JAQ: Joint Efficient Architecture Design and Low-Bit Quantization with Hardware-Software Co-Exploration

Mingzi Wang[1], Yuan Meng[2*], Chen Tang[1,2], Weixiang Zhang[1], Yijian Qin[2], Yang Yao[2], Yingxin Li[1], Tongtong Feng[2], Xin Wang[2], Xun Guan[1*], Zhi Wang[1*], Wenwu Zhu[2*]

[1]Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, China
[2]Department of Computer Science and Technology & BNRist, Tsinghua University, Beijing, China
wmz22@mails.tsinghua.edu.cn, yuanmeng@mail.tsinghua.edu.cn, {xun.guan,wangzhi}@sz.tsinghua.edu.cn, wwzhu@tsinghua.edu.cn

Abstract

The co-design of *neural network architectures, quantization precisions, and hardware accelerators* offers a promising approach to achieving an optimal balance between performance and efficiency, particularly for model deployment on resource-constrained edge devices. In this work, we propose the **JAQ** Framework, which jointly optimizes the three critical dimensions. However, effectively automating the design process across the vast search space of those three dimensions poses significant challenges, especially when pursuing extremely low-bit quantization. Specifical, the primary challenges include: (1) *Memory overhead in software-side*: Low-precision quantization-aware training can lead to significant memory usage due to storing large intermediate features and latent weights for back-propagation, potentially causing memory exhaustion. (2) *Search time-consuming in hardware-side*: The discrete nature of hardware parameters and the complex interplay between compiler optimizations and individual operators make the accelerator search time-consuming. To address these issues, **JAQ** mitigates the memory overhead through a channel-wise sparse quantization (**CSQ**) scheme, selectively applying quantization to the most sensitive components of the model during optimization. Additionally, **JAQ** designs **BatchTile**, which employs a hardware generation network to encode all possible tiling modes, thereby speeding up the search for the optimal compiler mapping strategy. Extensive experiments demonstrate the effectiveness of **JAQ**, achieving approximately 7% higher Top-1 accuracy on ImageNet compared to previous methods and reducing the hardware search time per iteration to 0.15 seconds. Code is available at https://github.com/wmz-opensource/JAQ/.

Introduction

Given the significant computational demands of Deep Neural Networks (DNNs), deploying them in resource-limited environments, such as the Internet of Things (IoT), remains a challenge. For example, even highly optimized Convolutional Neural Networks (CNNs) have recently struggled to perform efficiently on resource-constrained hardware devices (Li et al. 2023; Rashid, Kallakuri, and Mohsenin 2024). To speed up inference on real-world hardware while maintaining performance, hardware-aware techniques (e.g., quantization, and hardware-aware neural architecture search) have emerged to improve the model efficiency on the model-side. For example, HAQ (Wang et al. 2019), OFA (Cai et al. 2019), and ElasticViT (Tang et al. 2023) optimize the model for a fixed target device. On the other hand, accelerator-side methods design specialized accelerators (Chen et al. 2014; Liu et al. 2015; Parashar et al. 2017) to facilitate the deployment of DNNs, have received more attention recently.

However, the separated design on either the model-side or accelerator-side falls into sub-optimal (Fu et al. 2021; Hong et al. 2022) as (1) the model-size optimization will be up against efficiency loss when the hardware does not support certain operators and (2) the optimal accelerator design varies very different for various model structures and the corresponding quantized precision (Wang et al. 2019). This trend suggests the limitations of and the need for co-design of both neural networks, quantized bit-widths, and hardware accelerators.

The first principle of co-design involves efficiently navigating the vast design space. To achieve this, differentiable methods have been developed to facilitate end-to-end co-exploration. Notably, AutoNBA (Fu et al. 2021) utilizes learnable weights for determining the expected precision and architectural operator, along with designing a new objective for optimizing hardware components. DANCE (Choi et al. 2021) further introduces an MLP-based accelerator search strategy into the differentiable search framework. However, these methods support only high bit-width quantization (i.e., $\geq$ 4 bits), resulting in minimal performance degradation due to the significant redundancies that remain for compression (Esser et al. 2019). In contrast, we have observed that low-precision disrupts the optimization process, leading to a misguided search, as we will discuss later.

Therefore, we propose JAQ framework, which addresses challenges and achieves efficient joint exploration. For the first challenge, we propose channel-wise sparse quantization method. It selects a small subset of the most crucial activations channels for quantization, leaving other channels unquantized during the search process, effectively alleviating the issue of memory explosion. For the second challenge, we propose BatchTile approach that encodes all tile sizes within the search space as different batches, enabling us to deter-

* Corresponding authors.

Method	Model Arch	Low-Bit Quant	Acc. Arch
NAAS (Lin, Yang, and Han 2021)	×	×	✓
DANCE (Choi et al. 2021)	✓	×	✓
Auto-nba (Fu et al. 2021)	✓	×	✓
Ours	✓	✓	✓

Table 1: Comparison with other works on the search space dimension(Model Architecture, Low-Bit Quantization and Accelerator Architecture).

mine the optimal tiling strategies simultaneously, which significantly reduces the time overhead.

To summarize, the contributions of the paper are:

- We propose the JAQ framework, which enables efficient and effective co-exploration within the extensive optimization space. To the best of our knowledge, we are the first to explore the joint search among network architecture, ultra-low mixed-precision bit-width allocation, and accelerator architecture, as shown in Tab. 1.
- To tackle the challenge of memory explosion, we propose the channel-wise sparse quantization approach, achieving around 5× reduction in memory cost compared to non-optimized scenarios.
- We propose a hardware generation network to optimize accelerator design and BatchTile method to integrate the compiler mapping search efficiently, which reduces the search time per iteration to 0.15 seconds.
- Extensive experimental evaluations demonstrate that our framework surpasses the state-of-the-art. Our work opens up new possibilities for agile software-hardware co-design.

Related Work

Quantization and Neural Architecture Search

As a hardware-friendly lightweight technique, quantization has broad prospects for application. Mixed-Precision Quantization (MPQ) (Dong et al. 2019; Wang et al. 2019; Tang et al. 2022; Kim et al. 2024; Huang et al. 2022) allocates different bitwidths to the activations and weights of each layer, showing better accuracy-efficiency trade-off compared to fixed-precision quantization (Choi et al. 2018; Esser et al. 2019; Markov et al. 2023; Xu et al. 2023; Nagel et al. 2022). Recently, hardware increasingly supports mixed-precision (Sharma et al. 2018; Umuroglu, Rasnayake, and Själander 2018), which further pushes the research in MPQ. HAQ (Wang et al. 2019) leverages Reinforcement learning (RL) to allocate bitwidth to each layer. HAWQ (Dong et al. 2019) uses the information derived from the Hessian matrix to determine quantization sensitivity and guide the allocation of bitwidths for network parameters.

Neural Architecture Search (NAS) enables the automated design of high-performance DNN network structures, saving time and effort of the manual design. To reduce search cost, differentiable NAS (Liu, Simonyan, and Yang 2018; Qin et al. 2021) methods have merged, which integrate all candidate operators into an end-to-end trained supernet, and finally select the optimal subnet. Some studies have incorporated hardware performance metrics into the NAS via lookup tables (Zhang et al. 2020; Li et al. 2021), aiming to enhance the model's efficiency on actual hardware. However, all these works concentrate exclusively on algorithmic optimization without exploring hardware architecture, which may not yield optimal inference efficiency.

DNN Accelerators

To improve the performance of modern deep neural network computations, fixed-bitwidth DNN accelerators have emerged, featuring specialized components like MAC arrays, on-chip buffers, and network-on-chip architectures (Chen et al. 2016; Jouppi et al. 2017; Du et al. 2015). Recently, the concept of MPQ has paved the way for the development of bit-flexible accelerators (Sharma et al. 2018; Umuroglu, Rasnayake, and Själander 2018) that allow for varying bitwidths across individual layers. However, designing AI accelerators remains a complex and time-consuming task that demands significant hardware expertise. However, designing AI accelerators is complex and requires significant expertise. AI-driven methods, such as NAAS (Lin, Yang, and Han 2021) and GPT4AIGChip (Fu et al. 2023), streamline the process by autonomously evaluating design configurations. These approaches focus primarily on hardware architecture and often yield sub-optimal results compared to co-design methodologies that integrate both network and hardware exploration (You et al. 2023; Lou et al. 2023; Stevens et al. 2021; Reggiani et al. 2023).

Hardware-software Co-design

Some studies employ hardware-software co-design methods using reinforcement learning or evolutionary algorithm (Jiang et al. 2020; Abdelfattah et al. 2020), which require expensive training time and also suffer from limited search spaces. To address this issue, differentiable methods have been employed for co-exploration. EDD (Li et al. 2020) is an FPGA-based differentiable network design framework. However, it does not encompass the search for hardware parameters, such as the number of BRAMs or DSPs. While Dance (Choi et al. 2021) builds a pipeline to explore ASIC-based accelerator and network structure, it has a limitation that it does not take quantization into consideration. Auto-nba (Fu et al. 2021) is not suitable for low-bit quantization.

JAQ targets efficient joint search of network, low-bit mixed-precision bitwidths and accelerator architecture.

JAQ Framework

Preliminary

Differentiable Neural Architecture Search. Differentiable neural architecture search (DNAS) (Liu, Simonyan, and Yang 2018; Wu et al. 2019) transforms the entire search space into a supernet and each path in the supernet is equipped with an architecture parameter, which represents the probability of selecting this path. The incorpo-

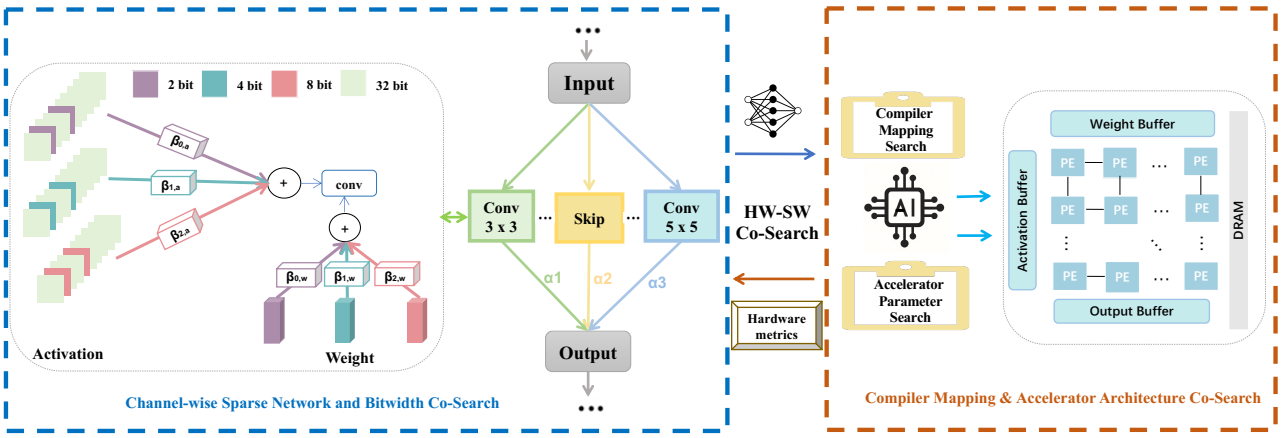

Figure 1: JAQ framework. The left part represents the optimization of network structure and bitwidths allocation, addressing the memory cost bottleneck through channel-wise sparse quantization. The right part depicts accelerator architecture search, including hardware parameters and compiler mapping strategy. Hardware metrics indicate accelerator performance (Energy, Latency and Area).

ration of the Gumbel-Softmax(Jang, Gu, and Poole 2016) function plays a pivotal role enabling these architecture parameters trainable through gradient-based optimization. After the training of the supernet, the optimal subnet is formed by the path with the highest architecture parameter in each layer. The function of Gumbel-Softmax is:

$$\beta_t = \frac{\exp\left(\frac{\beta_t+\epsilon_t}{\tau}\right)}{\sum_{i=1}^{N}\exp\left(\frac{\beta_i+\epsilon_i}{\tau}\right)}, \quad \epsilon \sim U(0,1), \tag{1}$$

where β represents the original parameter distribution, while ϵ is a number sampled from a uniform distribution ranging between 0 and 1. Additionally, the smoothness of the distribution can be regulated using the temperature coefficient τ.

Quantization. The quantization function $Q(\cdot)$, defined as:

$$Q(V) = \text{round}\left(\text{clip}\left(\frac{V}{s}, \min_b, \max_b\right)\right) \times s, \tag{2}$$

where $\mathbf{V}$ and $Q(\mathbf{V})$ denote the floating-point value and its dequantized value (quantization width is b bit). The parameter $s = \frac{\max(\mathbf{V})-\min(\mathbf{V})}{2^b-1}$, which represents the scale factor used in the quantization mapping, the interval $[min_b, max_b]$ specifies integer range.

Problem Formulation

Fig. 1 illustrates the overall framework of JAQ, which includes the joint search among network structure, ultra-low mixed-precision bitwidth allocation, and accelerator architecture. The formulation of the joint optimization problem is:

$$\begin{aligned} &\min_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\mathbf{w}} \mathcal{L}_{\text{CE}}\left(\mathbf{w}, \text{N}(\boldsymbol{\alpha}), \text{M}(\boldsymbol{\beta})\right) \\ &\text{s.t.} \quad \mathcal{E}_{\text{HW}}\left(\text{H}\left(\boldsymbol{\gamma}\right), \text{N}\left(\boldsymbol{\alpha}\right), \text{M}(\boldsymbol{\beta})\right) \le C \end{aligned} \tag{3}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ denote the operator architecture parameters and the bitwidth architecture parameters, respectively. $\boldsymbol{\gamma}$ denotes the hardware accelerator configuration. $\mathbf{w}$ represents the weights of the NAS supernet. $\text{N}(\boldsymbol{\alpha})$ indicates the network structure selected based on $\boldsymbol{\alpha}$. $\text{M}(\boldsymbol{\beta})$ denotes the bitwidths selection for each operator according to $\boldsymbol{\beta}$. $\text{H}(\boldsymbol{\gamma})$ depicts the accelerator architecture based on $\boldsymbol{\gamma}$. $\mathcal{E}_{\text{HW}}$ reflects the hardware-side performance, calculated by hardware metrics (Energy, Latency, and Area). $\mathcal{L}_{\text{CE}}$ represents the cross-entropy loss, and $\mathcal{E}_{\text{HW}}$. To track this optimization problem, we introduce a Lagrange multiplier λ for Eq (4):

$$\min_{\boldsymbol{\alpha},\boldsymbol{\beta},\boldsymbol{\gamma},\mathbf{w}} \left[\mathcal{L}_{\text{CE}}\left(\mathbf{w}, \text{N}(\boldsymbol{\alpha}), \text{M}(\boldsymbol{\beta})\right) + \lambda \mathcal{E}_{\text{HW}}\left(\text{H}\left(\boldsymbol{\gamma}\right), \text{N}\left(\boldsymbol{\alpha}\right), \text{M}(\boldsymbol{\beta})\right)\right], \tag{4}$$

Channel-wise Sparse Quantization (CSQ)

Memory Cost Bottleneck. DNAS segments the supernet into a series of cells. Each cell is structured as a directed acyclic graph (DAG) with several nodes (each node corresponds to a distinct operator), and each operator within the supernet must be stored in GPU memory during training. This indicates that adding extra search dimensions in DNAS-based methods can easily lead to GPU memory overload or require to reduce the training batch size to maintain the original utilization of GPU memory. Quantization, a memory-intensive process, involves storing numerous quantized parameters and additional quantization information. Therefore, integrating network architecture search with bitwidth selection significantly amplifies the GPU memory consumption. For example, as shown in Appendix A Fig. 4a, the utilization of GPU memory increases linearly with the number of available bitwidth options in each operator which is deemed unacceptable. All experiments are conducted on a single NVIDIA GeForce RTX 4090 GPU to prevent inaccuracies in memory measurements caused by multiple copies of the model in multi-GPU parallel setups. We find that as the batch size increases linearly, the GPU memory utiliza-

tion also increases linearly. Therefore, by measuring memory cost with a small batch size, we can estimate the memory utilization for larger batch sizes. We utilize FBNet (Wu et al. 2019) as the supernet, and add quantization process(Eq. 2) in each operator.

CSQ. The differentable network and bitwidths co-search framework in JAQ can be implemented by formulating as:

$$\mathbf{A}^{l+1} = \sum_{i=1}^{n} \boldsymbol{\alpha}_i^l \cdot \tilde{\mathbf{W}}_i^l \cdot \tilde{\mathbf{A}}_i^l, \text{ where}$$
$$\tilde{\mathbf{W}}_i^l = \sum_{k=1}^{m} \boldsymbol{\beta}_{w_{i,k}^l} \cdot Q(\mathbf{W}_{ik}^l), \text{ and } \tilde{\mathbf{A}}_i^l = \sum_{k=1}^{m} \boldsymbol{\beta}_{a_{i,k}^l} \cdot Q(\mathbf{A}^l), \tag{5}$$

where l denotes the layer index in network, and n is the number of operator candidates per layer, while m is the number of bit-width candidates per operator. $\tilde{W}$ and $\tilde{A}$ represent the sum of quantized weights and quantized activations under different precisions respectively. $\boldsymbol{\alpha}$ denotes the operator architecture parameters, while $\boldsymbol{\beta}_{w_{i,k}^l}$ and $\boldsymbol{\beta}_{a_{i,k}^l}$ are the architecture parameters of weights and activations for each precision. Q represents the quantization function (Eq. 2). As illustrated in Appendix A Fig. 4b, during the supernet training process, the memory requirement of weight quantization is trivial. Therefore, memory cost bottleneck in network and bitwidths co-search framework can be predominantly attributed to the quantization of activations. To alleviate this issue during supernet training, we propose channel-wise sparse quantization strategy for the quantization of activation. This can be implemented by reformulating Eq. 5 as:

$$\tilde{\mathbf{A}}_i^l = \left(\sum_{k=1}^{m} \boldsymbol{\beta}_{a_{i,k}^l} \cdot Q(\mathbf{A}^l(\Omega^l)) \right) \oplus \mathbf{A}^l(1 - \Omega^l), \tag{6}$$

where Ω is the indices of channels to be quantized and $\oplus$ denotes concatenation operation. $\mathbf{A}^l(\Omega^l)$ represents all channels selected in $\mathbf{A}^l$ according to Ω^l.

The core innovation of this method is to quantize only a few channels of activations during searching phase, while leaving other channels unquantized, which significantly reduces the demand on GPU memory. To achieve better search result (detailed explanation and experiments are provided in the ablation study), we need to select the most important channels from each activations. Inspired by a previous work (Liu et al. 2017), the scale factors in Batch Normalization (BN) can effectively represent the importance of each channel.

$$\hat{z} = \frac{z_{\text{in}} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad z_{\text{out}} = \gamma \hat{z} + \beta, \tag{7}$$

where z_{in} and z_{out} are the input and output of a BN layer, μ_B and σ_B represent the mean and standard deviation of the input activations across the batch B. The trainable parameters γ and β serve as scale and shift factors respectively.

Therefore, we choose to quantize only the top $K\%$ of the most important channels of each activations during search phase and defining:

$$\Omega^l \leftarrow \texttt{TopKChannelToQuantize}(\Gamma^l, K)$$
$$\Gamma_j^l = \sum_{i=1}^{n} \boldsymbol{\alpha}_i^{l-1} \gamma_{ij}^{l-1} \quad j \in N^{l-1}, \tag{8}$$

where Γ is the importance indicator of each channel, and γ represents the scale factors defined in Eq. 7, while N is the output channels number of l - 1 layer.

During the search phase, scale factors are trainable to dynamically adjust the importance indicator for each channel. Finally, as Appendix A Fig. 4c demonstrates, the GPU utilization in our algorithm is significantly reduced to acceptable bounds.

Accelerator Architecture Search

Hardware parameters in accelerators are non-differentiable. Although it is possible to optimize these parameters using reinforcement learning (Lin, Yang, and Han 2021), the time overhead is particularly substantial. Therefore, there is a demand for exploring efficient methods to search for these parameters. Furthermore, compiler mapping is crucial for the latency and energy consumption of DNNs inference on accelerators. Therefore we incorporate compiler mapping optimization into the joint search framework, reducing its searching time to less than 0.15 seconds per iteration.

Accelerator Search Space. Our accelerator search space is divided into two categories. The first category involves accelerator parameters, which include the shape and number of processing elements (PEs), the size of the on-chip cache used for storing weights, activations, and outputs, as well as the inter-connection of PEs, described as the dataflow type of the parallel dimension. The second category focuses on optimizations of the compiler mapping, including sizes of tiles and loop order of tiling.

Accelerator Parameters Search. First, we identify the current optimal subnet within the supernet and encode each operator in the subnet as the operator encoding vector in Fig. 2: Kernel, Stride, Output Row, Input Channel, Output Channel, Activation Bitwidth, and Weight Bitwidth. Then, the encoded operators are sent into the accelerator parameters search part, constituted by five layers of residual blocks. The final layer maps the hidden states into seven elements in the accelerator parameters encoding Vector in Fig. 2: PE_x, PE_y, Activation Cache, Weight Cache, Output Cache, Dataflow Type, and Tile Order.

Gumbel-Softmax (Jang, Gu, and Poole 2016) is used as the activation function in each classifier, ensuring that the output values closely resemble the inputs for hardware cost estimation, as well as maintaining the gradient propagation during the training stage.

Compiler Mapping Search: BatchTile. As shown in Tab. 2, Tile size is crucial for the model inference performance on accelerator. In JAQ, the accelerator is configured to process one image at a time, hence the batch size is one. Consequently, each operator requires tiling across four dimensions: input channel, output channel, output width, and

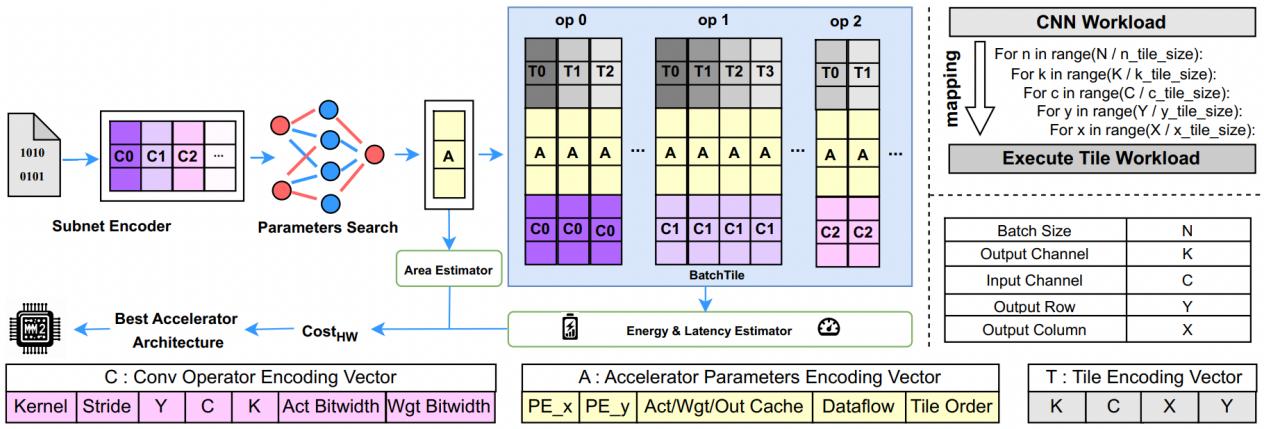

Figure 2: The overall accelerator search framework of JAQ. The right part represents the executing workload of a CNN operator after compiler mapping, which can be segmented into tiles across five dimensions. The left part displays an optimization pipeline including subnet encoder, accelerator parameters search, and the BatchTile method. The bottom section elaborates on the meanings of each field within the three distinct vectors.

	B/b	OW/ow	OH/oh	IC/ic	OC/oc	Lat. (ms)	Ene. (mJ)
Best Tiling	1/1	7/1	7/1	35/16	35/16	3.2	2.6
Case 0	1/1	2/4	2/4	276/2	2/512	56.6	8.3
Case 1	1/1	4/2	4/2	18/32	5/128	4.7	5.1

Table 2: The latency (Lat.) and energy (Ene.) for the optimal tiling method and two randomly selected tiling methods applied to a specific pair of operator and accelerator (Operator: kernel size of 5, stride of 1, output size of 7, both input and output channels at 552, with activation and weight bitwidths of 8 bits. Accelerator: PE Array dimensions of 16x16, with 384KB Act/Wgt/Out Cache sizes).

output height. To achieve peak performance for model inference on accelerator, optimal tile sizes for each operator should be determined during the compiler mapping stage. However, finding the optimal tile size for all operators in a subnet is time-consuming (around 50s), which is unfriendly to end-to-end joint search. To efficiently find the optimal tile size, we propose the BatchTile method. The BatchTile method initially encodes each operator's tiling strategies across four dimensions as illustrated in Fig. 2: Output Channel, Input Channel, Output Column, and Output Row. Subsequently, we concatenate accelerator parameters encoding vector, operator encoding vector of each operator, and different tile encoding vectors to form various ⟨Operator, Accelerator Parameters, Tiling Strategy⟩ pairs. These pairs, as different batches, are fed into the Energy & Latency Estimator (the principle of the estimator follows (Choi et al. 2021)) to simultaneously identify the optimal tiling strategy for each operator. Finally, the BatchTile method reduces the entire compiler mapping search time to less than 0.15 seconds(comparison experiment is in Tab. 3).

Method	Search Time
Auto-nba (Fu et al. 2021)	30
Ours	0.15

Table 3: Comparison of Accelerator search time(s) between JAQ and previous work.

Our methodology capitalizes on hardware design principles, estimating the hardware from the perspective of area, energy, and latency metrics. Combining these three metrics, the hardware cost function included in Eq. 4 is:

$$\mathcal{E}_{\text{HW}} = \lambda_E \cdot \texttt{Energy} + \lambda_L \cdot \texttt{Latency} + \lambda_A \cdot \texttt{Area}, \quad (9)$$

where λ_E, λ_L, and λ_A are adjustable among these cost metrics.

The Overall Joint Pipeline

JAQ consists of the search stage and the retrain stage. The search stage integrates the channel-wise sparse quantization method into the model (network architectures and bitwidths) searching, and incorporates the BatchTile approach into the accelerator searching.

For searching, each iteration consists of two steps. The first step is to update the weights ($\mathbf{w}$) in supernet, which doesn't require interfacing with the accelerator. The second step, collaborating with the accelerator, involves updating the architecture parameters ($\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$) and the accelerator configuration ($\boldsymbol{\gamma}$), as defined in Eq. 4. In the second step, after forward propagation in the supernet, the current optimal subnet is encoded and passed into the accelerator search framework. Then, we optimize the accelerator parameters and compiler mapping strategy. Subsequently, the $\texttt{Cost}_{\text{HW}}$ obtained through Eq. 9 is bound to the architecture parameters, which will be updated during the backpropagation process.

For retraining, we retrain the optimal subnet obtained from the search stage. Finally, we achieve the optimal network structure and accelerator architecture, thus realizing the synergy between software and hardware design.

Experiments

Experimental Settings

Our experiments are conducted on the CIFAR-10/100, and ImageNet datasets. In search stage, We use 80% of the data to update the weights within the supernet and 20% of the data for the architecture parameters. The initial learning rate is 0.01, employing an annealing cosine learning rate schedule. The initial temperature for the Gumbel-Softmax is set to 5. For the CIFAR-10/100 and ImageNet datasets, we search for 90 and 45 epochs on eight NVIDIA GeForce RTX 4090 GPUs, respectively. In Eq. 8, we select K as 3. In Eq. 9, λ_E, λ_L, and λ_A are all set to 0.33. In retrain stage, we train the subnet for 600 epochs for CIFAR-10/100 and 180 epochs for ImageNet, respectively. We employ an annealing cosine learning rate schedule, with an initial learning rate of 0.01.

Search Space

We utilize FBNet (Wu et al. 2019) as the network search space. Except for stem and head layers, it comprises 22 blocks. Each block has 9 candidate operations, including a skip choice. We utilize BitFusion (Sharma et al. 2018) accelerator as the hardware template, which is a SOTA ASIC accelerator for mixed-precision models. For the search space of bitwidths, the weights and activations of each layer have three different options $\in [2, 4, 8]$. For the accelerator search space, PEx and PEy are selectable within a range of 3 to 64. The cache sizes for weights, activations, and outputs are configurable in increments of 16KB, ranging from 64KB to 528KB, offering 30 distinct choices. We choose three types of dataflows: Weight Stationary (WS) (Jouppi et al. 2017), Output Stationary (OS) (Du et al. 2015), and Row Stationary (RS) (Chen et al. 2016). For each operator, there are 120 possible permutations of the tile order across five dimensions: batch size, input channel, output channel, output height, and output width. For tile sizes, we set the batch size to only one, while in other dimensions, the tile size can vary from 2^0 to 2^n (The maximum value of n is 10).

Co-exploration Results

Compared with previous joint search framework (Fu et al. 2021), we conduct experiments on the CIFAR-10, CIFAR-100, and ImageNet (ILSVRC2012) datasets. In various comparative experiments, we adjusted λ parameter in Eq. 4 to achieve different balances between accuracy and hardware cost. Specifically, on the CIFAR-10 and CIFAR-100 datasets, the value of λ is set to 0.0005, 0.001, 0.002, and 0.004, while on ImageNet, it is set to 0.001, 0.002, and 0.005. As shown in Tab. 4, the experiments reveal that our method significantly outperforms baseline in low-bit joint search tasks.

To demonstrate the efficiency of the JAQ, we conduct comparative analyses with other search frameworks. As

	λ = 0.004		λ = 0.002		λ = 0.001		λ = 0.0005	
	A	**E**	**A**	**E**	**A**	**E**	**A**	**E**
Auto-nba	82.847	10	89.643	12.8	86.597	20	86.677	26
Ours	91.081	11.8	92.163	12.4	91.895	17.6	92.183	30

(a) CIFAR10

	λ = 0.004		λ = 0.002		λ = 0.001		λ = 0.0005	
	A	**E**	**A**	**E**	**A**	**E**	**A**	**E**
Auto-nba	60.169	4	52.837	6.2	56.468	14	48.542	18
Ours	72.440	2.6	72.956	7.8	73.264	13.4	73.651	14.2

(b) CIFAR100

	λ = 0.005		λ = 0.002		λ = 0.001	
	A	**E**	**A**	**E**	**A**	**E**
Auto-nba	62.423	32.48	61.781	253	62.787	503.1
Ours	69.132	26.238	69.473	230.1	70.197	498.6

(c) ImageNet

Table 4: Comparisons between our method and the baseline(Auto-nba (Fu et al. 2021)) on three distinct datasets: CIFAR-10, CIFAR-100, and ImageNet. EDAP ($J \cdot s \cdot m^2 \cdot 10^{-18}$) stands for the Energy-Delay-Area Product, which is a common hardware metric. A indicates accuracy and E indicates EDAP.

Method	**Net.**	**Bit.**	**Acc.**	**T**
NAAS (Lin, Yang, and Han 2021)	—	—	✓	1200
OQAT (Shen et al. 2021)	✓	✓	—	1200
BatchQuant (Bai et al. 2021)	✓	✓	—	1800
Auto-nba (Fu et al. 2021)	✓	✓	✓	180
Ours	✓	✓	✓	160

Table 5: Comparison of search space(Network, Bitwidth and Accelerator) and search time(T)(GPU hours) between JAQ and other works on the ImageNet dataset.

shown in Tab. 5, NAAS (Lin, Yang, and Han 2021) employs reinforcement learning (RL) to jointly search network structures and accelerator architectures. OQAT (Shen et al. 2021) and BatchQuant (Bai et al. 2021) utilize a one-shot approach for joint searching of network structures and bitwidths. In contrast, Auto-nba (Fu et al. 2021) and our work both present a triple search framework, but our work achieves better search efficiency within a large search space.

Hardware design must take into account the actual requirements for energy, latency, and area. Some accelerators are specifically designed to minimize power consumption and latency for deployment on embedded platforms, while others are produced to occupy a tiny area for integration into System on Chips (SoCs). The JAQ method can satisfy the sensitivity of a specific metric by adjusting the parameters in Eq. 9. As shown in Tab. 6, for instance, increasing the λ_L results in a low latency in the final result. Conversely, increasing the λ_A leads to a tiny area for the accelerator. Overall, this indicates that by adjusting the cost hyperparameters, JAQ can achieve a desired solution.

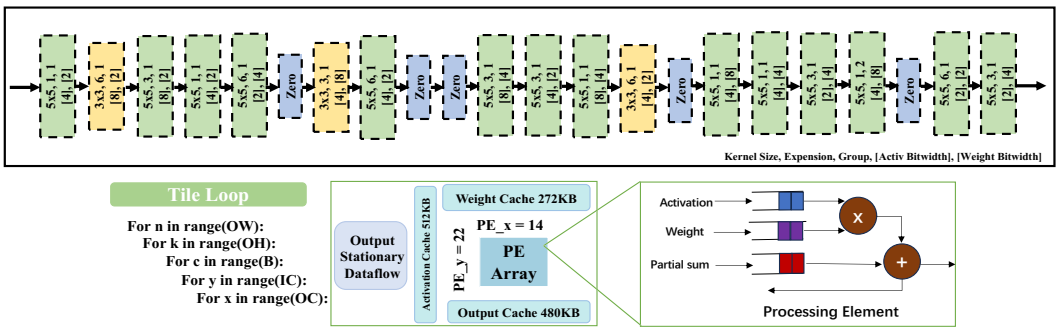

Figure 3: Visualization of searched network, bitwidths and accelerator on CIFAR-100.

	λ_E	λ_L	λ_A	Acc	Lat. (ms)	En. (mJ)	Ar. (mm^2)
Latency-S	0.1	0.8	0.1	75.099	1.94	1.52	1.36
Area-S	0.1	0.1	0.8	73.641	2.74	2.43	0.69

Table 6: Different hardware sensitivity(Latency-S and Area-S) experiments on CIFAR-100 dataset with different metrics(Latency, Energy and Area).

Method	Misguided Search (%)	Top-1 Accuracy
Auto-nba (Fu et al. 2021)	40	55.704
Channel 0	5	64.355
Ours(K=1)	0	65.377
Ours(K=5)	0	65.863

Table 7: Comparison of different methods for the joint search of network structures and the allocation of 2, 3, and 4-bit bitwidths without any constraint on the CIFAR-100 dataset.

Ablation Studies

Under low bit search condition, to demonstrate the effectiveness of the channel-wise sparse quantization algorithm in addressing GPU memory bottleneck problem, we contrast JAQ with a previous work (Auto-nba (Fu et al. 2021)) tackling the same problem. Auto-nba introduces a method called heterogeneous sampling which employs the Straight-Through Estimator (STE)(Bengio, Léonard, and Courville 2013) to mask the quantization operation during updating weight parameters. While updating architecture parameters, it employs hard Gumbel-Softmax to active only one bitwidth choice to save GPU memory. However, this method encounters two severe drawbacks under low bit search condition. First, as shown in Appendix B Fig. 5a, the architecture parameters of the operators suffer from significant parameter coupling during training, making it challenging to distinguish them effectively. Second, without any constraint, each bitwidth allocation will most likely converge to the maximum value within the candidate range, rather than selecting low bitwidths that severely impact performance. Yet, as depicted in Appendix B Fig. 5c, many operators ultimately select the 2-bit configuration, leading to a serious misguided search.

Furthermore, we conduct joint search of network structures and bitwidths allocation without any constraint. As shown in Tab. 7, in the first experiment, Auto-nba utilizes heterogeneous sampling to address memory explosion but suffers from severe misguided search, only achieving 55.7 top-1 accuracy. In the second experiment, we also employ the channel-wise concept but quantize only the first channel of each activation during the search process. This approach still suffers from 5% misguided searches, indicating that fixing the selection of channels is inappropriate. Instead, selecting important channels within each layer is preferable. The third and fourth experiments implement our channel-wise sparse quantization algorithm, setting K in Eq. 8 to 1 and 5, respectively. Using Eq. 6, we selectively quantize the most important channels, effectively eliminating misguided search problem and achieving significantly higher accuracy than the previous work.

Visualization

Fig. 3 indicates that convolutions with the kernel size of 5 are more compatible with the JAQ accelerator architecture, and larger kernel sizes can achieve higher accuracy with low bitwidth. Because there are more activations and outputs than weights, they are allocated a larger cache size in the search result. The output stationary dataflow is particularly well-suited to the network structure of JAQ, providing superior hardware performance.

Conclusion

In this paper, we present JAQ, which is the first to implement joint optimization across three dimensions: network structure, ultra-low mixed-precision bitwidths, and accelerator architecture. By addressing the challenges of memory explosion and search overhead of accelerator architecture, JAQ enables efficient joint optimization within a vast search space. When benchmarking with SOTA works, we achieve superior performance. We believe that JAQ can provide inspiration and support to the field of software-hardware co-design.

Acknowledgments

This work was supported by National Key Research and Development Project of China (Grant No. 2023YFF0905502), National Natural Science Foundation of China (Grant No. 62472249, No. 62402264), Shenzhen Science and Technology Program (Grant No. JCYJ20220818101014030), National Key R&D Program of China (2021YFA1001000), National Natural Science Foundation of China (U23A20282), Shenzhen Science, Technology and Innovation Commission (KJZD20231023094659002, JCYJ20220530142809022, WDZC20220811170401001). We thank anonymous reviewers for their valuable advice.

References

Abdelfattah, M. S.; Dudziak, Ł.; Chau, T.; Lee, R.; Kim, H.; and Lane, N. D. 2020. Best of both worlds: Automl codesign of a cnn and its hardware accelerator. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6. IEEE.

Bai, H.; Cao, M.; Huang, P.; and Shan, J. 2021. Batchquant: Quantized-for-all architecture search with robust quantizer. *Advances in Neural Information Processing Systems*, 34: 1074–1085.

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Cai, H.; Gan, C.; Wang, T.; Zhang, Z.; and Han, S. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*.

Chen, Y.; Luo, T.; Liu, S.; Zhang, S.; He, L.; Wang, J.; Li, L.; Chen, T.; Xu, Z.; Sun, N.; et al. 2014. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 609–622. IEEE.

Chen, Y.-H.; Krishna, T.; Emer, J. S.; and Sze, V. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1): 127–138.

Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P. I.-J.; Srinivasan, V.; and Gopalakrishnan, K. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*.

Choi, K.; Hong, D.; Yoon, H.; Yu, J.; Kim, Y.; and Lee, J. 2021. Dance: Differentiable accelerator/network co-exploration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 337–342. IEEE.

Dong, Z.; Yao, Z.; Gholami, A.; Mahoney, M. W.; and Keutzer, K. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 293–302.

Du, Z.; Fasthuber, R.; Chen, T.; Ienne, P.; Li, L.; Luo, T.; Feng, X.; Chen, Y.; and Temam, O. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd annual international symposium on computer architecture*, 92–104.

Esser, S. K.; McKinstry, J. L.; Bablani, D.; Appuswamy, R.; and Modha, D. S. 2019. Learned step size quantization. *arXiv preprint arXiv:1902.08153*.

Fu, Y.; Zhang, Y.; Yu, Z.; Li, S.; Ye, Z.; Li, C.; Wan, C.; and Lin, Y. C. 2023. Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 1–9. IEEE.

Fu, Y.; Zhang, Y.; Zhang, Y.; Cox, D.; and Lin, Y. 2021. Auto-NBA: Efficient and effective search over the joint space of networks, bitwidths, and accelerators. In *International Conference on Machine Learning*, 3505–3517. PMLR.

Hong, D.; Choi, K.; Lee, H. Y.; Yu, J.; Park, N.; Kim, Y.; and Lee, J. 2022. Enabling hard constraints in differentiable neural network and accelerator co-exploration. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 589–594.

Huang, X.; Shen, Z.; Li, S.; Liu, Z.; Xianghong, H.; Wicaksana, J.; Xing, E.; and Cheng, K.-T. 2022. Sdq: Stochastic differentiable quantization with mixed precision. In *International Conference on Machine Learning*, 9295–9309. PMLR.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jiang, W.; Yang, L.; Sha, E. H.-M.; Zhuge, Q.; Gu, S.; Dasgupta, S.; Shi, Y.; and Hu, J. 2020. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12): 4805–4815.

Jouppi, N. P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, 1–12.

Kim, H.-B.; Lee, J. H.; Yoo, S.; and Kim, H.-S. 2024. MetaMix: Meta-state Precision Searcher for Mixed-precision Activation Quantization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 13132–13141.

Li, C.; Yu, Z.; Fu, Y.; Zhang, Y.; Zhao, Y.; You, H.; Yu, Q.; Wang, Y.; and Lin, Y. 2021. Hw-nas-bench: Hardware-aware neural architecture search benchmark. *arXiv preprint arXiv:2103.10584*.

Li, Y.; Baik, J.; Rahman, M. M.; Anagnostopoulos, I.; Li, R.; and Shu, T. 2023. Pareto Optimization of CNN Models via Hardware-Aware Neural Architecture Search for Drainage Crossing Classification on Resource-Limited Devices. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 1767–1775.

Li, Y.; Hao, C.; Zhang, X.; Liu, X.; Chen, Y.; Xiong, J.; Hwu, W.-m.; and Chen, D. 2020. Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6. IEEE.

Lin, Y.; Yang, M.; and Han, S. 2021. Naas: Neural accelerator architecture search. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 1051–1056. IEEE.

Liu, D.; Chen, T.; Liu, S.; Zhou, J.; Zhou, S.; Teman, O.; Feng, X.; Zhou, X.; and Chen, Y. 2015. Pudiannao: A polyvalent machine learning accelerator. *ACM SIGARCH Computer Architecture News*, 43(1): 369–381.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, 2736–2744.

Lou, W.; Qian, J.; Gong, L.; Wang, X.; Wang, C.; and Zhou, X. 2023. NAF: Deeper Network/Accelerator Co-Exploration for Customizing CNNs on FPGA. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1–6. IEEE.

Markov, I.; Vladu, A.; Guo, Q.; and Alistarh, D. 2023. Quantized distributed training of large models with convergence guarantees. In *International Conference on Machine Learning*, 24020–24044. PMLR.

Nagel, M.; Fournarakis, M.; Bondarenko, Y.; and Blankevoort, T. 2022. Overcoming oscillations in quantization-aware training. In *International Conference on Machine Learning*, 16318–16330. PMLR.

Parashar, A.; Rhu, M.; Mukkara, A.; Puglielli, A.; Venkatesan, R.; Khailany, B.; Emer, J.; Keckler, S. W.; and Dally, W. J. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH computer architecture news*, 45(2): 27–40.

Qin, Y.; Wang, X.; Zhang, Z.; and Zhu, W. 2021. Graph differentiable architecture search with structure learning. *Advances in neural information processing systems*, 34: 16860–16872.

Rashid, H.-A.; Kallakuri, U.; and Mohsenin, T. 2024. TinyM2Net-V2: A Compact Low-power Software Hardware Architecture for M ulti m odal Deep Neural Networks. *ACM Transactions on Embedded Computing Systems*, 23(3): 1–23.

Reggiani, E.; Pappalardo, A.; Doblas, M.; Moreto, M.; Olivieri, M.; Unsal, O. S.; and Cristal, A. 2023. Mix-GEMM: An efficient HW-SW Architecture for Mixed-Precision Quantized Deep Neural Networks Inference on Edge Devices. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 1085–1098. IEEE.

Sharma, H.; Park, J.; Suda, N.; Lai, L.; Chau, B.; Kim, J. K.; Chandra, V.; and Esmaeilzadeh, H. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 764–775. IEEE.

Shen, M.; Liang, F.; Gong, R.; Li, Y.; Li, C.; Lin, C.; Yu, F.; Yan, J.; and Ouyang, W. 2021. Once quantization-aware training: High performance extremely low-bit architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5340–5349.

Stevens, J. R.; Venkatesan, R.; Dai, S.; Khailany, B.; and Raghunathan, A. 2021. Softermax: Hardware/software co-design of an efficient softmax for transformers. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 469–474. IEEE.

Tang, C.; Ouyang, K.; Wang, Z.; Zhu, Y.; Ji, W.; Wang, Y.; and Zhu, W. 2022. Mixed-precision neural network quantization via learned layer-wise importance. In *European Conference on Computer Vision*, 259–275. Springer.

Tang, C.; Zhang, L. L.; Jiang, H.; Xu, J.; Cao, T.; Zhang, Q.; Yang, Y.; Wang, Z.; and Yang, M. 2023. Elasticvit: Conflict-aware supernet training for deploying fast vision transformer on diverse mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 5829–5840.

Umuroglu, Y.; Rasnayake, L.; and Själander, M. 2018. Bismo: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 307–3077. IEEE.

Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; and Han, S. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 8612–8620.

Wu, B.; Dai, X.; Zhang, P.; Wang, Y.; Sun, F.; Wu, Y.; Tian, Y.; Vajda, P.; Jia, Y.; and Keutzer, K. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 10734–10742.

Xu, S.; Li, Y.; Lin, M.; Gao, P.; Guo, G.; Lü, J.; and Zhang, B. 2023. Q-detr: An efficient low-bit quantized detection transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 3842–3851.

You, H.; Sun, Z.; Shi, H.; Yu, Z.; Zhao, Y.; Zhang, Y.; Li, C.; Li, B.; and Lin, Y. 2023. Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 273–286. IEEE.

Zhang, L. L.; Yang, Y.; Jiang, Y.; Zhu, W.; and Liu, Y. 2020. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 692–693.