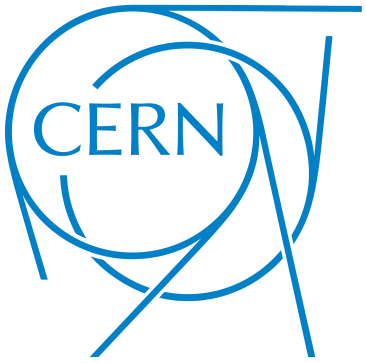
FITcompress for PQuant

Marlon Joshua Helbing

Supervisors: Roope Niemi & Vladimir Lončar

Date: 24 September 2025



Abstract

This work presents the integration of FITCompress into the PQuant framework, adapting the algorithm to support fixed-point quantization and various pruning methods for practical hardware deployment. FITCompress was modified to work with PQuant's Q(m,n) fixed-point representation by developing a principled approach for distributing bits between integer and fractional parts based on dynamic range analysis. To address the lack of explicit activation quantization in the original FITCompress paper, a calibration-based method was implemented that determines optimal bit allocations for activation units, pooling layers and model inputs. Our integration supports joint optimization of quantization and pruning for methods that use a global sparsity target, such as PDP and Wanda, while also offering quantization-only optimization for other pruning approaches. Experimental results on ResNet-20 with CIFAR-10 show that FITCompress achieves substantial compression with only minimal accuracy loss. With PDP and Wanda, the compressed models maintained accuracies within about one percentage point of the uncompressed baseline, even when reduced to just 6% (PDP) and 4% (Wanda) of the original model's size. FITCompress was also analyzed with other pruning methods that do not rely on a global sparsity target, demonstrating that the approach remains flexible even without direct pruning optimization. This integration bridges the gap between FITCompress's theoretical framework and hardware-oriented quantization, enabling efficient deployment on resource-constrained devices.

Contents

An Introduction to FITcompress and PQuant

1.1 FITcompress

1.1.1 Basic Setup

FITCompress [9] is a model compression algorithm that unifies **mixed-precision quantization** and **unstructured pruning** under a single theoretical framework. It formulates compression as a **path-planning problem** through a search space, guiding the model from an initial full-precision state to an optimally compressed configuration. In each iteration, FITcompress explores multiple candidate compression actions in parallel. Starting from the current best model, it generates L candidate models by quantizing exactly one layer's **weights** with a bit-width given by the **layerwise quantization scheduler** and 1 additional candidate model by applying pruning at the level dictated by the **pruning scheduler**. For each candidate, the **Fisher Information Trace** (FIT) [10] is used to approximate parameter sensitivity to the applied compression and calculate a **distance metric** to estimate how close the candidate is to an optimal compressed model. This distance metric is inspired by the A^* algorithm; it sums the distances from the initial to the current model and from the current to the final model, yielding the **full distance**. Among these $L+1$ candidates, the one with the lowest full distance is chosen as the next best candidate. By repeating this process iteratively until a pre-defined **compression rate** - defined as the fraction of the original model size (in bytes) that remains after applying pruning and quantization - is reached, FITcompress optimally moves through this **search space of candidate models**, interleaving quantization and pruning steps in a way that balances sparsity and precision while minimizing accuracy loss. FITcompress then returns optimal quantization settings per layer and a global pruning percentage. The main pipeline can be seen in Fig. 1.1.

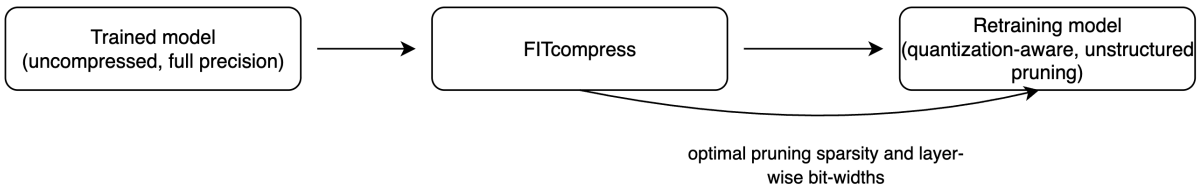


Figure 1.1: FITcompress pipeline.

1.1.2 Pruning

FITCompress performs global unstructured pruning by scoring every weight's **importance** with the Fisher Information Trace (FIT), then zeroing the lowest-scoring fraction κ across the entire network. The FIT scores capture each parameter's sensitivity, so pruning targets weights that minimally affect the loss.

1.1.3 Quantization

FITCompress uses **uniform symmetric quantization** for each layer's weights. The range is centered around zero and bounded by the maximum absolute weight value ma of the current layer, i.e. $[-ma, ma]$. This range is discretized into uniformly spaced levels determined by the chosen bit-width b, with the scale factor d controlling the spacing. Formally, the quantization step computes:

$$d = \frac{ma}{2^b - 1}, \quad mi = -ma,$$

$$q = \text{round}\left(\frac{w - mi}{d}\right),$$

and the corresponding dequantized weight is reconstructed as

$$\hat{w} = q \cdot d + mi.$$

1.1.4 Training

After FITCompress has determined an optimal compression configuration, the resulting quantized and pruned model is retrained to recover accuracy losses introduced by compression. Retraining is performed with **quantization-aware training** using the `brevitas` framework. In addition, the model undergoes **global unstructured pruning**, where the κ fraction of weights with the lowest importance scores are permanently masked out. Concretely, a pruning mask is generated at the beginning of training and applied across all layers, such that pruned weights are fixed to zero while the remaining parameters are retained for optimization.

1.2 PQuant

1.2.1 Basic Setup

PQuant is a library for training compressed machine learning models. It allows the user to define their model using the typical PyTorch layers and activations, such as nn.Linear and nn.ReLU(), and train the model while quantizing and pruning it. PQuant replaces the layers and activations it finds with a compressed (in the case of layers) or quantized (in the case of activations) variant. These automatically handle the quantization of the weights, biases and activations, and the pruning of the weights. Quantization, pruning and training parameters can be set easily via a **configuration file**.

1.2.2 Pruning

The library supports multiple pruning methods, namely Activation pruning (AP) [2], Autosparse (AS) [4], MDMM [6], continuous sparsification (CS) [7], DST [5], PDP [1] and Wanda [8]. The various pruning methods have different training steps, such as pre-training, normal training and fine-tuning.

1.2.3 Quantization

PQuant makes use of hardware-oriented numerical quantizers. In particular, it uses **fixed-point quantizers**. In this framework, numbers are represented in a $Q(m, n)$ format with one sign bit, m integer bits, and n fractional bits. To quantize a real value x into this format, the value is first scaled by 2^n and rounded to the nearest integer

$$q = \text{round}(x \cdot 2^n),$$

where q is the signed integer that encodes the bit pattern stored in memory. The corresponding dequantized real value is then reconstructed as

$$\hat{x} = q \cdot 2^{-n}.$$

Here, n determines the precision, since the step size between representable values is 2^{-n}. The integer part m does not appear directly in the equations, but it defines the maximum representable magnitude, i.e. the dynamic range. With m integer bits, the representable range is

$$x \in \left[-2^m,\, 2^m - 2^{-n}\right].$$

Integrating FITcompress into PQuant

2.1 Adding it into the PQuant Pipeline

Since FITCompress requires a pre-trained, uncompressed model as input, it was integrated between the pre-training and normal training phases. In this setup, a pre-trained model is passed to FITCompress, which determines optimal quantization bit-widths for the weight layers as well as a global pruning sparsity target. Once this search is completed, the discovered settings are written into the configuration file: the `layer_specific` field stores the quantization bit-width for each layer, while the `sparsity` field records the global pruning target. Finally, the pre-trained model together with these optimal compression settings is forwarded to the normal training and fine-tuning phase, where quantization-aware training and pruning-aware training is performed. The overall pipeline is illustrated in Fig. 2.1.

2.2 From symmetric to fixed point Quantization

Since the retraining in PQuant (i.e. normal training and fine-tuning) is carried out with fixed-point quantizers, the quantization settings determined during FITCompress must also be based on fixed-point quantization rather than the uniform symmetric quantization used in the original FITCompress implementation. To adapt FITCompress accordingly, first each possible bit-width assignment was reduced by one, since one bit must be reserved for the sign. Next, a logic for distributing the remaining bits between integer and fractional parts is required. Assume that at some iteration FITCompress assigns b bits to a given layer i. First the maximum absolute weight of that layer is computed, $\max |w|$. The number of integer bits must be large enough to cover this dynamic range, which is

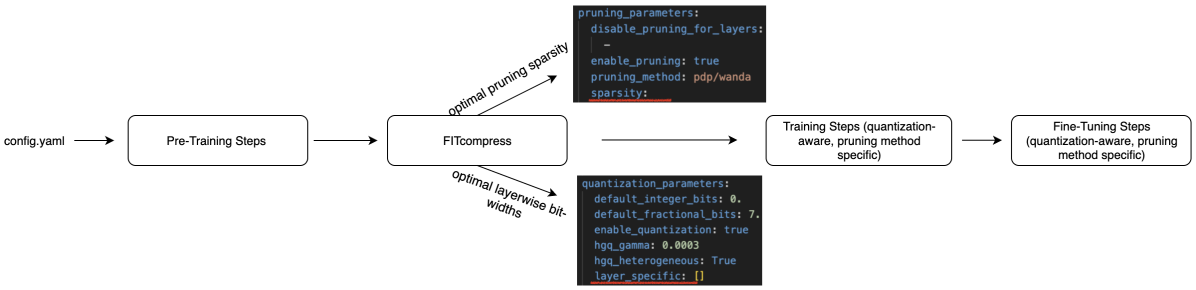


Figure 2.1: FITcompress integrated into the PQuant pipeline. Underlined in red are `sparsity` and `layer_specific`, which are set with the optimal settings found during FITcompress.

given by

$$n_{\text{int}} = \lceil \log_2(\max |w| + \epsilon) \rceil ,$$

where ϵ is a small constant to avoid numerical issues. The remaining bits from the assigned bit-width are then allocated to the fractional part,

$$n_{\text{frac}} = b - n_{\text{int}}.$$

In this way, layer i is guaranteed to have sufficient integer bits to represent its largest parameter, while the unused capacity is dedicated to fractional bits, thereby maximizing precision. Finally, it is important to note that in the original FITCompress code, quantization could be applied repeatedly on already quantized weights (e.g. quantizing a layer first to 7 bits and later directly from this 7-bit representation to 4 bits). Such repeated quantization introduces additional rounding errors and degrades accuracy. To avoid this, the procedure was modified so that each quantization step during FITCompress is always performed directly from the full-precision weights.

2.3 Quantization of Activations, Inputs and Pooling Layers

In the original FITCompress paper, the quantization of activations, input and pooling layers is not explicitly described. It can only be inferred from the reported results that the same number of bits found for a given weight layer were also applied to the corresponding activation unit, and this was repeated for each layer. Since PQuant relies on fixed-point quantization, a more detailed logic for determining activation quantization settings was required.

Our approach is as follows : After FITCompress has determined the optimal quantization settings per weight layer, first these settings are applied to the model and pruning is temporarily disabled. Using a calibration dataset, several forward passes through the quantized model are performed and then the values entering each activation unit (i.e. the outputs of the quantized weight layers) are recorded. For each activation unit, the mean of these inputs is computed and then the maximum absolute value is extracted. Based on these maximum values, the same logic described in Section 2.2 is applied, which distributes the assigned bit-width (i.e. the one of the respective weight layer) between integer and fractional parts. Note that since ReLU activation functions were used in our experiments, the outputs are non-negative, i.e.

$$\text{ReLU}(x) = \max(0, x)$$

Consequently, no sign bit is required for representing activation values, and the bit that would normally be reserved for the sign can instead be allocated to either the integer or the fractional part, thereby increasing the effective precision. In this way, a principled allocation of integer and fractional bits is obtained for the quantized activation units, consistent with the fixed-point representation used for the weights. The bit allocation for quantizing the model inputs and pooling layers follows the same principled approach, based on an analysis of the actual values entering the model and inputs entering the pooling layer, respectively. The bit allocation for quantizing the model inputs and pooling layers follows the same principled approach, based on an analysis of the actual values entering the model and those passed into the pooling layers. Since these modules do not have an associated weight layer, and thus no optimal bit-width determined by FITCompress, their integer and fractional bits are derived from a fixed starting point of 7 bits (with one bit reserved for the sign). Finally, note that in the current implementation of PQuant, BatchNorm layers are not supported for quantization and are therefore left unchanged. This might affect the results, as leaving BatchNorm unquantized can lead to improved performance compared to a fully quantized setting.

2.4 Integration of Pruning Methods

In the original FITCompress algorithm, retraining was performed using global unstructured pruning guided by calculated importance scores and the optimal pruning sparsity (Sec. 1.1.2). In contrast, PQuant supports a variety of different pruning methods, as described in Section 1.2.2, each of which comes with its own rules and constraints. Combining these two approaches is therefore not straightforward.

First, we experiment with pruning directly based on the **importance scores**, following the original FITCompress procedure. In this setting, no external pruning method from PQuant is applied—instead, pruning is driven solely by the importance scores computed within FITCompress itself.

For pruning methods such as PDP and Wanda, which both require a sparsity hyperparameter, this value could be set directly to the optimal pruning sparsity found by `FITCompress`. However, the importance scores from `FITCompress` could not be reused in these methods. In **PDP**, pruning is carried out by computing a global threshold over all weights: the absolute values of all parameters are collected, and a cutoff is chosen such that only the largest $(1 - \text{sparsity})$ fraction remain. Each layer's target sparsity is then initialized according to how many of its weights fall below this global threshold.

In **Wanda**, the ranking of weights is not purely based on magnitude, but on a score that scales weight magnitudes by the norm of the corresponding input activations to each layer. Pruning is performed by directly selecting the top $(1 - \text{sparsity})$ fraction of weights according to this importance metric.

An adaptation was attempted where, instead of ranking by magnitude (PDP) or the Wanda metric, weights were ranked by the **importance scores from** `FITCompress`. However, this failed in practice: a large fraction of importance scores were exactly zero, which caused the effective pruning threshold to collapse to zero. As a result, no weights were pruned at sparsities below 50%.

Other pruning methods supported in PQuant do **not** rely on a global pruning sparsity target and could therefore not make direct use of the optimal sparsity identified by FITCompress. Nevertheless, an optimal quantization configuration could still be determined independently. To accommodate these cases, FITCompress was extended to be able to **optimize either quantization or pruning in isolation**. In this way, FITCompress can flexibly support a broad range of pruning strategies within the PQuant framework.

2.5 Calling FITcompress from the configuration files

FITCompress can be conveniently enabled and configured directly through the configuration file (Fig. 2.1). A set of hyperparameters allows control over which aspects of the algorithm are applied and how the search procedure is carried out. The parameter `enable_fitcompress` specifies whether FITCompress is activated at all. If enabled, one can choose to optimize quantization (`optimize_quantization`) and/or pruning (`optimize_pruning`) during the search. The quantization schedule is defined via `quantization_schedule`, which lists the candidate bit-widths to be tested, while `pruning_schedule` describes a logarithmic curve (base 10) ranging from 0 to 1 over 40 steps. Setting `greedy_astar` disables the fallback mechanism of the A^* algorithm, meaning that once the next best candidate is selected, all remaining candidates are discarded. The parameter `approximate` enables the use of Fisher Traces from previous iterations to approximate FIT scores for quantization settings, as detailed in the section *Computational Details* of the original FITCompress paper [9]. Finally, `lambda` defines a multiplicative factor in the distance function used during search: the total distance between the uncompressed model and the optimal model is given by $g + \lambda f$, where g denotes the distance between the uncompressed model and the current best candidate, and f denotes the distance between the current best and the estimated optimal model. As in the original implementation, $\lambda = 1$ is adopted throughout our experiments.

Parameter	**Value**
enable_fitcompress	true
optimize_quantization	true
quantization_schedule	[7, 4, 3, 2, 1]
pruning_schedule	$\{\text{start} = 0,\ \text{end} = -3,\ \text{steps} = 40\}$
compression_goal	0.08
optimize_pruning	true
greedy_astar	true
approximate	true
lambda	1

Table 2.1: FITcompress hyperparameters as specified in the configuration file.

Experiments

3.1 Experimental Setup

All experiments were conducted on a ResNet-20 architecture [3], initialized from the same pre-trained model. Training used SGD with momentum 0.9 for 200 epochs, a batch size of 128, and an initial learning rate of 0.1 decayed at epochs 82, 123, and 164. Models were trained on CIFAR-10 (50,000 training and 10,000 test images from `torchvision`), with horizontal flips and random crops for augmentation, using an NVIDIA L40S GPU. Performance was measured in terms of classification accuracy. As discussed in Section 2.4, joint pruning–quantization optimization is supported only for PDP and Wanda, so the focus will be on these two methods. Accuracy, remaining weights, and mean bit-width will be analyzed across multiple compression rates. For comparison, results from pruning solely based on importance scores are reported, using the same training and FITcompress hyperparameters as PDP, but with only 200 fine-tuning epochs. Finally, results for quantization-only optimization at a fixed compression goal of 0.10 for pruning methods in PQuant that do not have a global sparsity target are evaluated. In all experiments, `greedy_astar` and `approximate` (see Section 2.5) are enabled, consistent with the original paper. Detailed hyperparameter settings for all pruning methods are given in Appendix B.

Results

4.1 FITcompress with Wanda,PDP and Importance Scores

Using either of the three pruning approaches, the compressed model maintains accuracy close to the baseline up when retaining up to 6% of bytes (Fig. 4.1). Beyond this point, PDP performance declines sharply as it struggles with joint pruning and quantization when the sparsity goal is greater than 50%. In contrast, Wanda exhibits a gradual drop: about 1% accuracy loss at 4% remaining bytes and roughly 6% at 2% remaining bytes. Beyond the 6% threshold, pruning directly with importance scores yields the best performance. Representative bit allocations and per-layer remaining weights for PDP at 6%, Wanda at 4% and importance scores at 2% remaining bytes are shown in Appendix A.

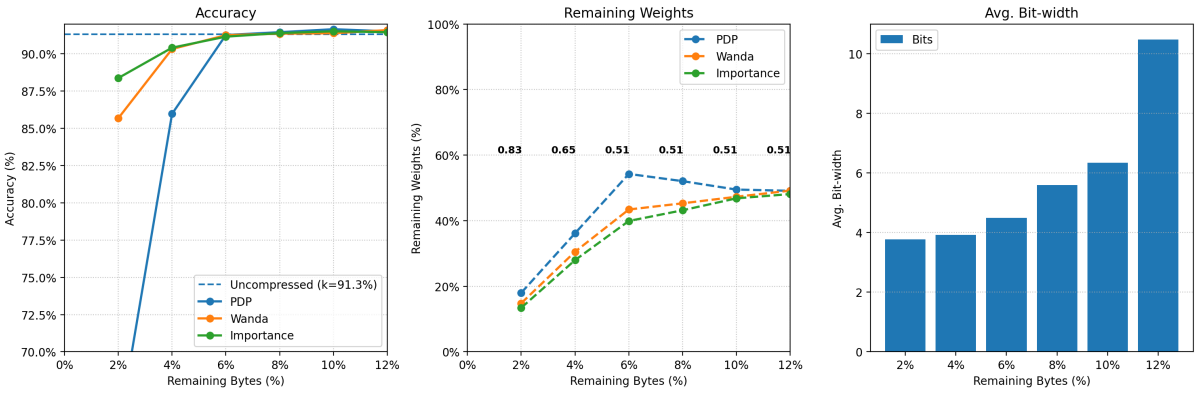


Figure 4.1: FITcompress with Wanda,PDP and importance scores evaluated at different compression strengths based on accuracy, remaining weights and total average bit width. The found pruning sparsity targets from FITcompress are denoted in the plot for Remaining Weights.

4.2 FITcompress with other pruning methods

With CS, accuracy remains close to the baseline even under heavy pruning (down to 15.92% remaining weights). AP, a structured pruning method, suffers a steeper decline to 87.29% accuracy at 39.42% remaining weights, while DST achieves the best accuracy overall but prunes the least amount of weights. Autosparse provides a balanced trade-off, retaining 91.06% accuracy while reducing the model to 52.86% of its weights and MDMM prunes the strongest, but retains the least accuracy (Table 4.1). The bit allocations identified by FITcompress across all methods are provided in Appendix A.

Model	**Accuracy**	**Remaining Weights**	**Mean Bit-Width**
Uncompressed model	91.31%	100%	full precision
FITcompress + CS	90.33%	15.92%	1.65/3.80
FITcompress + DST	91.45%	57.89%	1.65/3.80
FITcompress + AP	87.29%	39.42%	1.65/3.80
FITcompress + Autosparse	91.06%	52.86%	1.65/3.80
FITcompress + MDMM	83.77%	5.5%	1.65/3.80

Table 4.1: Performance comparison when only optimizing for quantization settings during FITcompress. Mean Bit-Width shows mean integer bits/mean fractional bits.

Conclusion

With this work, FITcompress was successfully integrated into the PQuant framework. Coupled with fixed-point quantization, FITcompress proves highly effective for practical hardware deployment, enabling efficient inference on resource-constrained devices without sacrificing accuracy. In combination with PDP and Wanda, the compressed models maintained accuracies within about one percentage point of the uncompressed baseline, even when reduced to only 6% (PDP) and 4% (Wanda) of the original bytes—demonstrating substantial compression with minimal loss. Yet, our experiments also revealed the limits of aggressive compression: Both Wanda and PDP exhibited a gradual decline in accuracy at more extreme compression levels. Interestingly, we find that leveraging the raw importance scores from FITcompress for direct one-shot pruning yields the strongest accuracy retention under aggressive compression. Finally, by extending the integration to additional pruning methods such as CS, DST, AP, AS and MDMM, it was demonstrated that FITcompress can remain flexible even without joint pruning optimization. This adaptability makes it a versatile tool capable of supporting a wide range of sparsification strategies while still delivering robust quantization settings.

5.1 Limitations and Future Work

While our work demonstrates the effectiveness of integrating FITCompress with fixed-point quantization and various pruning methods, several limitations remain that open avenues for future research. First, a deeper integration of the importance scores computed within FITCompress is desirable. In our current approach, these scores are not directly utilized by pruning methods such as PDP or Wanda, and developing strategies to incorporate them more effectively could possibly improve pruning efficiency. In fact, an open question is whether using the importance scores directly may be sufficient for pruning, potentially reducing the need for additional methods such as PDP or Wanda when combined with FITCompress. Another limitation lies in the handling of BatchNorm layers: since they were not quantized in our current implementation, the reported results may be somewhat optimistic, as leaving BatchNorm unquantized can artificially improve performance compared to a fully quantized setting. Finally, our experiments have so far been limited to relatively small-scale models, such as ResNet-20, evaluated on the comparatively simple CIFAR-10 dataset. Applying FITCompress in combination with fixed-point quantization to larger and more complex architectures would be an important next step in assessing the scalability and robustness of the method.

Appendix A

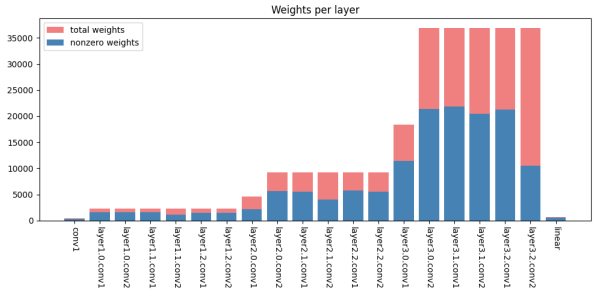


Figure A.1: Remaining weights (54.26%) after FITcompress at a compression goal of 0.06, using PDP. FITcompress found a sparsity goal of 0.51%.

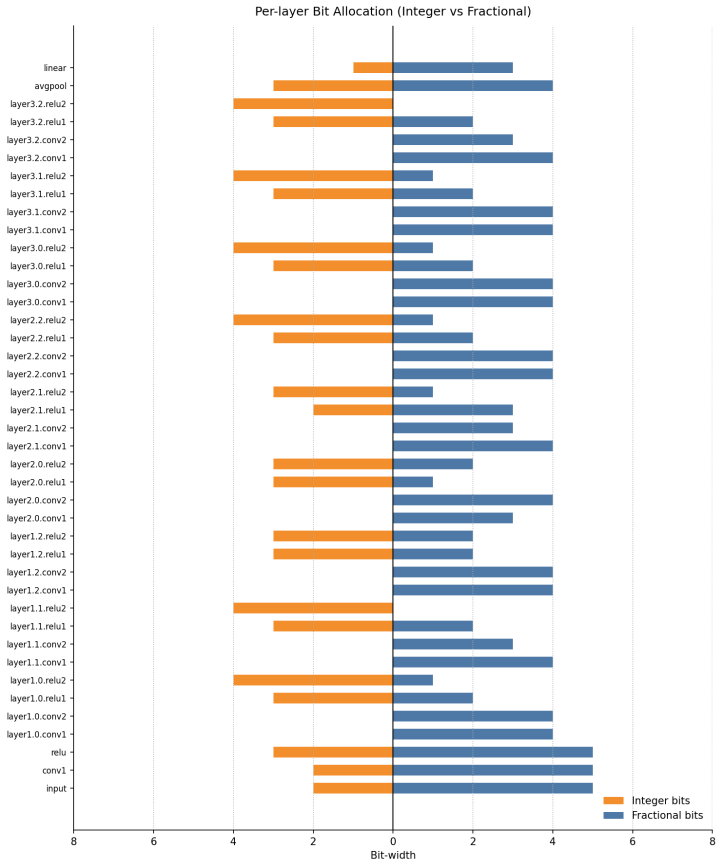


Figure A.2: Bit allocation after FITcompress at a compression goal of 0.06, using PDP.

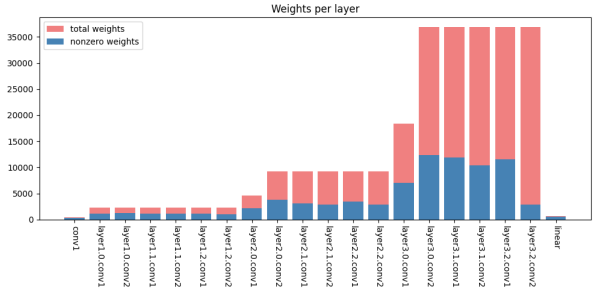


Figure A.3: Remaining weights (30.52%) after FITcompress at a compression goal of 0.04, using Wanda. FITcompress found a sparsity goal of 65%.

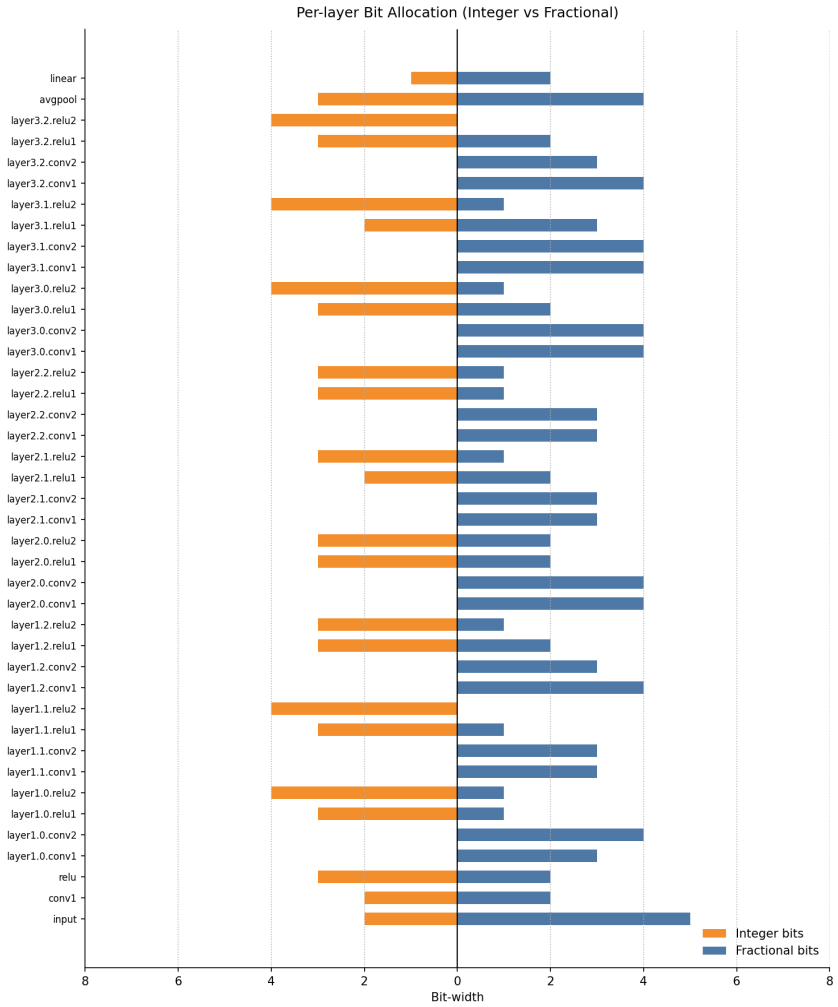


Figure A.4: Bit allocation after FITcompress at a compression goal of 0.04, using Wanda.

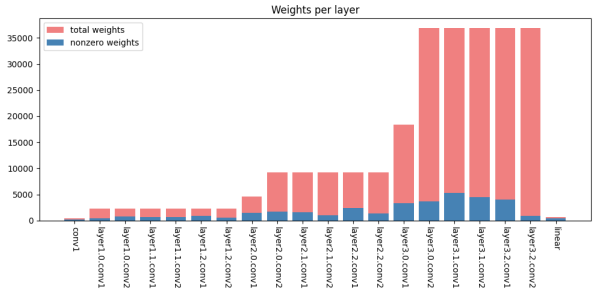


Figure A.5: Remaining weights (13.42%) after FITcompress at a compression goal of 0.02, using Importance Scores. FITcompress found a sparsity goal of 83%.

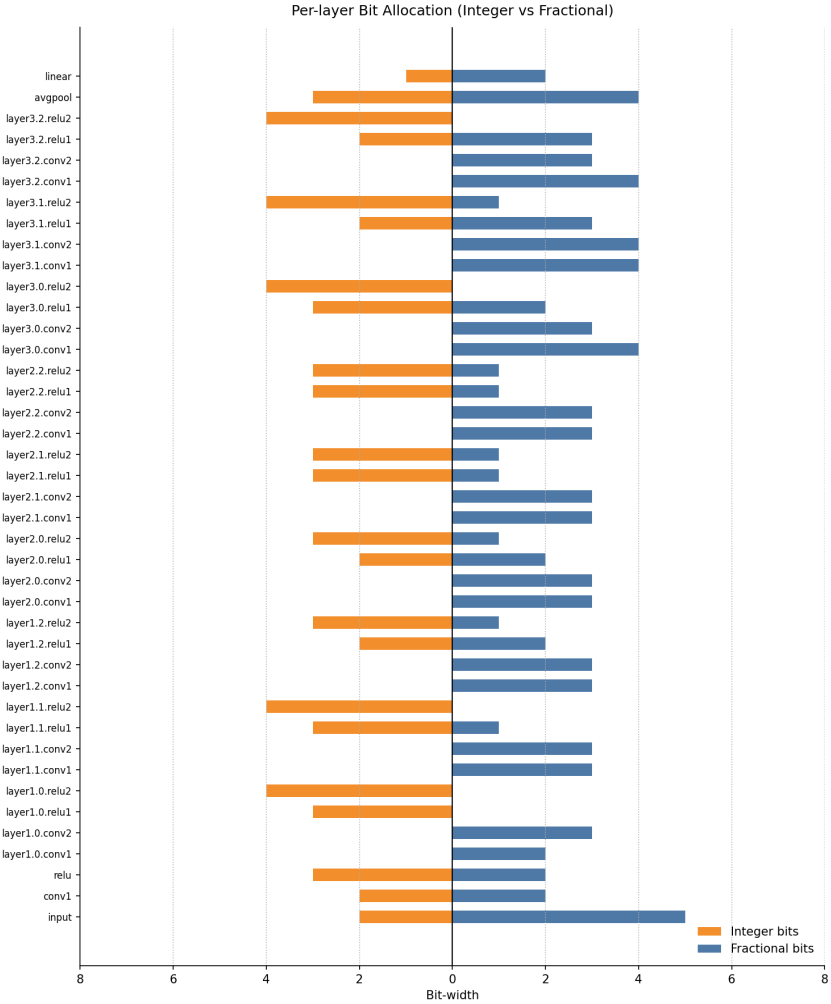


Figure A.6: Bit allocation after FITcompress at a compression goal of 0.02, using Importance Scores.

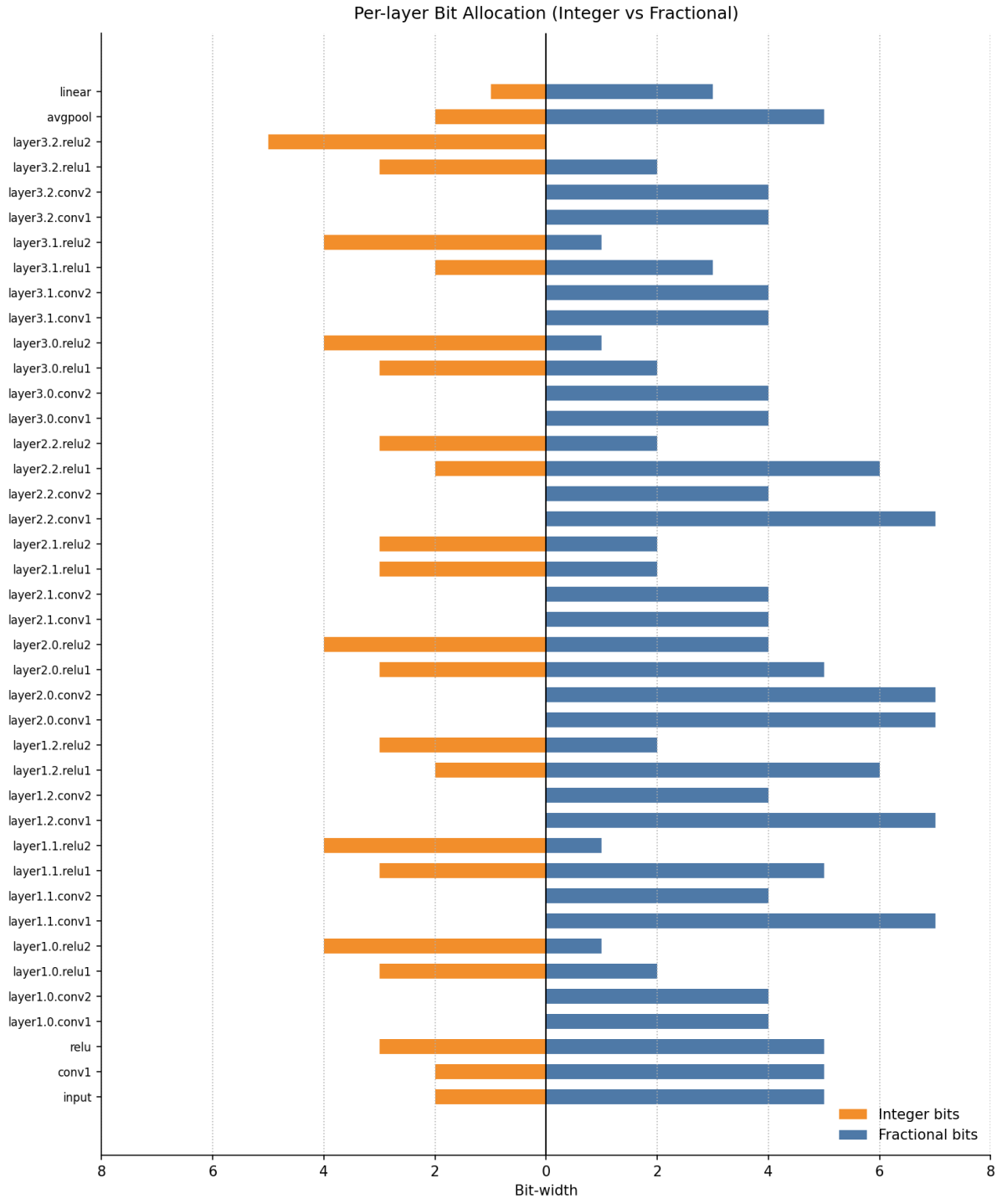


Figure A.7: Bit allocation after FITcompress at a compression goal of 0.10, using CS, DST, AP and Autosparse.

Appendix B

Hyperparameter	PDP	Wanda	CS	DST	AP	AS	MDMM
Training-specific							
Optimizer	SGD	SGD	SGD	SGD	SGD	SGD	SGD
Momentum	0.9	0.9	0.9	0.9	0.9	0.875	0.9
Scheduler	Cosine	Cosine	Multistep	Multistep	Cosine	Cosine	Multistep
Pre-Training Epochs	0	0	0	0	0	0	0
Training Epochs	100	200	85 (3r)	160	100	100	200
Fine-Tuning Epochs	100	0	85	0	0	0	30
Batch Size	64	64	64	64	64	64	64
Learning Rate	10^{-2}	10^{-2}	10^{-1}	10^{-2}	10^{-2}	10^{-2}	10^{-2}
Weight Decay	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	3.05×10^{-5}	10^{-4}
FITcompress-specific							
Pruning Schedule	$(1 - 10^{-n/13})$	$(1 - 10^{-n/13})$	/	/	/	/	/
Quantization Schedule	[7,4,3,2,1]	[7,4,3,2,1]	[7,4,3,2,1]	[7,4,3,2,1]	[7,4,3,2,1]	[7,4,3,2,1]	[7,4,3,2,1]
Compression Goals	[0.12,0.10,...,0.02]	[0.12,0.10,...,0.02]	0.10	0.10	0.10	0.10	0.10

Table B.1: Training- and FITcompress-specific hyperparameters across pruning methods. For the FITcompress-specific pruning scheduler, we have $n = 1, \ldots, 40$.

Hyperparameter	PDP	Wanda	CS	DST	AP	AS	MDMM
Epsilon	0.015	–	–	–	–	–	1×10^{-3}
Threshold	–	–	0	0	0.15	–	–
Temperature	1×10^{-5}	–	200 (final_temp)	–	–	–	–
Structured / N:M	False	False	–	–	–	–	False
t_delta	–	100	–	–	50	–	–
t_start_collecting	–	100	–	–	50	–	–
Threshold decay	–	–	1×10^{-9}	0	0	0	–
Threshold init	–	–	0	0	–	-5	–
Threshold type	–	–	–	channelwise	–	channelwise	–
alpha	–	–	–	–	–	0.5	–
alpha_reset_epochs	–	–	–	–	–	90	–
autotune_epochs	–	–	–	–	–	10	–
backward_sparsity	–	–	–	–	–	False	–
constraint_type	–	–	–	–	–	–	Equality
target_value	–	–	–	–	–	–	0.0
metric_type	–	–	–	–	–	–	UnstructSparse
target_sparsity	–	–	–	–	–	–	0.9
rf	–	–	–	–	–	–	1
scale	–	–	–	–	–	–	10.0
damping	–	–	–	–	–	–	1.0
use_grad	–	–	–	–	–	–	False
l0_mode	–	–	–	–	–	–	coarse

Table B.2: Pruning-specific hyperparameters for each method.

Bibliography

[1] Minsik Cho, Saurabh Adya, and Devang Naik. Pdp: Parameter-free differentiable pruning is all you need. *arXiv preprint arXiv:2305.11203*, 2023.

[2] Siavash Golkar, Michael Kagan, and Kyunghyun Cho. Continual learning via neural pruning. *arXiv preprint arXiv:1903.04476*, 2019.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[4] Abhisek Kundu, Naveen K. Mellempudi, Dharma Teja Vooturi, Bharat Kaul, and Pradeep Dubey. Auto$_{\text{sparse}}$: Towards automated sparse training of deep neural networks. *arXiv preprint arXiv:2304.06941*, 2023.

[5] Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden K. H. So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. *arXiv preprint arXiv:2005.06870*, 2020.

[6] John C. Platt and Alan H. Barr. Constrained differential optimization. In *Neural Information Processing Systems (NIPS) 1987*, 1987.

[7] Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. *arXiv preprint arXiv:1912.04427*, 2019.

[8] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[9] Ben Zandonati, Glenn Bucagu, Adrian Alan Pol, Maurizio Pierini, Olya Sirkin, and Tal Kopetz. Towards optimal compression: Joint pruning and quantization. *arXiv preprint arXiv:2302.07612*, 2023.

[10] Ben Zandonati, Adrian Alan Pol, Maurizio Pierini, Olya Sirkin, and Tal Kopetz. Fit: A metric for model sensitivity. *arXiv preprint arXiv:2210.08502*, 2022.