# BAQE: Backend-Adaptive DNN Deployment via Synchronous Bayesian Quantization and Hardware Configuration Exploration

Wenqian Zhao, Shuo Yin, Chen Bai, *Member, IEEE*, Zixiao Wang, and Bei Yu, *Senior Member, IEEE*

***Abstract*—Efficiently deploying deep learning (DL) algorithms on different hardware backends has become a time-consuming challenge. Achieving ultimate inference efficiency on hardware requires both algorithm-level model compression techniques, such as model quantization, and hardware-level optimization, such as operation reconfiguration and scheduling. In this article, we propose *BAQE*, a unified deployment framework that bridges the gap between algorithm-level and backend-level optimization. By constructing a global search space, we can synchronously optimize both the model quantization settings and backend configuration parameters. To accelerate this laborious and time-consuming process, we propose a searching strategy based on multiobjective Bayesian optimization (BO) using a Gaussian model with deep kernel learning as the surrogate model. More importantly, BAQE can easily adapt to various backends with different hardware resources efficiently and effectively. Each inner step of the optimization process is aware of the genuine hardware resources, ensuring that all accuracy/latency metrics and historical knowledge/feedback are evaluated directly on the device within each iteration. Empirical results demonstrate that our approach achieves both superior inference time and accuracy with a faster optimization process.**

***Index Terms*—Hardware-aware acceleration, model quantization, neural network compression.**

## I. Introduction

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

Quantization, compared to other DNN model compression techniques like model pruning or knowledge distillation, is generally more dependent on hardware. The hardware support for quantized data types strongly influences the available quantized bit-width candidates and acceleration ratio. For example, NVIDIA's Ampere GPU architecture with third generation tensor core provides unprecedented acceleration for tensor operations at Int1/4/8, bfloat16, and FP16 precision.

Received 2 February 2024; revised 14 August 2024; accepted 3 October 2024. Date of publication 9 October 2024; date of current version 21 March 2025. This work was supported in part by Research Grants Council of Hong Kong, SAR under Grant CUHK14208021, and in part by the MIND Project under Grant MINDXZ202404. This article was recommended by Associate Editor X. Lin. *(Corresponding author: Bei Yu.)*

The authors 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).

Digital Object Identifier 10.1109/TCAD.2024.3476450

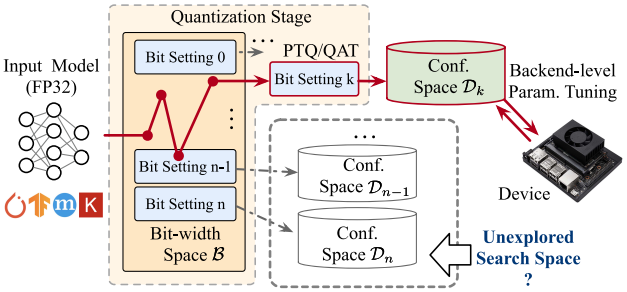


Fig. 1. Current quantization model deployment is a 2-stage flow composed of quantization at the model level and compilation optimization at the backend level. PTQ/QAT denote post-training quantization and quantization aware training. Decoupling these two steps may lead to potential unexplored search space. Each "Bit setting" denotes a bit-width assignment for layers in the model.

On the other hand, ARM Neon with its single instruction multiple data (SIMD) architecture can support 8, 16, 32, and 64-bit vector operations. Additionally, the effectiveness of quantization on acceleration may vary depending on the available hardware resources. With versatile bit-width support of advanced hardware, algorithm designers have more flexibility to apply mixed-precision quantization for different layers. Many previous works [1], [2], [3], [4], [5], [6], [7] have dug into this direction to find the optimal bit-width setting for each layer of a DNN model.

However, selecting bit-width and tuning the model is not the final stage in the real scenario of quantization deployment. As shown in Fig. 1, 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.

Although some researchers [1], [8], [9], [10], [11] 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 [12] 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

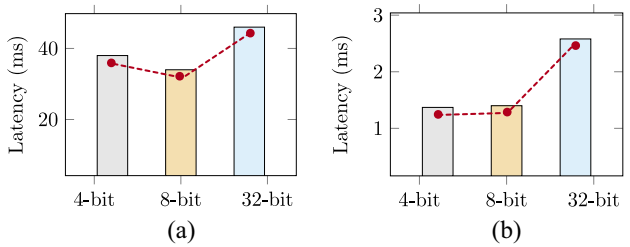


Fig. 2. Inference latency comparison with example model Resnet-50 and data type in Int4, Int8, and Int32. Red dotted line indicates the speed-up ratio from quantization, which differs for Jetson Orin NX and RTX 3070. Speed-up from quantization on NX is marginal and Int8 inference is even slightly faster than Int4. In comparison, the RTX3070 speed-up is more noticeable. (a) Jetson orin NX. (b) RTX 3070.

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 along with the predetermined model bit-width setting. As visualized in the lower area in Fig. 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 suboptimal 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 Fig. 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 [13] relies on a simulator to retrieve the power/latency of an FPGA-based accelerator. HAWQ-V3 [11]'s ILP solver requires precollected 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. First, 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 deployment flow to simultaneously search for quantization bit-width settings and backend configuration. First, we built a unified search space, including model-level and backend-level parameters. To realize the goal of adapting to different hardware backends, we assume that no prior knowledge of the backend is available in our scenario. First, we use the transductive experimental design (TED)-based-sampling strategy to coarsely sample from the search space as the initial data points. Afterward, we evaluate the initial sample to fit a Gaussian process (GP) with deep kernel learning (DKL-GP) as our surrogate model. Then we build multiobjective Bayesian optimization (BO) to efficiently search for better-bit-width settings and backend configuration together. At each searching iteration, we auto-tune the scaling

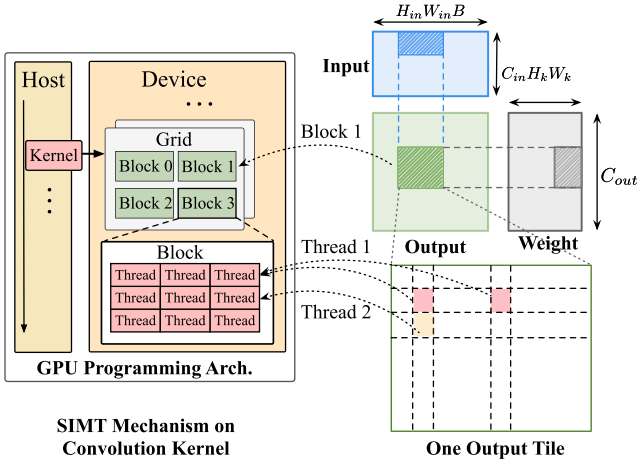


Fig. 3. Example of flatten 2-D convolution workload partition on CUDA programming architecture with hierarchical parallelism. 2-D matrix operations are split into tiles, which are assigned to multiple thread blocks in GPU device.

factor and model weight as well as loop scheduling parameters with a small batch of calibration data to test the potential performance of each sampled candidate.

The contribution of this article is listed as follows.

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

## II. Preliminaries

### A. DNN Model Quantization

The key idea of quantization is to replace the numerical representation FP32 with half-precision FP16 or even Int8/Int4 form, which is commonly formulated as

$$\text{quantize}(r) = \text{round}(r/S) - Z \tag{1}$$

where quantize$(\cdot)$ denotes the quantization operation: the original weight/activation $r$ in FP32 precision is first scaled with a real value factor $S$ and then rounded to the nearest integer value (with truncation). $Z$ is the integer value zero point to calibrate the mapping value shift from floating to integer range.

Many previous works, such as [11], [14], [15], [16], [17], [18], [19], and [20], have utilized weights or activation quantization in low-bit precision. Among various approaches, integer quantization has shown prominent efficacy in real deployment scenarios for its hardware friendliness. In general, fixed-point arithmetic operations, such as multiplication and addition, are much simpler and faster with only shift and

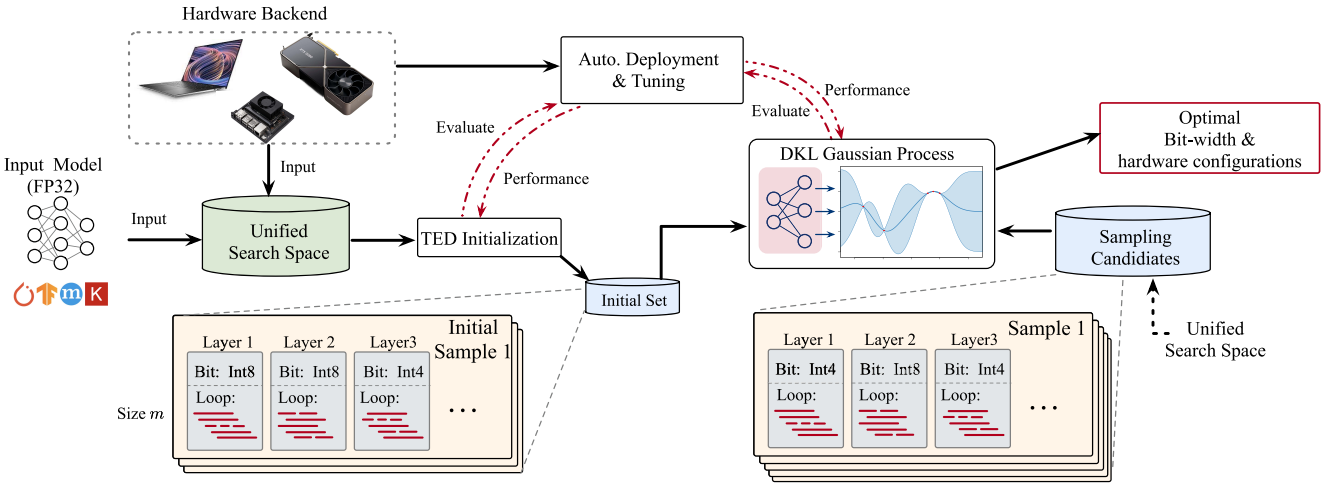


Fig. 4. Overview of the framework of BAQE. Our framework searches for bit-width settings $\boldsymbol{b}$ and backend configuration parameters $\boldsymbol{hp}$ (loop scheduling parameters) simultaneously. Quantization scaling factor $\boldsymbol{s}$ and network parameters are tuned on the device at each searching step.

add, casting off time-consuming steps, such as normalization, de-normalization, or exponent alignment in floating point arithmetic.

Quantization may possibly lead to accuracy degradation, especially for layers that are sensitive to the precision deduction. Mixed-precision [1], [2], [3], [4], [5], [6], [7] approaches have been proposed to compensate for this problem by assigning different bit precision for each layer. This fine-grained bit-width distribution leads to a precision assignment problem, where the number of choices grows exponentially as the network grows deeper. Wu et al. [21] used the differentiable NAS (DNAS) method to fastly select the bit candidates while [1] chose bit-width based on the trace of the Hessian matrix. Yao et al. [11] used integer learning programming to optimize bit-width.

### B. Hardware-Aware Quantization

One of the ultimate goals of DNN quantization is to improve inference latency, which is strongly hardware-dependent. Many methods [1], [8], [9], [10], [11] considered hardware resource limit and reformulated this into a constrained optimization problem. The objective becomes minimizing, such as information loss or accuracy degradation, while maintaining memory/speed metrics, such as model size or GFLOPs. Wang et al. [13] used reinforcement learning (RL) to determine the bit-width setting while mapping to a simulator to retrieve energy/latency feedback. Yao et al. [11] extended the awareness by directly compiling all layers at each bit-width on hardware to get real latency before precision selection.

### C. Hardware Backend Deployment Optimization

Nowadays, most DL layers are dense tensor operations that can be decomposed into multilevel for-loop representation. The original execution order of these loops may not reach the utmost utilization of computation units or memory bandwidth. Many approaches, such as loop reordering, loop unrolling, or loop tiling, can enhance execution efficiency or reduce memory cache miss. The same effect occurs if the data layout is transformed to align with the loop order. In the case of the GPU Backend, the selection on "tile size" is the number of parallel threads assigned to each CUDA block to execute, which needs to be properly selected to balance communication and parallelism as shown in Fig. 3.

Rescheduling these loops with the operations above can search for the optimal backend configuration by reorganizing the low-level implementation. Several approaches [22], [23], [24], [25], [26], [27] in the DNN compilation field have proposed automatically tuning these backend parameters. Chen et al. [22], [23] first used a simulated annealing (SA) algorithm as a search strategy. Mu et al. [24] and Zheng et al. [26] used guided genetic algorithm and evolution search to explore the backend configuration search space.

## III. Methodology

### A. Overview of BAQE Framework

Fig. 4 visually describes the framework of BAQE. First, we build a unified large search space in Section III-B with model-level/backend-level parameters to optimize. Second, a TED-based method is applied in Section III-C as the initial step to generate initial samples on bit-width $\boldsymbol{b}$ and backend configuration parameters $\boldsymbol{hp}$ that widely spread in the search space. Afterward, BAQE utilizes multiobjective BO algorithm to search for optimal $\boldsymbol{b}$ and $\boldsymbol{hp}$ (Section III-D). All performance feedback is evaluated on the device with automatic deployment and tuning at each iteration (Section III-F).

### B. Unified Global Search Space

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

$$\mathcal{S} = (b_1, b_2, \ldots, b_l, w_1, w_2, \ldots, w_l \; s_1, s_2, \ldots, s_l, hp_1, hp_2, .., hp_j) \quad (2)$$

where $\boldsymbol{b}$: $[b_1, b_2, \ldots, b_l]^\top$ are the bit-width of each layer $l$ to quantize and $\boldsymbol{s}$: $[s_1, s_2, \ldots, s_l]^\top$ are the corresponding scaling factors. $\boldsymbol{w}$: $[w_1, w_2, \ldots, w_l]^\top$ are the model weights. $\boldsymbol{hp}$:

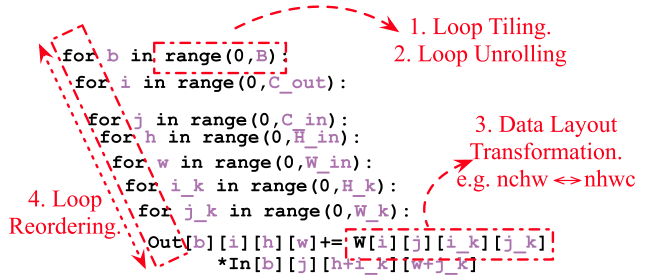


Fig. 5. Example of pseudo implementation of the 2-D convolution kernel. Annotations in red color denote backend-level optimization methods on this multiloop code piece, which can increase parallelism or optimize memory read/write efficiency.

$[hp_1, hp_2, .., hp_j]^\top$ 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 $b_1, b_2, \ldots, b_l$ and $hp_1, hp_2, .., hp_j$ are discrete variables while $s_1, s_2, \ldots, s_l$ are continuous. In correspondence, our framework chose both auto-tuning and discrete sampling strategies.

Concerning the algorithm-level search space, the bit-width $\boldsymbol{b}$: $[b_1, b_2, \ldots, b_l]^\top$ selection range for each layer encompasses Int4, Int8, and Float32, maintaining a discrete and uniform framework across all layers in all models. As outlined in the initial manuscript, all three platforms are equipped with GPU architectures featuring third generation tensor cores that facilitate Int4/Int8 quantized inference, while retaining the original Float32 as a viable option. Each layer $i$ possesses its individual bit-width selection denoted by $b_i$, as referenced in Section III-B. The search space for scaling factors $s_1, s_2, \ldots, s_l$ and model parameter weights $\boldsymbol{w}$: $[w_1, w_2, \ldots, w_l]^\top$ comprises floating-point variables with continuous exploration within the range of $[FP32_{\min}, FP32_{\max}]$.

To clarify what backend configuration parameters $\boldsymbol{hp}$ represents, Fig. 5 is the visualization of a 2-D convolution. As depicted in Fig. 5, reordering the loop topology enables the inner loops to fetch and compute consecutive data on memory without affecting the output value, therefore, relieving memory pressure as the cache miss rate is reduced. Data layout with dimension "nchw" renders more efficient memory fetch when loop on $H_{\text{in}}/W_{\text{in}}$ is inside loop $C_{\text{in}}$. Loops can also be unrolled into tiles of multithreads and mapped to hardware thread blocks for parallel execution, as shown in Fig. 3.

In the context of hardware-level exploration, the search space presents greater complexity, prompting the utilization of abstract notation $hp_1, hp_2, \ldots, hp_j$ as detailed in Section III-B. To elucidate further, the hardware configuration search space mirrors that of AutoTVM [22], encompassing an array of discrete or binary variables. Illustrated in Fig. 5, for each layer, the initial variable pertains to data layout, with discrete options $nhwc, nchw, hwcn$ representing memory data arrangement. Additionally, tile size variables for width, height, and channel dimensions of each layer $l_i$: $[\text{tile}_w, \text{tile}_h, \text{tile}_c]i$ are discrete integers within ranges like $1, 2, \ldots, w_i$. Other hardware variables for each layer $l_i$ consist of a binary variable indicating the use of tensor cores: 1, 0. For CUDA program generation, variables, such as CUDA block size and CUDA thread number within each block, are represented as 3-D variables $[b_x, b_y, b_z, t_x, t_y, t_z]i$. Finally, the execution order of loops in each layer encompasses all permutations of execution variables denoted by $[n, \text{tile}_w, \text{tile}_h, \text{tile}_c, b_x, b_y, b_z, t_x, t_y, t_z]i$. The unrolling factor, with a search range of $[0, 1, \ldots fmax]$, enables loop level unfolding for parallelism during the compilation stage, with a maximum threshold set at 512.

### C. TED-Based Initial Sampling

As we introduced above, BAQE is a backend-adaptive framework where backend domain knowledge is not available. Considering the scenario that no domain-specific knowledge of input hardware backend is available at the beginning stage, collecting genuine latency/accuracy data with online tuning/evaluation is inevitable. Therefore, the acceleration burden on the optimization flow falls to efficient sampling and searching.

TED is an efficient sampling strategy to optimize the sample quality for regression without label/prediction value of the sample data, which aligns with our problem background because the hardware platform is a opaque system with no prior knowledge. The main objective of experimental design is to select a set of candidates $\{(\boldsymbol{b}, \boldsymbol{hp})_0, (\boldsymbol{b}, \boldsymbol{hp})_1, \ldots, \}$ that are maximally informative. Here, we denote each candidate as $\boldsymbol{x}$

$$\boldsymbol{x}_i = \left[\boldsymbol{b}^\top, \boldsymbol{hp}^\top\right]_i = \left[b_1, b_2, .., b_l, hp_1, hp_2, \ldots, hp_j\right]. \quad (3)$$

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

$$\min_{\boldsymbol{w}} \Phi(\boldsymbol{w}) = \sum_{i=1}^{m} \left(\boldsymbol{w}^\top \boldsymbol{x}_i - y_i\right)^2 + \mu \|\boldsymbol{w}\|^2. \quad (4)$$

Here, $m$ is the number of samples, $\boldsymbol{w}$ is the regression weight, and $\mu$ is a regularization coefficient. Given maximum likelihood estimate $\hat{\boldsymbol{w}} = argmin_{\boldsymbol{w}} \Phi(\boldsymbol{w})$, the estimation error $\boldsymbol{w} - \hat{\boldsymbol{w}}$ has 0 mean and covariance $\sigma^2 C_w$. Here, $\sigma$ is a constant and $C_w$ is the inverse Hessian of $\Phi(\boldsymbol{w})$

$$\begin{aligned} \mathbf{C_w} &= \left([\boldsymbol{B}, \boldsymbol{HP}]^\top [\boldsymbol{B}, \boldsymbol{HP}] + \mu \mathbf{I}\right)^{-1} \\ &= \left(\boldsymbol{X}^\top \boldsymbol{X} + \mu \mathbf{I}\right)^{-1} \end{aligned} \quad (5)$$

where all $m$ sampled bit-width and backend parameters are $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m]$. Note that the covariance indicates the confidence of the estimation. Higher confidence means higher informativeness of sampled data. In other words, maximizing the trace $Tr(C_w)$ can lead to higher informativeness. However, one of the potential flaws is that $C_w$ may not align with the quality of all other data points to search.

When sample $m$ candidate $(\boldsymbol{b}, \boldsymbol{hp})$ from search space $\mathcal{S}$, let $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, \ldots, ]^\top$ denotes all data points to search, each $\boldsymbol{v}_i$ is a $(\boldsymbol{b}, \boldsymbol{hp})$ bit-width/backend configuration combination. The regression error of predicted performance on $\boldsymbol{V}$ is $\sigma C_{\boldsymbol{V}}$ where

$$\begin{aligned} \mathbf{C}_{\boldsymbol{V}} &= \boldsymbol{V} \mathbf{C_w} \boldsymbol{V}^\top \\ &= \boldsymbol{V}\left([\boldsymbol{B}, \boldsymbol{HP}]^\top [\boldsymbol{B}, \boldsymbol{HP}] + \mu \mathbf{I}\right)^{-1} \boldsymbol{V}^\top \\ &= \boldsymbol{V}\left(\mathbf{X}^\top \mathbf{X} + \mu \mathbf{I}\right)^{-1} \boldsymbol{V}^\top. \end{aligned} \quad (6)$$

**Algorithm 1** TED-Based Initial Sampling

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

TABLE I
VARIABLE DEFINITION AT INITIAL SAMPLING STAGE

| Notation | Variable Definition |
|---|---|
| $\boldsymbol{X}$ | Initial samples: sampled set from search space. |
| $m$ | Size of $\boldsymbol{X}$: Initial sample size. |
| $\mathbb{S}$ | Unified search space Equation (2). |
| $\boldsymbol{V}$ | Global data: target candidates in $\mathbb{S}$ where $\boldsymbol{X}$ are sampled. |
| $\boldsymbol{K}$ | Distance Matrix: Kernalized distance: $\boldsymbol{K}_{ij} = k(\boldsymbol{v}_i, \boldsymbol{v}_j)$. |
| $\boldsymbol{K}_{\boldsymbol{Vx}}, \boldsymbol{K}_{\boldsymbol{xV}}$ | Rows/columns of distance matrix: $(\boldsymbol{K}_{\boldsymbol{Vx}})_i = k(\boldsymbol{v}_i, \boldsymbol{x})$. |
| $k(\cdot)$ | Non-linear kernel function: $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ as distance between $\boldsymbol{x}_i$ $\boldsymbol{x}_j$. |
| $\boldsymbol{x}^*$ | Optimal candidate sampled at each iteration. |

With the original goal of optimizing sample informativeness, we replace the $C_w$ and switch to minimize trace of $C_{\boldsymbol{V}}$. In addition, we try to add nonlinearity with a kernel function $k$: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = [(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)/2\sigma^2]$, replacing original sample data matrix with element distance. In the covariance matrix, all data points $\boldsymbol{V}$ is replaced with $K_{\boldsymbol{VX}}$ that $(K_{\boldsymbol{VX}})_{ij} = k(\boldsymbol{v}_i, \boldsymbol{x}_j)$. The sample product $\boldsymbol{X}^\top \boldsymbol{X}$ is replaced with $K_{\boldsymbol{XX}}$ where $(K_{\boldsymbol{XX}})_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Then our initial sampling objective becomes

$$\begin{aligned} \max_{\boldsymbol{X}} \quad & \mathrm{Tr}\Big(\mathbf{K}_{\boldsymbol{VX}}(\mathbf{K}_{\boldsymbol{XX}} + \mu\mathbf{I})^{-1}\mathbf{K}_{\boldsymbol{XV}}\Big) \\ \text{s.t.} \quad & \boldsymbol{X} \subset V, |\boldsymbol{X}| = m. \end{aligned} \tag{7}$$

After deriving the optimization objective, the initial sampling strategy of BAQE is constructed and listed in Algorithm 1. In order the further illustrate the initial sampling clearly, here we necessarily list definitions of all variable notations in Table I. Note that here the kernel function $k(\cdot, \cdot)$ is replaceable with any nonlinear distance function. The initial sampling of BAQE iteratively chooses the most informative bit-width/backend parameters to sample $\boldsymbol{x}^*$ while dynamically updating $K$ for each iteration.

In real implementation, the dimension of $\boldsymbol{K}$ and $\boldsymbol{V}$ are not constructed by traversing the complete search space, as this would not be practical or feasible in real applications. The construction of the target data matrix $\boldsymbol{V}$ is instead done using a very first batch of 500 representative samples from the search space. This initial batch is likely chosen to capture the diversity and distribution of the overall search space, acting as a good proxy for the full dataset. Similarly, the initial sampling of 10 points for the $\boldsymbol{K}$ matrix is selected from among these 500 batch representatives. This ensures that the $\boldsymbol{K}$ matrix, of size $500 \times 500$, spans a meaningful subspace of the overall search space. The target data $\boldsymbol{X}$ will then be a $500 \times dim_{\boldsymbol{x}}$ matrix, where $dim_{\boldsymbol{x}}$ is the dimensionality of the input data. By constructing $\boldsymbol{K}$ and $\boldsymbol{V}$ in this way, we can efficiently approximate the full search space without the need to traverse it exhaustively. This makes the algorithm scalable and practical for real-world applications where the full search space may be prohibitively large or inaccessible.

### D. Multiobjective Exploration

Once the initial dataset of $(\boldsymbol{b}, \boldsymbol{hp})$ is collected, we then explore the search space. It is nontrivial to obtain optimal quantized bit widths and backend configuration parameters to tradeoff the on-device inference time, accuracy, and model size within a limited time budget. The reason lies in two folds. First, the concrete form $f$ between a post-quantized DNN model and its on-device inference time and model accuracy is complex and unknown. Second, tuning and evaluating a DNN model's accuracy takes some time. We propose multiobjective exploration based on BO to solve the problem. BO consists of a *surrogate model* and an *acquisition function*. The initial dataset generated from the proposed algorithm (Section III-C) is used to build the surrogate model. The GP is often chosen as the surrogate model to represent the concrete form $f$ mentioned earlier. The acquisition function guides the multiobjective exploration direction. It decides which candidate solutions (bit widths and loop scheduling parameters) should be applied for the DNN model based on predictions from the surrogate model. So, we can only evaluate the quantized DNN model's on-device inference time, accuracy, *etc.*, with the candidate solution to save the overall exploration runtime. BO selects a new candidate solution iteratively. Next, we illustrate our proposed surrogate model and the acquisition function for multiobjective exploration.

*Surrogate Model:* Regarding the exploration focusing on multiobjectives, we propose the DKL-GP as the surrogate model. Without loss of generality, suppose a set of candidates $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, 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 $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^\top$, where $y_{il}$ refers to the value of the $i$th input $\boldsymbol{x}_i$ for $l$th metric. DKL-GP places a GP prior $f_l$ for these metrics, respectively, as shown in

$$y_{il} \sim \mathcal{N}\Big(f_l(\boldsymbol{x}_i), \sigma_l^2\Big) \tag{8}$$

where $\sigma_l$ is the variance for the $l$th metric. We define a positive semi-definite matrix $\boldsymbol{K}^f$ as the interobjective similarities and $\boldsymbol{K}^x$ as the covariance function over the input $\boldsymbol{x}_i$. The interobjective similarity demonstrates an observation of one metric can affect the predictions of another metric. For example, a DNN model with a large size often incurs a high-on-device inference time due to more computations required. DKL-GP models correlations between metrics with

$$\mathrm{cov}\big(f_l(\boldsymbol{x}), f_k(\boldsymbol{x}')\big) = \boldsymbol{K}^f_{lk}\boldsymbol{K}^x\big(\boldsymbol{x}, \boldsymbol{x}'\big) \tag{9}$$

where $\boldsymbol{K}^f_{lk}$ characterizes the interobjective similarities between metrics $l$ and $k$. Introducing (9) in DKL-GP helps to capture correlations between metrics [28]. And these correlations are important in trading off the on-device inference time, model

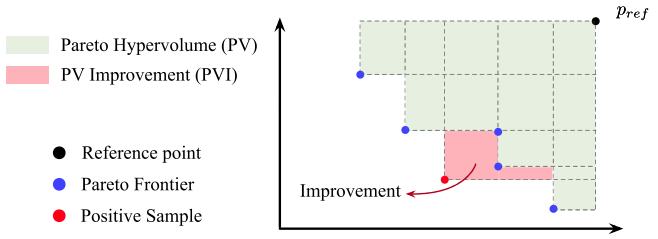


Fig. 6. 2-D visualization of the idea of Pareto hypervolume (PV) and EPVI.

accuracy, and size in the exploration procedure. For example, given a new input $\boldsymbol{x}^*$, DKL-GP predict its metric mean values with

$$f_l(\boldsymbol{x}^*) = \left(\boldsymbol{K}^f_{.l} \otimes \boldsymbol{K}^x(\boldsymbol{x}^*, \boldsymbol{X})\right)^\top \Sigma^{-1}\boldsymbol{Y}$$
$$\Sigma = \boldsymbol{K}^f \otimes \boldsymbol{K}^x(\boldsymbol{X}, \boldsymbol{X}) + \boldsymbol{D} \otimes \boldsymbol{I} \tag{10}$$

where $\boldsymbol{K}^f_{.l}$ denotes the $l$th column of $\boldsymbol{K}^f$, $\otimes$ is the Kronecker product, $\boldsymbol{D}$ is the diagonal matrix in which the diagonal elements are $\sigma_l^2$, and $\boldsymbol{I}$ is the identify matrix. We stack multilayer perceptrons to parameterize the kernels for $\boldsymbol{K}^f$ and $\boldsymbol{K}^x$. The formulated deep kernels are more robust than previous kernel formulations like automatic relevance determinant (ARD). Equation (11) shows the example deep kernels

$$\text{cov}(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left(-\boldsymbol{\beta}^\top \Sigma^{-1} \boldsymbol{\beta}\right)$$
$$\boldsymbol{\beta} = \left(\phi(\boldsymbol{x}, \boldsymbol{w}) - \phi(\boldsymbol{x}', \boldsymbol{w})\right) \tag{11}$$

where $\boldsymbol{w}$ denotes weights of multilayer perceptrons, and $\phi$ are nonlinear transformations.

*Acquisition Function:* BAQE chooses the expected improvement of Pareto hypervolume (EPVI) as the acquisition function to guide the multiobjective search. During the BO process, the performance evaluation on bit-width $\boldsymbol{b}$ and backend configuration parameters $\boldsymbol{hp}$ includes on-device inference time, accuracy, and model size. However, the GP can only handle a single performance metric as evaluation. Given such a situation, we determine to formulate a new objective that is sufficient to cover all three dimensions of the objective. This is somewhat tricky as all these three factors are negatively correlated. For example, compressing the model size or bit-width may lead to accuracy loss. Increasing accuracy with higher-precision costs higher-inference latency. In this way, our optimization goal switch from co-optimization to finding a good tradeoff point with overall optimality. In BAQE, we pick the EPVI as the acquisition function to efficiently guide the optimization process to the Pareto frontier of three objectives.

We demonstrate the idea of Pareto Hypervolume and EPVI in Fig. 6 with a 2-D visualization. Given a reference point $p_{\text{ref}}$ in the 3-D space of (accuracy, latency, model size) denoted as $\Omega(x, y, z)$, Pareto Hypervolume is the "half-arch" space bounded by $p_{\text{ref}}$ and the optimal points (blue dots in Fig. 6) on Pareto frontier $\mathcal{P}$. Such space volume is a *Lebesgue measure* of the Pareto optimality of 3 dimensions, which is formulated as

$$PV_\Omega(\mathcal{P}, p_{\text{ref}}) = \sum_{p\in\Omega}\left[\mathbb{1}(p \succcurlyeq p_{\text{ref}})\left[1 - \prod_{p'\in\mathcal{P}} \mathbb{1}\left(p \not\preccurlyeq p'\right)\right]\right] \tag{12}$$

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 $p_{\text{ref}}$ is the green area in Fig. 6. $p_{\text{ref}}$ is manually selected on space $\Omega$.

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 $\Omega$, that surpasses the old frontier (red area in Fig. 6)

$$\text{PVI}_\Omega\left(\mathcal{P}, p_{\text{ref}}, p^+\right) = \text{PV}_\Omega\left(\mathcal{P} \cup p^+, p_{\text{ref}}\right) - \text{PV}_\Omega(\mathcal{P}, p_{\text{ref}}) \tag{13}$$

while the EPVI is

$$\text{EPVI}_\Omega(\mathcal{P}, p_{\text{ref}}) = \mathbb{E}_{p^+|\Omega}\left[\text{PVI}_\Omega\left(\mathcal{P}, p_{\text{ref}}, p^+\right)\right]$$
$$= \sum_{p^+\in\Omega} \text{Prob}\left(p^+|\Omega\right) \cdot \text{PVI}_\Omega\left(\mathcal{P}, p_{\text{ref}}, p^+\right). \tag{14}$$

Our acquisition function is built based on (14) to select the point $p^+$ that maximizes the expected improvement. Probability $\text{Prob}(p^+|\Omega)$ is modeled as the multiobjective GP for (accuracy, latency, model size). BAQE optimizes by iteratively selecting $\boldsymbol{x}^*$ from candidates $\boldsymbol{X}$ such that the likelihood of DKL-GP $f(\cdot)$ is maximally increased

$$\boldsymbol{x}^* = \underset{f(\boldsymbol{x})=p^+}{\text{argmax}}\, \text{EPVI}_\Omega(\mathcal{P}, p_{\text{ref}}). \tag{15}$$

### *E. Complete Exploration Flow*

In addition to the visualization and description on each component above, we illustrate more on the complete flow description here. The multiobjective exploration of BAQE is shown in Algorithm 2.

First, the TED-based initial sampling generate the first batch $\boldsymbol{X}_{\text{init}}$ as the training dataset to fit the DKL-GP surrogate model. In our experiments, the selection on the initial sample size $m$ is not stressful, any number larger than 10 shows well convergence. In our case, we choose 10 as $m$ for both cases of benchmark model: Resnet-18 and Resnet-50. As discussed in Section III-C in this article, the TED-based sampling maximizes the trace $K_{VX}(K_{XX} + \mu I)^{-1}K_{XV}$, which indirectly minimize the covariance of the performance regression model. From a high-level understanding, this step will reduce the prediction uncertainty of the regression model with a more sparse sampling on the search space $\mathcal{S}$.

Afterwards, the BO process keeps sampling among the all candidates $\boldsymbol{V}$, picking the sample $(\boldsymbol{b}, \boldsymbol{hp})$ combination with maximized expected Pareto hypervolume improvement. Each candidate is embedded with a multilayer perceptron model $\phi(\cdot)$. In our experiment, this multilayer perceptrons $\phi(\cdot)$ is composed of four linear layer with the first three layers followed with a ReLU layer to break the linearity. The embedding dimensions of each linear layer are 1000, 500, 50, and 48. Each candidate $\boldsymbol{x}$ is embedded as feature before calculating the covariance in DKL-GP.

**Algorithm 2** BAQE Complete Exploration Flow

**Input:** Bit/backend parameters space $\mathbb{S}$, Stopping iteration number $N$, initial smaple size $m$, regularization coefficient $\mu$.
**Output:** Pareto optimal solution set $\boldsymbol{X}_{optim}$.
$\boldsymbol{X} \leftarrow \emptyset, \boldsymbol{V} \leftarrow \mathbb{S}$;
`//Initial sampling`
$(\boldsymbol{b}, \boldsymbol{hp}) \leftarrow \mathbf{TED}(\mathbb{S}, \mu, m)$;
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 $\boldsymbol{X}$;
    $\boldsymbol{x}^* \leftarrow argmax_{\boldsymbol{x} \in \boldsymbol{V}} EPVI(\boldsymbol{x}|\boldsymbol{V})$;
    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 $\boldsymbol{X}_{optim}$;

TABLE II
VARIABLE DEFINITION AT BAQE COMPLETE FLOW

| Notation | Variable Definition |
|---|---|
| $\mathbf{TED}(\cdot)$ | TED-based initial sampling in Algorithm 1. |
| $\boldsymbol{y}$ | Performance: hypervolume from evaluation. |
| $\boldsymbol{X}_{optim}$ | Optimal solutions on Pareto-frontier of $\boldsymbol{X}$. |
| $N$ | Iteration number for search. |

### F. Automatic Deployment and Tuning

At each BO iteration, a data sample $\boldsymbol{x} = (\boldsymbol{b}, \boldsymbol{hp})$ is derived from a large number of sample candidates. However, weight $\boldsymbol{w}$ and scaling factor $\boldsymbol{s}$ are still at the initial value. At each iteration, BAQE tunes the continuous $w_0, w_1, \ldots, w_l$ and $s_0, s_1, \ldots, s_l$. To attain performance feedback with genuine evaluation, BAQE inserts automatic deployment and tuning at each sampling stage to update $\boldsymbol{w}$ and $\boldsymbol{s}$. We refuse to fine-tune with fake quantization because simulated quantization usually shows a large accuracy bias as networks go deep.

Our framework is not restricted to PTQ or QAT. Our implementation follows baseline's QAT scheme for fair comparison. During tuning, only a small batch of calibration data is sufficient (0.01% of ImageNet). The fine-tuning stage does not necessarily need to be thorough. We only need an indicator of on-device performance as feedback instead of a well-trained model. In addition, we can also further tune the backend-level parameters during the on-device compilation and execution.

TABLE III
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 IV
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** |

In the end, BAQE omits some explored $(\boldsymbol{b}, \boldsymbol{s}, \boldsymbol{w}, \boldsymbol{hp})$ for given hardware backend, with vairable definition in Table II.

The hardware support of the specific tensor virtual machine (TVM) version utilized in this study has been thoroughly examined. the platform is capable of supporting both ARM and X86 CPU architectures, as well as CUDA-based GPUs. Furthermore, the integration of low-level virtual machine (LLVM) suggests the potential for adaptation to a wider range of heterogeneous accelerators. This broad hardware compatibility is a crucial factor, as it enables the deployment of the proposed solution across a diverse set of computing environments.

Given that the backend optimization process is grounded in the TVM framework, we can reasonably conclude that the search and co-optimization stages can be effectively implemented without the need for extensive customization. By leveraging the off-the-shelf toolkit provided by TVM, we can capitalize on the established optimization algorithms and techniques, thereby streamlining the development process and ensuring the robustness of the proposed solution.

## IV. EXPERIMENTS

*Platforms:* We evaluate the performance of all benchmark models on three different level platforms. The first one is System-on-Module (SoM) edge device NVIDIA Jetson Orin NX with 16-GB LPDDR5 RAM. This device is built on NVIDIA's Ampere architecture with 1024 CUDA cores and 32 tensor cores. It has a 8-core ARM Cortex-A78E CPU and three power mode: 10 W/15 W/20 W. For fair comparison, all experiments in this article are conducted on 15-W mode. The second platform is PC device with 14-core Intel i7-12700H CPU and 64-GB RAM. The PC device is equipped with NVIDIA RTX3070 GPU at 130-W power, which is also built on Ampere architecture with 5160 CUDA cores and

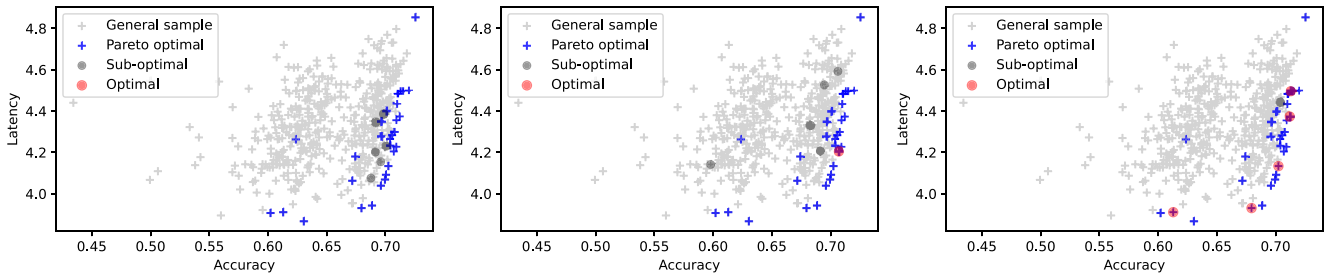


Fig. 7. Comparison of performance of searched samples. Blue dots denote all Pareto optimal points, which is a projection from (Accuracy, Latency, Size) to (Accuracy, Latency). Red dots are found in Pareto points by searching. Gray dots denote searched suboptimal points that are not on the Pareto frontier. Our BAQE found the most optimal points within the same search time. (a) Random search. (b) SA + XGBoost. (c) Ours.

184 tensor cores and 8-GB GDDR6 RAM on board. The last platform is a server platform with 10-core Intel Xeon 4210R CPU and 128-G RAM. It is equipped with NVIDIA RTX 3090 GPU with power 350 W, which is also on Ampere architecture and has 10496 CUDA cores and 328 tensor cores with 24-GB RAM on board. In our implementation, we choose all three platforms with GPU on the same architecture so they have the same computation units, such as CUDA core and tensor core, but differ in quantities. In this way, they can support same bit-with types and handle the same bit-width with same computations to eliminate other environment variance. We decided to use the 15-W power setting as it offered a fair basis for comparison among the available options (10 W, 15 W, 20 W) for all our experiments. We also examined the model's inference speed across these power settings and observed that higher power resulted in faster processing speeds, not affecting the performance finding in this work.

*Dataset:* We use ImageNet-1K as the dataset for both training and evaluation on the classification task. For fair accuracy evaluation, we use all 50 000 images in the validation set to test Top-1 score. For auto-tuning on model weight and scaling factor during optimization. We use only 0.01% of the training set for Resnet-18 and 0.05% for Resnet-50. For final evaluation on the quantized performance, we use the complete ImageNet-1K evaluation set.

*QAT/PTQ:* We chose QAT in practice but only slightly tune the model. We use 0.01% of ImageNet for Resnet-18 (0.05% for Resnet-50) with batch size 128 and LR 0.0001 for 1 epoch. It is decoupled from the exploration stage. Such small size can also be calibration set for PTQ as well.

*Implementation Details:* We choose the most commonly deployed model: Resnet-18 and Resnet-50, as the benchmark model. We build our BAQE framework using BOTorch and TVM [22], and we extend TVM [22] framework with the third gen. Tensor core support and more bit-width (Int4, Int8, FP32, and Int32) to enable practical mixed-precision quantization. In our implementation, each layer's weight and activation value are always at the same precision to align with genuine mul/add calculations on hardware. All accuracy Top-1 score and latency, including the baseline methods, are obtained in our own platform and environment for fair comparison.

The proposed algorithm can be applied to a range of DNN architectures beyond just ResNet, including other popular CNN models like [29] and InceptionV3 [30]. The results demonstrate consistent performance improvements across different model sizes. We choose ResNet-18 and ResNet-50 to align with the baseline HAWQ-V3 and to show the performance on models with different sizes. There is a common problem for all quantization techniques for Transformer-based models. Transformer-based models have more nonlinear layers, and the Softmax layer within each transformer block is particularly problematic. The computation of Softmax, $\sigma(z_i) = (e^{z_i}/[\sum_{j=1}^{J} e^{z_j}])$, involves an exponential operation, whose output value is very sensitive to the input of the exponent. Small value changes caused by quantization noise can lead to significant accuracy drops. Given the same quantization problem setting, the performance drop is significant on both the baseline and our methods. Many techniques have been proposed recently to address this, including custom data types with more exponent representation or new hardware modules. However, these are beyond the scope of this article.

TABLE V
SEARCHING COMPARISON ON RESNET-18 WITH/WITHOUT TED-BASED INITIAL SAMPLING. TRIAL NUMBER IS 40 FOR ALL EVALUATIONS HERE. "10 + 30" TRIAL NUMBER INCLUDE 10 FOR TED-BASED INITIAL SAMPLING AND 30 FOR FOLLOWING DKL-GP SEARCH

| Device | With TED | Trial Num. | Pareto Optimal |
|---|---|---|---|
| Jetson NX Orin (Edge device) | × | 40 | 3 |
| | ✓ | 10 + 30 | **5** |
| RTX 3070 (PC) | × | 40 | 4 |
| | ✓ | 10 + 30 | **6** |
| RTX 3090 (Server) | × | 40 | 1 |
| | ✓ | 10 + 30 | **4** |

TABLE VI
SEARCHING COMPARISON ON RESNET-50 WITH/WITHOUT TED-BASED INITIAL SAMPLING. TRIAL NUMBER IS 40 FOR ALL EVALUATIONS HERE. "10 + 30" TRIAL NUMBER INCLUDE 10 FOR TED-BASED INITIAL SAMPLING AND 30 FOR FOLLOWING DKL-GP SEARCH

| Device | With TED | Trial Num. | Pareto Optimal |
|---|---|---|---|
| Jetson NX Orin (Edge device) | × | 40 | 0 |
| | ✓ | 10 + 30 | **3** |
| RTX 3070 (PC) | × | 40 | 1 |
| | ✓ | 10 + 30 | **2** |
| RTX 3090 (Server) | × | 40 | 0 |
| | ✓ | 10 + 30 | **2** |

## A. Search Strategy Analysis

We evaluate BAQE's search efficiency in bit-width and backend configuration optimization with both visual and

TABLE VII
PERFORMANCE COMPARISON WITH SOTA HAWQ-V3 [11] 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 [11] | 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 [11] | 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 |

numerical comparisons. We record the first thirty sampling trials. Our baseline methods include random search in our unified search space and SA with XGBoost as the performance prediction model, which is the strategy of TVM [22].

*Visual Analysis:* As a visual demonstration, we choose the most representative Jetson Orin NX as backend and evaluate the search process on Resnet-18. We record the first thirty sampling trials. For clear visualization, we use the 2-D coordinate with latency and Top-1 score. As shown in Fig. 7, our BAQE search strategy can reach the most optimal bit-width and backend configurations with inference speed and accuracy. Even the rest suboptimal points are near the frontier, which indicates that BAQE possesses better-searching quality.

*Numerical Analysis:* We conduct numerical comparisons to evaluate both search quality and efficiency. To show the adaptiveness and generality of BAQE, we evaluate all three backends and two benchmark models in Tables III and IV. For efficiency, we collect normalized search time of collecting the same number of optimal points. We collect the number of Pareto optimal points within thirty trials for quality evaluation. In addition, we calculate another metric: the average distance of to reference set (ADRS). ADRS indicates how far the sample performance $\vec{Y}$ are away from the Pareto frontier points $\mathcal{P}$

$$\text{ADRS}(\boldsymbol{Y}, \mathcal{P}) = \frac{1}{|\boldsymbol{Y}|} \sum_{p \in \boldsymbol{Y}} \min_{p' \in \mathcal{P}} D(p, p') \tag{16}$$

where $D$ denotes Euclidean distance in the performance space. Results in Tables III and IV 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 initial 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. The searching process itself is rather fast, costing less than 1 min overhead. On the other hand, the compilation time cost at each iteration can be reduced given that a semi-optimized set of parameters are provided with searching algorithm.

*Ablation on TED-Based Initial Sampling:* Ablation on TED-Based Initial Sampling. In order to discuss the importance of TED-based initial sampling and the contribution of this step to the modeling effectiveness of the surrogate model, we conduct an ablation study to evaluate the search without TED-based sampling. As a fair comparison, we directly conduct the exact same deep-kernel-learning GP to explore the design space for the same number of trials. At the end, we list the search result comparison in Tables V and VI. We have conducted the same search process on the same three backends and two benchmark networks. The result on the Pareto-optimal points collected within 40 trials demonstrates the efficiency, indicating how far the searching process can reach the most optimal solutions. In Tables V and VI, the numbers clearly show that TED-based initial sampling helps the searching stage to find more optimal points, therefore demonstrating a positive effect on the modeling of the DKL-GP process.

TABLE VIII
COMPARISON WITH MULTIPLE BASELINES IN MODEL SIZE (MB), INFERENCE LATENCY (MS), AND TOP-1 SCORE (%) FOR RESNET-50 ON RTX3090 (SERVER)

| Method | Model Size | Latency | Top-1 Score |
|---|---|---|---|
| PACT [31] | 16.0 | 0.37 | 75.59 |
| HAQ [13] | 9.62 | NA | 75.0 |
| HAWQ-V3 [11] | 18.57 | 0.37 | 76.02 |
| AdaRound [18] | 24.5 | NA | 75.01 |
| BAQE (Ours) | 18.19 | 0.37 | 76.07 |
| | 16.43 | 0.37 | 75.98 |
| | 16.80 | 0.35 | 75.55 |

## B. Quantization Performance Analysis

We compare the performance of Pareto points of BAQE and the-state-of-the-art hardware-aware quantization HAWQ-V3 [11] on all three backends with on-device inference speed and accuracy in Table VII. HAWQ-V3 [11] uses the bit-width setting generated with ILP solver. Note that our search strategy output a group of samples on the Pareto frontier with optimal performance tradeoff. For each benchmark and backend, we list three samples with different performances in accuracy, speed, and model size, indicating BAQE's searching efficiency and ability. For a fair comparison, we compare with HAWQ-V3 [11] with our Pareto set in hypervolume, which is defined in Section III-D, we set the reference point as (0, 0, 0) and normalized the final value. As shown in Table IX, BAQE surpasses baseline with 6%–96% improvement.

More importantly, this result table aligns with our assumption: given different backends, higher latency does not necessarily lead to a slower inference. The interference of backend configuration breaks the linear relations between these metrics. Results also verify our statement that indirect

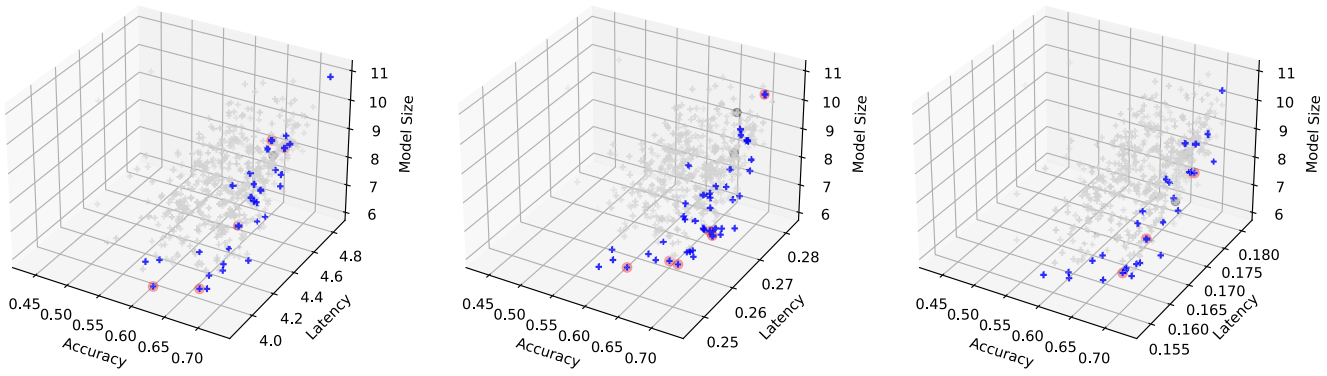


Fig. 8. 3-D performance distribution of all $(\boldsymbol{b}, \boldsymbol{hp})$ on three different backends with benchmark model Resnet-18. (a) Jetson Orin NX. (b) PC + RTX 2030. (c) Server + RTX 3090.

TABLE IX
COMPARISON IN NORMALIZED HYPERVOLUME

| | p | | yp | |
|---|---|---|---|---|
| Benchmark | Method | Jetson Orin NX | RTX 3070 | RTX 3090 |
| Resnet-18 | HAWQ-V3 [11] | 1 | 1 | 1 |
| | BAQE (Ours) | **1.06** | **1.08** | **1.13** |
| Resnet-50 | HAWQ-V3 [11] | 1 | 1 | 1 |
| | BAQE (Ours) | **1.96** | **1.52** | **1.53** |

TABLE X
EXAMPLE COMPARISON OF LATENCY WITH/WITHOUT LOOP-LEVEL OPTIMIZATION ON ORIN NX

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

metrics, such as the number of parameters, and FLOPs/BOPs are not trustworthy indicators; on-device evaluation with the ultimate feedback, such as on-device latency and accuracy, reflects the real performance. Other baselines are compared in Table VIII, we can tell that HAWQ-V3 remains the most performant baseline on the three objectives. Other baselines like HAQ or AdaRound require specific hardware or have adaptive bias terms, which add overhead and hinder inference latency. They also cannot go through normal backend optimization for proper latency evaluation.

To further investigate into the quantized model performance on real hardware and the relative performance, we collect the quantized model bit-width distribution of all optimal solutions in Table VII. As we looked into the Pareto-optimal results, nearly all layers seem to tend tohquantize to Int8 or Int4. This finding is within our anticipation as model parameters in 8-bit integer are usually capable of containing enough information with lossless compression and significant speed-up. For Resnet-18, we have notice that the average ratio for Int4 is 49.1%, the average ratio for Int8 is 50.8% on Jetson NX. On the on RTX 3070 (PC), the average ratio for Int4 is 44.0%, the average ratio for Int8 is 56.0%. On RTX 3090 (Server), the average ratio for Int4 is 43.8%, the average ratio for Int8 is 56.1%. From this trend we can tell that the ratio of lower-bit-width (Int4) roughly equals to the higher-bit-width (Int8) when the hardware resources are limited. The ratio of lower bits slightly decrease as hardware resource limit relaxes. For larger model Resnet-50, on Jetson NX, the average ratio for Int4 is 48.4%, the average ratio for Int8 is 51.5%. On the RTX 3070 (PC), the average ratio for Int4 is 49.03%, the average ratio for Int8 is 50.96%, which are quite similar. the ratio only decreases when it comes to RTX 3090 (Server), the average ratio for Int4 is 43.8%, the average ratio for Int8 is 56.20%. As we compare in terms of model size, we notice that the ratio of lower-bit-widths decreases later when the model is bigger. This is an intuitive conclusion such the bottleneck turns from compute-bound to memory-bound as the hardware resource becomes sufficient.

*Ablation on* $\boldsymbol{hp}$ *Optimization:* In order to justify our motivation that hardware parameter optimization is essential as well as bit-width during quantization, we also conduct ablative study with/without loop-level optimization in Table X.

### C. Pareto Optimality Analysis

Given the performance visualization in 2-D projection on (Accuracy, Latency) space to show the performance distribution as well as the searched optimal samples with Pareto optimal performance. The original 3-D performance distribution in the search space is visualized here in Figs. 8 and 9. The notations are the same as in the main paper where blue dots denote all Pareto optimal points and red dots are found in Pareto points by searching. Gray dots denote searched suboptimal points that are not on the Pareto frontier. The performance distribution of the target data from the unified search space in the 3-objective performance space appears highly disordered and sparse. This indicates the importance of selecting proper candidate solutions within the search space during deployment, highlighting the necessity of efficient design space exploration at this step. Second, aside from the "accuracy" and "model size" dimensions, the points in the "latency" dimension exhibit a blocking relationship, such that the points in the front physically block the points behind them when visualized. Careful examination of the color-coded

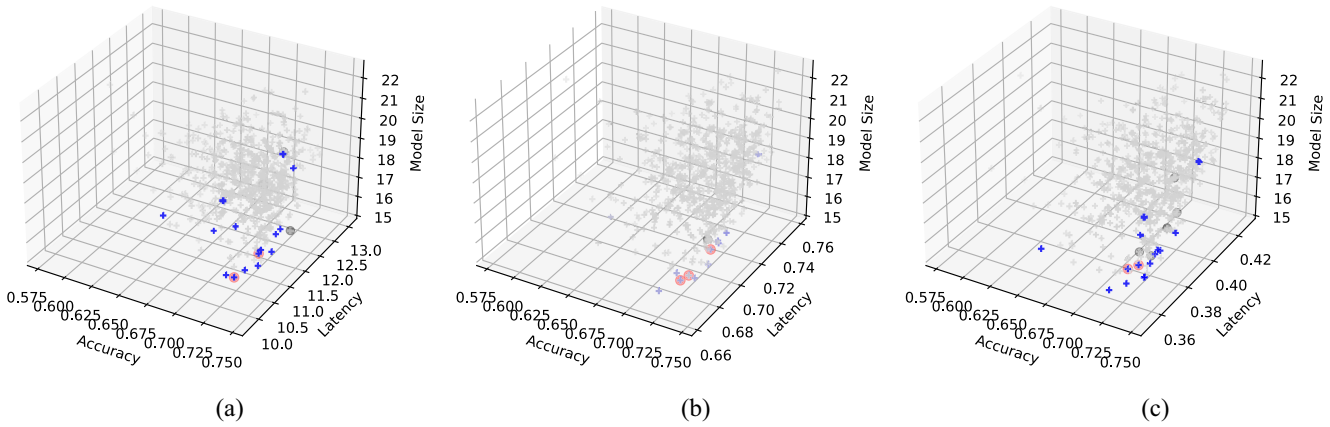


Fig. 9. 3-D performance distribution of all $(\boldsymbol{b}, \boldsymbol{hp})$ on three different backends with benchmark model Resnet-50. (a) Jetson Orin NX. (b) PC + RTX 2030. (c) Server + RTX 3090.

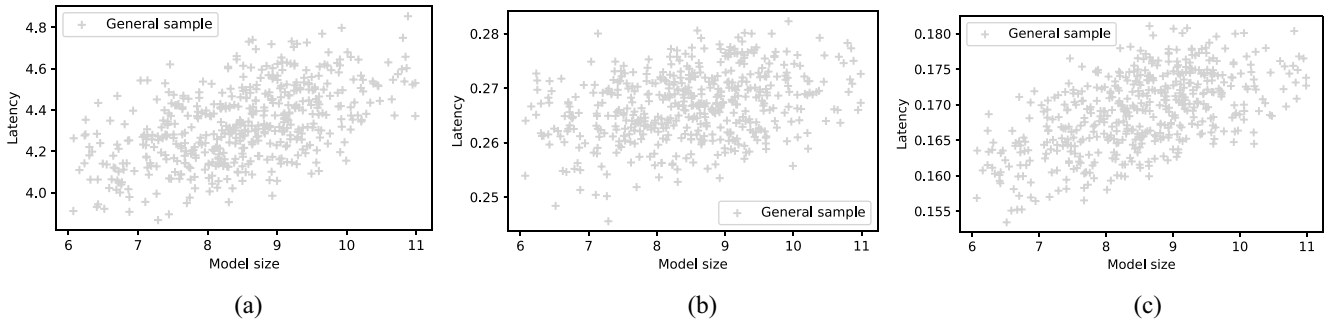


Fig. 10. Relation of latency and model size for all $(\boldsymbol{b}, \boldsymbol{hp})$ on three different backends with benchmark model Resnet-18. (a) Jetson Orin NX. (b) PC + RTX 2030. (c) Server + RTX 3090.

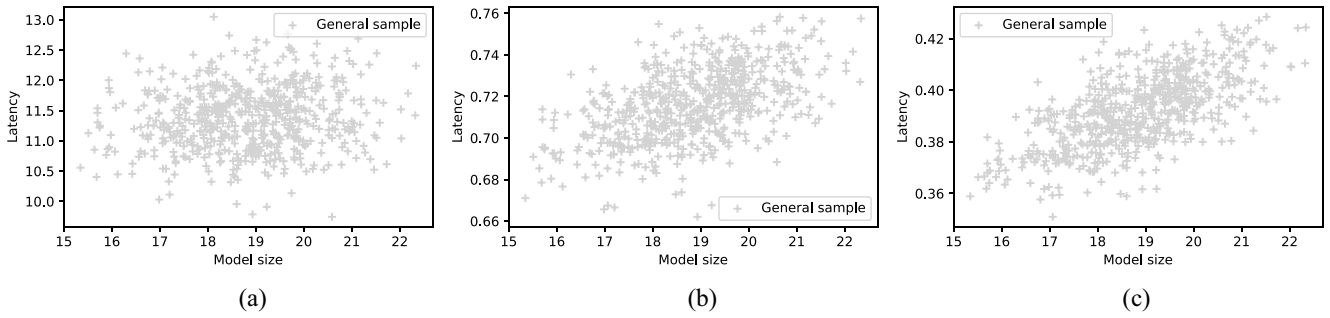


Fig. 11. Relation of latency and model size for all $(\boldsymbol{b}, \boldsymbol{hp})$ on three different backends with benchmark model Resnet-50. (a) Jetson Orin NX. (b) PC + RTX 2030. (c) Server + RTX 3090.

points can help delineate their relative positions. Third, while the readers may not need to determine the precise position of each individual point, the contrast between the sparse gray dots and the distribution of the Pareto-optimal points provides a general understanding of the Pareto frontier. The blue points represent the existing frontier, while the red points denote the searched optimal solutions, which appear to be relatively spread out on the frontier rather than centralized around a single corner.

Note that we have claimed the point in both introduction and experiments that model size does not guarantee the indication on performance but only a metric on the memory consumption. On-device performance is usually affected by many reasons especially when the hardware resources are limited. Here, we plot the relation of latency in Figs. 10 and 11. We can tell that latency and model show slight positive relation with large variance. As the hardware sources shrink (from server to edge device Jetson NX), such positive relation is fading. This aligns with our claim that using model size as indirect speed or latency indicator is not a valid choice. On-device evaluation is necessary.

## V. Conclusion

In this article, 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 multiobjective 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.

## REFERENCES


[1] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ: Hessian aware quantization of neural networks with mixed-precision," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 293–302.

[2] H. V. Habi, R. H. Jennings, and A. Netzer, "HMQ: Hardware friendly mixed precision quantization block for CNNs," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 448–463.

[3] P. Hu, X. Peng, H. Zhu, M. M. S. Aly, and J. Lin, "OPQ: Compressing deep neural networks with one-shot pruning-quantization," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, vol. 35, 2021, pp. 7780–7788.

[4] Z. Qu, Z. Zhou, Y. Cheng, and L. Thiele, "Adaptive loss-aware quantization for multi-bit networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2020, pp. 7988–7997.

[5] T. Wang et al., "APQ: Joint search for network architecture, pruning and quantization policy," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.(CVPR)*, 2020, pp. 2078–2087.

[6] S. Zhao, T. Yue, and X. Hu, "Distribution-aware adaptive multi-bit quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2021, pp. 9281–9290.

[7] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, "Adaptive quantization for deep neural network," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, vol. 32, 2018, pp. 1–9.

[8] B. Hawks, J. Duarte, N. J. Fraser, A. Pappalardo, N. Tran, and Y. Umuroglu, "Ps and Qs: Quantization-aware pruning for efficient low latency neural network inference," *Front. Artif. Intell.*, vol. 4, Jul. 2021, Art. no. 676564.

[9] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 784–800.

[10] Y. Wang, Y. Lu, and T. Blankevoort, "Differentiable joint pruning and quantization for hardware efficiency," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020, pp. 259–277.

[11] Z. Yao et al., "HAWQ-V3: Dyadic neural network quantization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 11875–11886.

[12] L. Liu, M. Shen, R. Gong, F. Yu, and H. Yang, "NNLQP: A multi-platform neural network latency query and prediction system with an evolving database," in *Proc. Int. Conf. Parallel Process. (ICPP)*, 2022, pp. 1–14.

[13] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 8612–8620.

[14] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 525–542.

[15] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–14.

[16] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2018, pp. 2704–2713.

[17] W. Zhao et al., "A high-performance accelerator for super-resolution processing on embedded GPU," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 10, pp. 3210–3223, Oct. 2023.

[18] 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. (ICML)*, 2020, pp. 7197–7206.

[19] Y. Li et al., "BRECQ: Pushing the limit of post-training quantization by block reconstruction," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021, pp. 1–16.

[20] Z. Pei, X. Yao, W. Zhao, and B. Yu, "Quantization via distillation and contrastive learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 23, 2023, doi: 10.1109/TNNLS.2023.3300309.

[21] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of ConvNets via differentiable neural architecture search," 2018, *arXiv:1812.00090*.

[22] T. Chen et al., "TVM: AN automated end-to-end optimizing compiler for deep learning," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2018, pp. 579–594.

[23] T. Chen et al., "Learning to optimize tensor programs," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 31, 2018, pp. 3393–3404.

[24] J. Mu, M. Wang, L. Li, J. Yang, W. Lin, and W. Zhang, "A history-based auto-tuning framework for fast and high-performance DNN design on GPU," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[25] Y. Bai et al., "GTCO: Graph and tensor co-design for transformer-based image recognition on tensor cores," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 2, pp. 586–599, Feb. 2024.

[26] L. Zheng et al., "Ansor: Generating high-performance tensor programs for deep learning," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2020, pp. 863–879.

[27] Y. Bai, W. Zhao, S. Yin, Z. Wang, and B. Yu, "ATFormer: A learned performance model with transfer learning across devices for deep learning tensor programs," in *Proc. Conf. Empir. Methods Nat. Lang. Process.*, 2023, pp. 4102–4116.

[28] E. V. Bonilla, K. Chai, and C. Williams, "Multi-task Gaussian process prediction," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 20, 2007, pp. 1–11.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, vol. 25, 2012, pp. 1–9.

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 2818–2826.

[31] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," 2018, *arXiv:1805.06085*.