FREE BITS: PLATFORM-AWARE LATENCY OPTIMIZATION OF MIXED-PRECISION NEURAL NETWORKS FOR EDGE DEPLOYMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixed-precision quantization, where a deep neural network's layers are quantized to different precisions, offers the opportunity to optimize the trade-offs between model size, latency, and statistical accuracy beyond what can be achieved with homogeneous-bit-width quantization. However, the search space for layer-wise quantization policies is intractable, and the execution latency of mixed-precision networks is related non-trivially and non-monotonically to precision, depending on the deployment target. This establishes the need for hardware-aware, directed heuristic search algorithms. This paper proposes a hybrid search methodology for mixed-precision network configurations consisting of a hardware-agnostic differentiable search algorithm followed by a hardware-aware heuristic optimization to find mixed-precision configurations latency-optimized for a specific hardware target. We evaluate our algorithm on MobileNetV1 and MobileNetV2 and deploy the resulting networks on a family of multi-core RISC-V microcontroller platforms, with different hardware characteristics. We achieve up to 29.2% reduction of end-to-end latency compared to an 8-bit model at a negligible accuracy drop from a full-precision baseline on the 1000-class ImageNet dataset. We demonstrate speedups relative to an 8-bit baseline even on systems with no hardware support for sub-byte arithmetic at negligible accuracy drop. Furthermore, we show the superiority of our approach to both a purely heuristic search and differentiable search targeting reduced binary operation counts as a proxy for inference latency.

1 INTRODUCTION

The number of internet of things (IoT) devices deployed is growing rapidly and is projected to reach 19.1 billion by 2025 (Statista, Inc., 2022). Concurrently, the field of deep learning (DL) has developed very rapidly, with DL-based algorithms currently representing the state of the art in applications such as image recognition (Brock et al., 2021), natural language processing (Bao et al., 2021), processing of biomedical data (Ingolfsson et al., 2020) and many more. To process the data collected by IoT sensor nodes accurately, efficiently and with an adequate level of privacy, the trend has been to execute DL algorithms directly on the edge devices that collect it. The application scenarios for IoT nodes dictate that these devices be battery-powered and low-cost, which imposes severe constraints on their power consumption and memory available on the node. Microcontroller units (MCUs) have been a popular target for edge deployment of Deep Neural Network (DNN) due to their ubiquity and low cost, and extensive research has been conducted into designing efficient DNN models and techniques to enable inference on MCUs (Lin et al., 2021; Banbury et al., 2021).

An important technique to reduce the memory footprint of DNNs is *quantization*, where model parameters and intermediate activations are represented in low-precision formats. It has been shown that integer quantization to 8 bits does not impact a model's statistical performance (Jain et al., 2020), and new generations of MCUs make use of Single Instruction Multiple Data (SIMD) instructions to enable faster and more energy-efficient execution of programs implementing 8-bit arithmetic (Gautschi et al., 2017; Arm Ltd.). Quantization to even lower bit-widths has also seen widespread interest (Choi et al., 2018; Zhou et al., 2017; Alemdar et al., 2017) and the hardware community has followed suit, proposing low-precision DNN execution engines as well as instruction set architecture (ISA) extensions to accelerate networks quantized to sub-byte precision. However, homogeneous

quantization to sub-byte precisions often incurs a non-negligible accuracy penalty. To find the best trade-off between execution latency and statistical accuracy, *mixed-precision quantization* proposes to quantize different parts (usually at the granularity of individual layers) of the network to different precisions. As the search space of mixed-precision configurations for a given network is exponential in the number of layers and thus intractable, various works have proposed directed search algorithms for such configurations. Multiple works have applied differentiable neural architecture search (DNAS) to mixed-precision search. However, these approaches generally rely on a proxy for latency, such as binary operation (BOP) count, to guide the search (van Baalen et al., 2020; Cai & Vasconcelos, 2020). On real hardware platforms, latency is highly dependent on factors such as hardware implementation, memory hierarchy, tiling, and kernel implementation, none of which are directly linked to the number of BOPs in a network. In this work, we propose a mixed-precision latency optimization method consisting of a hardware-agnostic differentiable search step based on the Bayesian Bits algorithm (van Baalen et al., 2020), followed by a hardware-aware, profiling-based heuristic which both reduces execution latency and improves accuracy by increasing the precision in layers where higher precisions achieve lower latency. To reach a desired target latency, the configurations can be further refined in an optional greedy search step. In evaluations on MobileNetV1 (MNv1) and MobileNetV2 (MNv2), deployed to a cycle-accurate RISC-V multi-core simulator, our approach results in an accuracy-latency trade-off curve that dominates those produced by either differentiable search or greedy heuristics on their own. To our best knowledge, we demonstrate for the first time end-to-end deployment of mixed-precision networks to an MCU-class platform that exhibit not only a reduced memory footprint but also reduced execution latency by up to 29.2% at full-precision equivalent classification accuracy. Our key contributions are the following:

- We present a lightweight method to find latency-optimized mixed-precision quantization configurations for DNNs, consisting of a hardware-agnostic differentiable model search and hardware-aware heuristics, allowing the quick generation of optimized configurations for different platforms,
- we compose an end-to-end flow consisting of precision search, training, generation of integerized models and deploy the found configurations on a cycle-accurate simulator for high-performance RISC-V MCU systems and
- we analyze the resulting accuracy-latency trade-offs, showing that our approach achieves an end-to-end latency reduction by up to 29.2% vs. 8-bit quantization at full-precision equivalent classification accuracy and finds Pareto-dominant configurations with respect to homogeneous 4-bit quantization.

2 Related Work

Approaches to mixed-precision configuration search in literature can be broadly clustered into three categories: *Differentiable search* algorithms which jointly train precision-selection parameters and model parameters, *static precision assignment*, where the precision of each layer is determined off-line based either on heuristic criteria or model statistics and *reinforcement learning (RL)*-based approaches which train a RL agent to assign precisions to each layer.

van Baalen et al. (2020) adopt a differentiable-search approach, which we adapt for the first step of our algorithm. In their Bayesian Bits algorithm, quantization to a given precision is decomposed by first quantizing a tensor to the lowest supported bit-width, and higher precisions are recovered by adding the difference to the next quantization level to the quantized tensor. During training, the addition of the error tensors is controlled by stochastic, continuous-value gates whose distribution is parametrized by trainable parameters. To encourage low-precision quantization, a regularization term is added to the loss, with each precision gate contributing a term proportional to the BOP cost its associated layer incurs in the precision level controlled by the gate. EdMIPS (Cai & Vasconcelos, 2020) uses a less complex approach, training gate parameters for each available precision which are used to assemble the final tensor by weighting its differently quantized versions with the softmax distribution parametrized by the gates. The regularizer term used is analogous to that of Bayesian Bits, with a term proportional to the expected value of BOPs being added to the classification loss. Nikolić et al. (2020) take a similar approach, learning a single continuous parameter used to interpolate between bit-widths per layer, and Wu et al. (2018) train only weight precisions, using the Gumbel Softmax technique (Jang et al., 2017) to sample the precision gates' values. All of these

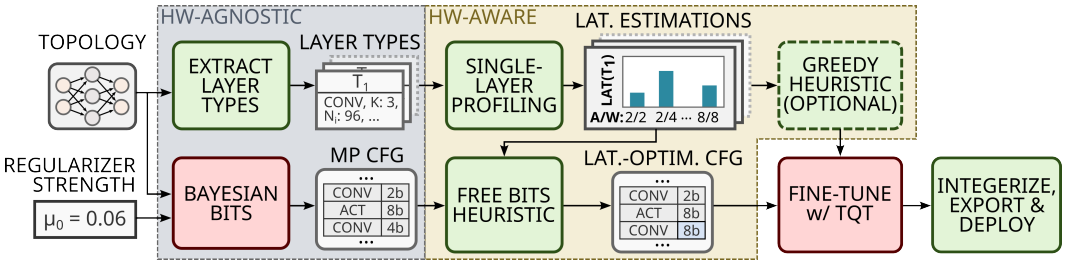


Figure 1: Overview of the proposed algorithm. Mixed-precision configurations (**MP CFGs**) for a given topology are first generated using Bayesian Bits. These configurations are then optimized for the target platform using profiling results for the layer types occurring in the topology by increasing the precisions in layers where this results in lower latency (**LAT.**). Optionally, the resulting configurations can be further refined to meet a latency target using a greedy heuristic. The final configuration is fine-tuned with Quantization-Aware Training (QAT) using the Trained Quantization Thresholds (TQT) algorithm, after which it can be deployed to the target hardware. Green boxes represent steps with low computational effort while steps with high computational effort (i.e., involving neural network training) are represented by red boxes.

algorithms have in common that the objective functions use BOP count (or another quantity that is monotonically related to precision, such as model size) as the target metric. As we show in Section 3, BOP count does not necessarily correlate with latency, and there is no trivial adaptation to enable these algorithms to optimize for execution latency directly. Furthermore, differentiable search does not allow for enforcing hard constraints on the configurations during the search.

In contrast, techniques performing static assignment of layer-wise precisions may target any performance metric, including execution latency, but can not directly optimize statistical accuracy at the same time. Rusci et al. (2019) greedily reduce weight precision to allow a network to fit in on-device storage and decrease activation precision of those layers which would not otherwise fit into device memory. As their target platform is an off-the-shelf MCU without native sub-byte arithmetic support, the low-precision operations incur runtime overhead, and the reduced memory footprint comes at the cost of increased runtime latency. Yao et al. (2020) calculate a second-order sensitivity metric for each layer and use integer linear programming (ILP) to optimize layer-wise precisions for execution latency, model size or BOP count while minimizing the estimated cumulative disturbance due to quantization.

In the category of RL-based algorithms, Wang et al. (2019) use deep deterministic policy gradient (DDPG) and restrict the agent's action space to networks that fulfill a user-determined (latency, energy, or model size) constraint, such that the reward function only considers the evaluated network's accuracy drop relative to full-precision accuracy. Elthakeb et al. (2020) use a combined reward function incorporating both the state of quantization and the classification accuracy with the proximal policy optimization (PPO) actor-critic algorithm to find weight-only quantization policies.

3 Free Bits

Free Bits is a multi-step method to find mixed-precision configurations of DNNs optimized for low latency on a given target hardware platform. In the first step, we apply a variant of Bayesian Bits with different regularizer strengths to find reduced-precision configurations of the targeted network architecture. The Bayesian Bits regularizer penalizes high BOP counts and has no concept of latency (and indeed does not support targeting latency optimization directly, see Section 3.1). Thus, the resulting configurations will parametrize mixed-precision networks which are not optimized for any particular hardware platform and which may even exhibit higher inference latency than, e.g., an all-8-bit baseline on the target platform.

In the second step, we optimize the mixed-precision configurations found with Bayesian Bits for a specific target. To achieve this, we first profile the latency of each unique layer type in the network in all allowed precision configurations on the target platform. This profiling data is used to update the initial configuration by choosing, for every layer in the network, the lowest-latency precision setting

that is higher or equal to that of the initial configuration. As the precision increase is expected to improve both latency and statistical accuracy, we name this step the *free bits* heuristic.

The latency of a mixed-precision network resulting from the first two steps is not bounded by a hard constraint, and if no satisfactory configuration is found, we apply a greedy heuristic in the third step to reach the latency target. When starting from a configuration with latency higher than the target, the heuristic successively decreases the precision of layers where this results in the largest latency reduction until the latency target is met. When starting from a configuration with a latency lower than the target, precision is increased in those layers where the latency penalty is the lowest.

The final configuration is then fine-tuned using a modified version of the TQT algorithm (Jain et al., 2020) and automatically converted to an integer-only model, which can be fed to a deployment backend for the target platform.

Our method emphasizes versatility and efficiency: Decoupling the differentiable search for low-precision configurations from the platform-specific latency optimization means that the computationally intensive first step only needs to be performed once for every network. From the mixed-precision configurations it produces, latency-optimized configurations can be generated for different hardware targets efficiently, as the second and third steps only require profiling data from the target platform.

3.1 Differentiable Mixed-Precision Search

To find the initial mixed-precision configurations which are latency-optimized in the second step, we apply two variants of the Bayesian Bits algorithm. Bayesian Bits aims to reduce a network's total BOP count with a regularizer that penalizes each precision gate's contribution to the expected BOP count individually (see also Section 2). There are two reasons why the algorithm cannot target latency directly. First, Bayesian Bits treats activation and linear operator layers individually, summing up the regularizer terms for every layer and optimizing each layer's gates independently. However, the latency of a single layer is determined jointly by the precision of input activations and weights and cannot be decomposed into additive contributions from each precision. Second, the regularizer terms for an individual layer's gates are added together, yielding a monotonously increasing penalty as a layer's expected precision increases. As shown in Figure 4, this is not necessarily the case for the execution latency of a layer: Instead, a layer may have lower latency when executed in a higher precision than in a lower one, depending on the hardware platform. In addition to the original Bayesian Bits algorithm, we also employ a modified version where the gate parameters for each precision are shared between a linear operator layer and its input activation, enforcing equal input and weight precisions. This modification accounts for the fact that on our target platforms, the theoretical throughput for a layer with non-equal activation and weight precisions is bounded by the higher of the two precisions.

3.2 Free Bits Heuristic

We update the configurations found by the latency-agnostic Bayesian Bits with a target-specific heuristic using profiling data from the hardware target platform. This step relies on two core ideas: First, we assume our target platform executes networks layer-by-layer, which implies $L_{net} \approx \sum_{i=1}^{N} L_i$ for the total execution latency L_{net} of an N-layer network where the i-th layer is executed with latency L_i. This is the case for most systems on which DNNs inference is run. Second, increasing a layer's input activation or weight precision never decreases the network's statistical accuracy.

Following the first assumption, we characterize each unique linear operator in the target network as a *layer type*. A layer type is defined as the tuple of all parameters which determine how a computational kernel is invoked, e.g., input dimensions, number of input/output channels, number of channel groups, and kernel size. For each layer type occurring in the network, we measure the execution latency on the target platform for all supported combinations of input and weight bit-widths (b_{in}, b_{wt}). With this estimation of $\{L_i\},\ i \in [1, N]$, we iterate over the layers in the network found by Bayesian Bits and check for every layer if higher-precision configurations of the same layer type with a lower estimated latency exist. If so, we update the layer's precision configuration to that with the lowest latency estimation. By the two assumptions above, the resulting network's execution la-

Algorithm 1 Profiling-based Heuristic Precision Search

Input:

LD: Latency dictionary mapping layer type c and precisions (b_{in}, b_{wt}) to a measured latency
C_{net}: Dictionary of layer types and precisions representing a mixed-precision network, of the form $\left\{i : \left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right)\right\}_{i=1}^{N}$
P_{all}: Set of allowed combinations (b_{in}, b_{wt}) of input and weight precisions

Output:

C'_{net}: Latency-optimized mixed-precision configuration of the input network

function HIGHER($(b_{in,1}, b_{wt,1}), (b_{in,2}, b_{wt,2})$) ▷ Returns True if precision 1 $\geq$ precision 2
  return $(b_{in,1} \geq b_{in,2}) \wedge (b_{wt,1} \geq b_{wt,2})$
end function
$C' \leftarrow C$
for all $i, \left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right) \in C_{net}$ **do**
  $lat^i_0 \leftarrow LD\left[\left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right)\right]$ ▷ Initial latency
  $cands \leftarrow \{(b_{in}, b_{wt}) \,|\, (b_{in}, b_{wt}) \in P_{all},$ ▷ Select lower-or-equal-latency configurations with higher-or-equal precisions
    $LD\left[\left(t^i, (b_{in}, b_{wt})\right)\right] \leq lat^i_0,$
    HIGHER$((b_{in}, b_{wt}), (b^i_{in}, b^i_{wt}))\}$
  $best \leftarrow \underset{(b_{in}, b_{wt}) \in cands}{\arg\min} LD[(t^i, (b_{in}, b_{wt}))]$ ▷ Select lowest-latency candidate
  $C'_{net}[i] \leftarrow (t^i, best)$ ▷ Update net configuration
end for

tency and statistical accuracy will be upper-bounded and lower-bounded, respectively, by those of the configuration found by Bayesian Bits. As it is expected to produce strictly superior configurations in terms of latency and statistical accuracy, we call this procedure the *free bits* heuristic. A pseudocode description of the procedure is shown in Algorithm 1.

3.3 Greedy Mixed-Precision Search

The configurations produced by the free bits heuristic can optionally be further refined with a greedy heuristic. Given a latency target L_{tgt} and a network configuration C_0 with estimated execution latency L_0, we aim to modify C_0 to reach a latency lower than, but as close as possible to, L_{tgt}. If $L_0 < L_{tgt}$, we apply the heuristic in the upward direction, seeking to increase the precision of as many layers as possible to maximize the accuracy improvement. Accordingly, we increase the precision of layers where this carries the lowest latency penalty. Conversely, in the downward direction (i.e., $L_0 > L_{tgt}$), we want to decrease the precision of as few layers as possible to minimize accuracy drop and thus choose the layers where a precision reduction results in the largest latency reduction. In the case $L_0 > L_{tgt}$, we additionally try to keep the layer-wise precision reductions applied "small", e.g, we prefer modifying two layers' configurations from 8b/8b to 8b/4b to modifying a single layer from 8b/8b to 8b/2b to achieve the same latency improvement. This is based on the experience that very low bit-widths have a disproportionate impact on classification accuracy. A pseudocode description of the two versions of the greedy heuristic is given in Appendix A.2.

This greedy heuristic can be applied to arbitrary C_0, and, applied to homogeneous-precision baseline configurations, serves as a test to assess the utility of the initial differentiable-search step: If applying this low-cost heuristic directly yields an accuracy-latency curve that is not Pareto-dominated by that resulting from the full algorithm, the computationally expensive differentiable search serves no useful purpose. The results of this comparison are detailed in Section 4.

3.4 Quantization-Aware Fine-Tuning and Deployment

Having arrived at a latency-optimized mixed-precision configuration, we perform QAT to fine-tune the network's parameters using a generalized version of the TQT algorithm. Our implementation of TQT differs from the original algorithm only in that we do not force clipping bounds to be exact powers of two. For the conversion of full-precision networks to fake-quantized (FQ) models, QAT

and generation of deployable integer-only models, we use the [omitted for double-blind review][1] framework, which allows automating large parts of this flow. For integerization, we follow the procedure referred to as integer channel norm (ICN) by Rusci et al. (2019) and *dyadic quantization* by Yao et al. (2020). The inputs to element-wise addition nodes which occur in networks with residual connections are quantized to 8 bits, and an equal quantization step size is enforced during the QAT phase. Likewise, the outputs of adder nodes are always quantized to 8-bit precision.

4 RESULTS

4.1 EXPERIMENTAL SETUP

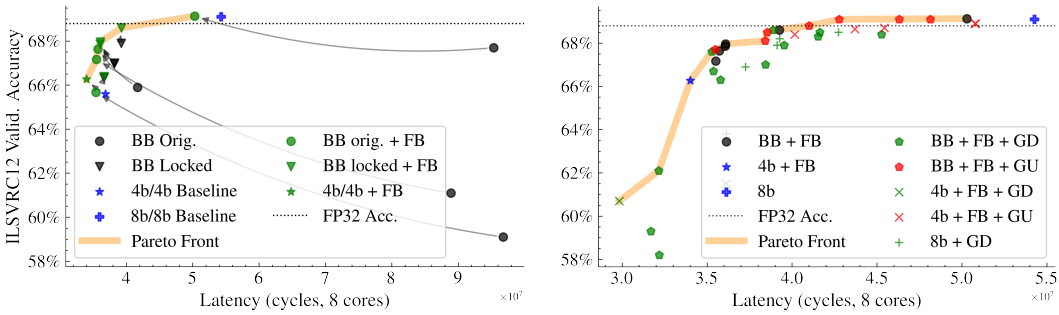
We performed experiments on two network architectures, applying the procedure proposed in Section 3 to MNv1 (Howard et al., 2017) and MNv2 (Sandler et al., 2018). We used width multipliers of 0.75 for MNv1 and 1.0 for MNv2. The input resolution was 224×224 for both networks. We train our networks on the ILSVRC2012 (Russakovsky et al., 2015) 1000-class dataset and report top-1 classification accuracies on the validation set.

Differentiable Mixed-Precision Search and QAT Fine-Tuning We applied the two variants of Bayesian Bits described in Section 3.1 to the MNv1 and MNv2 network topologies. The hyperparameters for Bayesian Bits training are listed in Table 3. The configurations produced by our algorithm (as well as those produced by Bayesian Bits in the case of MNv1) were fine-tuned with TQT using the hyperparameters listed in Table 2. In accordance with the capabilities of our hardware target (see below), the precisions Bayesian Bits can select from are 2, 4, and 8 bits for both weights and activations. We did not use the pruning mechanism of Bayesian Bits, i.e., 2-bit gates are always fully turned on.

Profiling, Deployment and Hardware Targets We use [omitted for double-blind review]'s automated integerization flow to generate precision-annotated, integer-only ONNX models, which are consumed by the DORY (Burrello et al., 2020) deployment backend. DORY generates compilable C code leveraging the PULP-NN (Garofalo et al., 2020a) kernel library, which we run on GVSOC, a cycle-accurate, open-source simulator for multi-core RISC-V systems. The hardware platforms we target are open-source RISC-V MCUs of the parallel ultra-low-power (PULP) family. One core, designated the fabric controller, orchestrates system operation, while compute-intensive tasks are executed on a PULP cluster of 8 RI5CY cores (Gautschi et al., 2017). System memory is split into two parts. A low-bandwidth L2 memory (parametrized to $512\,\mathrm{KiB}$ for MNv1 and $640\,\mathrm{KiB}$ for MobileNetV2 to accommodate the larger code size) stores program code and data and is used for partial result storage. The cluster cores operate on $64\,\mathrm{KiB}$ of high-bandwidth L1 scratchpad memory, optimized for low access contention. This hierarchical memory structure necessitates *tiled* execution of a network's layers with each tile's input, output, and weight data fitting into the L1 scratchpad. Tiling is automatically performed by DORY. If the total size of a layer's inputs, outputs, and weights exceeds the size of the L2 memory, an off-chip HyperRAM memory is used to store intermediate activations. Off-chip memory is also used to store the weights for all network layers.

All cores in the target system implement the base RV32IMF ISA in addition to the custom XpulpV2 extensions. We evaluate our found configurations on three systems, each containing a cluster whose cores have varying degrees of sub-byte arithmetic support. The first system's cluster implements only XpulpV2, which supports only 8-bit SIMD arithmetic. We refer to this system as *XpulpV2*. The second system implements the XpulpNN (Garofalo et al., 2020b) extension, which additionally provides support for packed-SIMD sub-byte arithmetic (for 2- and 4-bit data). However, XpulpNN's sub-byte arithmetic instructions require operands to have equal bit-widths. For operands of mismatching precisions, the lower-precision operands must first be unpacked in software to the larger data size. We refer to this version of the system as *XpulpNNv1*. The third system's cluster implements an improved version of XpulpNN which eliminates the runtime overhead from unpacking lower-precision operands by performing it transparently in hardware. We refer to this version of the system as XpulpNNv2. To generate the profiling data used by the heuristic steps of our algorithm, we again use DORY to generate and export dummy networks for all layer types in all precision configurations.

[1] Code will be open-sourced upon publication



(a) (b)

Figure 2: Latency-Accuracy trade-off of mixed-precision MobileNetV1 configurations running on a PULP system with XpulpNNV1 ISA extensions. a shows the configurations found by Bayesian Bits before and after applying the free bits heuristic, with grey arrows indicating the effect of applying the heuristic. b shows the Pareto-optimal configurations from a along with configurations produced by the greedy search described in Section 3.3. **BB orig./locked**: Configurations found by the original Bayesian Bits algorithm and the modified version enforcing symmetric activation/weight precisions, respectively. **FB**: Free bits. **GU/GD**: Greedy search in the upward/downward direction.

4.2 Latency-Accuracy Trade-Offs for XpulpNNv1

MobileNetV1 Figure 2a shows the latency-accuracy trade-off for MNv1 deployed to a PULP system with the XpulpNNv1 ISA extensions, with the effect of the free bits heuristic indicated. We observe that the original Bayesian Bits algorithm generally does not produce low-latency configurations. This confirms that even on hardware with native sub-byte arithmetic support, low BOP count does not directly translate to low latency. With two exceptions, applying the free bits heuristic improves the latency of all configurations substantially while increasing classification accuracy. For the homogeneous-precision $4\,\text{b}/4\,\text{b}$ baseline, the heuristic increases the precision of 12 layers, improving latency and accuracy by 7% and 0.7 percentage points, respectively. As symmetric activation and weight precisions are theoretically optimal for XpulpNNv1's hardware implementation of sub-byte arithmetic, this is a non-trivial result. The free bits heuristic lifts the previously uncompetitive configurations found by the original Bayesian Bits algorithm to the Pareto front, leading to accuracy and latency gains of $1.4 - 6.6$ percentage points and $12.3\% - 61.6\%$, respectively. The most accurate configuration matches the 8b/8b baseline in statistical accuracy at 69.1% while reducing execution latency by 7.6%, and the configuration at the Pareto front's knee point improves the execution latency by 27.8% at a classification accuracy within 0.2 percentage points of the 32-bit floating-point baseline of 68.8%.

Figure 2b shows the effect of applying the greedy search (see Section 3.3) to the configurations produced by Bayesian Bits and the freebie heuristic, as well as the homogeneous-precision baselines. The greedy heuristic produces mostly non-optimal configurations when applied in the downward direction. In contrast, when applied in the upward direction to configurations found by our combined algorithm, it yields Pareto-optimal networks, refining the original Pareto front and finding a configuration that reduces latency by 29.2% at an accuracy drop of only 0.3% from the full-precision baseline. Finally, we note that while the configurations produced by the upward greedy heuristic starting from the 4b/4b baseline are completely dominated by those found by Bayesian Bits and the free bits heuristic, they form a Pareto front which lies within 0.3 percentage points of classification accuracy.

We conclude that i) the differentiable-search step is indeed helpful in finding mixed-precision configurations optimized for low end-to-end latency, ii) the greedy search step applied to these configurations in the upward direction reliably refines the Pareto front, and iii) greedy search which increases layer-wise precisions starting from a low-precision baseline can provide a low-cost alternative to the multi-step procedure.

MobileNetV2 Figure 3 shows the latency-accuracy trade-off of MNv2 configurations produced by Bayesian Bits modified with the free bits heuristic running on the XpulpNNv1 system. The baseline

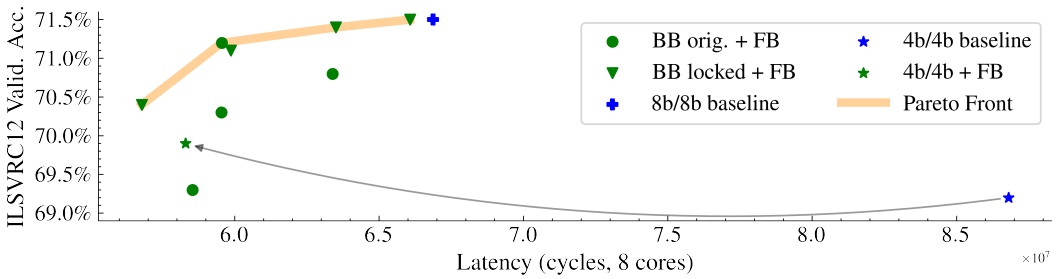


Figure 3: Latency-Accuracy tradeoff of MobileNetV2 configurations optimized for XpulpNNv1. The grey arrow indicates the effect of applying the heuristic to the 4b/4b baseline.

Acc. Margin	ISA	MobileNetV1		MobileNetV2	
		Lat. vs. 8b	Acc.	Lat. vs. 8b	Acc.
8b Baseline	*all*	+0%	69.1%	+0%	71.5%
0.5 pp.	XPv2	−5.5%	69.3%	−3.4%	71.0%
	XPNNv1	−27.9%	68.6%	−10.9%	71.2%
	XPNNv2	−28.6%	68.6%	−15.3%	71.0%
1.5 pp.	XPv2	−5.5%	69.3%	−6.3%	70.7%
	XPNNv1	−34.4%	67.6%	−15.1%	70.4%
	XPNNv2	−35.1%	67.6%	−15.3%	71.0%
4b + FB	XPv2	−3.5%	67.7%	−7.7%	70.9%
	XPNNv1	−37.1%	66.3%	−12.8%	69.9%
	XPNNv2	−39.8%	66.6%	−25.7%	69.6%
4b Baseline	XPv2	+49.9%	65.6%	+37.0%	69.3%
	XPNNv1	−32.3%	65.6%	+48.9%	69.3%
	XPNNv2	−38.3%	65.6%	−23.4%	69.3%

Table 1: Configurations within margins of 0.5 and 1.5 percentage points (**pp.**) of 8b/8b classification accuracy for PULP systems implementing different ISA extensions: XpulpV2 (**XPv2**), XpulpNNv1 (**XPNNv1**) and XpulpNNv2 (**XPNNv2**). The configurations listed here were found without the greedy heuristic search step. **4b+FB**: target-specific free bits heuristic applied to homogeneously quantized 4b/4b network.

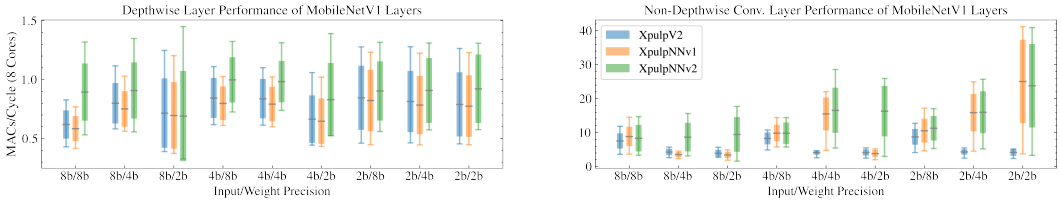
4b/4b configuration contains many asymmetric-precision convolutional layers due to adder node outputs being quantized to 8 bits. This leads to a latency higher than that of the 8b/8b baseline, which is reduced by the free bits heuristic by 46%. At the same time, classification accuracy is improved by 0.6 percentage points. Nevertheless, the resulting configuration is not Pareto-optimal with respect to those produced by our algorithm. In particular, the locked-precision version of Bayesian Bits, when combined with the free bits heuristic, produces configurations that dominate the optimized 4b/4b baseline and the 8b/8b baseline. The configuration at the Pareto front's knee point reduces execution latency by 10.9% at an accuracy penalty of only 0.3 percentage points relative to the 8b/8b baseline.

4.3 Free Bits Across Different Target Platforms

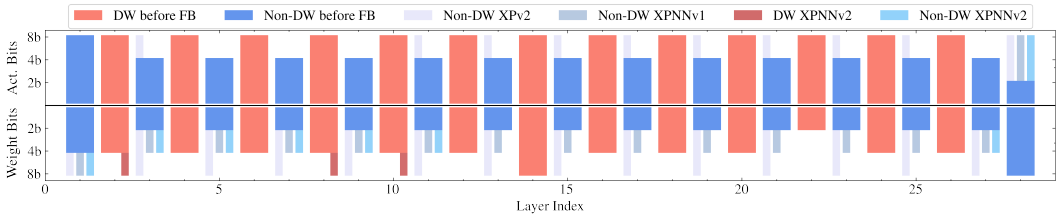
To evaluate the portability of our algorithm, we optimized MNv1 and MNv2 configurations found with Bayesian Bits for the three different PULP systems with different levels of support for sub-byte arithmetic, described in Section 4.1. Table 1 shows the lowest-latency configurations within 0.5 and 1.5 percentage points of classification accuracy of the 8b/8b baseline. The configurations listed were found with only the first two steps of our algorithm. Notably, our approach achieves latency reductions even on the XpulpV2 system without hardware support for sub-byte arithmetic, which can be attributed to a lower data movement overhead thanks to larger tile sizes. While relative latency reduction and the resulting accuracies are very similar between XpulpNNv1 and XpulpNNv2 on MNv1, significant differences can be observed on MNv2's 4b/4b baselines, both before and after applying the free bits heuristic. On XpulpNNv2, both exhibit significantly lower latency than the

8b/8b baseline, while on XpulpNNv1 the unoptimized baseline is uncompetitive and the optimized configuration is dominated in accuracy and latency by other configurations.

4.4 ANALYSIS



(a)



(b)

Figure 4: (a): Results of the layer-wise profiling of MNv1 on systems implementing XpulpV2 (**XPv2**), XpulpNNV1 (**XPNNv1**) and XpulpNNV2 (**XPNNv2**). Notably, depthwise (**DW**) layers may see a speedup from sub-byte quantization even on XpulpV2, which has no hardware support for sub-byte arithmetic. (b) shows the modifications performed by the free bits heuristic for the three target platforms. The free bits (**FB**) heuristic acts differently according to each hardware platform's characteristics.

To explain the mechanism by which the free bits heuristic reduces latency on different target platforms, it is helpful to consider the results of the layer-wise profiling step, shown in Figure 4a. Despite having no native sub-byte arithmetic support, XPv2 sometimes exhibits speedups from reduced precisions due to reduced tiling overheads in the memory-bound depthwise (DW) layers as well as in the early layers of the network, which have the largest activation tensor sizes. Conversely, XPNNv1 supports sub-byte arithmetic, but non-DW layers with lower weight than activation precisions incur overhead from weight unpacking, which results in lower performance. XpulpNNv2 does not see this performance degradation as the unpacking is performed by the hardware with no latency penalty. This explains the action of the free bits heuristic (Figure 4b): Activation and weight precisions of non-DW layers are increased to 8 bits on many layers for XPv2 and set to be equal for XPNNv1, while for XPNNv2, many layers are left in asymmetric precision.

5 CONCLUSION

In this paper, we have presented *Free Bits*, an efficient method to find latency-optimized mixed-precision network configurations for inference on edge devices. Taking advantage of the fact that, depending on the target platform, increasing input or weight precision may lead to lower execution latency, the method optimizes mixed-precision configurations found by the hardware-agnostic Bayesian Bits differentiable search algorithm. To further refine the precision configurations found in this way, a greedy heuristic can be applied. Deploying the MNv1 and MNv2 configurations found with our algorithm on a family of high-performance MCU-class RISC-V platforms, we find that, i) with hardware support for sub-byte arithmetic, MNv1 end-to-end latency can be reduced by 30% while retaining full-precision equivalent accuracy, ii) even without such hardware support, mixed-precision quantization enables a latency reduction of up to 7.7%, and iii) the found configurations offer a superior accuracy-latency trade-off to homogeneous 4-bit and 8-bit quantization.

REFERENCES

Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frederic Petrot. Ternary neural networks for resource-efficient AI applications. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2017-May, pp. 2547–2554, 2017. ISBN 9781509061815. doi: 10.1109/IJCNN.2017.7966166.

Arm Ltd. *Armv8-M Architecture Reference Manual*. Arm Ltd.

Colby Banbury, Chuteng Zhou, Igor Fedorov, Ramon Matas Navarro, Urmish Thakker, Dibakar Gope, Vijay Janapa Reddi, Matthew Mattina, and Paul Whatmough. MicroNets: Neural Network Architectures for Deploying TinyML Applications on Commodity Microcontrollers. In A. Smola, A. Dimakis, and I. Stoica (eds.), *Proceedings of Machine Learning and Systems*, pp. 517–532, 2021.

Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. BEiT: BERT Pre-Training of Image Transformers. Number Mim, pp. 1–17, jun 2021. URL http://arxiv.org/abs/2106.08254.

Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. High-Performance Large-Scale Image Recognition Without Normalization. 2021. URL http://arxiv.org/abs/2102.06171.

Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. DORY: Automatic end-to-end deployment of real-world DNNs on low-cost IoT MCUs. *arXiv*, (732631):1–14, 2020. ISSN 23318422.

Zhaowei Cai and Nuno Vasconcelos. Rethinking Differentiable Search for Mixed-Precision Neural Networks. 2020. URL http://arxiv.org/abs/2004.05795.

Jungwook Choi, Pierce I.Jen Chuang, Zhuo Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Bridging the accuracy gap for 2-bit Quantized Neural Networks (QNN). *arXiv*, pp. 1–10, 2018. ISSN 23318422.

Ahmed T Elthakeb, Prannoy Pilligundla, Fatemehsadat Mireshghallah, Amir Yazdanbakhsh, and Hadi Esmaeilzadeh. ReLeQ : A Reinforcement Learning Approach for Automatic Deep Quantization of Neural Networks. *IEEE Micro*, 40(5):37–45, sep 2020. ISSN 0272-1732. doi: 10.1109/MM.2020.3009475. URL https://ieeexplore.ieee.org/document/9141414/.

Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. Pulp-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2164), 2020a. ISSN 1364503X. doi: 10.1098/rsta.2019.0155.

Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Luca Benini, and Davide Rossi. XpulpNN: Enabling energy efficient and flexible inference of quantized neural network on RISC-V based IoT end nodes. *arXiv*, 14(8):1–16, 2020b. ISSN 23318422.

Michael Gautschi, Pasquale Davide Schiavone, Andreas Traber, Igor Loi, Antonio Pullini, Davide Rossi, Eric Flamand, Frank K. Gurkaynak, and Luca Benini. Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2700–2713, oct 2017. ISSN 1063-8210. doi: 10.1109/TVLSI.2017.2654506. URL https://ieeexplore.ieee.org/document/7864441/.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. ISSN 15071367. doi: 10.1016/S1507-1367(10)60022-3. URL http://arxiv.org/abs/1704.04861.

Thorir Mar Ingolfsson, Michael Hersche, Xiaying Wang, Nobuaki Kobayashi, Lukas Cavigelli, and Luca Benini. EEG-TCNet: An Accurate Temporal Convolutional Network for Embedded Motor-Imagery Brain–Machine Interfaces. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2958–2965. IEEE, oct 2020. ISBN 978-1-7281-8526-2. doi: 10.1109/SMC42975.2020.9283028. URL https://ieeexplore.ieee.org/document/9283028/.

Sambhav R. Jain, Albert Gural, Michael Wu, and Chris H. Dick. Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks. In *Proceedings of Machine Learning and Systems*, pp. 112—-128, 2020. URL http://arxiv.org/abs/1903.08066https://proceedings.mlsys.org/paper/2020/file/e2c420d928d4bf8ce0ff2ec19b371514-Paper.pdf.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–13, 2017.

Ji Lin, Wei-Ming Chen, Han Cai, Chuang Gan, and Song Han. MCUNetV2: Memory-Efficient Patch-based Inference for Tiny Deep Learning. (NeurIPS):1–17, 2021. URL http://arxiv.org/abs/2110.15352.

Miloš Nikolić, Ghouthi Boukli Hacene, Ciaran Bannon, Alberto Delmas Lascorz, Matthieu Courbariaux, Yoshua Bengio, Vincent Gripon, and Andreas Moshovos. BitPruning: Learning Bitlengths for Aggressive and Accurate Quantization. pp. 1–9, 2020. URL http://arxiv.org/abs/2002.03090.

Manuele Rusci, Alessandro Capotondi, and Luca Benini. Memory-Driven Mixed Low Precision Quantization For Enabling Deep Network Inference On Microcontrollers. 2019. URL http://arxiv.org/abs/1905.13082.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. ISSN 15731405. doi: 10.1007/s11263-015-0816-y.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 81(2):4510–4520, feb 2018. ISSN 10636919. doi: 10.1109/CVPR.2018.00474.

Statista, Inc. Number of internet of things (iot) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/, May 2022.

Mart van Baalen, Christos Louizos, Markus Nagel, Rana Ali Amjad, Ying Wang, Tijmen Blankevoort, and Max Welling. Bayesian bits: Unifying quantization and pruning. *Advances in Neural Information Processing Systems*, 2020-Decem(NeurIPS), 2020. ISSN 10495258.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization with mixed precision. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:8604–8612, 2019. ISSN 10636919. doi: 10.1109/CVPR.2019.00881. URL http://arxiv.org/abs/1811.08886.

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search. pp. 1–11, 2018. URL http://arxiv.org/abs/1812.00090.

Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael W. Mahoney, and Kurt Keutzer. HAWQV3: Dyadic Neural Network Quantization. 2020. URL http://arxiv.org/abs/2011.10680.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. *Proceedings - 2017 International Conference on Learning Representations*, pp. 1–14, feb 2017.

A APPENDIX

A.1 TRAINING HYPERPARAMETERS

Table 2: QAT hyperparameters for MobileNetV1 and MobileNetV2 trained with TQT

	MobileNetV1	MobileNetV2
Epochs	11	13
Batch Size	256	340
Opt.	SGD w/ momentum 0.9	SGD w/ momentum 0.9
LR_0	0.001	0.00075
LR decr. [a]	4,7	5,8
Quant. start [b]	0	1
Act. clip init. [c]	Const. 6.0	Const. 6.0/Max[d]

[a] LR is decreased by a factor of 0.1 at specified epochs
[b] Activations and weights are quantized starting from specified epoch
[c] Const. x: clipping bounds initialized to x,
Max: clipping bounds initialized to maximum value observed during unquantized training
[d] Activations inserted before and after adder nodes have clipping bounds initialized to Max, all others to Const. 6.0

Table 3: Bayesian Bits hyperparameters for MobileNetV1 and MobileNetV2

	MobileNetV1	MobileNetV2
Epochs	11	12
Batch Size	256	150
Net Opt.	SGD w/ momentum 0.9	SGD w/ momentum 0.9
$LR_{0,net}$	0.001	0.00075
Prec. Gate Opt.	Adam	Adam
LR_{gate}	0.0001	0.0002
LR decr. [c]	4,7	4,7
Quant. start	0	0
Act. clip init.	Const. 6.0	Const. 6.0
Φ_0 [a]	2.0	2.0
μ_0 [b]	0.01, 0.03, 0.06, 0.12	0.01, 0.03, 0.06, 0.12

[a] Initialization of precision gating parameters
[b] Global regularizer strength
[c] Only network parameters' learning rate is decreased

A.2 GREEDY LATENCY-MATCHING HEURISTICS

Algorithm 2 Greedy Heuristic Precision Search - Latency Reduction

Input:

LD: Latency dictionary mapping layer type t and precisions (b_{in}, b_{wt}) to a measured latency
C_{net}: Dictionary mapping layer indices to layer types and precisions representing a mixed-precision network with latency L_0, of the form $\left\{i : \left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right)\right\}_{i=1}^{N}$ with $\sum_{i=1}^{N} LD[C_{net}[i]] = L_0$
P_{all}: Set of allowed combinations b_{in}, b_{wt} of input and weight precisions
O_p: Ordering of P_{all}, e.g. $\{2\text{b} : 0,\ 4\text{b} : 1,\ 8\text{b} : 2\}$
$L_{tgt} > L_0$: Target latency

Output:

C'_{net}: Latency-optimized mixed-precision configuration of the input network

function DIST($(b_{in,1}, b_{wt,1}), (b_{in,2}, b_{wt,2})$) ▷ Returns distance between two layer precisions based on O_p
  return $O_p\left[b_{in,1}\right] - O_p\left[b_{in,2}\right] + O_p\left[b_{wt,1}\right] - O_p\left[b_{wt,2}\right]$
end function
function GET_MOVES(cfg, s_{max}) ▷ Returns net configuration with largest latency decrease for each layer by decreasing each layer's precision at most by s_{max}
  $moves \leftarrow cfg$ ▷ Initialize with starting configuration
  for all $i, \left(t_i, \left(b^i_{in}, b^i_{wt}\right)\right) \in cfg$ **do** ▷ Iterate over network layers
    $lat^i_0 \leftarrow LD\left[cfg\left[i\right]\right]$ ▷ Initial latency of layer i
    $cands \leftarrow \{(b_{in}, b_{wt}) \mid (b_{in}, b_{wt}) \in P_{all},\ LD\left[(t_i, (b_{in}, b_{wt}))\right] \leq lat^i_0,\ \text{DIST}((b_{in}, b_{wt}), \left(b^i_{in}, b^i_{wt}\right)) < s_{max}\}$ ▷ All precisions within $step_{max}$ precision distance with latency $\leq lat^i_0$
    $best \leftarrow \underset{(b_{in}, b_{wt}) \in cands}{\arg\min} LD\left[(t_i, (b_{in}, b_{wt}))\right]$ ▷ Select lowest-latency candidate for layer i
    $moves\left[i\right] \leftarrow (t_i, best)$ ▷ Update configuration
  end for
  return $moves$
end function
$C' \leftarrow C$ ▷ Initialize to original net configuration
for all $step_{max} \in [0, ..., 2\left|O_p\right|]$ **do** ▷ Prioritize low-distance modifications
  $moves \leftarrow \text{GET\_MOVES}(C, step_{max})$
  if $\sum_{i=1}^{N} LD\left[moves\left[i\right]\right] < L_{tgt}$ **then** ▷ Modified configuration satisfies latency target
    while $\sum_{i=1}^{N} LD\left[C'\left[i\right]\right] > L_{tgt}$ **do** ▷ Not all layers may need to be modified
      $move_{appl} \leftarrow \underset{i}{\arg\max}\, LD\left[C\left[i\right]\right] - LD\left[moves\left[i\right]\right]$ ▷ Find highest-impact modification and apply it
      $C'\left[move_{appl}\right] \leftarrow moves\left[move_{appl}\right]$
      $moves \leftarrow moves \setminus moves\left[move_{appl}\right]$ ▷ Discard applied move
    end while
    return C'
  end if
end for
return Failure ▷ If no configuration was found, declare failure

Algorithm 3 Greedy Heuristic Precision Search - Latency Increase

Input:

LD: Latency dictionary mapping layer type t and precisions (b_{in}, b_{wt}) to a measured latency
C_{net}: Dictionary mapping layer indices to layer types and precisions representing a mixed-precision network with latency L_0, of the form $\left\{i : \left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right)\right\}_{i=1}^{N}$ with $\sum_{i=1}^{N} LD[C_{net}[i]] = L_0$
P_{all}: Set of allowed combinations b_{in}, b_{wt} of input and weight precisions
O_p: Ordering of P_{all}, e.g. $\{2\text{b} : 0,\ 4\text{b} : 1,\ 8\text{b} : 2\}$
$L_{tgt} < L_0$: Target latency

Output:

C'_{net}: Latency-optimized mixed-precision configuration of the input network

function HIGHER($(b_{in,1}, b_{wt,1}), (b_{in,2}, b_{wt,2})$) ▷ Returns True if precision 1 > precision 2
  return $((b_{in,1} > b_{in,2}) \wedge (b_{wt,1} \geq b_{wt,2})) \vee ((b_{in,1} \geq b_{in,2}) \wedge (b_{wt,1} > b_{wt,2}))$
end function
function GET_MOVES(cfg) ▷ Return layer-wise precision increases with lowest latency degradation
  $moves \leftarrow cfg$ ▷ Initialize to input config.
  for all $i, \left(t_i, \left(b^i_{in}, b^i_{wt}\right)\right) \in cfg$ **do** ▷ Loop over network layers
    $cands \leftarrow \{(b_{in}, b_{wt}) \mid (b_{in}, b_{wt}) \in P_{all}, \text{HIGHER}((b_{in}, b_{wt}), \left(b^i_{in}, b^i_{wt}\right))\}$ ▷ Find all higher-or-equal precision configurations
    if $cands \neq \emptyset$ **then**
      $best \leftarrow \underset{(b_{in}, b_{wt}) \in cands}{\arg\min} LD\left[(t_i, (b_{in}, b_{wt}))\right]$ ▷ Choose lowest-latency candidate
      $moves\left[i\right] \leftarrow (t_i, best)$ ▷ Modify layer's configuration
    end if
  end for
  return $moves$
end function
$C' \leftarrow C$ ▷ Initialize to original net configuration
while True do ▷ Loop until failure or success
  $moves \leftarrow \text{GET\_MOVES}(C')$ ▷ Increase layer-wise precision at minimal latency penalty
  if $moves == C'$ **then**
    return Failure ▷ If no moves are found, the algorithm fails
  end if
  if $\sum_{i=1}^{N} LD\left[moves\left[i\right]\right] < L_{tgt}$ **then** ▷ Moves do not increase latency past the target...
    $C' \leftarrow moves$ ▷ ...so apply all of them
  else ▷ Moves can increase latency past the target
    while True do
      $move_{appl} \leftarrow \underset{i}{\arg\min}\, LD\left[moves\left[i\right]\right] - LD\left[C'\left[i\right]\right]$ ▷ Lowest-latency move
      if $\sum_{\substack{i=1 \\ i \neq move_{appl}}}^{N} LD\left[C'\left[i\right]\right] + LD\left[moves\left[move_{appl}\right]\right] > L_{tgt}$ **then**
        return C' ▷ If the move would increase latency past the target, return
      else
        $C'\left[move_{appl}\right] \leftarrow moves\left[move_{appl}\right]$ ▷ Otherwise, apply it
      end if
    end while
  end if
end while