# PQ Bench: Benchmarking Pruning and Quantization Techniques

Jonas Schulze
jonas.schulze@hpi.de
Hasso Plattner Institute
University of Potsdam, Germany

Nils Strassenburg
nils.strassenburg@hpi.de
Hasso Plattner Institute
University of Potsdam, Germany

Tilmann Rabl
tilmann.rabl@hpi.de
Hasso Plattner Institute
University of Potsdam, Germany

## Abstract

Deep learning models are increasingly deployed in performance-critical applications, where computational efficiency must be balanced with accuracy. Model compression techniques such as pruning and quantization help address this challenge. However, they are often evaluated solely on accuracy, overlooking their impact on model size and inference time for a given hardware setup.

To support practitioners in selecting and evaluating different model compression techniques, we introduce our benchmarking framework, PQ Bench. It pre-implements a set of popular compression techniques and automates the process of benchmarking their effects on accuracy, inference speed, and memory footprint. In our evaluation, we demonstrate the use of PQ Bench and provide key insights into the trade-offs across various models, compression strategies, and configurations.

**ACM Reference Format:**
Jonas Schulze, Nils Strassenburg, and Tilmann Rabl. 2025. PQ Bench: Benchmarking Pruning and Quantization Techniques. In *Workshop on Data Management for End-to-End Machine Learning (DEEM '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3735654.3735940

## 1 Introduction

Deep learning (DL) models are getting increasingly popular and are tightly integrated in database management systems (DBMSs). Popular examples include (1) optimizing query processing (learned indices [2, 21], learned query optimizers [26, 28], learned join ordering [27], or models for cost prediction [11]), (2) answering queries that include the evaluation of unstructured data (e.g., having filter predicates on images or text [19, 52]), (3) or providing a natural language interface to DBMSs by translating natural text to SQL [50].

Despite the growing popularity of DL in DBMSs, their high computational complexity impacts query processing performance, which leads to a trade-off between accuracy and performance. On the one hand, a complex model typically makes more accurate predictions than a simpler model. On the other hand, a complex model is slower and increases the query processing time. Consequently, the key challenge lies in identifying models that benefit the DBMS by making accurate predictions while minimizing its impact on query processing time.

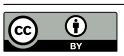
Model compression techniques, such as pruning and quantization, aim to address this challenge by reducing model size and complexity as much as possible while minimizing the loss of accuracy. Because these techniques are primarily developed within the machine learning (ML) community, associated publications tend to focus on improving model accuracy and only discuss size reduction and inference speed up from a theoretical viewpoint. Even the PyTorch-based benchmarking framework ShrinkBench [1] evaluates pruning techniques with a predefined set of models and datasets but does not include quantization techniques or measured inference times. This is a problem for engineers: while they can assess models' prediction quality, understanding how size reductions impact system performance in terms of memory footprint, single-item and batched inference speed remains challenging. To address this gap, we make the following contributions:

(1) We present an overview of pruning and quantization techniques and identify a subset of popular methods by examining their availability as options provided in widely used DL frameworks.
(2) We develop the benchmarking framework PQ Bench, which automatically applies various compression techniques to a given model and dataset, and benchmarks their impact on accuracy and other performance-critical metrics to provide users with a rough estimate of expected performance.
(3) Using PQ Bench, we evaluate pruning and quantization methods on common DL models and offer insights into the performance of pruning and quantization techniques.

## 2 Background

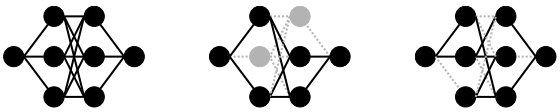
**Pruning**. Pruning reduces a model's size and computational complexity by removing single parameters or entire neurons from the model architecture while minimizing accuracy drops. Han et al. categorize pruning techniques among the dimensions of: granularity, criterion, and ratio [44]. The pruning *granularity* determines the pattern used to prune the model. We differentiate between structured and unstructured pruning. *Structured pruning* prunes groups of weights that structurally belong together, such as all parameters associated with a neuron, entire filters, or channels (Figure 1b). This preserves the overall structural integrity of the model but alters the layer's input and output shapes. In contrast, *unstructured pruning* removes individual parameters from the model, without a specific pattern (Figure 1c). The resulting model loses its structural integrity [15, 29]. The pruning *criterion* determines what parameters or neurons to prune. Two common pruning methods are *random* and *L1* pruning [38]. Random pruning selects neurons or weights at random [39], while L1 pruning removes the least important parameters based on their L1 norm. *Regular L1* pruning calculates L1 scores per layer, whereas *global L1* pruning considers all model parameters [7, 13]. The pruning *ratio* defines the relative number of parameters or neurons to prune.

**Quantization**. DL frameworks typically use 32-bit floating point numbers (*FP32*). Quantization reduces model size and complexity




*DEEM '25, Berlin, Germany*

ACM ISBN 979-8-4007-1924-0/2025/06
https://doi.org/10.1145/3735654.3735940



(a) base model (b) structured (c) unstructured

**Figure 1: Model pruning techniques**

by converting parameters to lower-precision types like 16-bit floats (*FP16*), 8-bit integers (*INT8*), or single bits [8, 51]. We can quantize a model during or after the training. Quantization aware training integrates quantization into training by simulating it in the forward pass and storing quantized values. Post training quantization uses a calibration dataset to map FP32 activations to lower precisison while trying to minimizing information loss [40].

**TensorRT & ONNX**. TensorRT is a machine learning ecosystem by NVIDIA used to optimize models for low-latency and high-throughput inference on NVIDIA GPUs [33]. TensorRT benchmarks layers and computes an optimal execution schedule, minimizing the combined cost of kernel executions and format transforms. Additionally, TensorRT eliminates dead computations, folds constants, reorders or combines operations, and supports post training quantization to FP16 and INT8 [35, 36]. TensorRT is often used in combination with the open neural network exchange (ONNX) format built to allow the representation, exchange, and conversion of models across different ML platforms [49].

## 3 Approach

In this section, we give an overview of our model compression benchmarking framework PQ Bench[1], describe the implemented compression techniques, and explain how to extend it.

**Overview**. We provide a configuration file specifying: (1) the model to compress, (2) the dataset to evaluate the model on, (3) the compression technique(s) together with their parametrization, (4) and benchmarking hyperparameters such as the batch size and the number of iterations. PQ Bench takes the model, applies the compression techniques, benchmarks the model using the provided dataset, and saves the benchmarking results to a CSV file. The results include: the time it took to compress the model, the compressed model size, the model accuracy, and the batch inference speed for all specified batch sizes. To measure the compressed model size, we calculate the model object size for PyTorch models or the file size for TensorRT engines. To measure the accuracy, we use the average accuracy across each batch and iteration of the test set. To measure the compression time, we use Python's time package and encapsulate the responsible code fragments. To benchmark GPU inference speed (excluding data loading), we use Torch Profiler with a ten-iteration warm-up.

**Pruning**. We use PyTorch's pruning module [41], which replaces pruned weights with zeroes (*zero-fill pruning*), leaving the model architecture unchanged to implement unstructured pruning. This covers the configurations of local and global pruning, in combination with different L*n* norms such as L1 and L2 norms, and a dimension parameter for structured pruning. With a pruning dimension of zero, we prune neurons and channels, while with a pruning dimension of one, we disconnect neurons from their inputs. To implement structured pruning, we use the TorchPruner [25] library and offer random pruning, weight norm pruning [23], sensitivity pruning [30], and Taylor pruning [31]. By this, we cover the most frequently used techniques, representing a large subset of the techniques described in Section 2. For further techniques, PQ Bench allows easy integration of custom model pruning techniques without any changes to its automated benchmarking.

**Quantization**. To quantize a model, we use ONNX [49] and NVIDIA's TensorrtExecutionProvider (TEP) in a two-step process [20, 33, 37]. First, we convert the PyTorch model to the ONNX format, which includes calibrating the quantization parameters using a small subset of the dataset, optimizing execution subgraphs [54], and returning a quantized model together with a calibration file. Second, we use a full precision ONNX model, the calibration settings, and the quantization data type, such as FP16 or INT8, to specify a TensorRT TEP that applies the quantization logic and is used to execute the quantized model. Currently, PQ Bench offers post-training FP16 and INT8 quantization. To support other quantization techniques, we provide examples in our repository on how to implement a custom pruning object.

## 4 Evaluation

In this section, we use PQ Bench to evaluate the impact of pruning and quantization on pre-trained convolutional and transformer models using the ImageNette [5] and ImageWoof [6] datasets, each containing ten classes from ImageNet [45]. We focus on CNNs due to the availability of pre-trained models and on widely used post-training compression techniques with existing implementations. We leave more advanced approaches, such as compression-aware training [53] or post-pruning fine-tuning, for future work.

The CNN models include AlexNet [22], MobileNet [43], ResNet [10], DenseNet [12], GoogLeNet [47], Inception [48], SqueezeNet [14], ShuffleNet [24] and SimpleNet [9]. As a vision transformer, we include VIT_B_16 [4]. We run our experiments on an NVIDIA V100 GPU with 32GB memory and have Python 3.11.5, PyTorch 2.4.1, TensorRT 10.4.0, CUDA 12.2, and ONNX 1.16.2 with ONNX Runtime GPU support 1.19.2 installed. As metrics, we consider accuracy (Section 4.1), inference speed (Section 4.2), as well as model size (Section 4.3), and report the median of three runs not showing confidence intervals due to low variance. We focus on post-training FP16 and INT8 quantization [17], on magnitude-based pruning, and mainly discuss the results for the unoptimized AlexNet architecture and the highly optimized MobileNet architecture, as they represent the extreme points in our measurements. For results on other architectures, we refer to our technical report [18].

### 4.1 Accuracy

**Zero-fill Pruning**. Comparing random and L1 norm pruning, we find that L1 norm consistently outperforms random pruning, as it uses a principled weight-magnitude criterion rather than arbitrary selection. Thus, we focus on local and global L1 norm pruning with pruning degrees between zero and 90% in Figure 2. AlexNet is less affected by pruning than MobileNet, and global pruning outperforms local pruning for both models and datasets. With local pruning, AlexNet drops in accuracy after 50%, while MobileNet declines sharply at 10%. Global pruning minimizes accuracy loss,

[1] GitHub repo: https://github.com/hpides/pq-bench

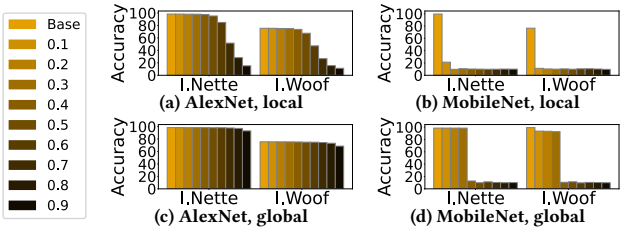



**Figure 2: Impact of unstructured L1 pruning on accuracy.**

allowing AlexNet to maintain accuracy up to 80% pruning and MobileNet up to 30%. AlexNet is less affected by pruning due to its redundancy, while MobileNet and similarly optimized models like Inception are more sensitive [22, 43, 46]. Global pruning outperforms local pruning because not all layers of a neural network are equally large and important. While local pruning prunes the same percentage of parameters from every layer (regardless of importance), global pruning primarily prunes layers resistant to pruning while keeping critical layers untouched.

**Structured Pruning**. Figure 3 shows that structured L1 norm pruning causes significant accuracy drops even for low pruning degrees, regardless whether we prune on the channels axis (*Dim*=0), the neurons axis (*Dim*=1), or use a L2 instead of L1 norm. In structured pruning, vision models are typically pruned at the filter level. While many parameters within a filter can be removed without significantly impacting accuracy, some are performance-critical. Thus, removing entire filters leads to poor accuracy compared to, for example, global unstructured pruning.

**Overall View**. Figure 4 shows accuracy numbers for a pruning degree of 0.1 and 0.5 for local and global L1 unstructured pruning respectively, as it highlights differences between model architectures well. Models designed for edge devices, i.e., ShuffleNet and MobileNet, are unreceptive to pruning. Inception seems to be an outlier, given its performance being the poorest in the local setting.

**Quantization**. Figure 4d shows that, except for SimpleNet (which has issues with dynamic tensor indexing when exporting the model to ONNX), INT8 quantization causes no significant drop in accuracy. We find the same for FP16 which is in line with related work [32, 42].

## 4.2 Inference Speed

**Zero-fill Pruning**. Analyzing the inference speed of models pruned using zero-fill pruning, we observe negligible speed improvements compared to a non-pruned model, even at high pruning degrees. The reason is that zero-fill pruning does not remove parameters, but instead sets them to zero. While modern GPUs support specific sparsity patterns for efficient execution, these do not match the random patterns produced by the applied pruning techniques. E.g. NVIDIA's TensorRT promises support for sparse matrix inference on selected GPUs, but they demand significant preprocessing steps and only work on exactly fifty percent sparse models [16]. As a result, the computational workload and complexity remain largely unchanged, leading to no speed-up.

**Structured Pruning**. With structured pruning, we examined a steady increase of inference speed, up to 2x when comparing the base model to a 90 percent pruned model, which we expect, as we reduce the number of parameters involved in computations. However, faster computations do not justify the poor performance of structurally pruned models as seen in Figure 3.

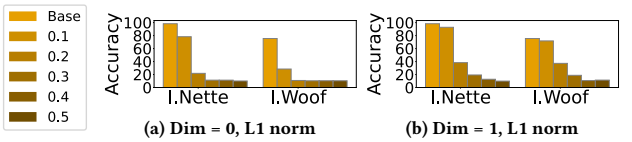



**Figure 3: AlexNet pruned across structural axes (Dim).**

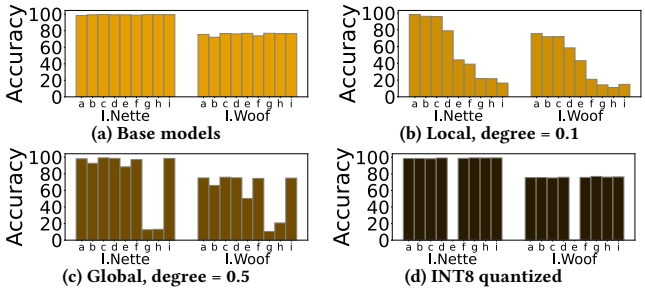



**Figure 4: Compressed models. a) Alex, b) Squeeze, c) Dense, d) Res, e) Simple, f) GoogLe, g) Shuffle, h) Mobile, i) Inception.**

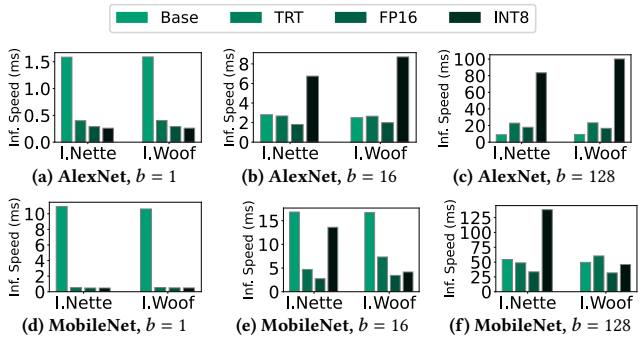
**Quantization**. We compare the inference speed of AlexNet and MobileNet models implemented in PyTorch (Base), with optimization techniques including TensorRT optimizations without quantization (TRT), FP16, and INT8 quantization across different batch sizes. Without considering model or hardware characteristics, we expect to see a reduction in inference time compared to the base model when applying TRT because of execution graph optimizations and further improvements with every reduction of parameter precision due to reduced model size and computation complexity.

In Figure 5, for a batch size of one, inference time drastically decreases when using TensorRT and gets even shorter when applying quantization. For a batch size of 16 and AlexNet, we see marginal to no improvements between Base and TRT, noticeable improvements when using FP16 quantization, and significantly higher inference times for INT8 quantization. For a batch size of 16 and MobileNet, we see large improvements when applying TRT, additional improvements with FP16 quantization, and a marginal to significant increase in inference time when quantizing to INT8. For a batch size of 128, we notice an increase in inference time for AlexNet regardless of the applied optimization. For MobileNet, quantizing to FP16 reduces the inference time, while all other optimizations bring no improvements. These patterns are similar across all our evaluated models.

One of TensorRT's main optimizations is to fuse layers. This improves performance most when the GPU is not fully utilized, for example, because of small models or batch sizes. For small batch sizes, this fusion can be highly effective since the number of operations per input is lower. Thus, TensorRT improves the inference time most for MobileNet and a batch size of one, and fails to optimize the inference time for AlexNet in combination with a batch size of 128. The reason for the poor performance of INT8 quantization are missing INT8 tensor cores, resulting in a fallback to FP16 computations and an overhead for data type conversions [34].

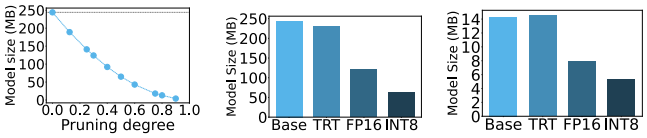
## 4.3 Model Size

**Pruning**. In zero-filled pruning, pruned values are replaced with zero values, leading to no size reduction when using a dense matrix format. Both coordinate (COO) and compressed sparse column (CSC) formats linearly decrease model size with increasing pruning




(a) AlexNet, $b = 1$ (b) AlexNet, $b = 16$ (c) AlexNet, $b = 128$
(d) MobileNet, $b = 1$ (e) MobileNet, $b = 16$ (f) MobileNet, $b = 128$

**Figure 5: Comparing quantization impact on inference speed with AlexNet and MobileNet at different batch sizes ($b$).**




(a) Struc. Pruned (b) AlexNet (c) MobileNet

**Figure 6: Model size after optimization and compression.**

degrees. However, they only reduce the model size compared to a dense matrix representation at 80% and 70% sparsity, respectively, due to the overhead introduced by index values. As shown in Figure 6a, structured pruning decreases the model size logarithmically. We assume the reason to be TorchPruner's cascading side-pruning operations which prune models beyond the given pruning degree to guarantee the compatibilities of inputs and outputs between layers.

**Quantization**. In Figure 6, we compare the sizes of unquantized AlexNet and MobileNet models with their optimized, quantized TensorRT engine versions. For both architectures, the non-quantized TensorRT engine remains similar in size to the unoptimized PyTorch model, as it stores an execution plan and full parameter representation, similar to PyTorch's format. Quantizing to FP16 and INT8 reduces size by factors of two and four, respectively, as FP16 uses half and INT8 a quarter of the bits of FP32. Surprisingly, we see this effect only for AlexNet and MobileNet, while all other models share only the size reduction when being quantized to FP16, but no further reduction for INT8.

## 4.4 Combining Pruning and Quantization

**Convolutional Architectures**. In this section, we investigate the effect of combining pruning and quantization. We first perform global unstructured pruning with a pruning degree of 90% for AlexNet and 30% for MobileNet on ImageNette with a batch size of eight. Afterward, we optimize the model using TensorRT and quantize it to FP16 and INT8.

Overall, the combination only marginally reduces the model accuracy (less than 6 percent when comparing a pruned and quantized model to the base model for both AlexNet and MobileNet). Figure 7 compares the inference speed of quantized models to those of pruned and quantized models. For AlexNet, we see an average improvement of 10–15% when combining pruning and quantization. For MobileNet, here with a pruning degree of 0.3, we see an increase in inference time. We assume the reason to be TensorRT's ability to exploit very sparse weight matrices.


(a) AlexNet, 90% pruned (b) MobileNet, 30% pruned

**Figure 7: Inference speed of global pruning and quantization.**

**Transformer Architectures**. We also analyze the impact of pruning and quantization on a VIT_B_16 vision transformer [3]. When pruning globally and unstructured, we can remove roughly 40% of the parameters without a significant loss in accuracy. For quantization, the accuracy does not degrade irrespective of using FP16 or INT8. Regarding inference speed, only quantization has an effect on inference time (from 8 ms for a non-quantized TensorRT model to 4 ms) and halves the model size from 346 MB to 170 MB irrespective of FP16 or INT8 quantization. We assume this is because even high-end NVIDIA GPUs do not fully support integer operations on any given architecture type.

## 4.5 Discussion

In summary, we find that quantization outperforms pruning in all aspects of model size reduction, inference speed up (for small batch sizes), and accuracy preservation but comes with a high overhead of external dependencies and depends on the availability of an NVIDIA GPU with matching tensor cores. Pruning shows the best results when we apply unstructured rather than structured pruning. Combining pruning and quantization can lead to further improvements, especially when GPUs support the architectural structures of the compressed model. We observe inconsistent INT8 quantization behavior in both inference speed and model size, even on an NVIDIA A100. We suspect this is due to unsupported layer types and fallback to FP16, leading to costly data type conversions. Investigating these issues is a key direction for future work.

We advise engineers to use PQ Bench and apply quantization for best results if they have a high-end GPU at their disposal and are willing to set up TensorRT and its dependencies. If they seek for an easy and quick application or do not have sufficient hardware, we recommend global unstructured pruning.

## 5 Conclusion

In this paper, we provide an overview of popular pruning and quantization techniques. We implement these model compression methods in PQ Bench, a framework that, given a model and dataset, enables engineers to efficiently benchmark various configurations and assess improvements in model accuracy, size, and inference speed. Through our evaluation using PQ Bench and popular DL models, we find that, when properly configured, quantization significantly (and pruning to a moderate extent) reduces the model size and inference time while only marginally impacting accuracy. Future work includes, among other things, testing on a wider range of model architectures, exploring quantization on mid-range GPUs and below INT8, and developing a recommendation system for selecting compression techniques based on a given model.

## Acknowledgments

This work was partially funded by the German Research Foundation (ref. 414984028 and ref. 556566056) and the European Union's Horizon 2020 research and innovation programme (ref. 957407).

## References


[1] Blalock, Davis and Gonzalez Ortiz, Jose Javier and Frankle, Jonathan and Guttag, John. 2020. *Shrinkbench*. https://github.com/JJGO/shrinkbench Accessed: -2024-11-27.

[2] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 969–984. doi:10.1145/3318464.3389711

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR* abs/2010.11929 (2020). arXiv:2010.11929 https://arxiv.org/abs/2010.11929

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*. https://openreview.net/forum?id=YicbFdNTTy

[5] fast.ai. 2019. ImageNette. https://github.com/fastai/imagenette. Accessed: 2025-02-05.

[6] fast.ai. 2019. ImageWoof. https://github.com/fastai/imagenette?tab=readme-ov-file#imagewoof. Accessed: 2025-02-05.

[7] Eirik Fladmark, Muhammad Hamza Sajjad, and Laura Brinkholm Justesen. 2023. Exploring the Performance of Pruning Methods in Neural Networks: An Empirical Study of the Lottery Ticket Hypothesis. arXiv:2303.15479 [cs.LG] https://arxiv.org/abs/2303.15479

[8] RaviKiran Gopalan and Oliver M. Collins. 2009. An Optimization Approach to Single-Bit Quantization. *IEEE Transactions on Circuits and Systems I: Regular Papers* 56, 12 (2009), 2655–2668. doi:10.1109/TCSI.2009.2019392

[9] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. 2023. Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures. arXiv:1608.06037 [cs.CV] https://arxiv.org/abs/1608.06037

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV] https://arxiv.org/abs/1512.03385

[11] Benjamin Hilprecht and Carsten Binnig. 2022. Zero-shot cost models for out-of-the-box learned cost prediction. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2361–2374.

[12] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2018. Densely Connected Convolutional Networks. arXiv:1608.06993 [cs.CV] https://arxiv.org/abs/1608.06993

[13] Zhongzhan Huang, Wenqi Shao, Xinjiang Wang, Liang Lin, and Ping Luo. 2021. Rethinking the pruning criteria for convolutional neural network. *Advances in Neural Information Processing Systems* 34 (2021), 16305–16318.

[14] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv:1602.07360 [cs.CV] https://arxiv.org/abs/1602.07360

[15] Java Point. 2023. Pruning in Machine Learning. https://www.javatpoint.com/pruning-in-machine-learning. Accessed: 2024-10-18.

[16] Jeff Pool, Abhishek Sawarkar, Jay Rodge. 2021. Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT. https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/. Accessed: 2024-11-25.

[17] Joel Nicholls. 2018. Quantization in Deep Learning. https://medium.com/@joel_34050/quantization-in-deep-learning-478417eab72b. Accessed: 2024-10-18.

[18] Jonas Schulze. 2024. Master Thesis - Benchmarking of Pruning and Quantization Techniques. https://github.com/sjoze/master-thesis/blob/main/master_thesis_final_Jonas_Schulze.pdf. Accessed: 2025-01-31.

[19] Daniel Kang, John Guibas, Peter D. Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2022. TASTI: Semantic Indexes for Machine Learning-based Queries over Unstructured Data. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1934–1947. doi:10.1145/3514221.3517897

[20] Katsuya Hyodo. 2024. TensorRT Excecution Provide. https://zenn.dev/pinto0309/scraps/42587e1074fc53. Accessed: 2024-11-27.

[21] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 489–504. doi:10.1145/3183713.3196909

[22] Alex Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. arXiv:1404.5997 [cs.NE] https://arxiv.org/abs/1404.5997

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning Filters for Efficient ConvNets. arXiv:1608.08710 [cs.CV] https://arxiv.org/abs/1608.08710

[24] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. arXiv:1807.11164 [cs.CV] https://arxiv.org/abs/1807.11164

[25] Marco Ancona. 2020. TorchPruner. https://github.com/marcoancona/TorchPruner/tree/master. Accessed: 2024-11-12.

[26] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul23. [n. d.]. Neo: A Learned Query Optimizer. *Proceedings of the VLDB Endowment* 12, 11 ([n. d.]).

[27] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management* (Houston, TX, USA) *(aiDM'18)*. Association for Computing Machinery, New York, NY, USA, Article 3, 4 pages. doi:10.1145/3211954.3211957

[28] Ryan Marcus and Olga Papaemmanouil. 2019. Towards a Hands-Free Query Optimizer through Deep Learning. In *Proceedings of the 9th Biennial Conference on Innovative Data Systems Research (CIDR '19)*.

[29] Mark Kurtz. 2020. What is Pruning in Machine Learning? https://opendatascience.com/what-is-pruning-in-machine-learning/. Accessed: 2024-10-18.

[30] Deepak Mittal, Shweta Bhardwaj, Mitesh M. Khapra, and Balaraman Ravindran. 2018. Studying the Plasticity in Deep Convolutional Neural Networks using Random Pruning. arXiv:1812.10240 [cs.LG] https://arxiv.org/abs/1812.10240

[31] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning Convolutional Neural Networks for Resource Efficient Inference. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJGCiw5gl

[32] Prateeth Nayak, David Zhang, and Sek Chai. 2019. Bit efficient quantization for deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 52–56.

[33] NVIDIA. 2016. TensorRT. https://developer.nvidia.com/tensorrt. Accessed: 2024-11-17.

[34] NVIDIA. 2017. NVIDIA TESLA V100 GPU ARCHITECTURE. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf. Accessed: 2025-01-28.

[35] NVIDIA. 2020. NVIDIA TensorRT Documentation. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html. Accessed: 2024-11-25.

[36] NVIDIA. 2020. NVIDIA TensorRT's capabilities. https://docs.nvidia.com/deeplearning/tensorrt/10.8.0/architecture/capabilities.html. Accessed: 2025-02-02.

[37] ONNX. 2021. ONNX Execution Provider. https://onnxruntime.ai/docs/execution-providers/TensorRT-ExecutionProvider.html. Accessed: 2024-11-24.

[38] PyTorch. 2020. L1 Unstructured, PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.utils.prune.l1_unstructured.html. Accessed: 2025-02-02.

[39] PyTorch. 2020. Random Unstructured, PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.utils.prune.random_unstructured.html. Accessed: 2025-02-02.

[40] PyTorch. 2022. Post Training Quantization (PTQ). https://pytorch.org/TensorRT/tutorials/ptq.html. Accessed: 2025-02-02.

[41] PyTorch. 2023. Torch Prune. https://pytorch.org/docs/stable/nn.html. Accessed: 2024-11-19.

[42] Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. 2022. A comprehensive survey on model quantization for deep neural networks. *arXiv preprint arXiv:2205.07877* (2022).

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381 [cs.CV] https://arxiv.org/abs/1801.04381

[44] Song Han. 2024. TinyML and Efficient Deep Learning Computing Lecture (MIT). https://hanlab.mit.edu/courses/2024-fall-65940. Accessed: 2025-01-31.

[45] Stanford Vision Lab, Stanford University. 2007. ImageNet. https://www.image-net.org/. Accessed: 2025-02-05.

[46] Jiang Su, Julian Faraone, Junyi Liu, Yiren Zhao, David B Thomas, Philip HW Leong, and Peter YK Cheung. 2018. Redundancy-reduced mobilenet acceleration on reconfigurable logic for imagenet classification. In *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 14th International Symposium, ARC 2018, Santorini, Greece, May 2-4, 2018, Proceedings 14*. Springer, 16–28.

[47] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. arXiv:1409.4842 [cs.CV] https://arxiv.org/abs/1409.4842

[48] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. arXiv:1512.00567 [cs.CV] https://arxiv.org/abs/1512.00567

[49] The Linux Foundation. 2017. ONNX. https://onnx.ai/. Accessed: 2025-01-31.

[50] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Ugur Cetintemel, and Carsten Binnig. 2020. DBPal: A Fully Pluggable NL2SQL Training Pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 2347–2361. doi:10.1145/3318464.3380589

[51] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. 2020. Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. arXiv:2004.09602 [cs.LG] https://arxiv.org/abs/2004.09602

[52] Zhihui Yang, Zuozhi Wang, Yicong Huang, Yao Lu, Chen Li, and X Sean Wang. 2022. Optimizing machine learning inference queries with correlative proxy models. *Proceedings of the VLDB Endowment* 15, 10 (2022), 2032–2044.

[53] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley J. Osher, Yingyong Qi, and Jack Xin. 2019. Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets. *CoRR* abs/1903.05662 (2019). arXiv:1903.05662 http://arxiv.org/abs/1903.05662

[54] Yuxiao Zhou, Zhishan Guo, Zheng Dong, and Kecheng Yang. 2023. TensorRT Implementations of Model Quantization on Edge SoC. In *2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 486–493.