



Towards a tailored mixed-precision sub-8-bit quantization scheme for Gated Recurrent Units using Genetic Algorithms

Miccini, Riccardo; Cerioli, Alessandro; Laroche, Clément; Piechowiak, Tobias; Sparsø, Jens; Pezzarossa, Luca

Published in:
Proceedings of tinyML Research Symposium'24

Publication date:
2024

Document Version
Peer reviewed version

[Link back to DTU Orbit](#)

Citation (APA):
Miccini, R., Cerioli, A., Laroche, C., Piechowiak, T., Sparsø, J., & Pezzarossa, L. (in press). Towards a tailored mixed-precision sub-8-bit quantization scheme for Gated Recurrent Units using Genetic Algorithms. In *Proceedings of tinyML Research Symposium'24*

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Towards a tailored mixed-precision sub-8-bit quantization scheme for Gated Recurrent Units using Genetic Algorithms

Riccardo Miccini
Alessandro Cerioli

rimi@dtu.dk
alceri@dtu.dk

Technical University of Denmark
Kongens Lyngby, Denmark

Clément Laroche
Tobias Piechowiak
claroche@jabra.com
topiechowiak@jabra.com
GN Audio
Ballerup, Denmark

Jens Sparsø
Luca Pezzarossa
jspa@dtu.dk
lpez@dtu.dk
Technical University of Denmark
Kongens Lyngby, Denmark

ABSTRACT

Despite the recent advances in model compression techniques for deep neural networks, deploying such models on ultra-low-power embedded devices still proves challenging. In particular, quantization schemes for Gated Recurrent Units (GRU) are difficult to tune due to their dependence on an internal state, preventing them from fully benefiting from sub-8bit quantization. In this work, we propose a modular integer quantization scheme for GRUs where the bit width of each operator can be selected independently. We then employ Genetic Algorithms (GA) to explore the vast search space of possible bit widths, simultaneously optimizing for model size and accuracy. We evaluate our methods on four different sequential tasks and demonstrate that mixed-precision solutions exceed homogeneous-precision ones in terms of Pareto efficiency. Our results show a model size reduction between 25% and 55% while maintaining an accuracy comparable with the 8-bit homogeneous equivalent.

KEYWORDS

Neural networks, Quantization, Neural architecture search

ACM Reference Format:

Riccardo Miccini, Alessandro Cerioli, Clément Laroche, Tobias Piechowiak, Jens Sparsø, and Luca Pezzarossa. 2024. Towards a tailored mixed-precision sub-8-bit quantization scheme for Gated Recurrent Units using Genetic Algorithms. In *Proceedings of tinyML Research Symposium (tinyML Research Symposium '24)*. San Francisco, CA, USA, 7 pages.

1 INTRODUCTION

In recent years, deep neural networks have been making strides in a wide range of fields, including computer vision, natural language processing, and audio processing. In particular, Recurrent Neural Networks (RNNs) — specifically their Long Short-Term Memory (LSTM) [14] and Gated Recurrent Unit (GRU) [6] variants — have been successfully applied to a variety of sequence modelling tasks, such as speech recognition, machine translation, keyword spotting, and speech enhancement. However, deploying such models on ultra-low-power embedded devices still proves challenging, due to their high computational complexity and memory footprint. Specifically,

data transfer — and by extension model size — is a major contributor to the overall energy footprint of a deep learning model [15].

Model compression techniques such as quantization have been successfully applied to Convolutional Neural Networks (CNNs), allowing them to be deployed on embedded devices with limited computational resources. Remarkably, the quantization of RNNs has not been explored as extensively, potentially due to the additional complexity introduced by their recurrent nature. Among the most notable works, [1] propose binary, ternary, and quaternary quantization schemes for RNNs and evaluate it on sentiment analysis, [11] combines structural pruning and 8-bit quantization to optimize LSTMs for speech enhancement on a Cortex-M7 embedded platform, [20] presents quantization schemes for the standard LSTM and its variants, based on fixed-point arithmetic, evaluating them on speech recognition; finally [26] employs mixed-precision FP16 and 8-bit integer quantization to deploy speech enhancement models based on LSTMs or GRUs on a RISC-V embedded target.

Neural Architecture Search (NAS) is a recently introduced technique for automating the design of neural networks. It consists of exploring the search space of possible neural network architectures to optimize them according to one or several metrics, such as model accuracy, size, or computational complexity. Within NAS, several methods for traversing the vast search space of possible architectures have been proposed, including gradient-based methods [27], Reinforcement Learning [5], and Genetic Algorithms (GA). In particular, the latter has been adopted in computer vision as a way to find an optimal CNN structure for face recognition [24] and, more recently, achieving better performances than manually-derived CNNs for image classification [23]. Furthermore, GA have been demonstrated to be effective in the refinement of RNNs for natural language processing [19]. Vector extensions of genetic algorithms have also been proposed: [21] proposes NSGA-Net, a multi-objective GA for optimizing neural networks for image classification, while [28] uses GA to simultaneously optimize inference time and model size for anomaly detection. For a comprehensive overview of the field, we refer the reader to [4, 17].

Nevertheless, most NAS solutions focus on optimizing architectural aspects of the network such as the number of layers or the type of activation functions, whereas the hyperparameters associated with quantization are often handled manually. This can be particularly strenuous, especially in mixed-precision quantization settings where the search space of quantization parameters is vast. Addressing this gap, we contribute to the field of quantization by employing GA to efficiently derive mixed-precision quantization schemes for GRUs which simultaneously maximizes accuracy and

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

tinyML Research Symposium '24, April 2024, San Francisco, CA

© 2024 Copyright held by the owner/author(s).

minimizes model size. Such a quantization scheme is particularly relevant for embedded devices with limited storage, memory, or computational resources, where model weights and activation size are critical factors. In this work we contribute by:

- Designing an integer-only modular quantization scheme for GRUs that can be used for sub-8-bit computation;
- Employing Genetic Algorithms to derive Pareto-optimal mixed-precision quantization schemes;
- Conducting a comprehensive evaluation of the solution, showcasing its effectiveness across four diverse sequence classification tasks of varying complexity.

2 BACKGROUND

This section provides a theoretical background on the main building blocks of this work, namely GRUs, quantization, and GA.

2.1 Gated Recurrent Unit

Recurrent Neural Networks, are a class of neural networks first introduced in [25] as a way to process sequential data of arbitrary length. RNNs process data sequentially over several time steps: at each time step, the network takes as input the current data and the previous output, allowing the network to maintain an internal state. Because of this, RNNs are well suited for processing data such as text or audio. However, in their most basic form, RNNs suffer from the so-called *vanishing gradient problem* [3], which makes them difficult to train on long sequences.

To address this issue, LSTM units [14], and subsequently GRUs [6], were introduced. Both LSTMs and GRUs implement a number of gates to modulate the flow of new and past information, allowing a more compact internal representation that does not suffer from vanishing gradient.

GRUs are a simplified version of LSTMs, which implement only two gates and use a single state vector instead of two. This is achieved through the following equations (based on the formulation in [7]):

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \quad (1a)$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \quad (1b)$$

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn})) \quad (1c)$$

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{(t-1)} \quad (1d)$$

where x_t is the input at the current time step, $h_{(t-1)}$ is the previous hidden state. The gates are the *reset gate* r_t and the *update gate* z_t , implemented in Eqs. (1a) and (1b), while the *hidden state* h_t , which also serves as output, is implemented in Eq. (1d). Finally, GRUs also include a *new state* (or *candidate activation*) n_t which is computed in Eq. (1c), and is used to compute the new hidden state. The reset gate controls how much of the previous state is kept in memory, while the update gate controls how much of the new state is added to the memory. The GRU architecture, along with all its building blocks and operations described above, is depicted in Fig. 1.

2.2 Quantization

In the field of signal processing, quantization is the process of mapping a continuous set of values to a finite one. When applied to neural networks, quantization is a compression technique aimed at

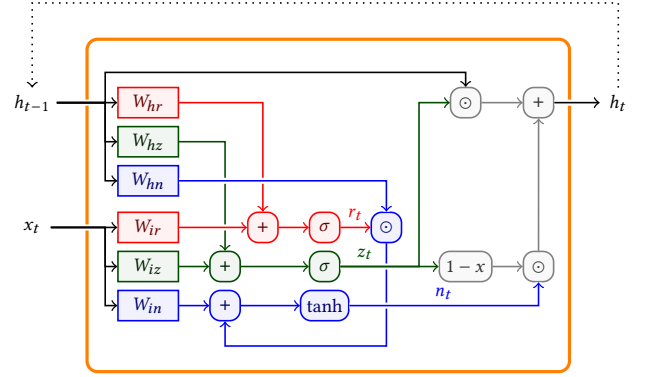


Figure 1: A GRU with its building blocks exposed; the different colors denote the operations associated with Eq. (1).

reducing the memory footprint and computational complexity of the model by limiting the precision of its weights and activations. Since most deep learning models are heavily over-parameterized, there is often a large margin for reducing their representational precision without significantly affecting performances [12].

In this work, we consider linear (or uniform) quantization, where the original real values are compressed into finite sets of evenly-spaced quantization levels. What follows is a brief overview of the technique; for a more comprehensive treatment please refer to [12, 16, 22]. Linear quantization is achieved by mapping a real value r to its corresponding quantized q using the following transforms:

$$q = \left\lfloor \frac{r}{S} \right\rfloor - Z \quad (2a)$$

$$r \approx \tilde{r} = S(q + Z) \quad (2b)$$

where \tilde{r} is the dequantized real value, S is the scaling factor, and Z is the zero-point, which are computed as follows:

$$S = \frac{\beta - \alpha}{2^b - 1} \quad ; \quad Z = \left\lfloor \frac{2^b - 1 - \beta}{S} \right\rfloor \quad (3)$$

where b is the bit-width of the quantization, and α and β are the lower and upper bounds of the clipping range, respectively. Choosing the clipping range is a crucial step in quantization, as it determines the range of representable values; a straightforward choice is often $\alpha = \min(r)$ and $\beta = \max(r)$.

When $\alpha \neq -\beta$ this is known as *asymmetric quantization*, since the quantization range is not symmetric around zero. Conversely, when the clipping range is centered around zero, i.e. $\alpha = -\beta$, this is known as *symmetric quantization*. When using symmetric quantization, we have that $Z = 0$, which simplifies Eq. (2). While asymmetric quantization is more expressive due to the tighter clipping range, symmetric quantization is preferred due to its lower computational overhead. Oftentimes, in practice, weights are quantized symmetrically while activations are quantized asymmetrically [12].

The quantization parameters S and Z can be computed by simply observing the minimum and maximum values of weights and activations over a representative calibration dataset: this is called *Post-Training Quantization* (PTQ). Alternatively, they can be computed during *Quantization Aware Training* (QAT) by adding quantization and dequantization nodes to the model graph — namely

implementing Eq. (2) — and training the model with the quantization nodes in place. In this latter case, since the quantization step in Eq. (2a) is not differentiable, the *Straight-Through Estimator* (STE) [2] is used to backpropagate the gradients through the quantization nodes. This additional step allows the model to learn to adapt to the quantization, which often results in better performances than PTQ. In this work, we evaluate our method on both quantization strategies, enabling us to fully appreciate its versatility.

2.3 Genetic Algorithms

Genetic algorithms [13] are a class of optimization metaheuristics based on the principles of natural selection and biological evolution. They mimic the logic of biological and genetic adaptation of individuals (solutions) to an environment (search space) and are extensively used in a large variety of domains to optimize multi-dimensional functions that are not differentiable, where classical gradient descent algorithms are not adequate. Starting from an initial population, the algorithm iterates over the following phases, illustrated in Fig. 2: evaluation, survival, selection, crossover and mutation, until an exit condition is met. A fundamental aspect of GA is the *genome*, a set of parameters representing a potential solution to the given optimization problem, as well as the basis for the crossover and mutation phases. In this context, each parameter is a *gene*. Genomes can be encoded in a variety of ways, including binary strings, real-valued vectors, or more sophisticated data structures allowing for discrete or continuous genes. Due to their reliance on a population of solutions, GA may be computationally demanding but are also suitable for parallel computing.

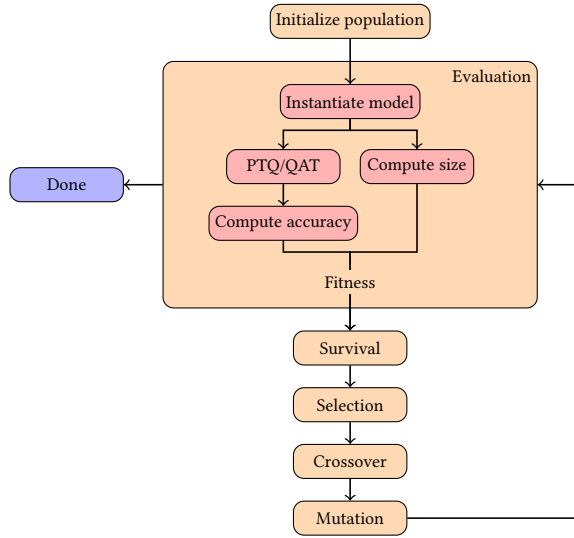


Figure 2: Typical flow of a genetic algorithm, with red blocks indicating the custom evaluation pipeline.

There exist many variants of GA. The Non-dominated Sorting Genetic Algorithm (NSGA) uses a vector fitness function, making it suitable for multi-objective optimization problems. In the specific case of NSGA-II [8], which we use in our work, a two-dimensional vector and selection heuristic based on the concept of Pareto efficiency (or optimality) is used. According to it, a solution whereby

all the objectives are worse than those of another solution is said to be “dominated”. Conversely, a solution that fares better than the alternatives by at least one objective is considered to be “non-dominated”. In the selection phase, the solutions are sorted based on non-dominated sorting and crowding distance. This approach has already been used in the context of NAS [21] and presents itself as an adequate solution when there is a need to optimize a neural network on the basis of different objectives, such as accuracy but also other performance or efficiency-related objectives like latency, model size, and memory footprint.

3 MODULAR QUANTIZATION

In this section, we present our approach to integer quantization of GRU layers, as well as a method for exploring the search space of possible bit-widths using the GA described in Section 2.3.

3.1 Quantization scheme for GRU

Building upon the methods described in Section 2.2, we propose a linear quantization scheme for GRUs where each operation is quantized independently, allowing for heterogeneous bit-widths.

By inspecting Fig. 1, we can see that a GRU layer is composed of 4 main types of operations: linear/dense layers, element-wise additions, element-wise multiplications, and non-linear activation functions. For each operation, we can define a quantization scheme by applying Eq. (2b) to the real-valued parameters involved in the operation definition — namely its inputs, outputs, and relevant weights — and then solving for the quantized output.

To accommodate integer-only hardware, we further restrict the quantization to integer-only computation. This is done by representing the combined scaling factors M_* in Eqs. (9), (13) and (17) as fixed-point with an n -bit fractional part, so that $M_* \approx 2^{-n}M_0$ (where $*$ indicates any of the specific qualifiers introduced in the aforementioned equations). This scaling can then be trivially implemented with a multiplication and a shift operation.

In the following subsections, we elaborate on how each of the operations involved in a GRU layer is quantized.

3.1.1 Linear layer. A linear (or dense) layer is defined as:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (4)$$

where \mathbf{x} is the input vector, \mathbf{W} is the weight matrix, \mathbf{b} is the bias vector, and \mathbf{y} is the output vector. By applying Eq. (2b) to each of the parameters, we obtain:

$$S_y(q_y - Z_y) = S_w(q_w - Z_w)S_x(q_x - Z_x) + S_b(q_b - Z_b) \quad (5)$$

Subsequently, we can assume symmetric quantization for weights and biases ($Z_w = Z_b = 0$), and adopt a shared scaling factor for matrix multiplication and addition ($S_b = S_wS_x$). Applying this and solving for the quantized output q_y yields:

$$S_y(q_y - Z_y) = S_wS_x(q_wq_x - Z_xq_w + q_b) \quad (6)$$

$$q_y = \frac{S_wS_x}{S_y}(q_wq_x - Z_xq_w + q_b) + Z_y \quad (7)$$

We further simplify computation by substituting the following:

$$\frac{S_wS_x}{S_y} = M \quad ; \quad q_b - Z_xq_w = q_{\text{bias}} \quad (8)$$

where M is the combined scaling factor (representable as a fixed-point number as mentioned earlier) and q_{bias} is a pre-computed term. This finally gives us:

$$q_y = M(q_w q_x + q_{\text{bias}}) + Z_y \quad (9)$$

3.1.2 Element-wise sum. When quantizing an element-wise sum, we similarly apply Eq. (2b) to each of the parameters involved in the sum, and then solve for the quantized output:

$$\mathbf{y} = \mathbf{x}_1 + \mathbf{x}_2 \quad (10)$$

$$S_y(q_y - Z_y) = S_1(q_1 - Z_1) + S_2(q_2 - Z_2) \quad (11)$$

$$q_y = \frac{S_1}{S_y}(q_1 - Z_1) + \frac{S_2}{S_y}(q_2 - Z_2) + Z_y \quad (12)$$

We are now left with two combined scaling factors, $M_\alpha = S_1/S_y$ and $M_\beta = S_2/S_y$. Notice how M_β contributes to matching the quantization grid of the two operands, while M_α matches the final output range. After substituting the scaling factors, we obtain:

$$q_y = M_\alpha(q_1 - Z_1 + M_\beta(q_2 - Z_2)) + Z_y \quad (13)$$

3.1.3 Element-wise product. We once again apply the same familiar process as above:

$$\mathbf{y} = \mathbf{x}_1 \odot \mathbf{x}_2 \quad (14)$$

$$S_y(q_y - Z_y) = S_1(q_1 - Z_1) \odot S_2(q_2 - Z_2) \quad (15)$$

$$q_y = \frac{S_1 S_2}{S_y}(q_1 q_2 - q_1 Z_2 - q_2 Z_1 + Z_1 Z_2) + Z_y \quad (16)$$

This time, after some simplification, we are left with a single scaling factor $M_Y = S_1 S_2 / S_y$:

$$q_y = M_Y(q_1 q_2 - q_1 Z_2 - q_2 Z_1 + Z_1 Z_2) + Z_y \quad (17)$$

3.1.4 Activations. Finally, we consider the quantization of the sigmoid and hyperbolic tangent activation functions, here both denoted as σ . In this case, we build lookup tables (LUT), according to the following process:

- (1) Generate a vector of length L of quantized input values q_x between 0 and $2^b - 1$, where b is the given bit-width;
- (2) Compute its real-valued equivalent \tilde{x} using Eq. (2b);
- (3) Compute the real-valued activation $\tilde{y} = \sigma(\tilde{x})$;
- (4) Quantize \tilde{y} using Eq. (2a) to obtain the LUT q_y .

During inference, $\lfloor L/2^b - 1 \rfloor \cdot q_x$ is used as the table index, and the corresponding quantized output q_y is returned.

3.2 Quantization scheme search

As mentioned in Section 2.3, genetic algorithms operate on a genome representing each potential solution. In the context of our optimization problem, the genome is a vector of 17 genes, each representing the bit-width of a quantized operation in a GRU cell (i.e., each of the blocks in Fig. 1), encoded as an integer value in the range between 2 and 8. The combinations of genes and their bit-width values correspond to the search space of our problem.

We formulate our search task as a multi-objective maximization problem, where the objectives under consideration are the model accuracy and \widehat{M}^c , i.e. the normalized complement of the model size, defined as follows:

$$\widehat{M}^c = 1 - \frac{M_Q}{M_{\text{FP16}}} \quad (18)$$

where M_Q is the quantized model size and M_{FP16} is the model size of the floating-point baseline, both in bits. The model sizes M_* are computed as:

$$M_* = \sum_{\phi \in \Phi} \text{numel}(W^\phi) \cdot N_b^\phi + \text{numel}(b^\phi) \cdot 32 \quad (19)$$

where Φ is the set of all linear layers in the model and W^ϕ , b^ϕ , and N_b^ϕ are the weight matrix, bias vector, and chosen bit-width for the given layer ϕ , respectively; finally, $\text{numel}()$ is the number of elements in its argument. Note that the bit-width of the bias vector is fixed to 32 bits, which we assume to be the size of the accumulator register.

The overall genetic search workflow, along with the aforementioned evaluation pipeline, is depicted in Fig. 2. During the survival stage, a rank and crowding strategy is adopted, whereby solutions are first sorted into fronts based on non-domination, assigning lower ranks to Pareto-optimal solutions. To maintain diversity, preference is given to solutions with higher crowding distance, which measures the proximity of solutions within the same front; this ensures an even distribution across the Pareto front. Subsequently, mating candidates are selected through binary tournaments, based on their ranks and crowding distances, and new solutions are generated by means of Simulated Binary Crossover and Polynomial Mutation [9]. These two are crossover and mutation strategies, respectively, that are suitable for real-valued genes.

4 EXPERIMENTAL EVALUATION

We conducted a number of experiments to evaluate our methods, spanning 4 different sequential classification tasks: row-wise sequential MNIST, pixel-wise sequential MNIST, keyword spotting with 4 keywords, and keyword spotting with 10 keywords. In the following subsections, we describe the experimental setup used to conduct the genetic search, along with the details of each task, and discuss the results.

4.1 Setup

Each task was performed using a simple recurrent model with a single GRU layer followed by a linear layer. For each task, we trained an FP16 model and quantized it homogeneously, using either PTQ or QAT, with bit-widths ranging from 3 to 8; we use these models as our baselines. During training, we minimize the cross-entropy loss between the model predictions and the ground truth labels. Subsequently, we used NSGA-II to search for optimal mixed-precision quantization schemes. Table 1 provides an overview of the hyperparameters used during FP16 training, homogeneous quantization, and mixed-precision quantization (through genetic search).

During the genetic search, we randomly generated the genome for the initial population by sampling the search space — i.e., by generating a 17-element vector of uniformly-distributed integers in the closed interval [2, 8]. The search was terminated after 20 generations. This process was repeated for each task. We use the NSGA-II implementation provided by *pymoo*¹.

We then evaluated the models on the test set using the same metrics described in Section 3.2. To avoid overfitting on our test set, during the survival and selection phases of the genetic search

¹<https://pymoo.org/algorithms/moo/nsga2.html>

Table 1: Hyper-parameters for model training; mixed-precision values correspond to those used for QAT.

Parameter	FP16	Homogeneous	Mixed-precision
Batch size	256	1024/2048	1024
Epochs	50	30	12
Train. split	100%	100%	10%
Valid. split	5%	5%	5%
Valid. every	5	5	3
Optimizer	Adam [18]	Adam	Adam
Learning rate	10^{-3}	5×10^{-5}	5×10^{-5}

(see Section 2.3), the accuracy of each individual is computed on a separate validation split corresponding to an unseen 10% of data.

4.2 Sequential tasks

The following subsections describe the datasets, hyperparameters, and training procedures used for each task.

4.2.1 Sequential MNIST. For these experiments, we use the MNIST dataset [10], adapted to work for sequential tasks. The dataset comprises 28×28 gray-scale images of handwritten digits, divided into 60 000 training and 10 000 test samples.

For the row-wise variant, we feed each data point as a sequence of 28 time steps, each consisting of a 28-pixel feature vector; thus, each row of pixels is treated as a time step. For the pixel-wise variant, we feed each data point as a sequence of individual pixels, each treated as a one-dimensional feature vector. Since training with sequences of length 784 is exceedingly expensive, we resize the images to 19×19 pixels using bilinear interpolation before converting them to sequences, resulting in sequences of length 361.

For this set of experiments, we perform QAT on the models; for an overview of the training hyperparameters, please refer to Table 1. The models feature 128 and 256 hidden features for the row-wise and pixel-wise task variants, respectively.

4.2.2 Keyword spotting. For these experiments, we use the Speech Commands dataset [29], which comprises 65 000 one-second audio clips of 30 different words, divided into 20 000 training, 5 000 validation, and 10 000 test samples. The input data consists of 80 Mel-frequency log-power features extracted from the STFT of the audio signal, computed on 32-millisecond windows with 50% overlap, resulting in sequences of 63 time steps.

Although the original dataset contains 30 words, for our experiments we only consider the first 4 and 10 words, respectively. This is because, given its relatively simple architecture, the accuracy of the full-precision model on the full dataset was already quite low.

To further test the viability of our quantization scheme and parameter search, we perform PTQ on these models, instead of QAT. In this case, we use the entire training set for calibration. Similarly to the previous experiments, the models feature 128 and 256 hidden features for the 4-word and 10-word task variants.

4.3 Results

Fig. 3 shows their progressive improvement over each generation. As we can see, the model accuracy generally increases over time,

while the model size decreases. This trend can be observed in all four experiments, although the rate of improvement decreases steeply over time, and the maximum accuracy is often reached early on in the process. Indeed, in all the experiments besides *SpeechCmd10*, the median accuracy across generations oscillates dramatically while maintaining an upward trend, and we observe a relatively large variance for accuracy figures. In *SpeechCmd10*, the accuracy continues to increase up to the very last generations, hinting that extending the genetic search could have further improved our results. Conversely, the model size progressively decreases and converges in all experiments except *SpeechCmd4*, where a larger variance is observed. These observations are likely explained by the lack of inequality constraints on the objectives, causing lightweight but low-accuracy solutions to survive. Similarly, the absence of an *exploitation* phase prevents our search to further refine our high-quality specimens.

Fig. 4 shows the Pareto fronts for the search, for each of the four experiments. When comparing the Pareto front projected by the homogeneously-quantized baseline models with the one obtained by the mixed-precision search, we can see that the latter consistently dominates the former, with the exception of the 8-bit QAT/PTQ models, which achieve better accuracy than any of the heterogeneously-quantized models. In all the other cases, however, we achieve better or comparable accuracy and lower model size with our method. We can also see that there appear lots of mixed-precision solutions with chance-like accuracy, presenting as vertical stripes on the left side of the plot. However, these often comprise points from the earliest generations, further proving the effectiveness of the genetic search. While portions of our Pareto front present solutions with very low model size, oftentimes these provide very low accuracy, making them potentially unsuitable to fulfill the given task; nevertheless, they could be employed as part of an ensemble model, leveraging parallelization.

Furthermore, we tried to verify whether the discovered optimal solutions shared any characteristics that could be used to inform the network design and quantization. To do this, we fit a t-SNE dimensionality reduction model on the bit-widths associated with each building block of the GRU and subsequently color the dots according to the accuracy achieved. The emergence of clusters, as shown in Fig. 5, is evidence that there are some shared quantization patterns that are common among the best-performing models. Most notably, these clusters of high-performing solutions seem to occupy different regions of the t-SNE space, validating the need for bespoke quantization. Intuitively, this could be motivated by the different requirements of each task, e.g. problems featuring long sequences require more emphasis, and therefore higher resolution, on the model's internal state.

5 CONCLUSION

In this work, we presented a novel quantization scheme for GRUs and used Genetic Algorithms to simultaneously optimize for accuracy and model size by selecting the appropriate bit-width of each operation. Based on our preliminary results on a variety of simple sequence classification tasks, the mixed-precision solutions achieve better Pareto efficiency for our chosen metrics. Namely, in all experiments except one, we achieve a model size reduction

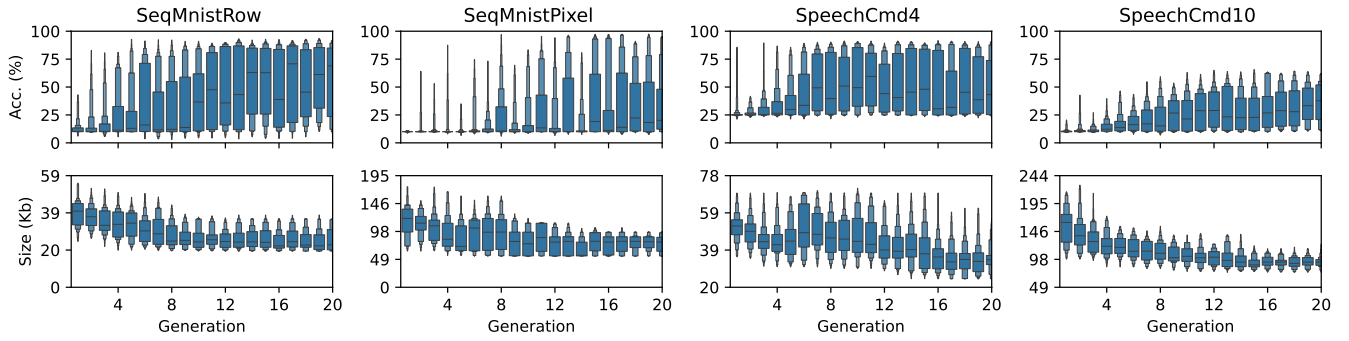


Figure 3: Enhanced box plot showing the distributions of accuracy and model size across generations for the given tasks.

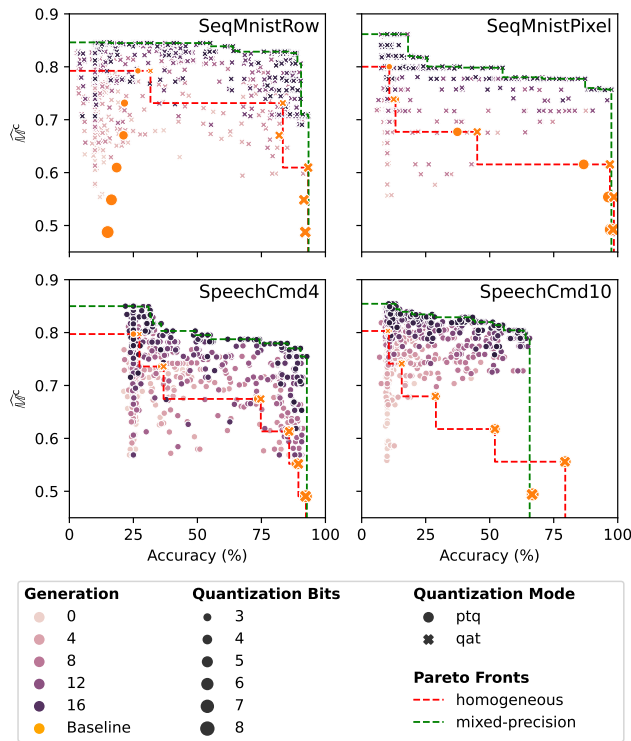


Figure 4: Pareto fronts (higher is better) for homogeneous (baseline) vs. mixed-precision (genetic search) quantization; each dot is a solution/individual.

between 25% and 55% while maintaining the same (or better) accuracy as the 8-bit homogeneously-quantized model. Similarly, when considering the relatively small 4-bit homogeneous baselines, we appreciate an increase in accuracy corresponding to between 2 to 4 times a similarly-sized heterogeneously-quantized model.

While promising, our solution presents some limitations. Most notably, to reap the efficiency and inference speed benefits of the heterogeneous quantization scheme, hardware support is essential. Furthermore, as an extension of this work, the genetic search could be enhanced by introducing an additional *exploitation* phase and

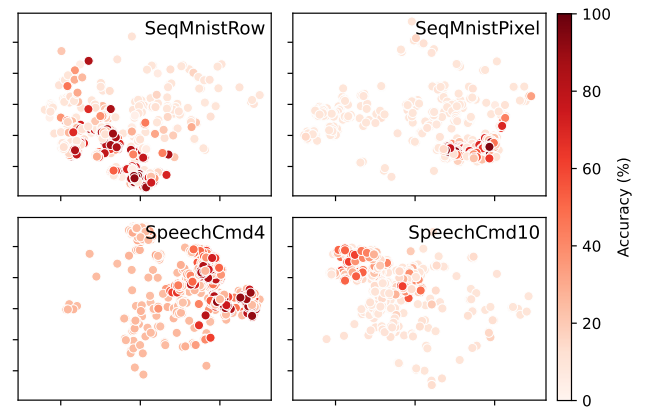


Figure 5: Dimensionally-reduced visualization (using t-SNE) of solutions' genomes, colored by accuracy.

by constraining our objectives to useful regions of the Pareto front, as mentioned in Section 4.

It remains to be determined whether our solution would scale to more challenging tasks such as speech enhancement, which would require a more complex model architecture and a larger dataset; these drawbacks could be ameliorated with knowledge distillation and dataset distillation techniques, respectively. Finally, it is worth noting that the quantized operators described here, along with the genetic search for optimal bit-widths, could also be applied to LSTMs; we leave this investigation for future work.

ACKNOWLEDGMENTS

We thank E. Njor for sharing his knowledge on NAS and R. James for the help in brainstorming the initial idea. This work has received funding from the European Union's Horizon research and innovation programme under grant agreement No 101070374.

REFERENCES

- [1] Md Zahangir Alom, Adam T. Moody, Naoya Maruyama, Brian C. Van Essen, and Tarek M. Taha. 2018. Effective Quantization Approaches for Recurrent Neural Networks. In *Proc. International Joint Conference on Neural Networks*. Institute of Electrical and Electronics Engineers Inc., Rio de Janeiro, Brazil, 8489341. <https://doi.org/10.1109/IJCNN.2018.8489341>

- [2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. <https://doi.org/10.48550/arXiv.1308.3432> arXiv:1308.3432 [cs].
- [3] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166. <https://doi.org/10.1109/72.79181>
- [4] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. 2021. A Comprehensive Survey on Hardware-Aware Neural Architecture Search. <https://doi.org/10.48550/arXiv.2101.09336> arXiv:2101.09336 [cs].
- [5] Krishna Teja Chitty-Venkata and Arun K. Somani. 2022. Neural Architecture Search Survey: A Hardware Perspective. *ACM Comput. Surv.* 55, 4, Article 78 (nov 2022), 36 pages. <https://doi.org/10.1145/3524500>
- [6] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). Association for Computational Linguistics, Doha, Qatar, 1724–1734. <https://doi.org/10.3115/v1/D14-1179>
- [7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. <http://arxiv.org/abs/1412.3555> arXiv:1412.3555 [cs].
- [8] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.
- [9] Kalyanmoy Deb, Karthik Sindhya, and Tatsuya Okabe. 2007. Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (London, England) (GECCO '07)*. Association for Computing Machinery, New York, NY, USA, 1187–1194. <https://doi.org/10.1145/1276958.1277190>
- [10] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.
- [11] Igor Fedorov, Marko Stamenovic, Carl Jensen, Li Chia Yang, Ari Mandell, Yiming Gan, Matthew Mattina, and Paul N. Whatmough. 2020. TinyLSTMs: Efficient Neural Speech Enhancement for Hearing Aids. *Proceedings of the Annual Conference of the International Speech Communication Association, Interspeech 2020- (2020)*, 4054–4058. <https://doi.org/10.21437/Interspeech.2020-1864>
- [12] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A Survey of Quantization Methods for Efficient Neural Network Inference. <http://arxiv.org/abs/2103.13630> arXiv:2103.13630 [cs].
- [13] David E Goldberg. 2013. *Genetic algorithms*. Pearson Education.
- [14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (11 1997), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735> arXiv:https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf
- [15] Mark Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, San Francisco, CA, USA, 10–14. <https://doi.org/10.1109/ISSCC.2014.6757323>
- [16] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Salt Lake City, UT, USA, 2704–2713. <https://doi.org/10.1109/CVPR.2018.00286>
- [17] Jeon-Seong Kang, Jinkyu Kang, Jung-Jun Kim, Kwang-Woo Jeon, Hyun-Joon Chung, and Byung-Hoon Park. 2023. Neural Architecture Search Survey: A Computer Vision Perspective. *Sensors* 23, 3 (2023), 1713.
- [18] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, Yoshua Bengio and Yann LeCun (Eds.). San Diego, CA, USA. <http://arxiv.org/abs/1412.6980>
- [19] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. 2022. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *IEEE Access* 10 (2022), 45736–45747.
- [20] Jian Li and Raziel Alvarez. 2021. On the quantization of recurrent neural networks. <http://arxiv.org/abs/2101.05453> arXiv:2101.05453 [cs].
- [21] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. 2019. NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (Prague, Czech Republic) (GECCO '19)*. Association for Computing Machinery, New York, NY, USA, 419–427. <https://doi.org/10.1145/3321707.3321729>
- [22] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. 2021. A White Paper on Neural Network Quantization. <http://arxiv.org/abs/2106.08295> arXiv:2106.08295 [cs].
- [23] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence (AAAI'19/IAAI'19/EAAI'19)*. AAAI Press, Honolulu, Hawaii, USA, Article 587, 10 pages. <https://doi.org/10.1609/aaai.v33i01.33014780>
- [24] Arash Rikhtegar, Mohammad Pooyan, and Mohammad Taghi Manzuri-Shalmani. 2016. Genetic algorithm-optimised structure of convolutional neural network for face recognition applications. *IET Computer Vision* 10, 6 (2016), 559–566. <https://doi.org/10.1049/iet-cvi.2015.0037> arXiv:https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-cvi.2015.0037
- [25] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (Oct. 1986), 533–536. <https://doi.org/10.1038/323533a0>
- [26] Manuele Rusci, Marco Fariselli, Martin Croome, Francesco Paci, and Eric Flamand. 2023. Accelerating RNN-Based Speech Enhancement on a Multi-core MCU with Mixed FP16-INT8 Post-training Quantization. *Communications in Computer and Information Science* 1752 (2023), 606–617. https://doi.org/10.1007/978-3-031-23618-1_41
- [27] Santanu Santra, Jun-Wei Hsieh, and Chi-Fang Lin. 2021. Gradient Descent Effects on Differential Neural Architecture Search: A Survey. *IEEE Access* 9 (2021), 89602–89618. <https://doi.org/10.1109/ACCESS.2021.3090918>
- [28] Chakkrit Termritthikun, Lin Xu, Yemeng Liu, and Ivan Lee. 2021. Neural Architecture Search and Multi-Objective Evolutionary Algorithms for Anomaly Detection. In *2021 International Conference on Data Mining Workshops (ICDMW)*. IEEE, Auckland, New Zealand, 1001–1008. <https://doi.org/10.1109/ICDMW53433.2021.00130>
- [29] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. arXiv:1804.03209 [cs.CL]