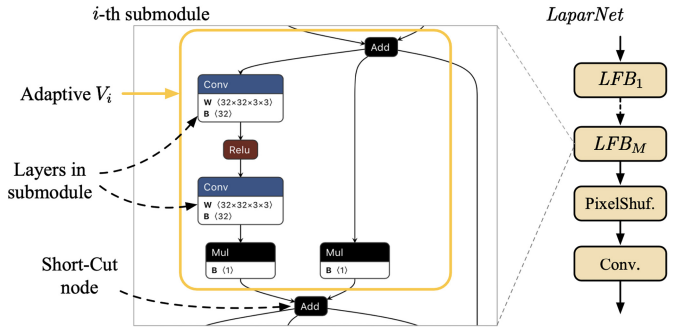

Fig. 9. Visualization of quantization submodule for $\boldsymbol{V}$ optimization. We divide *LaparNet* with each submodule either stacked with less than 4 layers or ended with an Short-Cut node. In practice, most submodules share the same structure with a residual block in *LaparNet*.

term $\boldsymbol{V}$ added before clipping

$$\boldsymbol{W} + \Delta \boldsymbol{W} = \text{clip}\left(\text{round}\left(\frac{W}{s}\right) + \sigma(\boldsymbol{V}), -\epsilon, \epsilon\right) \tag{19}$$

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 $\sigma(\cdot)$ [42] is to force the adaptive value within range $[0, 1]$ and has nonvanishing gradient around 0 or 1. The overall optimization objective is

$$\begin{aligned} \underset{\boldsymbol{V}}{\text{argmin}} \ & \mathbb{E}\left[\Delta \boldsymbol{\Phi}^{\top} \nabla^2_{\boldsymbol{\Phi}} \text{loss}(\boldsymbol{X}, \boldsymbol{W}) \Delta \boldsymbol{\Phi}\right] \\ & + \lambda \sum_i \left(1 - |2\sigma(\boldsymbol{V}_i) - 1|^{\tau}\right). \end{aligned} \tag{20}$$

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

Although we have the formulation in (20), it is still challenging to optimize all $\boldsymbol{V}$ for the whole network considering the size of the model. We optimize the above objective function with a finer granularity by dividing the network into a series of quantization submodules and tuning the $\boldsymbol{V}$ of each submodule iteratively. In this way, we are able to reduce the complexity of optimizing $\boldsymbol{V}$ and concentrate more on the submodule-wise quantization error. As the size of the submodule shrinks down, the computational complexity is smaller to optimize $\boldsymbol{V}$ for each submodule. On the other hand, finer-grained submodules may possibly lead to local optimal for each submodule and deviate from global optimal solution. As shown in Fig. 9, we choose the submodule size delicately after different trials to reach the highest restored accuracy. Given the network structure of *LaparNet*, we choose each single residual block as a single submodule and quantize submodules one by one with the first term in (20). The first term of objective in (20) is approximated by the $l2$-norm of difference in quantized/unquantized output tensors $\|\boldsymbol{\Phi}_{\boldsymbol{W}+\Delta \boldsymbol{W}} - \boldsymbol{\Phi}_{\boldsymbol{W}}\|_F^2$ at each submodule.

V. Experimental Results

A. Experimental Setup

Hardware Implementation: We validate our high-performance accelerator on NVIDIA Jetson Xavier NX, an edge device with embedded GPU. Metrics, including acceleration ratio and SR 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 1900 MHz, and 4/6 core 1400 MHz. The clock frequency of the GPU processor is 1100 MHz. 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 [54].

Dataset: The proposed accelerator is evaluated on common single image SR (SISR) Set5 [55], Set14 [3], B100 [56], Urban100 [57], Manga109 [45] dataset.

B. Performance Evaluation

To validate the speed-up effectiveness of our design, we make comparisons with the original model on both NVIDIA Jetson Xavier NX and RTX 2080 Ti devices. To show the generalization ability and soundness, we measure the inference in different input frame sizes and different scale ratios. The results are in 32-bit floating point precisions, and running times are shown in Table I. We successfully realize SR with the output of 540P quality to real-time inference. Our design surpasses the PyTorch with 352.27% faster inference on 2080 Ti. Overall, Our design surpasses TensorRT with 144.49% inference speed on 2080 Ti and 156.28% on Jetson Xavier NX on average. Furthermore, in comparison with TensorRT, our accelerator realizes an impressive +27.45%∼77.56% speed-up. It is interesting to notice Jetson Xavier NX presents more obvious acceleration than 2080 Ti. This implicitly verifies that our design is more effective for embedded GPUs with limited computation and communication resources.

In Table II, we also compare the quality of SR results with other famous models as baseline methods to show the performance of our design. The performance metrics are PSNR and SSIM, both are the mainstream common metrics to measure the qualities of restored HR frames. Higher values indicate better performance. Although our model is a compressed version with 90% of the dictionary slimmed out while the other baselines are not, it is still superior to almost all of the baseline models on both of these two metrics.

To verify the sparsity in the dictionary and the corresponding acceleration potentials, we conduct an ablation study on the slimming ratio in Fig. 10. 100% denotes the original dictionary without compression. It shows that for different scales, the time costs decrease linearly to the compression ratio. The

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

TABLE II
COMPARISONS ON MULTIPLE BENCHMARK DATASETS OF OUR MODEL (FULL-PRECISION) AND OTHER POPULAR SR NETWORKS. THE DICTIONARY IN OUR MODEL IS COMPRESSED TO 10% OF ORIGINAL SIZE FOR EVALUATION. PERFORMANCE METRICS ARE PSNR/SSIM. THE MAC IS CALCULATED CORRESPONDING TO A 1280×720 HR IMAGE. BOLD: **BEST** RESULTS

Scale	Method	Params	MAC	Set5	Set14	B100	Urban100	Manga109
$\times 2$	SRCNN [43]	57K	53G	36.66/0.9542	32.42/0.9063	31.36/0.8879	29.50/0.8946	35.74/0.9661
	FSRCNN [35]	12K	6G	37.00/0.9558	32.63/0.9088	31.53/0.8920	29.88/0.9020	36.67/0.9694
	VDSR [36]	665K	613G	37.53/0.9587	33.03/0.9124	31.90/0.8960	30.76/0.9140	37.22/0.9729
	DRRN [44]	297K	6,797G	37.74/0.9591	33.23/0.9136	32.05/0.8973	31.23/0.9188	37.92/0.9760
	LapSRN [45]	813K	30G	37.52/0.9590	33.08/0.9130	31.80/0.8950	30.41/0.9100	37.27/0.9740
	SRFBN-S [46]	282K	680G	37.78/0.9597	33.35/0.9156	32.00/0.8970	31.41/0.9207	38.06/0.9757
	FALSR-A [47]	1,021K	235G	37.82/0.9595	33.55/0.9168	32.12/0.8987	31.93/0.9256	-
	SRMDNF [48]	1,513K	348G	37.79/0.9600	33.32/0.9150	32.05/0.8980	31.33/0.9200	-
	TPSR [49]	60K	14G	37.38/0.9583	33.00/0.9123	31.75/0.8942	30.61/0.9119	-
	SESR-M11 [50]	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**
$\times 3$	SRCNN [43]	57K	53G	32.75/0.9090	29.28/0.8209	28.41/0.7863	26.24/0.7989	30.59/0.9107
	FSRCNN [35]	12K	5G	33.16/0.9140	29.43/0.8242	28.53/0.7910	26.43/0.8080	30.98/0.9212
	VDSR [36]	665K	613G	33.66/0.9213	29.77/0.8314	28.82/0.7976	27.14/0.8279	32.01/0.9310
	DRRN [44]	297K	6,797G	34.03/0.9244	29.96/0.8349	28.95/0.8004	27.53/0.8378	32.74/0.9390
	SelNet [51]	1,159K	120G	34.27/0.9257	30.30/0.8399	28.97/0.8025	-	-
	CARN [52]	1,592K	119G	34.29/0.9255	30.29/0.8407	29.06/0.8034	28.06/0.8493	-
	SRFBN-S [46]	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**
$\times 4$	SRCNN [43]	57K	53G	30.48/0.8628	27.49/0.7503	26.90/0.7101	24.52/0.7221	27.66/0.8505
	FSRCNN [35]	12K	5G	30.71/0.8657	27.59/0.7535	26.98/0.7150	24.62/0.7280	27.90/0.8517
	VDSR [36]	665K	613G	31.35/0.8838	28.01/0.7674	27.29/0.7251	25.18/0.7524	28.83/0.8809
	DRRN [44]	297K	6,797G	31.68/0.8888	28.21/0.7720	27.38/0.7284	25.44/0.7638	29.46/0.8960
	LapSRN [45]	813K	149G	31.54/0.8850	28.19/0.7720	27.32/0.7280	25.21/0.7560	29.09/0.8845
	CARN [52]	1,592K	91G	32.13/0.8937	28.60/0.7806	27.58/0.7349	26.07/0.7837	-
	SRFBN-S [46]	483K	1,037G	31.98/0.8923	28.45/0.7779	27.44/0.7313	25.71/0.7719	29.91/0.9008
	TPSR [49]	61K	4G	31.10/0.8779	27.95/0.7663	27.15/0.7214	24.97/0.7456	-
	SplitSR (HI=2) [53]	94k	99G	31.53/**0.8950**	28.18/**0.7887**	27.28/**0.7458**	25.20/0.7704	-
	SESR-M11 [50]	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**

dictionary query and filtering can be up to roughly $\times 20$ faster than the original version.

C. Quantized 8-Bit Analysis

We show the practicability of our adaptive 8-bit PTQ 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 III that even compressed with quantized 8-bit inference, our approach still achieves comparable or even better performance

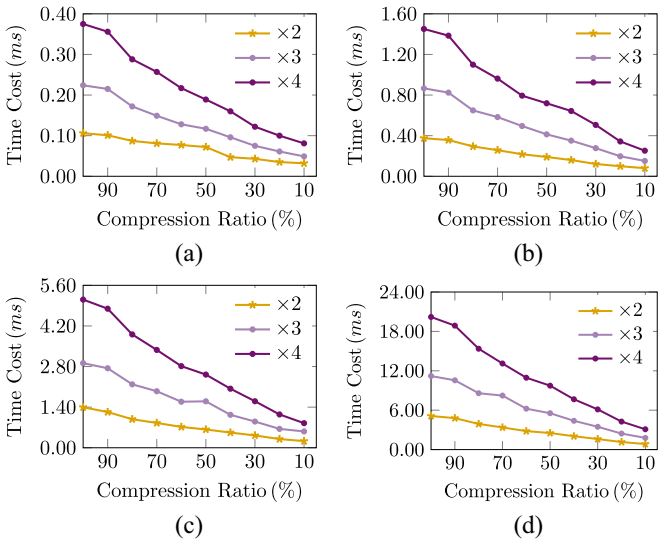

Fig. 10. Time consumption of the dictionary query and filtering with different compression ratios. Different input image sizes and scaling factors (from 2 to 4) are evaluated. (a) Input size 64×64. (b) Input size 128×128. (c) Input size 180×320. (d) Input size 360×640.

TABLE III
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**

with other SOTA baseline methods implemented in full precision.

We also analyze the inference speed of our implemented 8-bit inference. As shown in Table IV, 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 IV. This way, such a full-precision kernel will not introduce much overhead to the processing speed.

TABLE IV
QUANTIZED 8-BIT ACCELERATION ANALYSIS IN COMPARISON WITH FULL-PRECISION ICCAD21 [1]. "MUL" DENOTES THE TIME CONSUMPTION OF "MUL" KERNEL AND CORRESPONDING DATA TRANSFORMATION

Input size	Scale	ICCAD21 (mul) [1]	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**%

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

D. Discussions

We demonstrate the remarkable performance of domain-specific high-performance SR accelerator with all the experiment results, which is more effective for edge embedded GPU NVIDIA Jetson Xavier NX with limited power and hardware resources. Within our approaches, the key idea is to conquer the difficulties from dictionary learning algorithms used in SR task, which hold particular memory and computation patterns and are not feasible for existing deployment toolkits. Another challenge is the large sizes of both the input frame and the intermediate feature map, which bring huge memory pressure to hardware.

Moreover, as shown in Fig. 11, it is easy to notice a performance-scale tradeoff for SR task. Different models should be delicately selected for different hardware resources and scenarios. Although some models are compact with fast inference and large frames, they are somewhat limited in accuracy and may not restore enough texture details in HR frames. For a fair comparison of SR task scaling up to 720P with ×4 ratio under our consistent GPU hardware setting, although some super-lightweight SOTA models such as SESR-M11 [50] or TPSR [49] can infer under 3 ms per frame, the HR accuracy may not meet some requirement, as shown in Fig. 11. On the other hand, another SOTA baseline SplitSR [53] is more performant than former two baselines, however, with a much higher inference time 19 ms. Meanwhile, our dictionary-based approach shows a ruling performance over all other baselines within 16 ms per frame.

TABLE V
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

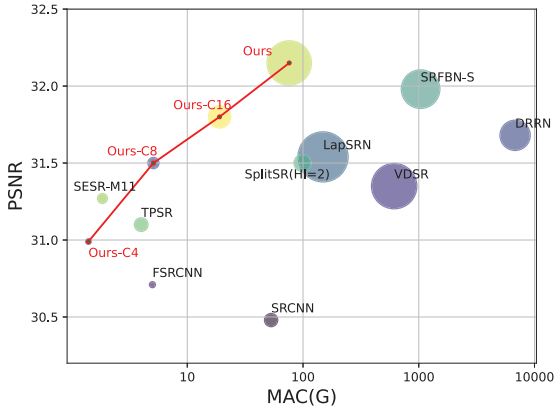

Fig. 11. Comparison between our proposed model after compression and other lightweight methods (<1M parameters) on Set5 for ×4 setting. Circle sizes are set proportional to the numbers of parameters. "Ours" denotes our full-size LAPAR model and "Ours-C16, 8, 4" denote our LAPAR models with embedding channel number reduced from 32 to 16, 8, and 4.

In addition, we added a Pareto curve in Fig. 11 to show the flexibility of our deployed dictionary-based algorithm. To plot the curve, we shrink our model from full-size to 1.56%, which is close to the size of the smallest baseline model. We reduce the model size by directly halving the embedding channel number of all residual blocks at each point from 32 to 16, 8, and 4. As shown in Fig. 11, our deployed algorithm achieves the highest performance-scale efficiency at channels 32, 16, and 8, except for the extremely compressed 4-channel case, which has slightly poorer PSNR than SESR-M11 [50].

To the best of our knowledge, our proposed accelerator is the first to achieve superior performance on SR applications on edge embedded GPUs.

VI. CONCLUSION

In this article, we design a domain-specific high-performance accelerator for SR 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.

REFERENCES

[1] W. Zhao et al., "A high-performance accelerator for super-resolution processing on embedded GPU," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2021, pp. 1–9.
[2] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 2, pp. 295–307, Feb. 2016.
[3] C. Ledig et al., "Photo-realistic single image super-resolution using a generative adversarial network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 4681–4690.
[4] S. Y. Kim, J. Oh, and M. Kim, "JSI-GAN: GAN-based joint super-resolution and inverse tone-mapping with pixel-wise task-specific filters for UHD HDR video," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 11287–11295.
[5] W. Li, X. Tao, T. Guo, L. Qi, J. Lu, and J. Jia, "MuCAN: Multi-correspondence aggregation network for video super-resolution," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 335–351.
[6] W. Li, K. Zhou, L. Qi, N. Jiang, J. Lu, and J. Jia, "LAPAR: Linearly-assembled pixel-adaptive regression network for single image super-resolution and beyond," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2020, pp. 20343–20355.
[7] C. Hao et al., "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.
[8] C. Guo et al., "Balancing efficiency and flexibility for DNN acceleration via temporal GPU-systolic array integration," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.
[9] H. Li, M. Bhargav, P. N. Whatmough, and H.-S. P. Wong, "On-chip memory technology design space explorations for mobile deep neural network accelerators," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, p. 131.
[10] Q. Sun, C. Bai, H. Geng, and B. Yu, "Deep neural network hardware deployment optimization via advanced active learning," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2021, pp. 1510–1515.
[11] X. Wei et al., "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2017, p. 29.
[12] R. Pinkham, S. Zeng, and Z. Zhang, "QuickNN: Memory and performance optimization of k-d tree based nearest neighbor search for 3D point clouds," in *Proc. IEEE Int. Symp. High Perform. Comput. Architect. (HPCA)*, 2020, pp. 180–192.
[13] Y. Bai and W. Wang, "ACPNet: Anchor-Center based person network for human pose estimation and instance segmentation," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, 2019, pp. 1072–1077.
[14] S. Cao et al., "Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity," in *Proc. ACM Int. Symp. Field-Program. Gate Arrays (FPGA)*, 2019, pp. 63–72.
[15] "NVIDIA TensorRT." Accessed: Mar. 15, 2021. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/index.html
[16] "Intel MKL-DNN." Accessed: Mar. 15, 2021. [Online]. Available: https://github.com/oneapi-src/oneDNN

[17] Y. Jung, Y. Choi, J. Sim, and L.-S. Kim, "eSRCNN: A framework for optimizing super-resolution tasks on diverse embedded CNN accelerators," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.
[18] T. Dai, J. Cai, Y. Zhang, S.-T. Xia, and L. Zhang, "Second-order attention network for single image super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 11065–11074.
[19] Z. Luo, Y. Huang, S. Li, L. Wang, T. Tan, "Unfolding the alternating optimization for blind super resolution," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 33, 2020.
[20] J. Yang, Z. Wang, Z. Lin, S. Cohen, and T. Huang, "Coupled dictionary training for image super-resolution," *IEEE Trans. Image Process.*, vol. 21, no. 8, pp. 3467–3478, Aug. 2012.
[21] Y. Romano, J. Isidoro, and P. Milanfar, "RAISR: Rapid and accurate image super resolution," *IEEE Trans. Comput. Imag.*, vol. 3, no. 1, pp. 110–125, Mar. 2017.
[22] P. Getreuer, I. Garcia-Dorado, J. Isidoro, S. Choi, F. Ong, and P. Milanfar, "BLADE: Filter learning for general purpose computational photography," in *Proc. ICCP*, 2018, pp. 1–11.
[23] C. Wang, Z. Li, and J. Shi. "Lightweight image super-resolution with adaptive weighted learning network." 2019, *arXiv:1904.02358*.
[24] J. Cheng, M. Grossman, and T. McKercher, *Professional CUDA C Programming*. New York, NY, USA: Wiley, 2014.
[25] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," 2020, *arXiv:2004.09602*.
[26] M. Nagel, M. V. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1325–1334.
[27] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, "Up or down? Adaptive rounding for post-training quantization," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 7197–7206.
[28] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 7948–7956.
[29] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2014, *arXiv:1412.7024*.
[30] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015, *arXiv:1510.00149*.
[31] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey, "Ternary neural networks with fine-grained quantization," 2017, *arXiv:1705.01462*.
[32] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 3123–3131.
[33] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 4107–4115.
[34] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 1398–1406.
[35] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 391–407.
[36] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 1646–1654.
[37] T. Chen et al., "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. USENIX Symp. Operat. Syst. Design Implement. (OSDI)*, 2018, pp. 578–594.
[38] J. R. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration," in *Proc. Conf. Neural Inf. Process. Syst. (NIPS)*, 2018, pp. 7587–7597.
[39] Y. Bai, X. Yao, Q. Sun, and B. Yu, "AutoGTCO: Graph and tensor co-Optimize for image recognition with transformers on GPU," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, 2021, pp. 1–9.
[40] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," 2009, *arXiv:0912.3995*.
[41] Y. Li et al., "BRECQ: Pushing the limit of post-training quantization by block reconstruction," in *Proc. Int. Conf. Learn. Represent.*, 2020.
[42] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," in *Proc. Int. Conf. Learn. Represent.*, 2018.
[43] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 184–199.
[44] Y. Tai, J. Yang, and X. Liu, "Image super-resolution via deep recursive residual network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2790–2798.
[45] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep Laplacian pyramid networks for fast and accurate super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5835–5843.
[46] Z. Li, J. Yang, Z. Liu, X. Yang, G. Jeon, and W. Wu, "Feedback network for image super-resolution," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 3867–3876.
[47] X. Chu, B. Zhang, H. Ma, R. Xu, and Q. Li, "Fast, accurate and lightweight super-resolution with neural architecture search," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, 2021, pp. 59–64.
[48] K. Zhang, W. Zuo, and L. Zhang, "Learning a single convolutional super-resolution network for multiple degradations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 3262–3271.
[49] R. Lee et al., "Journey towards tiny perceptual super-resolution," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 85–102.
[50] K. Bhardwaj et al., "Collapsible linear blocks for super-efficient super resolution," in *Proc. Mach. Learn. Syst.*, vol. 4, 2022, pp. 529–547.
[51] J.-S. Choi and M. Kim, "A deep convolutional neural network with selection units for super-resolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2017, pp. 1150–1156.
[52] N. Ahn, B. Kang, and K.-A. Sohn, "Fast, accurate, and lightweight super-resolution with cascading residual network," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 256–272.
[53] X. Liu et al., "Splitsr: An end-to-end approach to super-resolution on mobile devices," *Proc. ACM Interact. Mobile Wearable Ubiquitous Technol.*, vol. 5, no. 1, pp. 1–20, 2021.
[54] "Simple-SR." Accessed: Dec. 11, 2020. [Online]. Available: https://github.com/dvlab-research/Simple-SR
[55] M. Bevilacqua, A. Roumy, C. Guillemot, and M. L. Alberi-Morel, ''Low-complexity single-image super-resolution based on nonnegative neighbor embedding," in *Proc. BMVC*, 2012, pp. 1–10.
[56] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, vol. 2, 2001, pp. 416–425.
[57] C. Ledig et al., "Photo-realistic single image super-resolution using a generative adversarial network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 105–114.

Wenqian Zhao received the B.Sc. degree in computer science and engineering from The Chinese University of Hong Kong, Hong Kong, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering.

His research interests include machine learning for VLSI design automation and hardware-aware deep-learning acceleration.

Yang Bai received the B.S. degree in telecommunications engineering from Xidian University, Xi'an, China, in 2017, and the master's degree in computer science from the Chinese Academy of Sciences University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His research interests focus on the optimization for deep neural network training and inference via compilation techniques.

Qi Sun (Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2022.

He is currently a Postdoctoral Associate with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY, USA, since Fall 2022. His research interests include machine learning in electronic design automation, design space exploration, and deep neural network hardware acceleration.

Dr. Sun's work has been recognized with an ICCAD Best Paper Award, the Bronze Medal of the ICCAD Student Research Competition, and the Best Paper Award Nomination of DATE.

Wenbo Li (Graduate Student Member, IEEE) received the B.Eng. and M.S. degrees from Shanghai Jiao Tong University, Shanghai, China, in 2016 and 2019, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His primary research interests lie in low-level computer vision.

Mr. Li's paper was selected in the CVPR 2022 Best Paper Finalists. He also serves as a Reviewer for IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, *International Journal of Computer Vision*, IEEE TRANSACTIONS ON IMAGE PROCESSING, CVPR, ICCV, ECCV, ICLR, and NeurIPS.

Haisheng Zheng received the B.Eng. degree in internet of things engineering from Tiangong University, Tianjin, China, in 2020.

He is currently a Researcher with Shanghai AI Laboratory, Shanghai, China, since 2022. From 2020 to 2022, he was a Research and Development Engineer with SmartMore, Hong Kong. His research interests include artificial intelligence, embedded systems, high performance computing, and IC design.

Nianjuan Jiang (Member, IEEE) received the B.S. and Ph.D. degrees in electrical and computer engineering from National University of Singapore, Singapore, in 2007 and 2012, respectively.

From 2012 to 2016, she was with the Advanced Digital Sciences Center, a Singapore-based research center of the University of Illinois at Urbana–Champaign. From 2017 to 2020, she was with a retail 3-D+AI tech startup company in Shenzhen, China, as the Research and Development Lead. Since 2020, she has been with SmartMore Company Ltd., Hong Kong, as a Research and Development Lead. Her research interest includes computer vision, computer graphics, 3-D vision, and computational imaging.

Jiangbo Lu (Senior Member, IEEE) received the Ph.D. degree from Katholieke Universiteit Leuven, Leuven, Belgium, in 2009.

From 2009 to 2016, he was a Senior Research Scientist with the Advanced Digital Sciences Center, a Singapore-based research center of the University of Illinois at Urbana–Champaign. From 2017 to 2019, he was with a retail 3-D+AI tech startup company in Shenzhen, China, as the Chief Technology Officer (CTO). Since 2020, he has been with SmartMore Company Ltd., Hong Kong, as the Co-Founder and a CTO. Since 2021, he has also been an Adjunct Professor with the South China University of Technology, Guangzhou, China. His research interests include computer vision, 3-D vision, and image processing.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received nine Best Paper Awards from DATE 2022, ICCAD 2021 & 2013, ASPDAC 2021 & 2012, ICTAI 2019, *Integration, the VLSI Journal* in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, and six ICCAD/ISPD contest awards. He is an Editor of *IEEE TCCPS Newsletter*. He has served as the TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees.

Martin D. F. Wong received the B.Sc. degree in mathematics from the University of Toronto, Toronto, ON, Canada, in 1979, and the M.S. degree in mathematics and the Ph.D. degree in computer Science from the University of Illinois at Urbana–Champaign (UIUC), Champaign, IL, USA, in 1981 and 1987, respectively.

He was a Faculty Member with the The University of Texas at Austin (UT-Austin), Austin, TX, USA, from 1987 to 2002 and UIUC from 2002 to 2018. He was a Bruton Centennial Professor of CS with UT-Austin and an Edward C. Jordan Professor of ECE with UIUC. From August 2012 to December 2018, he was the Executive Associate Dean of the College of Engineering, UIUC. In January 2019, he joined The Chinese University of Hong Kong, Hong Kong, as the Dean of Engineering and a Choh-Ming Li Professor of Computer Science and Engineering. His main research interest is in electronic design automation (EDA). He has published around 500 papers and graduated over 50 Ph.D. students in EDA.

Prof. Wong is a Fellow of ACM.