

Optimization and deployment of NN

Wroclaw University of Science and Technology, 2025-01-13

Grzegorz Latosinski



ANTMICRO

- Founded in 2009
- Turning ideas into software-driven products
- Industrial IoT and embedded systems: AI/ML in defense/security, mining, agriculture, autonomous vehicles, robotics, aerospace, industrial automation
- We use, develop, advocate open source
- Introducing **new design methodologies and workflows** based on open source



OPEN SOURCE LEADERSHIP

We are members of the world's leading open source organizations and initiatives.



WHAT DO WE DO?

It's Open Source, see for yourself!

- How we code
 - Antmicro GitHub (737 repos!)
github.com/antmicro
 - Antmicro Open Source Portal
opensource.antmicro.com
- How we design hardware
 - openhardware.antmicro.com
- Our blog
 - blog.antmicro.com






CP1, WROCLAW



BALTYK, POZNAN



CONCORDIA, POZNAN

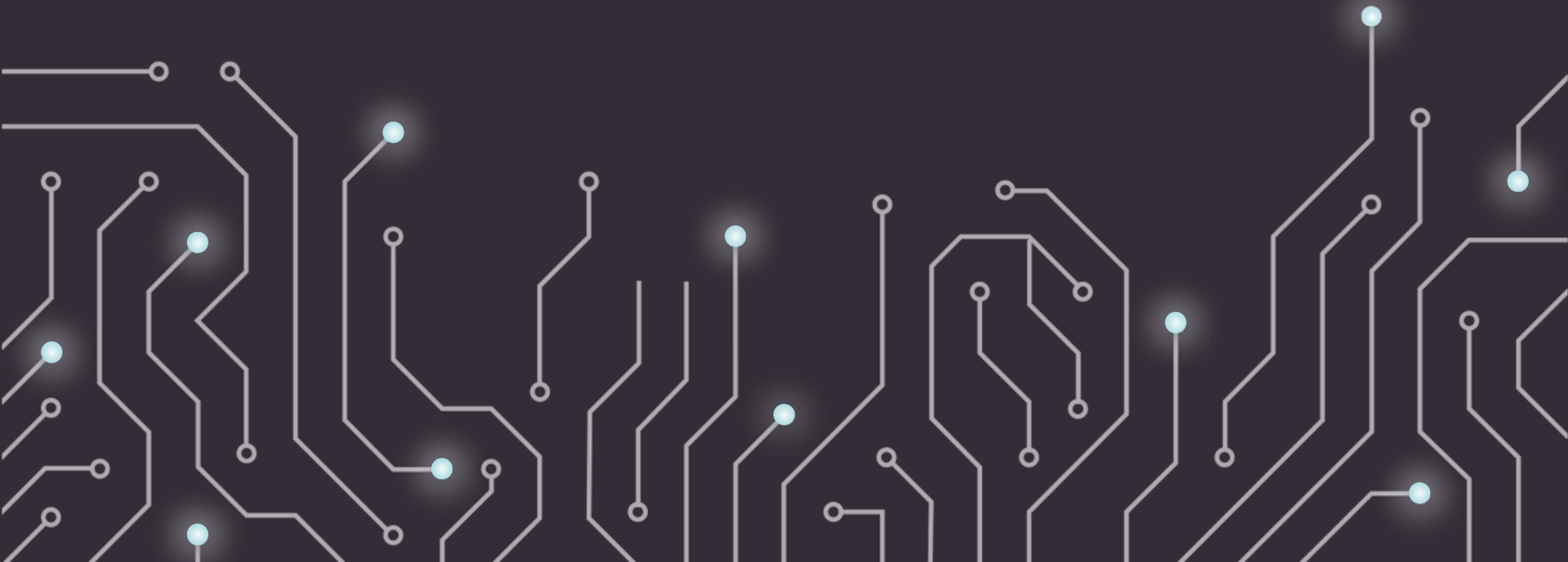


ETERNUM, GDANSK

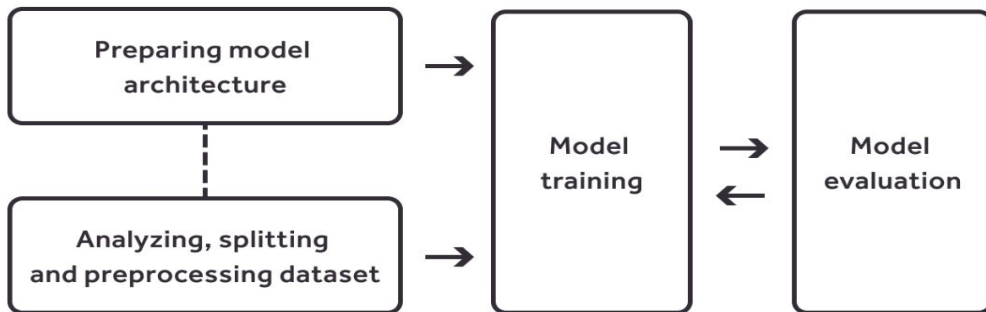


CITY GATE, GOTHENBURG

NEURAL NETWORKS DEPLOYMENT PROCESS



DEEP LEARNING DEPLOYMENT PROCESS



 TensorFlow

 PyTorch

 mxnet



POSSIBLE TARGET PLATFORMS

- Cloud platforms - GPU/TPU
- Desktop PCs - GPU/iGPU/CPU
- Single board/module computers
 - CPU: Raspberry Pi boards (ARM), HiFive boards (RISC-V), ...
 - GPU/eGPU: NVIDIA Jetson platforms, Asus Tinkerboard
 - Edge TPU: Google Coral
- External acceleration modules:
 - Intel: Neural Compute Stick, Myriad
 - Google: Google Coral TPUs (module or USB)
 - Hailo AI accelerators
- FPGAs
- Open source accelerators:
 - Apache VTA - <https://github.com/apache/tvm-vta>
 - Kelvin - <https://opensecurity.google.com/hw/kelvin/>
- Microcontrollers



WHY RUNNING DNN CAN BE DIFFICULT?

- High memory demand
- High computational demand
- Size vs quality trade-off
- Frameworks for NN development and training are large and memory demanding on their own
- Hardware does not support floating-point arithmetic, or runs it extremely slow

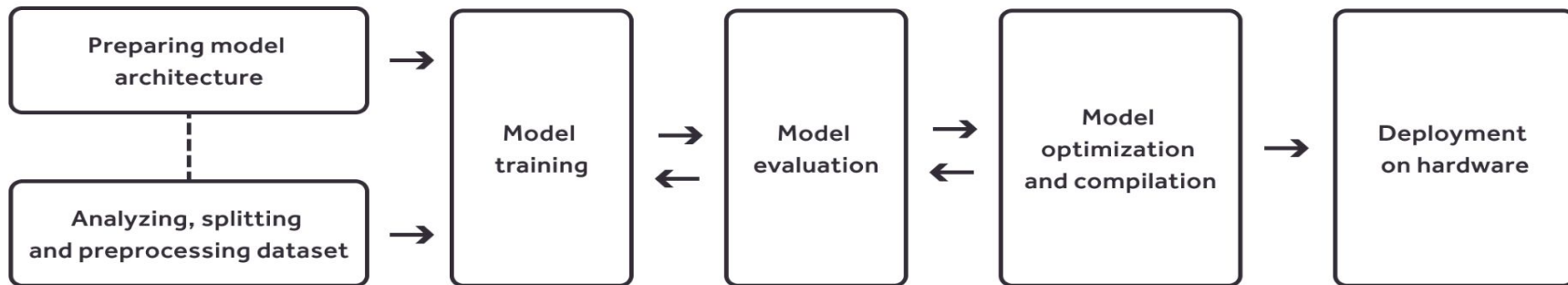


WHY RUNNING DNN LOCALLY?

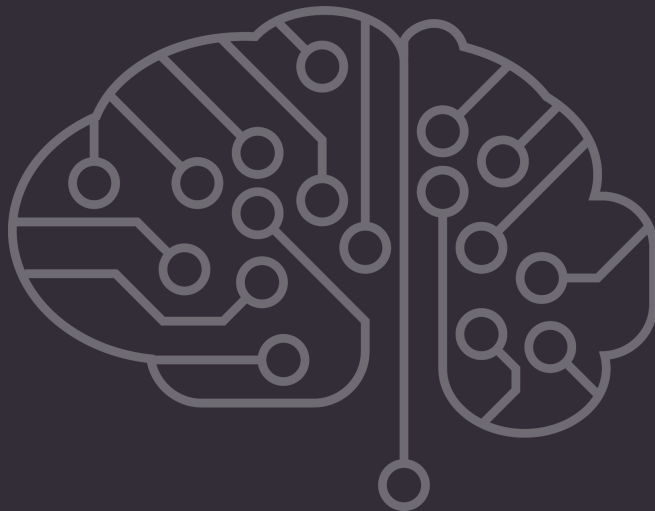
- **Cloud independence** - solution is offline
- **Security and privacy** - no potentially sensitive data is sent to a remote location
- **Latency** - passing data to the Cloud significantly increases processing time
- **Reliability** - the network communication is unreliable
- **Scalability** - in the long run, IoT-based machine learning solutions are far more scalable than centralized cloud solutions
- **Miniaturization**
- **Cost and energy efficiency**



DEEP LEARNING DEPLOYMENT PROCESS



MODEL COMPRESSION ALGORITHMS



MODEL COMPRESSION ALGORITHMS

- **Reasons for large sizes of models:**

- High-precision weights
- Lower cost of imposing regularization techniques and other overfitting-preventing strategies compared to several iterations of small model development

- **Possible fields of improvement:**

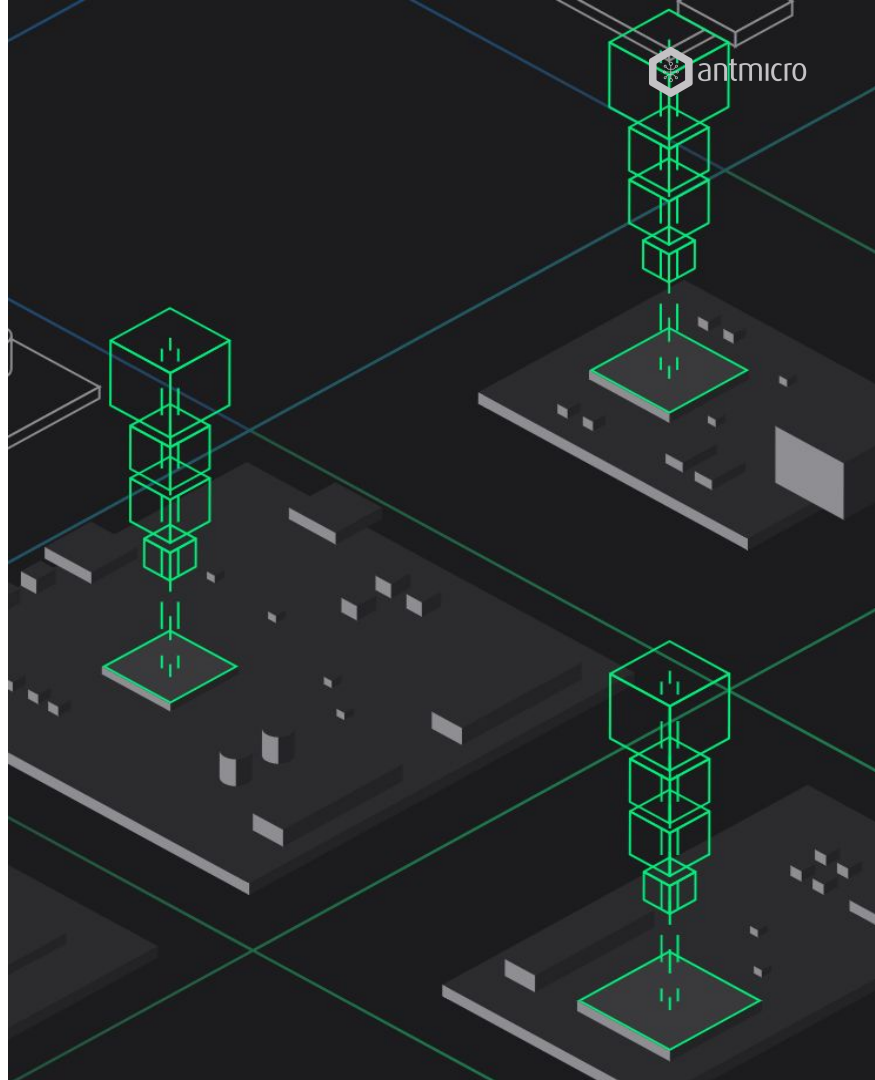
- Lower number of bits per weight
- Removal of insignificant weights/tensors
- Making tensors compression-friendly

- **Possible benefits:**

- Smaller size in storage
- Smaller size in memory
- Faster inference

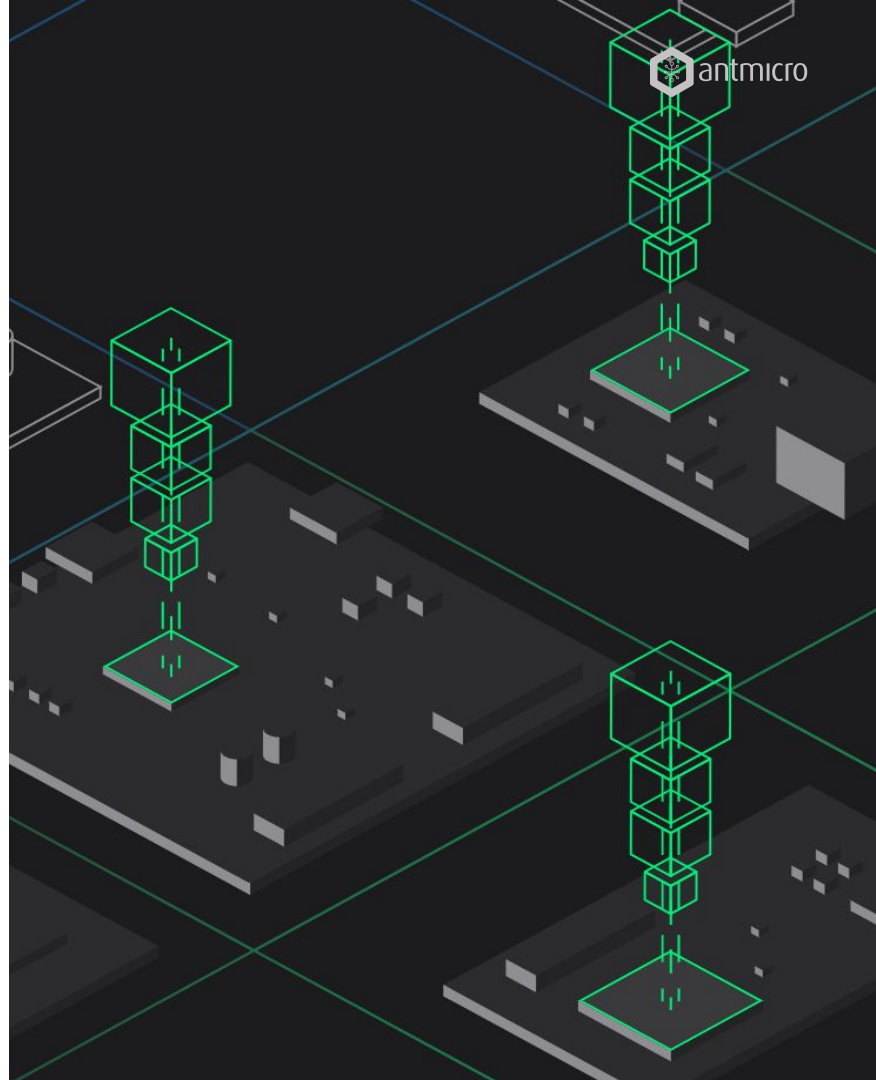
- **Possible problems:**

- Worse quality of predictions
- Slower inference

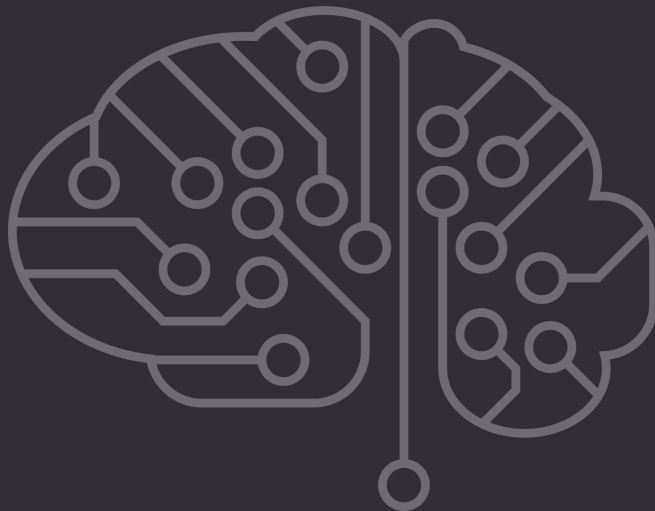


MODEL COMPRESSION ALGORITHMS

- **Quantization**
- **Pruning**
- **Knowledge distillation**
- Clustering
- Low-rank approximation
- ...



QUANTIZATION



FP32

QUANTIZATION

- Quantization is the process of reducing the number of bits used to represent weights in the neural network
- What is quantized?
 - Weights
 - Activations
- Target weights' types:
 - FP32 - $(-)^{\sim}1.17\text{E}-38 \dots \sim 3.4\text{E}+38$, 6-9 significant decimal digits prec.

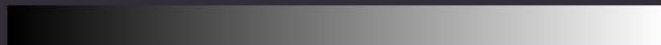


This is way too much bits, can we reduce it?

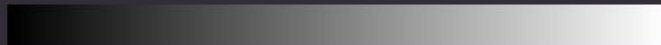
QUANTIZATION

- Quantization is the process of reducing the number of bits used to represent weights in the neural network
- What is quantized?
 - Weights
 - Activations
- Target weights' types:
 - FP32 - $(-)^{\sim}1.17\text{E-}38 \dots \sim 3.4\text{E}+38$, 6-9 significant decimal digits prec.
 - FP16 - $(-)^{\sim}5.96\text{E-}8 \dots 65504$, 4 significant decimal digits precision

FP32



FP16

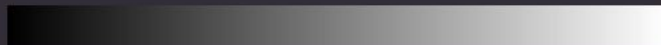


A little bit better, but still too much. Can we go lower?

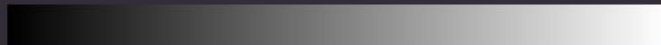
QUANTIZATION

- Quantization is the process of reducing the number of bits used to represent weights in the neural network
- What is quantized?
 - Weights
 - Activations
- Target weights' types:
 - FP32 - $(-)^{\sim}1.17\text{E-}38 \dots \sim 3.4\text{E}+38$, 6-9 significant decimal digits prec.
 - FP16 - $(-)^{\sim}5.96\text{E-}8 \dots 65504$, 4 significant decimal digits precision
 - INT8 - -128...127

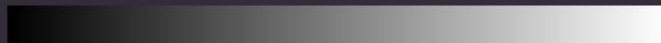
FP32



FP16



INT8

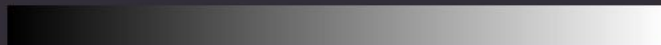


Nice, can we go lower?

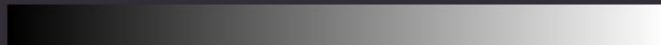
QUANTIZATION

- Quantization is the process of reducing the number of bits used to represent weights in the neural network
- What is quantized?
 - Weights
 - Activations
- Target weights' types:
 - FP32 - $(-)^{\sim}1.17\text{E-}38 \dots \sim 3.4\text{E}+38$, 6-9 significant decimal digits prec.
 - FP16 - $(-)^{\sim}5.96\text{E-}8 \dots 65504$, 4 significant decimal digits precision
 - INT8 - -128...127
 - INT4 - -8...7

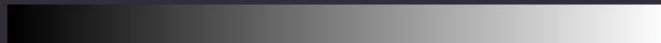
FP32



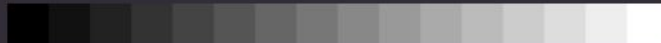
FP16



INT8



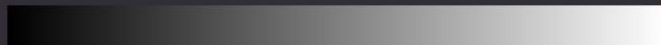
INT4



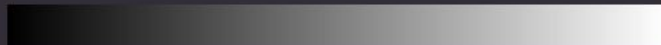
QUANTIZATION

- Quantization is the process of reducing the number of bits used to represent weights in the neural network
- What is quantized?
 - Weights
 - Activations
- Target weights' types:
 - FP32 - $(-)^{\sim}1.17\text{E-}38 \dots \sim 3.4\text{E+}38$, 6-9 significant decimal digits prec.
 - FP16 - $(-)^{\sim}5.96\text{E-}8 \dots 65504$, 4 significant decimal digits precision
 - INT8 - -128...127
 - INT4 - -8...7
 - INT3 - -4...3

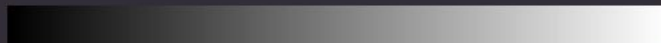
FP32



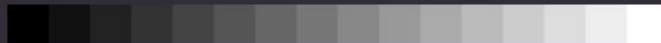
FP16



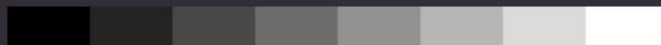
INT8



INT4



INT3



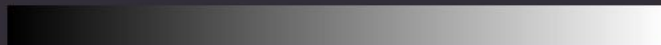
QUANTIZATION

- Quantization is the process of reducing the number of bits used to represent weights in the neural network
- What is quantized?
 - Weights
 - Activations
- Target weights' types:

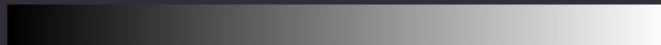
- FP32 - $(-)^{\sim}1.17E-38 \dots \sim 3.4E+38$, 6-9 significant decimal digits prec.
- FP16 - $(-)^{\sim}5.96E-8 \dots 65504$, 4 significant decimal digits precision
- INT8 - -128...127
- INT4 - -8...7
- INT3 - -4...3
- INT2/INT1.58/INT1 -

https://huggingface.co/blog/1_58_llm_extreme_quantization

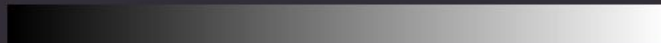
FP32



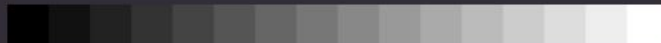
FP16



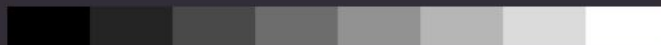
INT8



INT4



INT3



INT1



QUANTIZING VALUES

- Uniform quantization - even distribution of values in range from alpha to beta, most common quantization approach

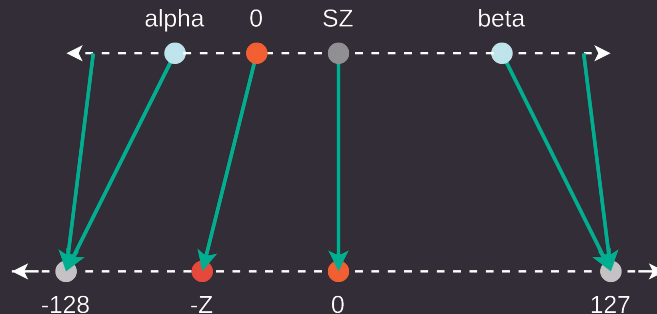
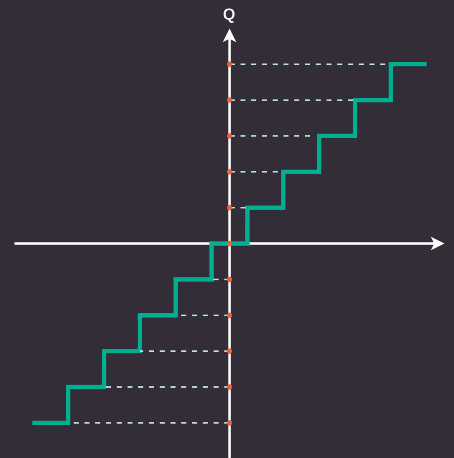
- Uniform quantization formula:

$$Q(r) = \text{Int}(\frac{r}{S}) + Z$$

- Where:

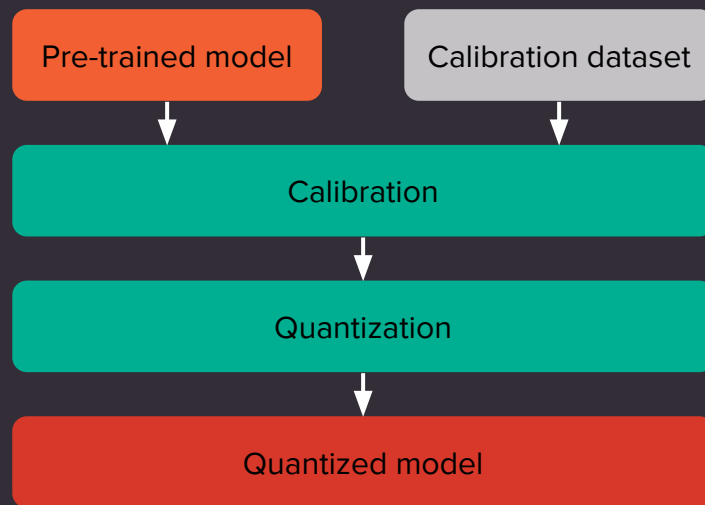
- Int - function mapping real value to an integer (using i.e. rounding or truncation)
- S - scaling factor (floating point value)
- Z - zero point (integer value)

- **Hyperparameters to establish: scaling factor and zero point**



POST-TRAINING QUANTIZATION

- **Post-training quantization** is the process of computing quantization parameters (clipping ranges/scale factors/zero points) based on a fixed pre-trained model and a calibration dataset
- Model is not trained in the process
- **Calibration dataset:**
 - Can be relatively small (much smaller than training dataset)
 - Should be **representative** (should be as diverse as possible, i.e. providing at least few samples for each class)
 - Does not in general needs to be labeled (in supervised learning)



POST-TRAINING QUANTIZATION

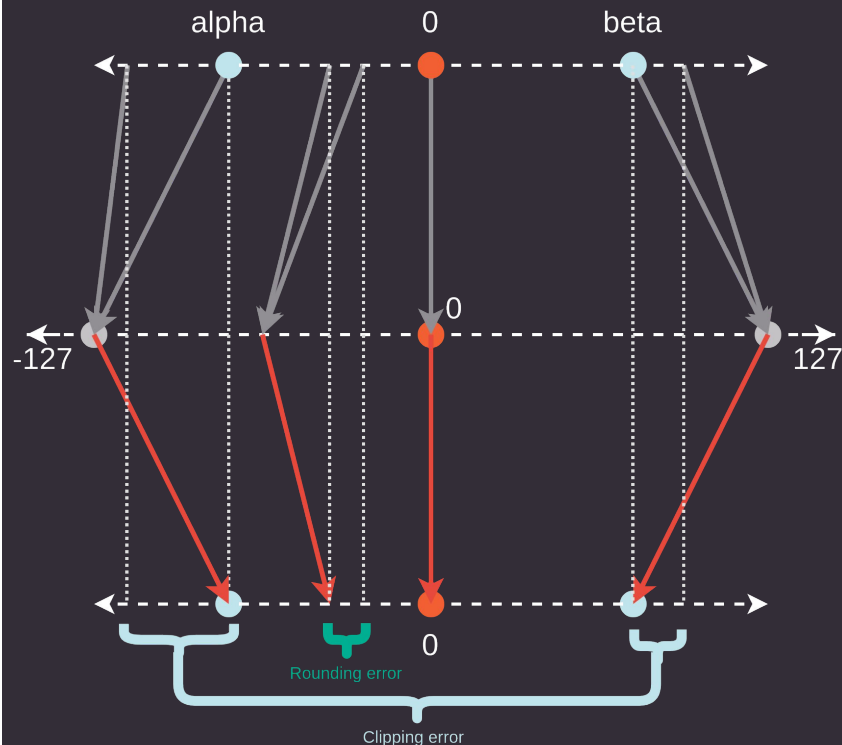
$$S = \frac{\beta - \alpha}{2^b - 1}$$

$$Q(r) = \text{Int}\left(\frac{r}{S}\right) - Z$$

- Calibration is the process of determining the clipping range, and based on this S and Z parameters
- Approaches:
 - **Min-max range** (prone to outliers):

$$\alpha = \min(R)$$

$$\beta = \max(R)$$
 - **Exponential moving average (EMA)**
 - **Optimization-based methods**



OPTIMIZATION-BASED POST-TRAINING QUANTIZATION

- Exponential moving average:

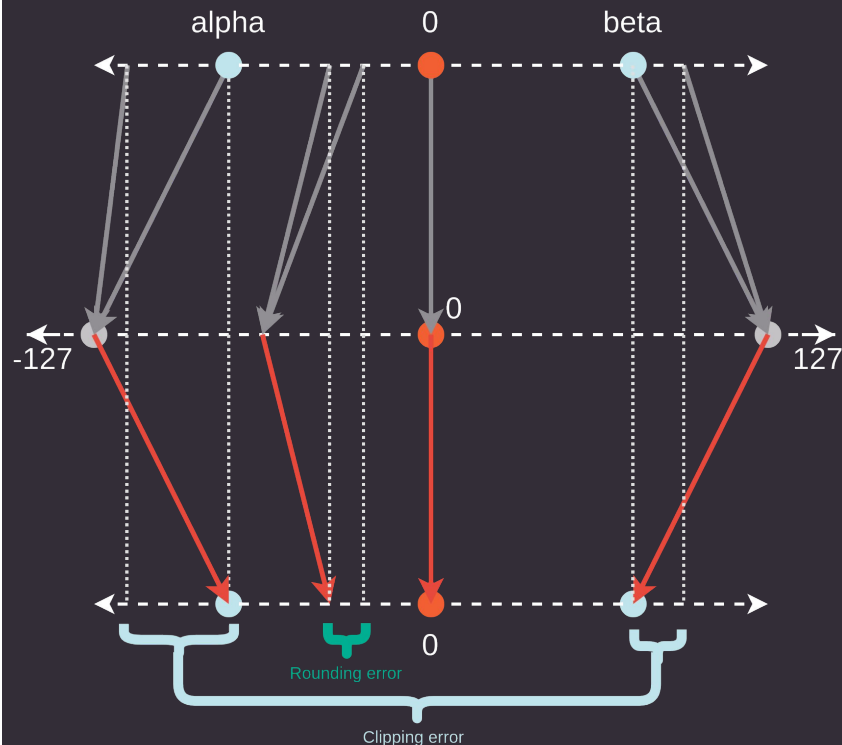
$$S_0 = X_0$$

$$S_t = decay * S_{t-1} + (1 - decay) * X_t$$

- Where:

- S_t is the current average value
- X_t is the current sample
- Decay is the smoothing factor telling how fast the previous observations fade when exposed to new data

- In EMA approach, we collect ranges of values in each activation tensor and compute α and β using EMA on observed values with decay value close to 1.0



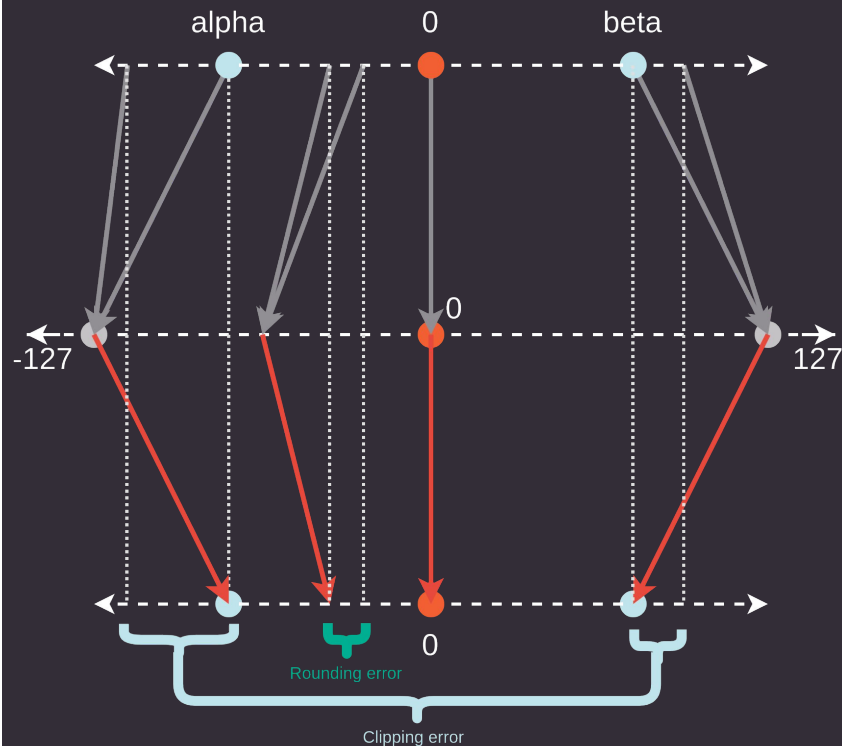
OPTIMIZATION-BASED POST-TRAINING QUANTIZATION

- Optimization-based methods:

$$\operatorname{argmin}_{\alpha, \beta} \mathcal{L}(R, \hat{R}(\alpha, \beta))$$

- Where:

- R is the set of real values to find the conversion parameters for (weights, activation values)
- α and β are the clipping range for real values
- $\hat{R}(\alpha, \beta)$ is the set of **dequantized** R values using the clipping range
- \mathcal{L} is the **loss function for quantization parameters**:
 - Mean-squared error
 - Cross-entropy
 - Kullback-Leibler divergence (relative entropy)



SYMMETRIC VS ASYMMETRIC QUANTIZATION

SYMMETRIC

$$Q(r) = \text{Int}(\frac{r}{s}) + Z$$

- $Z = 0$
- $-\alpha = \beta$
- The simplest case:
 - $\alpha = \beta = \max(|r_{\max}|, |r_{\min}|)$
- r_{\max}, r_{\min} are either max and min values of weights, or max and min observed values during calibration process
- Widely adopted in weights quantization
- Performs well only if distribution of values is not skewed, otherwise the symmetric clipping may significantly reduce the mapping quality

ASYMMETRIC

- $Z \neq 0$
- $-\alpha \neq \beta$
- The simplest case:
 - $\alpha = r_{\min}$
 - $\beta = r_{\max}$
- Adopted for activation values quantization
- More general, more flexible (due to the offset Z)
- Often has a significantly tighter clipping range, which is important when the quantized values are imbalanced, i.e. ReLU activation values

**Is there any benefit to using
symmetric quantization instead of
asymmetric quantization?**

SYMMETRIC VS ASYMMETRIC QUANTIZATION

$$Q(r) = \text{Int}(\frac{r}{s}) + Z$$

ASYMMETRIC

$$Y_r = W_r X_r$$

$$s_Y(Y - z_Y) = s_W(W - z_W)s_X(X - z_X)$$

$$s_Y(Y - z_Y) = s_W s_X (WX - Wz_X - Xz_W + z_W z_X)$$

$$Y = \frac{s_W s_X}{s_Y} WX - \frac{s_W s_X}{s_Y} Wz_X - \frac{s_W s_X}{s_Y} Xz_W + \frac{s_W s_X}{s_Y} z_W z_X + z_Y$$

SYMMETRIC

$$Y = \frac{s_W s_X}{s_Y} WX + z_Y$$

SYMMETRIC VS ASYMMETRIC QUANTIZATION

$$Q(r) = \text{Int}(\frac{r}{s}) + Z$$

ASYMMETRIC

$$Y_r = W_r X_r$$

$$s_Y(Y - z_Y) = s_W(W - z_W)s_X(X - z_X)$$

$$s_Y(Y - z_Y) = s_W s_X (WX - Wz_X - Xz_W + z_W z_X)$$

$$Y = \boxed{\frac{s_W s_X}{s_Y} WX} \boxed{- \frac{s_W s_X}{s_Y} Wz_X} \boxed{- \frac{s_W s_X}{s_Y} Xz_W} \boxed{+ \frac{s_W s_X}{s_Y} z_W z_X} \boxed{+ z_Y}$$

SYMMETRIC

$$Y = \boxed{\frac{s_W s_X}{s_Y} WX} \boxed{+ z_Y}$$

Can be pre-computed

Can't be pre-computed

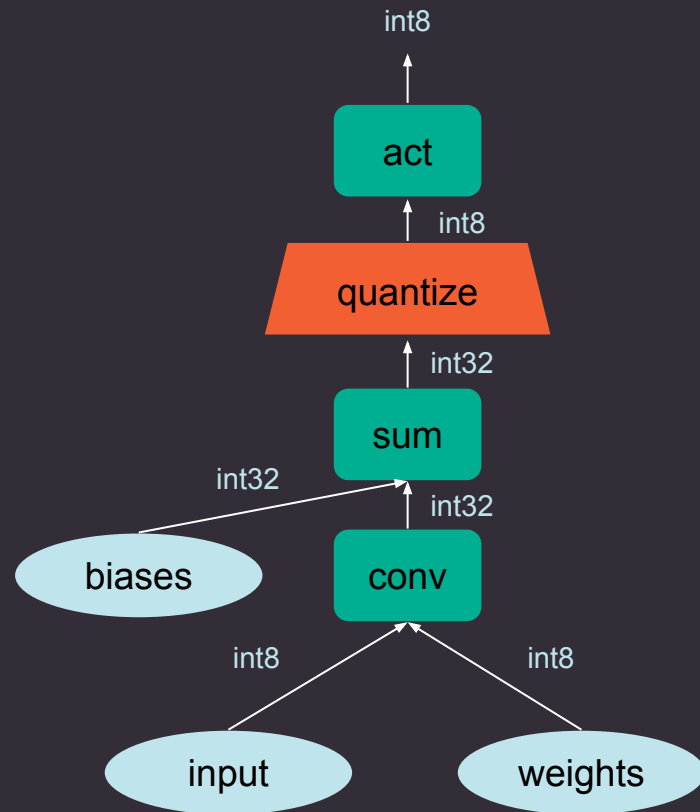
FULLY QUANTIZED INFERENCE

- **Quantizers** are blocks that quantize or requantize the input signals
- To prevent overflows, the convolution results and biases are represented as 32-bit integers (sum of int8 products requires a 32-bit accumulator)
- TensorFlow Lite CONV_2D specification ([check TFLite spec](#)):

CONV_2D

```

Input 0:
  data_type : int8
  range     : [-128, 127]
  granularity: per-tensor
Input 1 (Weight):
  data_type : int8
  range     : [-127, 127]
  granularity: per-axis (dim = 0)
  restriction: zero_point = 0
Input 2 (Bias):
  data_type : int32
  range     : [int32_min, int32_max]
  granularity: per-axis
  restriction: (scale, zero_point) = (input0_scale * input1_scale[...], 0)
Output 0:
  data_type : int8
  range     : [-128, 127]
  granularity: per-tensor
  
```



FULLY QUANTIZED INFERENCE

FULLY_CONNECTED

```

Input 0:
    data_type : int8
    range      : [-128, 127]
    granularity: per-tensor

Input 1 (Weight):
    data_type : int8
    range      : [-127, 127]
    granularity: per-tensor
    restriction: zero_point = 0

Input 2 (Bias):
    data_type : int32
    range      : [int32_min, int32_max]
    granularity: per-tensor
    restriction: (scale, zero_point) = (input0_scale * input1_scale[...],
0)
Output 0:
    data_type : int8
    range      : [-128, 127]
    granularity: per-tensor
  
```

- Formula for quantized GEMM output:

$$Y = \frac{s_W s_X}{s_Y} (W - z_W)(X - z_X) + z_Y = \frac{s_W s_X}{s_Y} * res + z_Y$$

- The scales are grouped into M:

$$M = \frac{s_W s_X}{s_Y} \in (0, 1)$$

- Since we operate on integers (INT8 in general, INT32 for intermediate results), M is represented as:

$$M = multiplier * 2^{-shift}$$

- Where multiplier is:

$$multiplier \in [0.5, 1.0)$$

- And is later stored as:

$$m = multiplier * 2^{31}$$

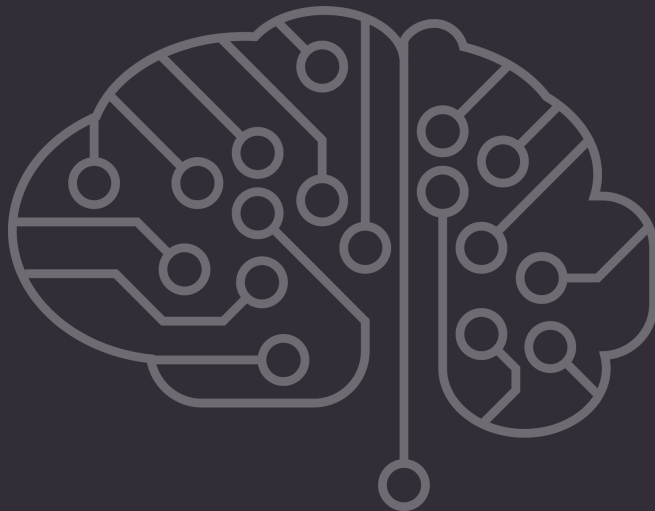
- In the end, the res is scaled as follows:

$$Y = (((res * m) + round) >> (31 - shift)) + z_Y$$

QUANTIZATION RESULTS TENSORFLOW MODEL OPTIMIZATION TOOLKIT

Model	FP32 Accuracy	FP32 size (MB)	INT8 Accuracy	INT8 size (MB)
MobileNetV2	0.8056691196511311	16	0.7756881984191878	5
MobileNetV3 small	0.8323793949304987	13	0.5622785500136277	4
ResNet50	0.7816843826655765	100	0.7835922594712456	26
InceptionV3	0.8833469610248024	94	0.8629054238212047	24
Xception	0.8661760697737804	90	0.8195693649495776	24

NETWORK PRUNING



OPTIMAL BRAIN DAMAGE



NETWORK PRUNING

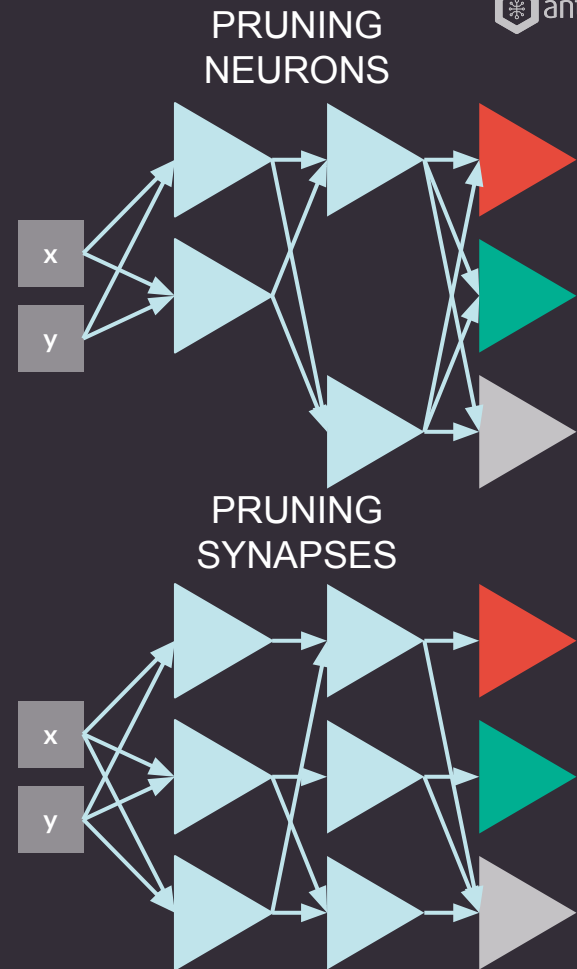
- Pruning is a family of algorithms that remove the least contributing parts of the network to a given task
- Formally, pruning is an algorithm that takes an input model

$$f(x; W)$$

And produces a new model:

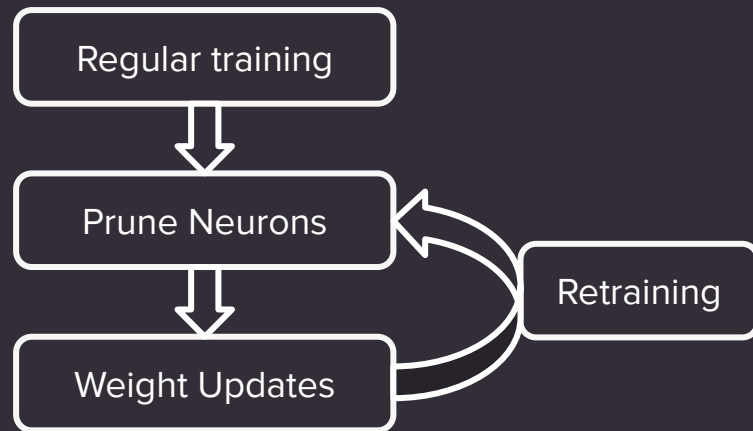
$$f(x; M \odot W')$$

- Where:
 - $f(x; \cdot)$ - a model architecture taking input x
 - W - initial weights of the model
 - $M \in \{0, 1\}^{|W|}$ - binary mask setting certain weights to 0
 - W' - fine-tuned weights
- What to prune?
 - Weights
 - Biases
 - Activations

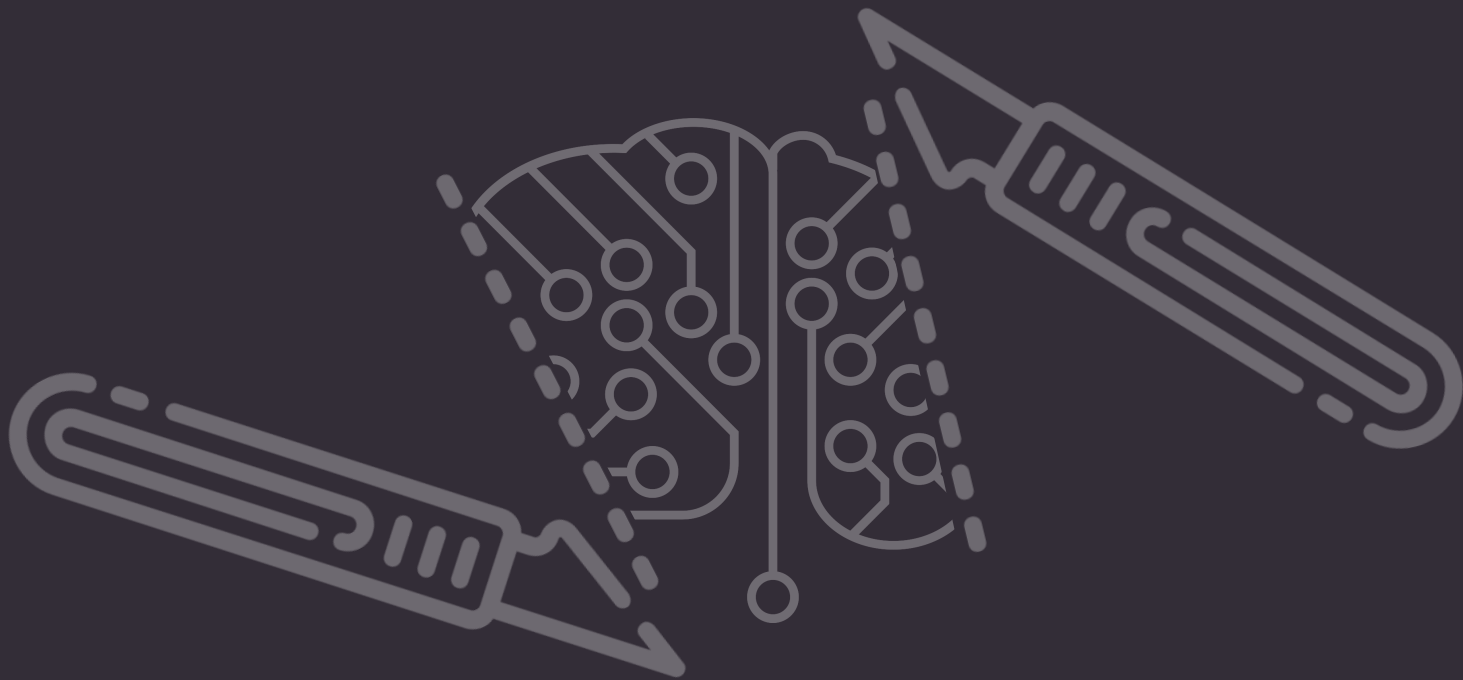


NETWORK PRUNING

- General flow is following:
 - Train the model
 - Analyse the network and create pruning masks
 - Apply pruning masks - remove connections and/or neurons
 - Fine-tune the model to recover from the pruning

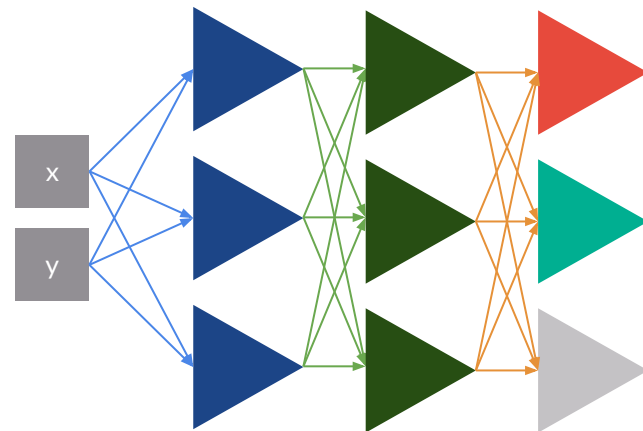


UNSTRUCTURED PRUNING



LEVEL PRUNER

- Belongs to a family of magnitude-based pruners
- Fairly simple and not overly invasive approach to pruning
- We define % of least significant weights to remove (e.g. 50%)
- Algorithm:
 - Sort the weights in the layer by their absolute values
 - Mask the smallest-magnitude weights until the desired sparsity is reached
- Usually requires little retraining
- Allows to compress the model, which is useful for storage
- Does not bring performance boost or size reduction at runtime



0.1	0.3
0.6	-0.2
-0.9	0.7

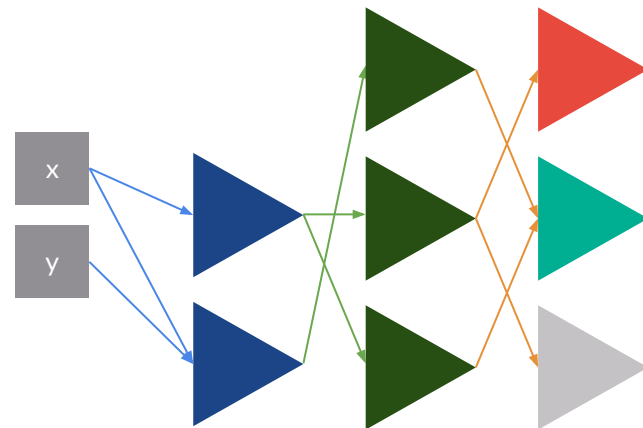
0.5	0.19	0.9
-0.3	-0.6	0.2
-0.1	0.93	0.1

-0.1	0.4	0.3
-0.4	0	-0.8
0.2	0.6	0.15

level=0.5

LEVEL PRUNER

- Belongs to a family of magnitude-based pruners
- Fairly simple and not overly invasive approach to pruning
- We define % of least significant weights to remove (e.g. 50%)
- Algorithm:
 - Sort the weights in the layer by their absolute values
 - Mask the smallest-magnitude weights until the desired sparsity is reached
- Usually requires little retraining
- Allows to compress the model, which is useful for storage
- Does not bring performance boost or size reduction at runtime



0	0
0.6	0
-0.9	0.7

0.5	0	0.9
0	-0.6	0
0	0.93	0

0	0.4	0
-0.4	0	-0.8
0	0.6	0

level=0.5

AUTOMATED GRADUAL PRUNER

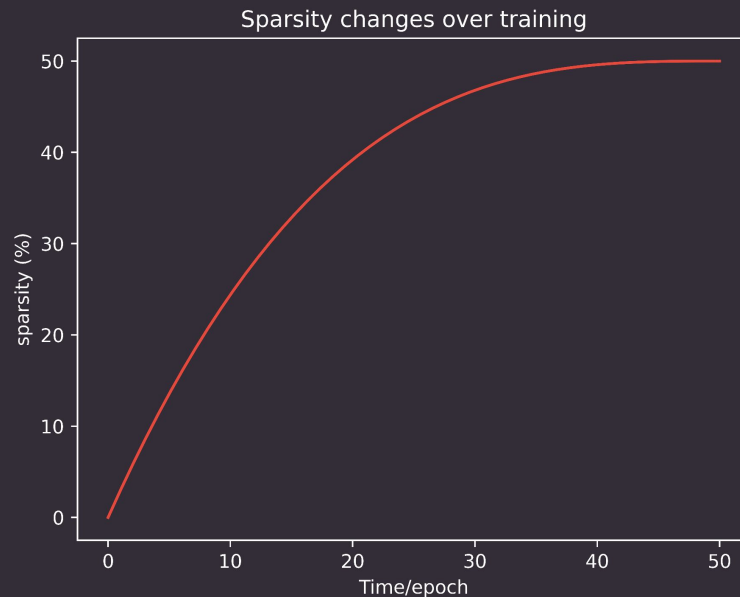
- Automated level pruner, **working during training**

- The sparsity curve follows this formula:

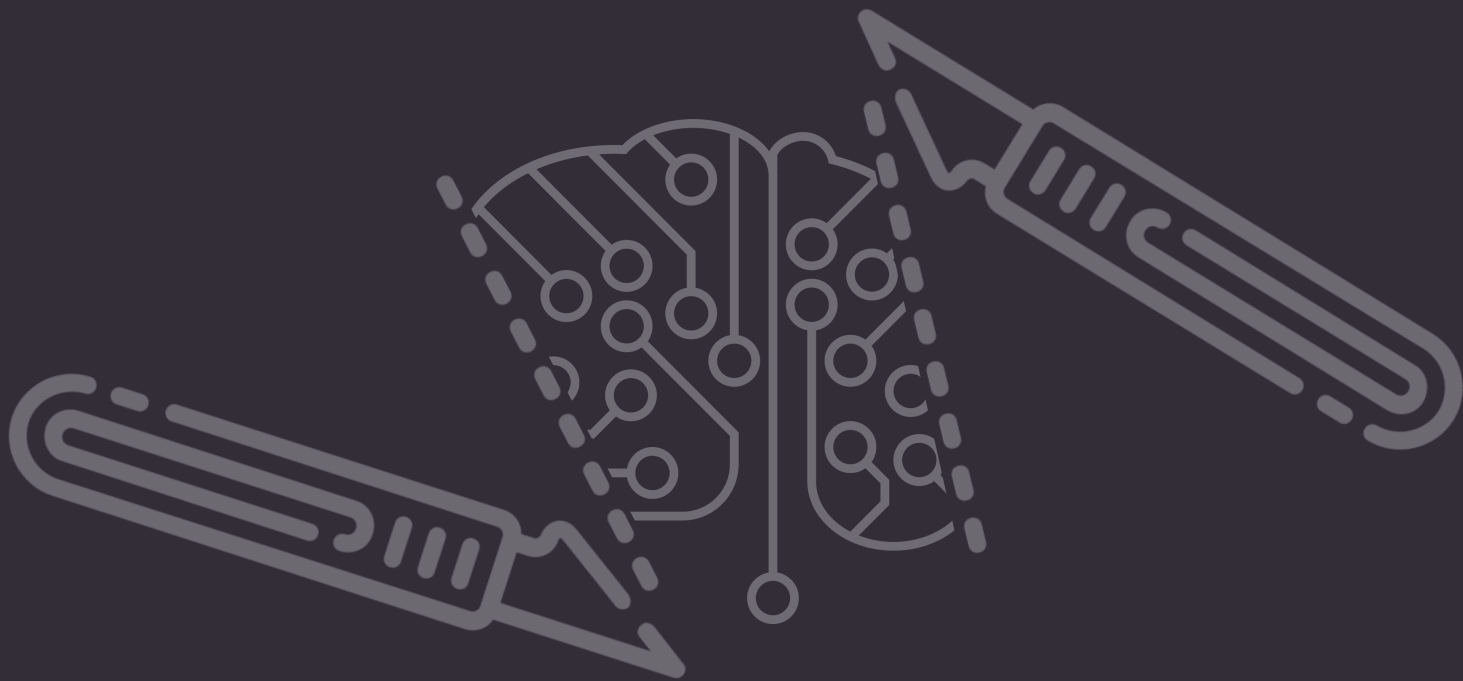
$$s_t = s_f + (s_i - s_f) \left(1 - \frac{t - t_0}{n\Delta t}\right)^3, t \in \{t_0, t_0 + \Delta t, \dots, t_0 + n\Delta t\}$$

- Where:

- s_i - initial sparsity
 - s_f - final sparsity
 - t_0 - start time
 - s_t - sparsity at time t
 - n - number of steps
 - Δt - step size
- The pruning requires minimal setting of hyperparameters - most of the setup happens automatically
 - There may be a need to adjust the learning rate policy to prevent too fast learning rate reduction, because the network may not recover from pruning

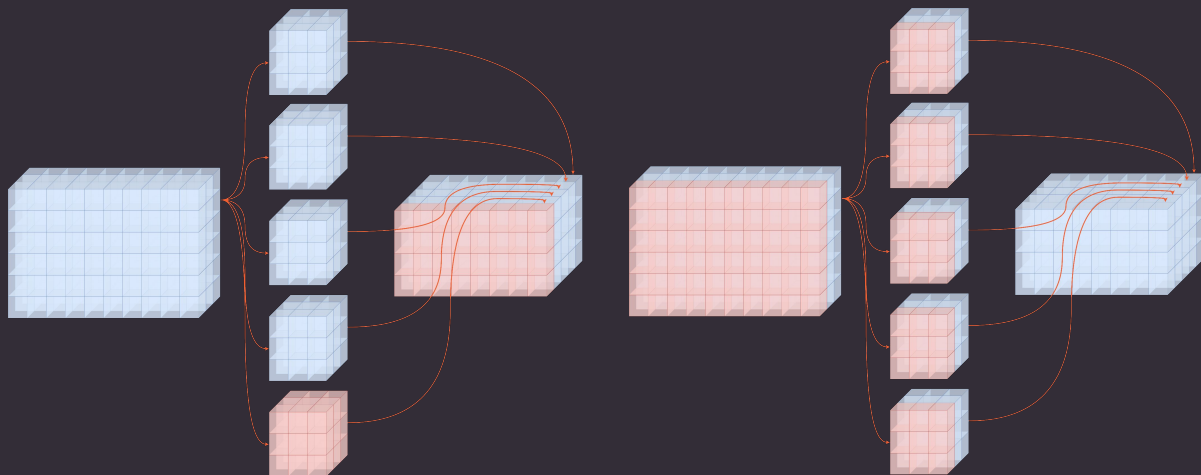
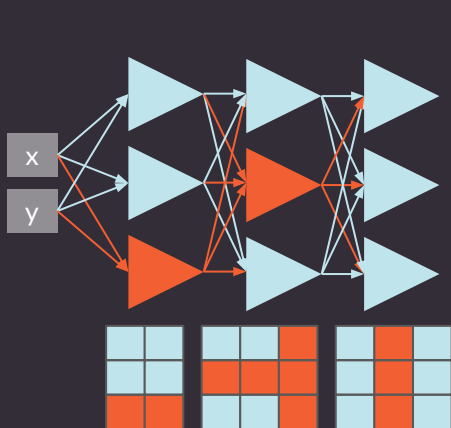


STRUCTURED PRUNING



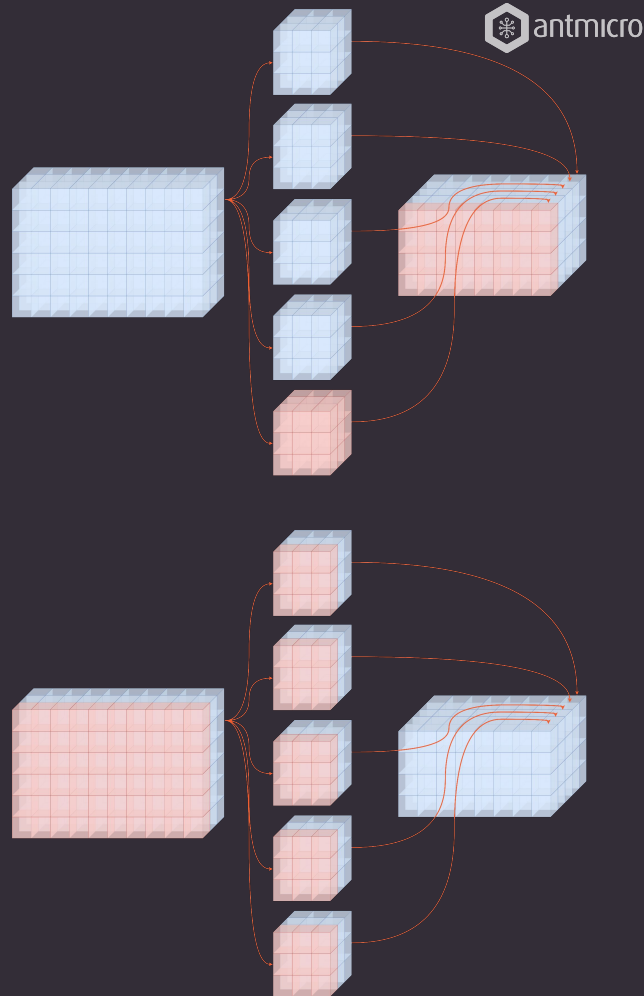
STRUCTURED PRUNING

- Pruning by neurons (fully connected layer)
- Pruning by filters (convolutional layer)
- Pruning by channels (convolutional layer)
- Significant memory and computation benefit
- Highly damaging for model's performance
- Structured pruning strategies are actively researched



L1/L2-RANKED STRUCTURE PRUNER

- Algorithm:
 - For each filter, calculate the sum of its absolute kernel weights
 - Sort the filters by s_j
 - Prune **m** filters with the smallest sum values and their corresponding feature maps. The kernels in the next convolutional layer corresponding to the pruned feature maps are also removed
 - Create a new kernel matrix for both the **i**-th and **(i+1)**-th layers, copy the remaining kernel weights to the new model
 - Fine-tune the new model until the quality of predictions is satisfactory

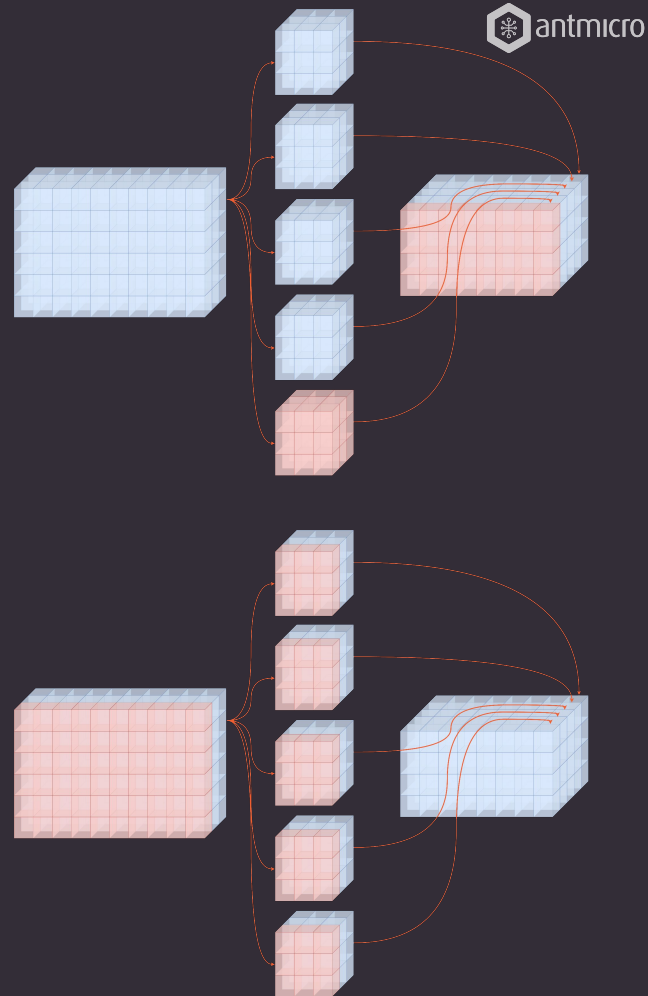


APOZ-RANKED STRUCTURE PRUNER

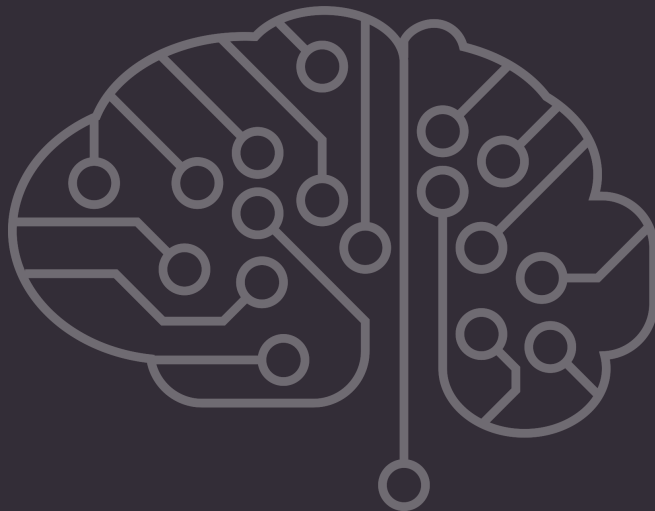
- APoZ - Average Percentage of Zeros
- APoZ is used to check the percentage of zero activations of a neuron/filter (i.e. after ReLU)
- APoZ for a c-th channel in the i-th layer is computed as follows:

$$APoZ_c^{(i)} = APoZ(O_c^{(i)}) = \frac{\sum_{k,j}^{NM} f(O_{c,j}^{(i)}(k) = 0)}{N \times M}$$

- Where:
 - $O_c^{(i)}$ - output of the c-th channel in the i-th layer
 - f - a function that equals 1 if the input is true, and 0 otherwise
 - M - represents the number of elements in output feature map
 - N - represents the number of validation examples



SECOND-ORDER DERIVATIVES FOR QUANTIZATION AND PRUNING



SECOND-ORDER DERIVATIVES FOR OPTIMIZATION

- [Optimal Brain Damage, 1989, Yann LeCun, John S. Denker, ...](#)
- [Optimal Brain Surgeon, 1992, Babak Hassibi, David G. Stork](#)
- The above papers revolved around pruning architectures of around 8000-20000 parameters (nowadays we have tens and hundreds of billions of parameters)
- The aim of pruning is to remove parameters with small “saliency” - parameters whose deletion will have the least effect on the network error (or training error, as in case of above papers)
- In pruning, most of the methods demonstrated earlier revolved around magnitude of the weight, which is an intuitive approximation of saliency - authors of above papers claim that small weights often are in fact necessary for low error
- Instead of magnitude, authors proposed pruning parameters based on minimal increase in training error after removing them
- The research from the above papers resulted in the usage of Hessian matrices, providing second-order derivatives

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

https://en.wikipedia.org/wiki/Hessian_matrix

SECOND-DERIVATIVE ANALYSIS OF WEIGHTS

- To address influence of the loss function by weights, one of the proposed ideas was to construct a local model of the error function and predict the effect of perturbing parameters in an analytic way
- They approximated the objective function L by a Taylor series shown to the right
- δW is the weight perturbation - change of weight from original value to 0
- This Taylor expansion has:
 - First order term
 - Second order term
 - Third order term
- The objective function is nearly quadratic, the third term is negligible
- We assume that the network training has converged, the first term is also negligible
- OBD claimed that we can also assume that every parameter is independent, which removes cross terms
- To sum up, we are only left with one component heavily based on the diagonal of the Hessian matrix, which can be used as our importance measurement for parameters

$$H = \frac{\partial^2 L}{\partial w^2}$$

$$g_i = \frac{\partial L}{\partial w_i}$$

$$h_{ij} = \frac{\partial^2 L}{\partial w_i \partial w_j}$$

$$\delta L = L(x; W) - L(x; W_p = W - \delta W)$$

$$\delta L = \underbrace{\sum_i g_i \delta w_i}_{\text{X}} + \underbrace{\frac{1}{2} \sum_i h_{ii} \delta w_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \delta w_i \delta w_j}_{\text{X}} + \underbrace{O(\|\delta W\|^3)}_{\text{X}}$$



$$\delta L \approx \frac{1}{2} \sum_i h_{ii} \delta w_i^2$$

$$importance_{w_i} \approx |\delta L_i| = \frac{1}{2} h_{ii} w_i^2$$

Paper



SECOND-ORDER DERIVATIVES FOR OPTIMIZATION

- [Optimal Brain Surgeon. 1992. Babak Hassibi. David G. Stork](#)
- Example on the right demonstrates XOR network with two inputs (and bias as a “third” input), 2 hidden neurons and 9 connections
- Hessian matrix is 9x9 matrix, where the darker the color, the lower the value is
- In network graph, the thicker the line, the higher magnitude the weight has (dashed lines represent negative weights)
- Looking at magnitudes:
 - The weight with the smallest magnitude is V_3 - it would be removed
 - After this, according to the paper, the network was unable to solve XOR problem
- Looking at Hessian:
 - Components for hidden-to-output weights are high, especially for V_1/V_3 values
- The bottom left plot represents the two-dimensional slice of the nine-dimensional error surface in the neighborhood of the starting point (state of weights before pruning)
 - OBS represents the point in the slice where U_{23} (picked by OBS) was zeroed out
 - Mag represents the point in the slice where V_3 (picked by magnitude pruner) was zeroed out

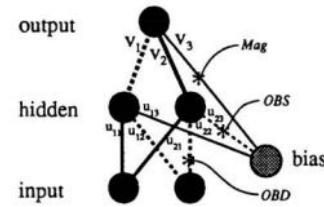


Figure 3: A nine weight XOR network trained to a local minimum. The thickness of the lines indicates the weight magnitudes, and inhibitory weights are shown dashed. Subsequent pruning using a magnitude based method (*Mag*) would delete weight v_3 ; using Optimal Brain Damage (*OBD*) would delete u_{22} . Even with retraining, the network pruned by those methods cannot learn the XOR problem. In contrast, Optimal Brain Surgeon (*OBS*) deletes u_{23} and furthermore changed all other weights (cf. Eq. 5) to achieve zero error on the XOR problem.

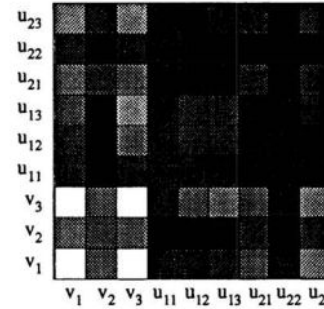


Figure 4: The Hessian of the trained but unpruned XOR network, calculated by means of Eq. 12. White represents large values and black small magnitudes. The rows and columns are labeled by the weights shown in Fig. 3. As is to be expected, the hidden-to-output weights have significant Hessian components. Note especially that the Hessian is far from being diagonal. The Hessians for all problems we have investigated, including the MONK's problems (below), are far from being diagonal.

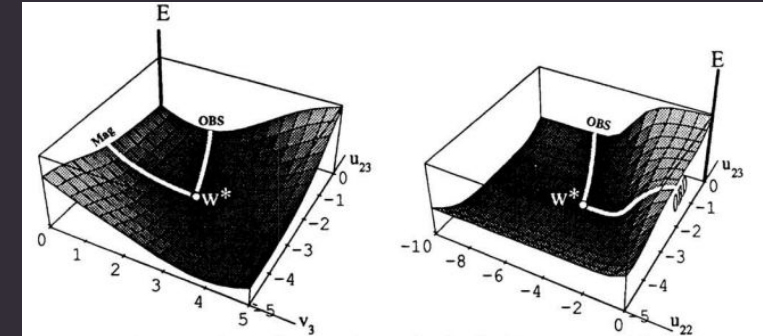


Figure 5: (Left) the XOR error surface as a function of weights v_3 and u_{23} (cf. Fig. 4). A magnitude based pruning method would delete weight v_3 whereas OBS deletes u_{23} . (Right) The XOR error surface as a function of weights u_{22} and u_{23} . Optimal Brain Damage would delete u_{22} whereas OBS deletes u_{23} . For this minimum, only deleting u_{23} will allow the pruned network to solve the XOR problem.

SECOND-ORDER DERIVATIVES FOR OPTIMIZATION

- Once LLMs emerged, the research regarding Hessian-based optimizations sped up, leading to all kinds of parallelization, batching and simplification of formulas for Hessians

$$\operatorname{argmin}_{\hat{W}} \|WX - \hat{W}X\|_2^2 \quad H = 2XX^T$$

- Overall, during recent years, following algorithms had emerged for LLMs:

- [Optimal Brain Compression/Optimal Brain Quantizer \(OBC/OBQ\) - 2022](#)

- Introduces single framework that can tackle both pruning and quantization
 - Picking next weight for pruning:

$$\operatorname{argmin}_p \frac{w_p^2}{[H^{-1}]_{pp}}$$

- Picking next weight for quantization:

$$\operatorname{argmin}_p \frac{(Q(w_p) - w_p)^2}{[H^{-1}]_{pp}}$$

- [Gradient-based Post Training Quantization \(GPTQ\) - 2023](#)

- <https://github.com/AutoGPTQ/AutoGPTQ>
 - Speeds up quantization from 1 hour in OBQ to under 1 minute for ResNet-50

- [SparseGPT - 2023](#)

- <https://github.com/IST-DASLab/sparsegpt>
 - Significantly improved the performance (and quality) of the OBC algorithm, giving one-shot pruner for LLMs
 - Has the similar algorithm as GPTQ, but prunes weights instead of quantizing them, and also allows pruning subgroups of weights

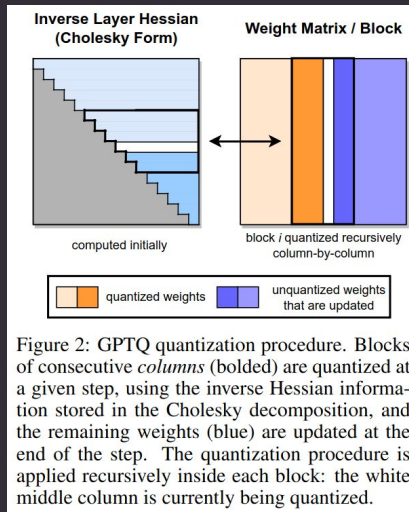


Figure 2: GPTQ quantization procedure. Blocks of consecutive *columns* (bolded) are quantized at a given step, using the inverse Hessian information stored in the Cholesky decomposition, and the remaining weights (blue) are updated at the end of the step. The quantization procedure is applied recursively inside each block: the white middle column is currently being quantized.



Algorithm 1 Prune $k \leq d_{\text{col}}$ weights from row w with inverse Hessian $H^{-1} = (2XX^T)^{-1}$ according to OBS in $O(k \cdot d_{\text{col}}^2)$ time.

```

M = {1, ..., d_col}
for i = 1, ..., k do
    p ← argmin_{p ∈ M} 1/[H^{-1}]_{pp} · w_p^2
    w ← w - H_{:,p}^{-1} 1/[H^{-1}]_{pp} · w_p
    H^{-1} ← H^{-1} - 1/[H^{-1}]_{pp} H_{:,p}^{-1} H_{p,:}^{-1}
    M ← M - {p}
end for

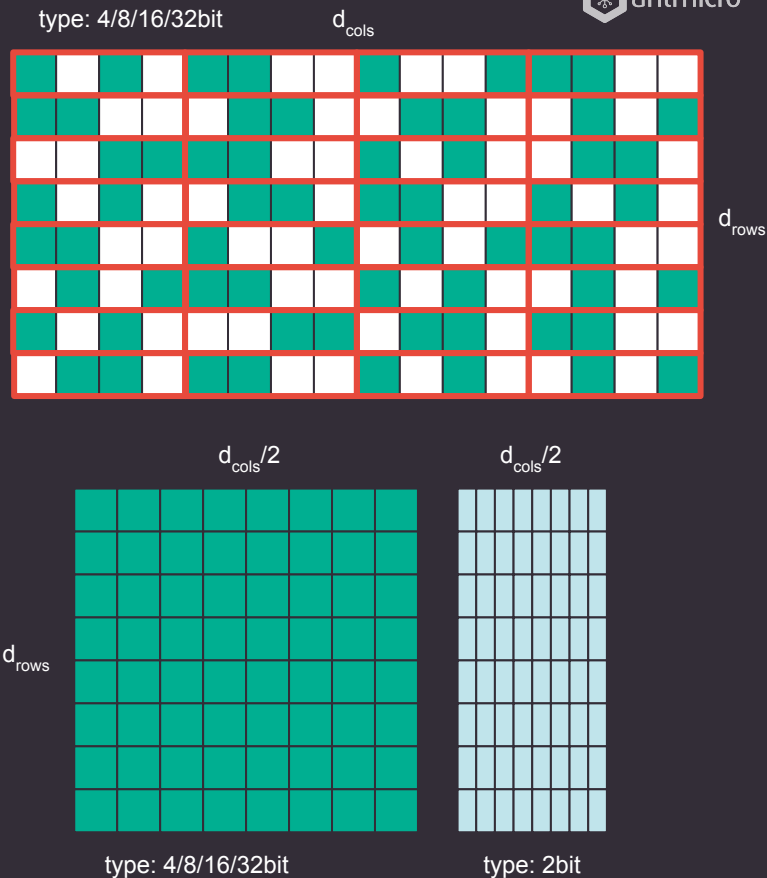
```

GPTQ COMPRESSION RESULTS

Model	Number of parameters	Original size (precision)	GPTQ 4-bit
Mistral-7B-v0.1	7.24 B	~14.48 GB (BF16)	4.16 GB
Starcoder	15.5 B	~64 GB (BF16)	8.91 GB
Vicuna-13B	13 B	~26 GB	7.26 GB
Zephyr 7B	7.24 B	~14.5 GB (BF16)	4.16 GB

SEMI-STRUCTURED PRUNING

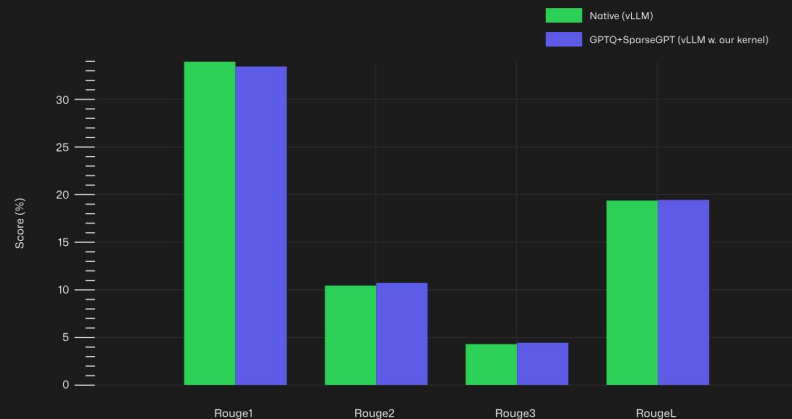
- N:M pruning - for M-element groups of weights we prune N weights
- Allows to create optimized representations, like value-index matrices
- Introduces quality reduction similar to unstructured pruning while allowing storage and memory usage optimizations similar to structured pruning
- Can be executed efficiently on certain hardware
- Algorithms supporting N:M pruning are:
 - Optimal Brain Surgeon (OBS, part of OBC)
 - <https://github.com/IST-DASLab/obc>
 - SparseGPT - an OBS-inspired pruning algorithm for LLMs
 - <https://github.com/IST-DASLab/sparsegpt>
 - Paper - [SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot](#)
- Both algorithms determine “significance” of weights using second-order derivatives (similarly to GPTQ)



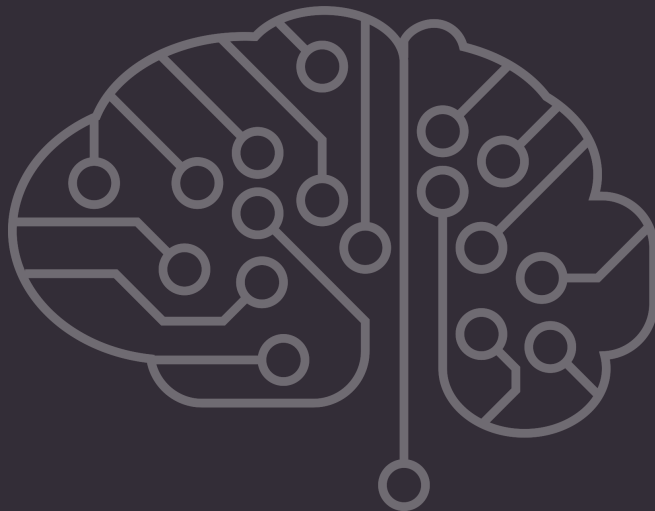
SIMULTANEOUS QUANTIZATION AND PRUNING OF LLMS

- [kenning.sparsegpt.sparsegpt](#) - our implementation of the LLM simultaneous quantization and pruning
- [kenning/sparsity_aware_kernel/custom_ext/gptq/q_compressed_gemm.cu](#) - vLLM kernel for sparse matrix multiplication (both done as Master Thesis within our internship)
- NVIDIA support for semi-structured sparse matrices:
 - NVIDIA supports 2:4 sparse matrix multiplication by dense matrix and vice versa (with various input types), including their edge platforms (NVIDIA Jetson Orin platforms)
- Along with quantization down to 4 bits, the models can reach ~20-25% of their original size without significant decrease in quality
- This, in turn, allows to deploy below models on smaller Jetson solutions, with even 4GB of available RAM
- Results of pruning and quantizing the network using GPTQ and SparseGPT:

Model	Original size (GiB)	GPTQ+SparseGPT size (GiB)	% original size	Quantization/pruning time
Mistral-7B	13.5	3.1	23%	1 hour
Phi-2	5.2	1.4	26%	20 minutes



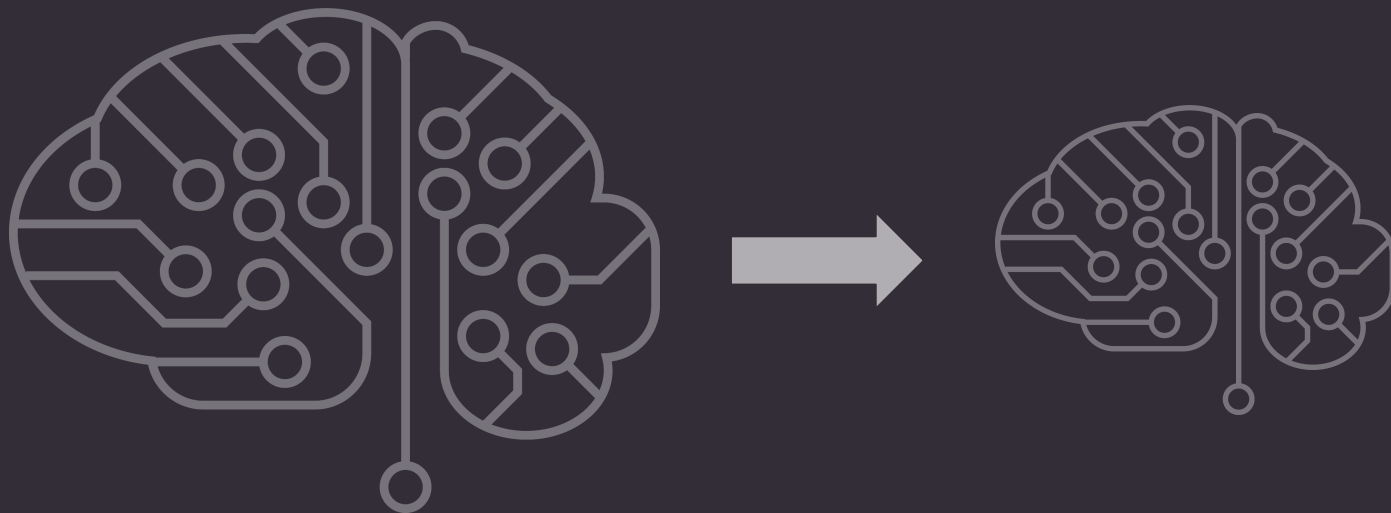
MODEL COMPRESSION SUMMARY



MODEL COMPRESSION SUMMARY

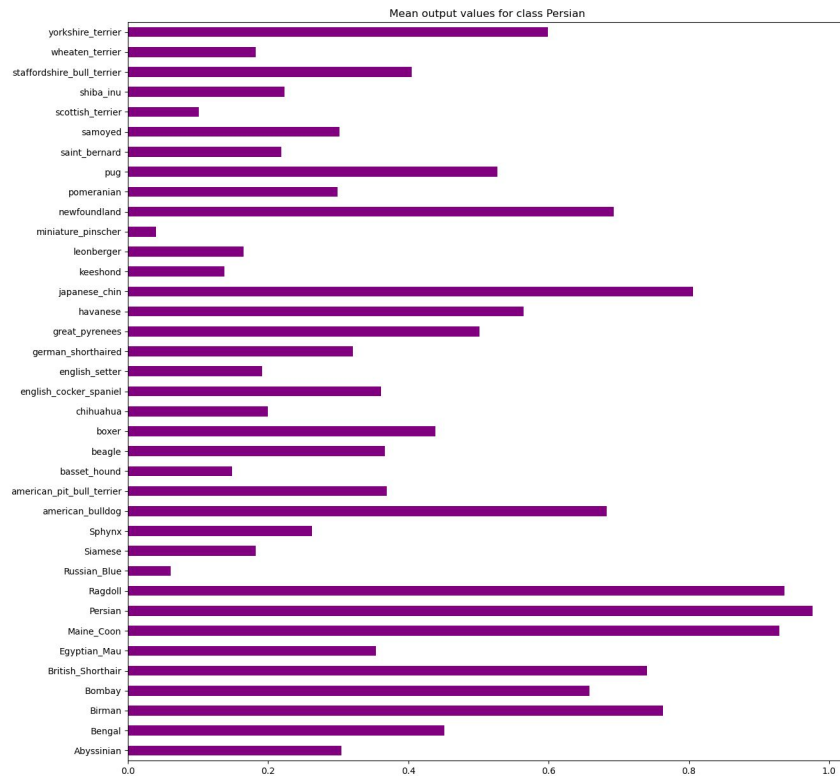
Model	Quantization	Unstructured/semi-structured pruning	Structured pruning
Smaller storage size	Always	If compressed	Always
Smaller memory footprint	Always	If library/hardware supports efficient storage of sparse matrices (e.g. NVIDIA GPUs since Ampere architecture)	Always
Faster execution	Depends on target type and hardware - most of the targets efficiently process INT8 values NVIDIA GPUs support wide range of types	Not widely available as for now, there are libraries like Blaze or Eigen for CPUs NVIDIA since Ampere architecture has Sparse Tensor Cores for unstructured pruning	Always
Requires retraining	For PTQ we don't need retraining, only calibration For LLMs, there are new quantization algorithms such as GPTQ that have few-shot calibration	The zeroing of several weights (40-50%) requires some retraining, but not too long	Removal of whole kernels/neurons requires heavy retraining
Risk of significant decrease in quality	Even with PTQ, the drop in quality (depending on application) should be negligible	Right after pruning, the quality of the network is severely decreased, but should require quite short training to bring it back to original state	After pruning, we need to run a longer training to bring back the original quality
Available optimization frameworks	TensorFlow Model Optimization Toolkit, Distiller (PyTorch), NNI (PyTorch), Kenning	TensorFlow Model Optimization Toolkit, NNI (PyTorch), Kenning	NNI (PyTorch), Kenning (experimental)
Available runtimes	TensorFlow Lite, Apache TVM, ...	TensorFlow Lite (unoptimized), TensorRT (CUDA)	Any framework that loads from PyTorch (ONNXRuntime, Apache TVM, ...)

KNOWLEDGE DISTILLATION



KNOWLEDGE OF LARGER MODELS

- The outputs of models are not one-hot vectors - there are almost no zero values
- In the properly trained model, the vector output element responsible for the appropriate class for the input should have the highest value
- The vector elements for classes similar to the true class usually have significantly higher values than other elements, i.e. for car the classes like bus, truck, motorcycle should have significantly higher values than dog, apple or toilet
- It means that outputs from large models, in comparison to ground truth, provide a crucial information about the similarities between the input and each class



“SIMILAR CLASSES” ACCORDING TO MOBILENETV2 (PET DATASET)

- English setter
- Top-5:
 - English setter 0.994704
 - English cocker spaniel 0.960606
 - German shorthaired 0.897741
 - Leonberger 0.812114
 - Newfoundland 0.786800



“SIMILAR CLASSES” ACCORDING TO MOBILENETV2 (PET DATASET)

- Yorkshire terrier
- Top-5:
 - Yorkshire terrier 0.998707
 - Havanese 0.948308
 - Pomeranian 0.871383
 - Wheaten terrier 0.852712
 - Scottish terrier 0.839262



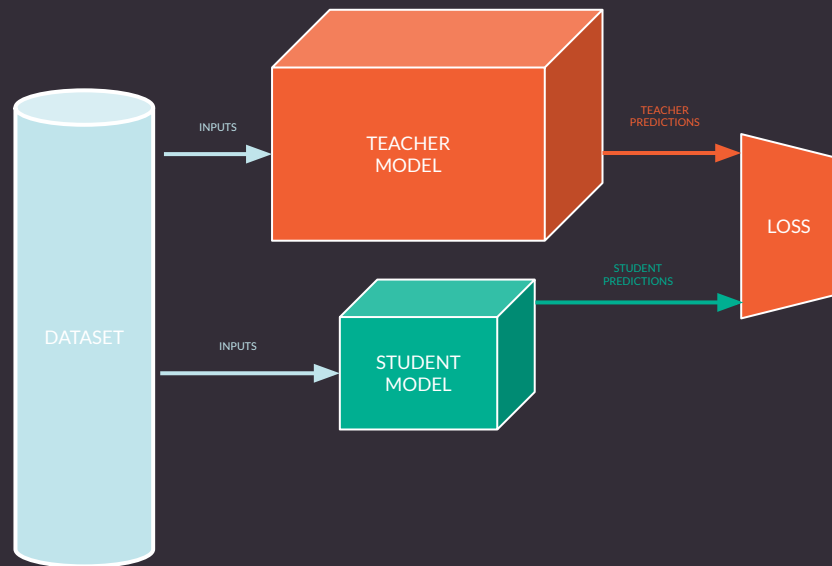
“SIMILAR CLASSES” ACCORDING TO MOBILENETV2 (PET DATASET)

- Chihuahua
- Top-5:
 - Chihuahua 0.976941
 - Sphynx 0.926055
 - Miniature pinscher 0.904815
 - Siamese 0.779801
 - Shiba inu 0.770530



KNOWLEDGE DISTILLATION

- Knowledge distillation is the process of utilizing the outputs for a given input from the larger model (a teacher) in the process of training the smaller model (a student)
- The similarities between objects reflected by teacher's output can be used in student's training as generalization hints:
 - Features shared between objects will be promoted
 - The reusability rate of kernels between classes of similar objects should be higher
 - The training process should converge faster and lead to significantly better model



Can we use the outputs from the teacher model as is?

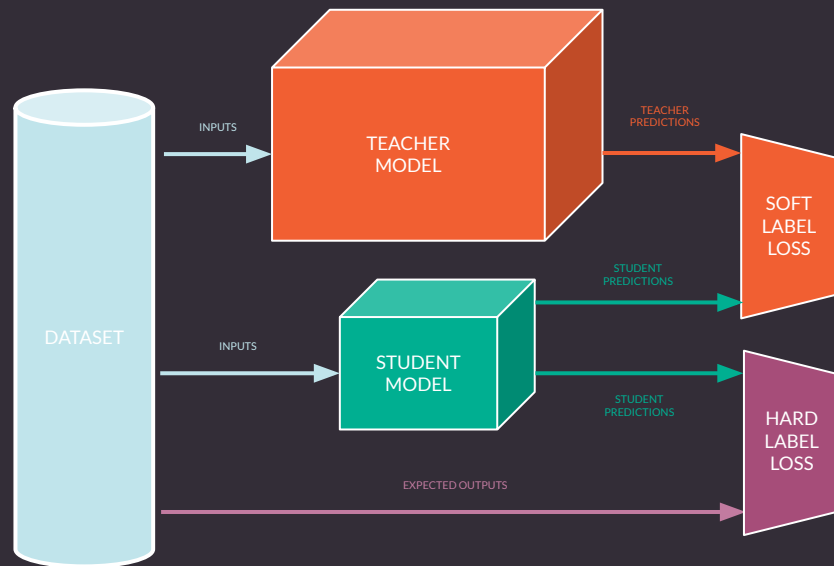
DARK KNOWLEDGE

- The similarities between classes are useful during training, but they can be also very misleading
- Using only or mostly the teacher's knowledge condemns the student to make the same mistakes as the teacher
- That is why students should also get the data from the dataset (libraries/resources)



KNOWLEDGE DISTILLATION

- Knowledge distillation is the process of utilizing the outputs for a given input from the larger model (a teacher) in the process of training the smaller model (a student)
- The similarities between objects reflected by teacher's output can be used in student's training as generalization hints:
 - Features shared between objects will be promoted
 - The reusability rate of kernels between classes of similar objects should be higher
 - The training process should converge faster and lead to significantly better model
- Teacher's knowledge may also slightly reduce errors coming from the dataset



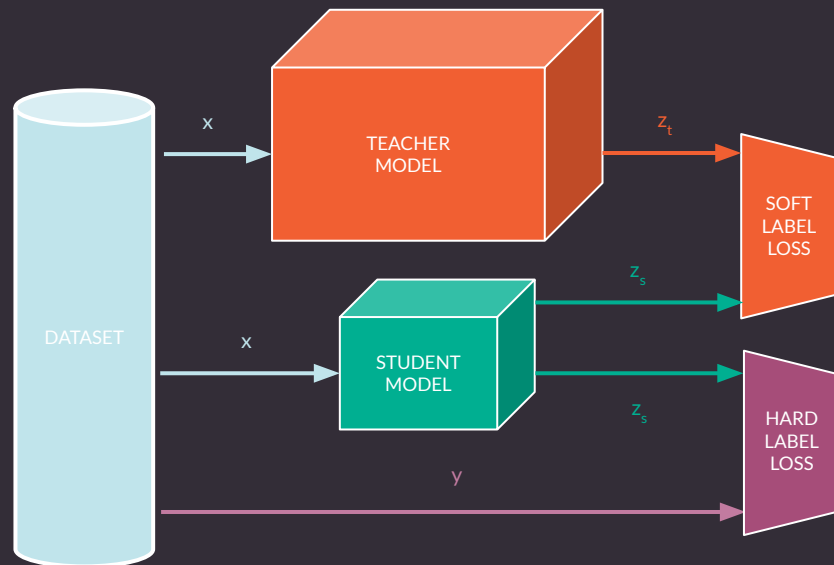
KNOWLEDGE DISTILLATION

- In classic knowledge distillation, the loss is computed in the following way:

$$L(x; W) = \alpha * H(y, \sigma(z_s; T = 1)) + \beta * H(\sigma(z_t; T = \tau), \sigma(z_s; T = \tau))$$

- Where:

- $L(x; W)$ - loss function for input x and current student weights W
- α - ground truth cross entropy loss coefficient
- β - teacher cross entropy loss coefficient (usually $\beta = 1 - \alpha$)
- σ - softmax function with temperature T
- z_s - student output vector
- z_t - teacher output vector
- τ - temperature for distillation soft labels, the higher the value, the richer in information the soft-labels distribution will be
- $H(y, \sigma(z_s; T = 1))$ - hard label loss
- $H(\sigma(z_t; T = \tau), \sigma(z_s; T = \tau))$ - soft label loss



EFFICIENT AND LIGHTWEIGHT DNN RUNTIMES



NEURAL NETWORK INTERPRETER

- **TensorFlow Lite**

- Repository:
<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite>
- Documentation: <https://www.tensorflow.org/lite>
- Very small library size (~1MB default, ~300kB for most popular operations)
- Very small and efficient model format (flatbuffers)
- Highly flexible - allows:
 - Enabling/disabling support for ops
 - Easy implementation of new ops
 - Easy delegation of ops to custom accelerators
- They use existing model formats and iterate over layers to process data
- They have per-layer optimized kernels
- Very flexible, with simple model replacement
- Less opportunities for interlayer or graph-wise optimizations



NEURAL NETWORK COMPILERS

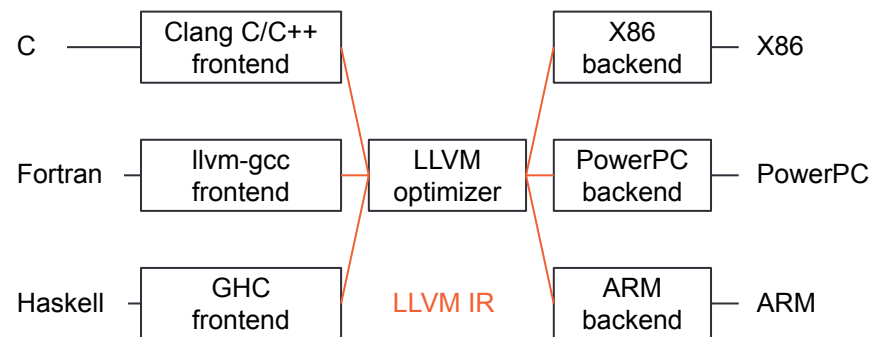
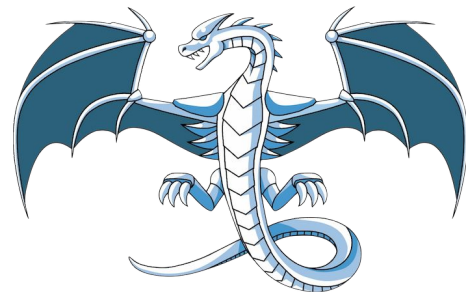
- **Apache TVM (Tensor Virtual Machine)**

- Homepage: <https://tvm.apache.org/>
- Repository: <https://github.com/apache/tvm>

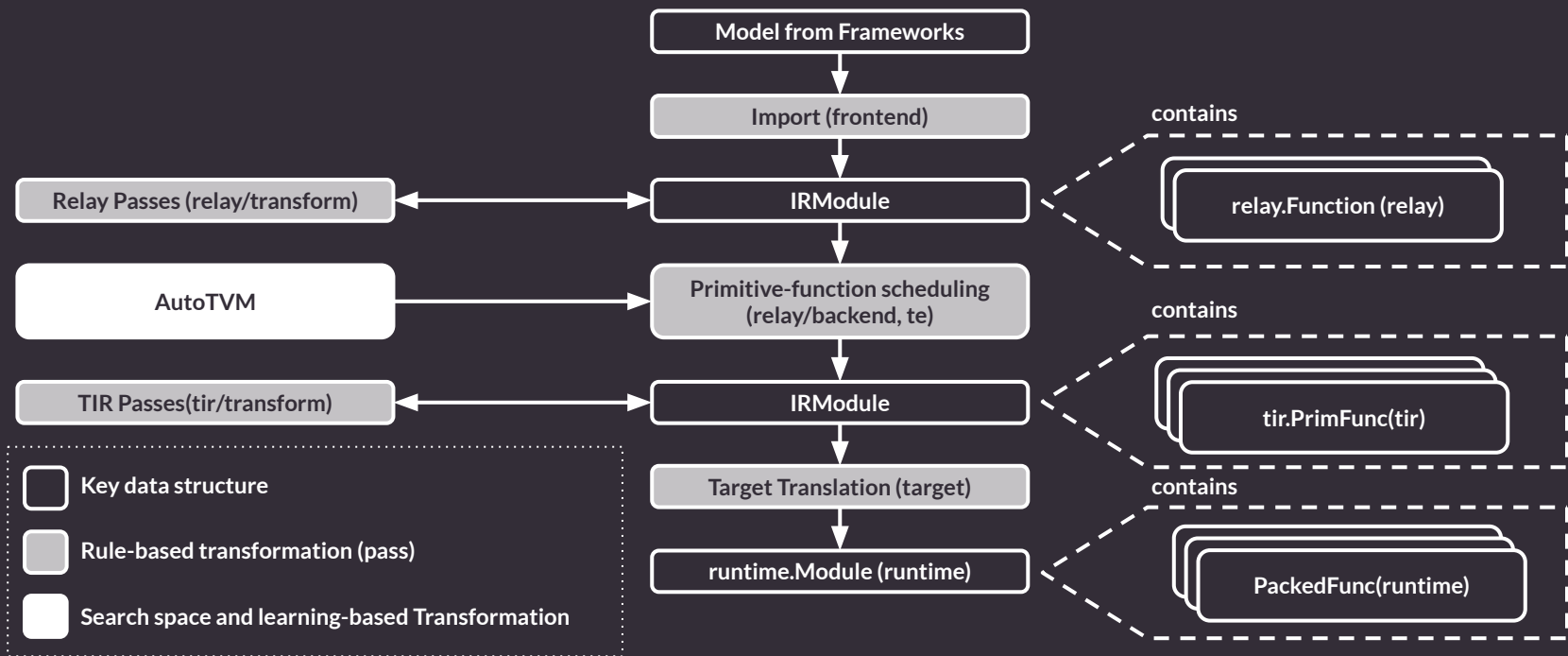
- **OpenXLA IREE**

- Homepage: <https://openxla.github.io/iree/>
- Repository: <https://github.com/openxla/iree>

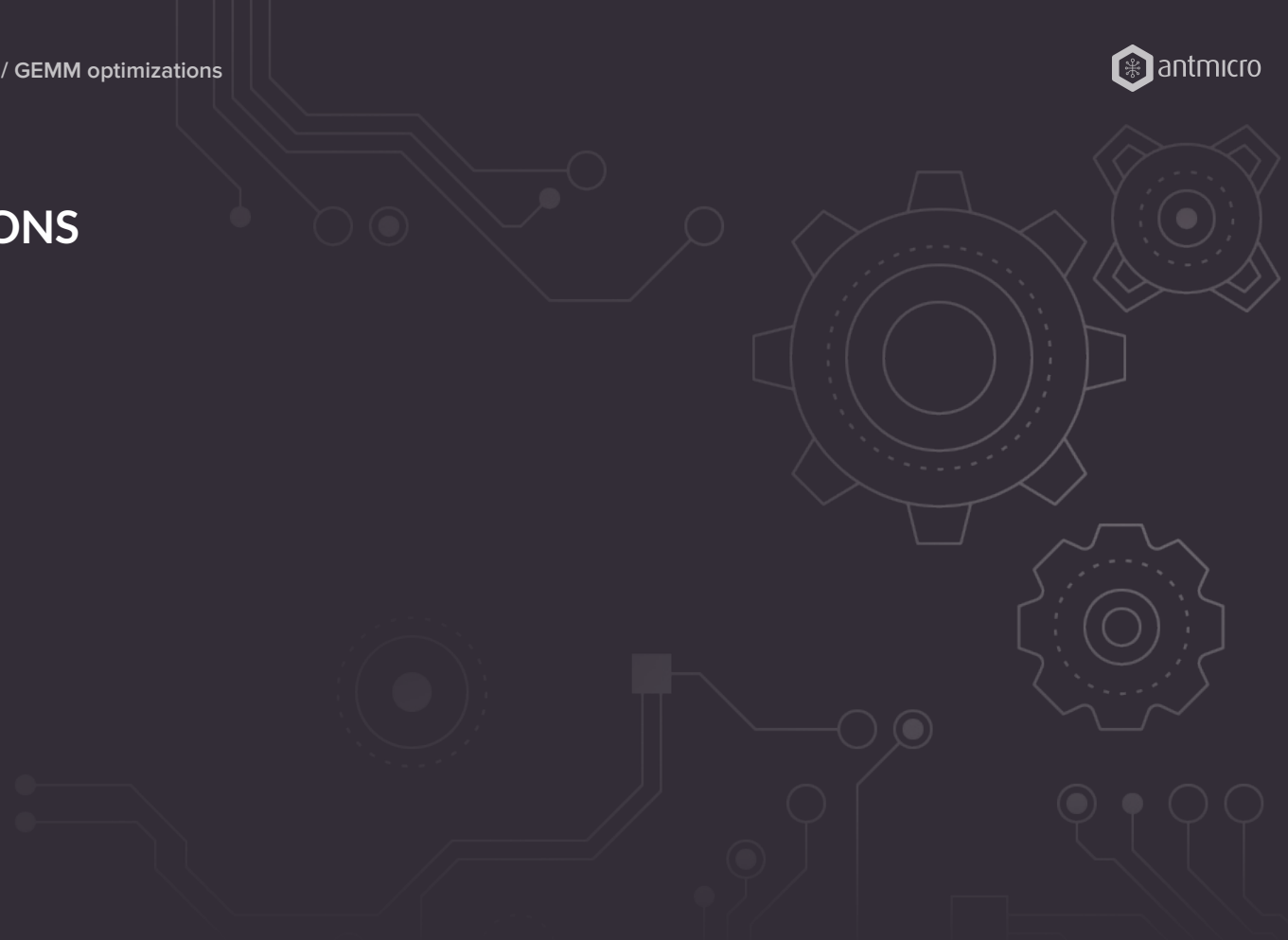
- They convert the model to a set of functions that are later compiled to a binary form, either an application, or shared library designed specifically for a given model and hardware
- Both are based on LLVM project, where models are converted to functional Intermediate Representation, which is later subjected to optimizations



TVM COMPILATION FLOW



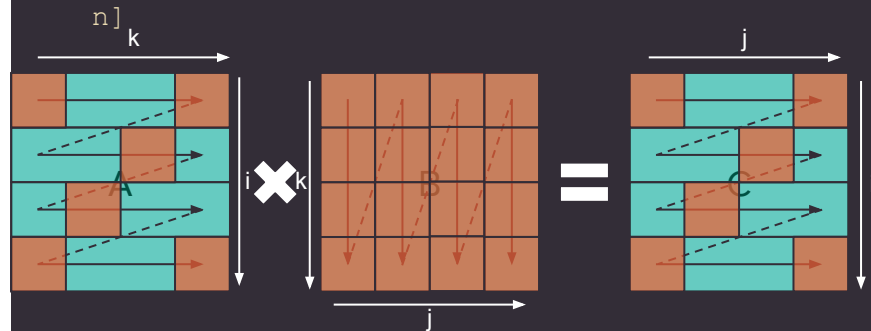
GEMM OPTIMIZATIONS



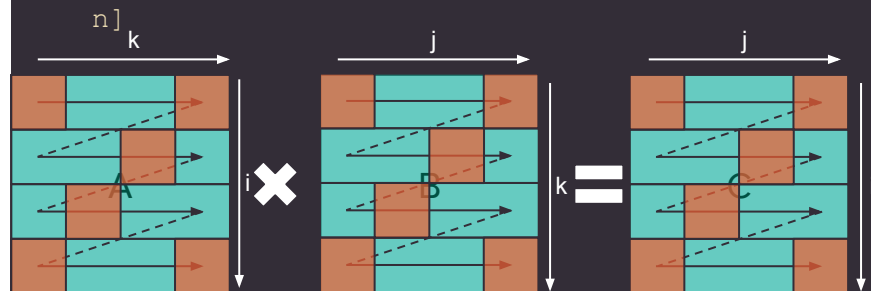
POSSIBLE GEMM OPTIMIZATIONS

- Loop permutations

```
for m in range(M):
    for n in range(N):
        for k in range(K):
            C[m, n] += A[m, k] * B[k,
```

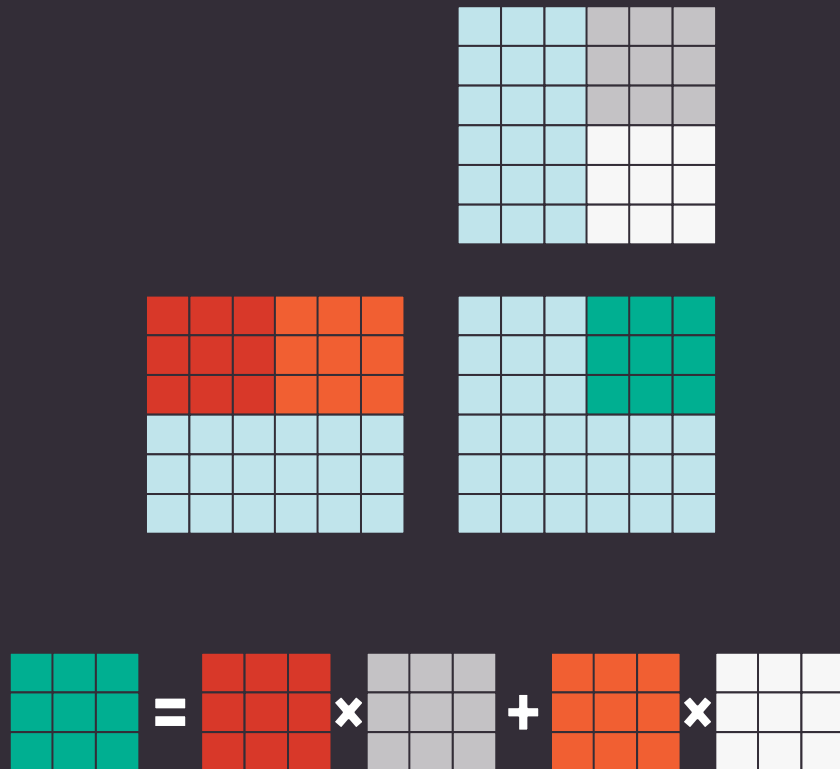


```
for m in range(M):
    for k in range(K):
        for n in range(N):
            C[m, n] += A[m, k] * B[k,
```



POSSIBLE GEMM OPTIMIZATIONS

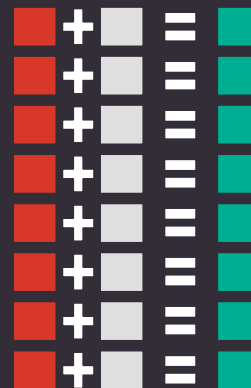
- Loop permutations
- Blocking/tiling



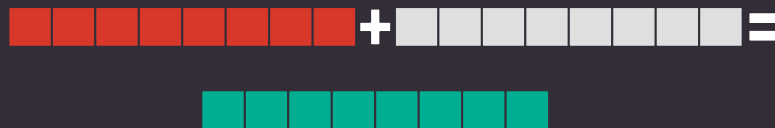
POSSIBLE GEMM OPTIMIZATIONS

- Loop permutations
- Blocking/tiling
- Vectorization:
 - x86 - AVX2, AVX512, ... extensions
 - ARM - Neon, SVE, ... ([CMSIS-NN library](#))
 - RISC-V - V Extensions ([MURISCV-NN library](#))

8 CPU cycles

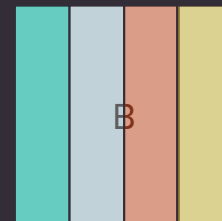
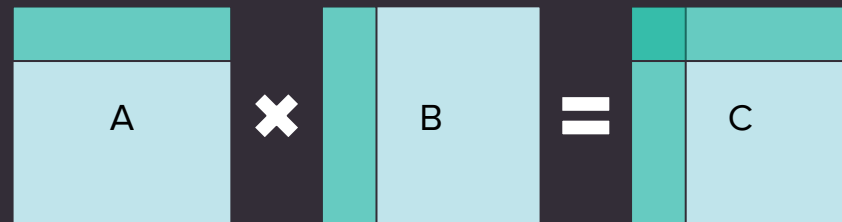


AVX: 1 CPU cycle



POSSIBLE GEMM OPTIMIZATIONS

- Loop permutations
- Blocking/tiling
- Vectorization:
 - x86 - AVX2, AVX512, ... extensions
 - ARM - Neon, SVE, ... ([CMSIS-NN library](#))
 - RISC-V - V Extensions ([MURISCV-NN library](#))
- Array packing



Typical B layout

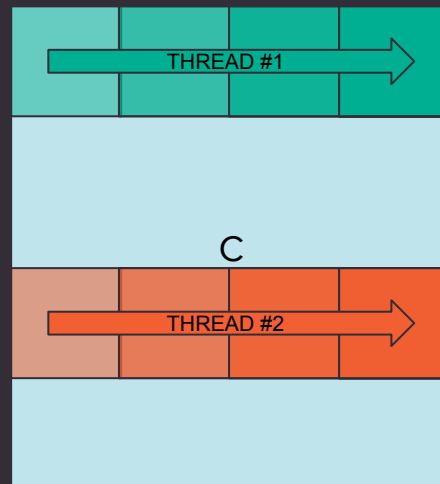


Array packed B layout



POSSIBLE GEMM OPTIMIZATIONS

- Loop permutations
- Blocking/tiling
- Vectorization:
 - x86 - AVX2, AVX512, ... extensions
 - ARM - Neon, SVE, ... ([CMSIS-NN library](#))
 - RISC-V - V Extensions ([MURISCV-NN library](#))
- Array packing
- Threading



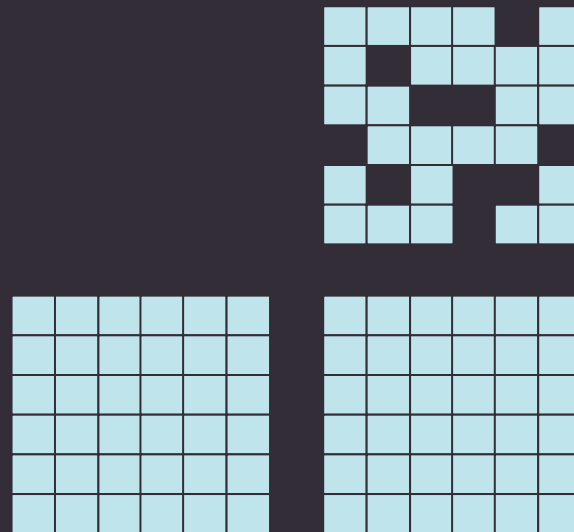
POSSIBLE GEMM OPTIMIZATIONS

- Loop permutations
- Blocking/tiling
- Vectorization:
 - x86 - AVX2, AVX512, ... extensions
 - ARM - Neon, SVE, ... ([CMSIS-NN library](#))
 - RISC-V - V Extensions ([MURISCV-NN library](#))
- Array packing
- Threading
- Unrolling

```
for m in range(M):  
    for k in range(K):  
        C[m, 0] += A[m, k] * B[k, 0]  
        C[m, 1] += A[m, k] * B[k, 1]  
        C[m, 2] += A[m, k] * B[k, 2]  
        C[m, 3] += A[m, k] * B[k, 3]  
        C[m, 4] += A[m, k] * B[k, 4]  
        C[m, 5] += A[m, k] * B[k, 5]  
        C[m, 6] += A[m, k] * B[k, 6]  
        C[m, 7] += A[m, k] * B[k, 7]  
        C[m, 8] += A[m, k] * B[k, 8]  
        # ...
```

POSSIBLE GEMM OPTIMIZATIONS

- **Loop permutations**
- **Blocking/tiling**
- **Vectorization:**
 - x86 - AVX2, AVX512, ... extensions
 - ARM - Neon, SVE, ... ([CMSIS-NN library](#))
 - RISC-V - V Extensions ([MURISCV-NN library](#))
- **Array packing**
- **Threading**
- **Unrolling**
- **Sparse matrix multiplication**
 - NVIDIA GPUs - Ampere+ architectures
 - NVIDIA Jetson Orin platforms
- Dead code elimination, constants unfolding, ...

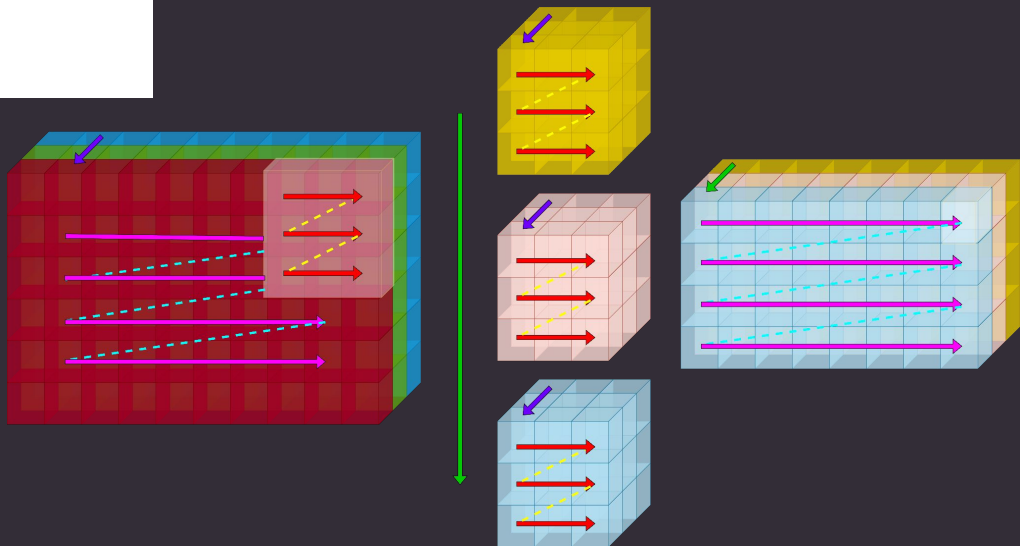


CONVOLUTION



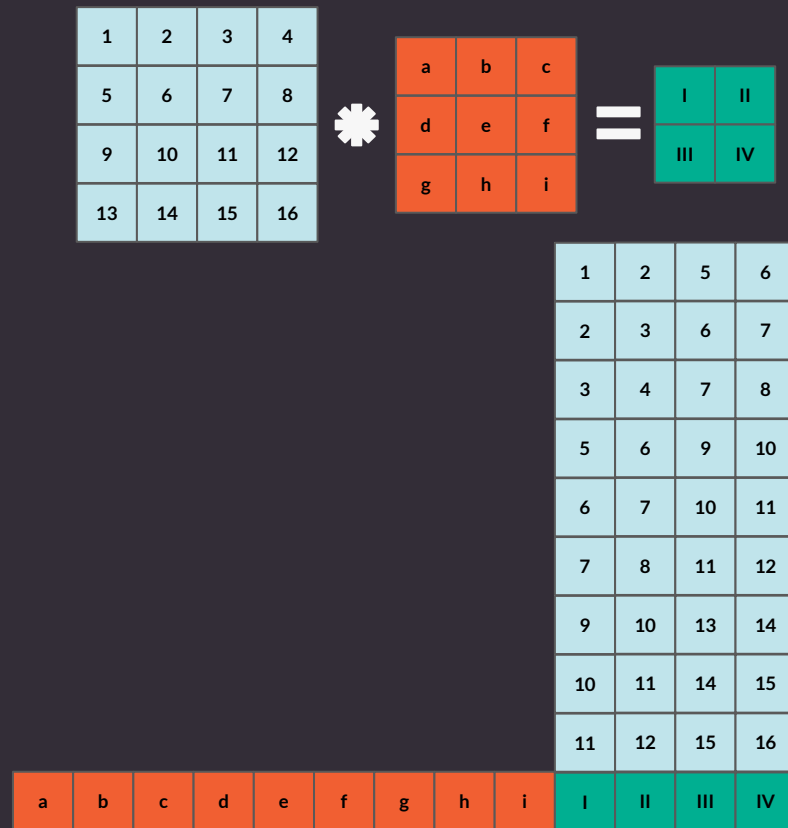
NAIVE CONV2D IMPLEMENTATION

```
for filter in range(num_filters):
    for channel in range(input_channels):
        for out_h in range(output_height):
            for out_w in range(output_width):
                for k_h in range(kernel_height):
                    for k_w in range(kernel_width):
                        output[filter, out_h, out_w] += (
                            kernel[filter, channel, k_h, k_w] * input[channel, out_h + k_h, out_w + k_w]
                        )
```



GEMM-BASED CONV2D IMPLEMENTATION - IM2COL

- Lots of the hardware platforms accelerate GEMM operations
- Also, there are lots of libraries that provide well-optimized implementations of the GEMM
- It is possible to convert the convolution to GEMM
- With the available accelerations for GEMM converting the convolution to GEMM is profitable
- To convert convolutions to GEMM, we need to rearrange the data in feature maps and kernels
- The algorithm for this rearrangement is called im2col
- Created matrices introduce significant amount of redundancy, but the execution time decrease compensates the memory overhead

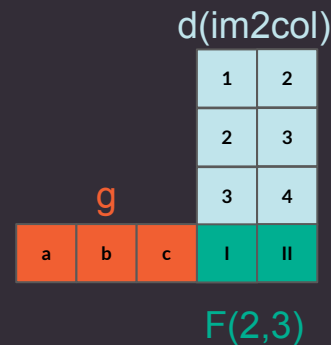
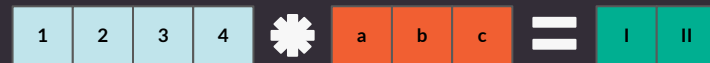


WINOGRAD CONVOLUTION

- The minimal filtering algorithm for computing **m** outputs with an **r**-tap FIR filter $F(m,r)$ requires $m+r-1$ multiplications
- E.g. for $F(2, 3)$ (2-element output, 3-element filter) we have 4-element input
- Standard algorithm uses **2*3=6 multiplications**
- Using modified Toom-Cook algorithm, we can compute convolution as follows:

$$F(2,3) = \begin{bmatrix} d_1 & d_2 & d_3 \\ d_2 & d_3 & d_4 \end{bmatrix} \begin{bmatrix} g_a \\ g_b \\ g_c \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}$$

- This solution gives **4 multiplications instead of 6 - 1.5 speedup**
- The 1D $F(m,r)$ and $F(n,s)$ algorithms can be nested to form minimal 2D algorithms for computing $m*n$ outputs with an $r*s$ filter, requiring $(m+r-1)*(n+s-1)$ multiplications
- Original convolution of 4x4 matrix by 3x3 kernel (to obtain 2x2 result) requires $3*3*2*2=36$ multiplications
- Winograd implementation requires **$(3+2-1)*(3+2-1)=16$ multiplications!**
- This gives us $36/16=2.25$ speedup!**
- Winograd can be used for convolutions with small kernels (3x3, 5x5, 7x7)



$$\begin{aligned} m_1 &= \left[\begin{array}{|c|} \hline 1 \\ \hline \end{array} - \begin{array}{|c|} \hline 3 \\ \hline \end{array} \right] \times \begin{array}{|c|} \hline a \\ \hline \end{array} \\ m_2 &= \left[\begin{array}{|c|} \hline 2 \\ \hline \end{array} + \begin{array}{|c|} \hline 3 \\ \hline \end{array} \right] \times \left[\begin{array}{|c|} \hline a \\ \hline \end{array} + \begin{array}{|c|} \hline b \\ \hline \end{array} + \begin{array}{|c|} \hline c \\ \hline \end{array} \right] \div \begin{array}{|c|} \hline 2 \\ \hline \end{array} \\ m_3 &= \left[\begin{array}{|c|} \hline 3 \\ \hline \end{array} - \begin{array}{|c|} \hline 2 \\ \hline \end{array} \right] \times \left[\begin{array}{|c|} \hline a \\ \hline \end{array} - \begin{array}{|c|} \hline b \\ \hline \end{array} + \begin{array}{|c|} \hline c \\ \hline \end{array} \right] \div \begin{array}{|c|} \hline 2 \\ \hline \end{array} \\ m_4 &= \left[\begin{array}{|c|} \hline 2 \\ \hline \end{array} - \begin{array}{|c|} \hline 4 \\ \hline \end{array} \right] \times \begin{array}{|c|} \hline b \\ \hline \end{array} \end{aligned}$$

To compute Precomputed

NEURAL NETWORK DEPLOYMENT ECOSYSTEM



NEURAL NETWORK DEPLOYMENT ECOSYSTEM



TRAINING



ONNX



Neural Network Intelligence

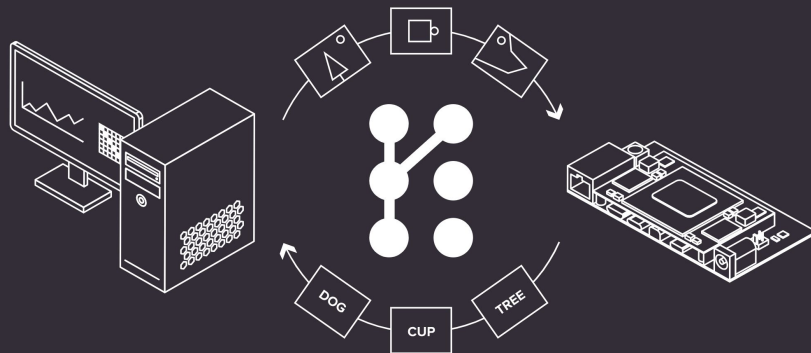


OPTIMIZATION + RUNTIME

How can we utilize various optimizations sparsely scattered across frameworks and runtimes?

KENNING

- Repository: github.com/antmicro/kenning
- Documentation: antmicro.github.io/kenning
- Kenning is a Python library for implementing pipelines for neural network optimization and deployment
- It aims towards providing wrappers for neural network deployment steps that can be seamlessly combined into pipelines regardless of underlying machine learning frameworks and compilers
- It also provides a consistent means for benchmarking models after applying certain optimizations and compilation on target platform directly on hardware platform



KENNING FLOW EXAMPLE

- Model: MobileNetV2
- Dataset: Pet Dataset for dogs and cats breeds classification
- Optimizations:
 - Full INT8 quantization with TensorFlow Lite
 - Compilation of model for Jetson AGX Orin device:
 - Target - x86 CPU with AVX2 vector extensions
- Runtime - execution using TVM-compiled model

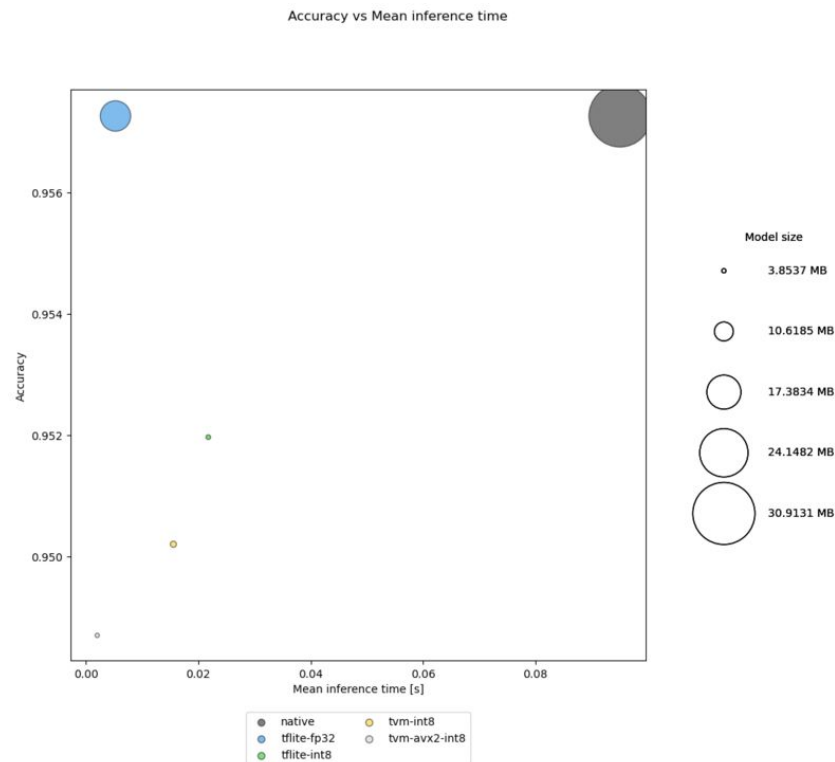
Other possible configuration:

- Target host - CPU used to execute the model
 - `llvm -mtriple=aarch64-linux-gnu`
- Target - GPU with CUDA cores, compute capability 8.7 and CUDNN/CUBLAS execution
 - `cuda -arch=sm_87 -libs=cudnn,cublas`

```
{
  "model_wrapper": {
    "type": "kenning.modelwrappers.TensorFlowPetDatasetMobileNetV2",
    "parameters": {
      "model_name": "mobilenetv2",
      "model_path": "./tensorflow_pet_dataset_mobilenetv2.h5"
    }
  },
  "dataset": {
    "type": "kenning.datasets.pet_dataset.PetDataset",
    "parameters": {
      "dataset_root": "./build/PetDataset"
    }
  },
  "optimizers": [
    {
      "type": "kenning.optimizers.tflite.TFLiteCompiler",
      "parameters": {
        "target": "int8",
        "compiled_model_path": "./build/int8.tflite",
        "inference_input_type": "int8",
        "inference_output_type": "int8"
      }
    },
    {
      "type": "kenning.optimizers.tvm.TVMCompiler",
      "parameters": {
        "target": "llvm -mcpu=core-avx2",
        "opt_level": 3,
        "conv2d_data_layout": "NCHW",
        "compiled_model_path": "./build/int8_tvm.tar"
      }
    }
  ],
  "runtime": {
    "type": "kenning.runtimes.tvm.TVMRuntime",
    "parameters": {
      "save_model_path": "./build/int8_tvm.tar"
    }
  }
}
```

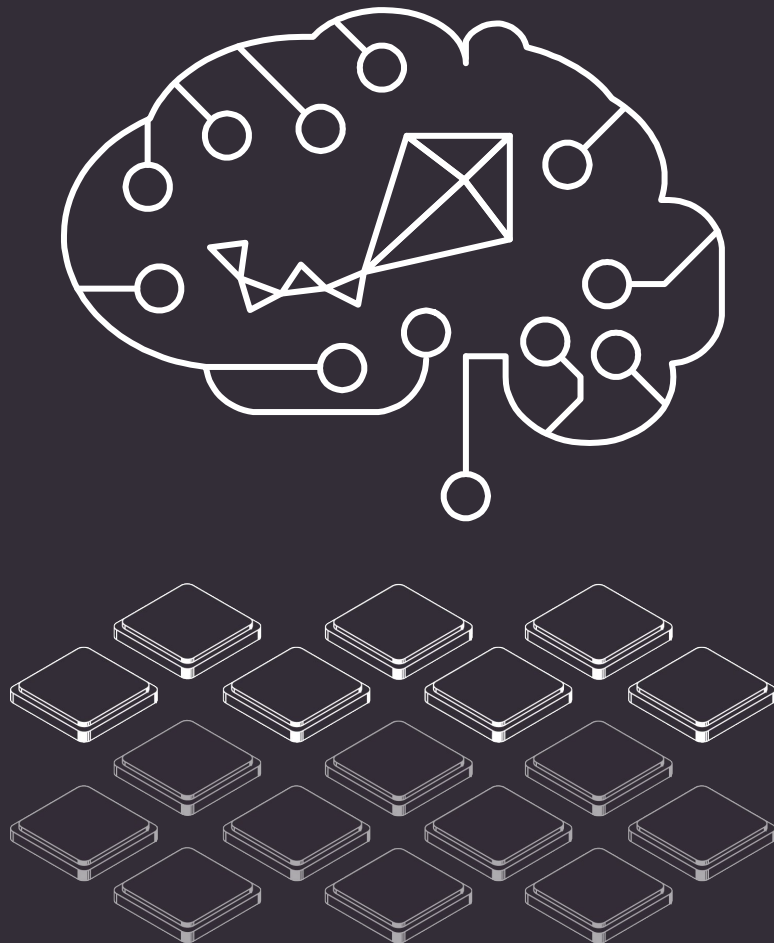

CPU DEPLOYMENT WITH VECTOR EXTENSIONS (AVX2)

Model Total params: 4,164,965 Trainable params: 1,906,981	Model accuracy	Speedup in comparison to native framework	Model size reduction in comparison to native framework
Native	0.805669119651131	1.00	1.00
TFLite FP32	0.805669119651131	2.12	1.97
TFLite INT8	0.775688198419187	0.39	7.02
TVM INT8 (TFLite input)	0.775688198419187	5.58	3.43
TVM INT8 with vector extensions (TFLite input)	0.775688198419187	16.07	5.85



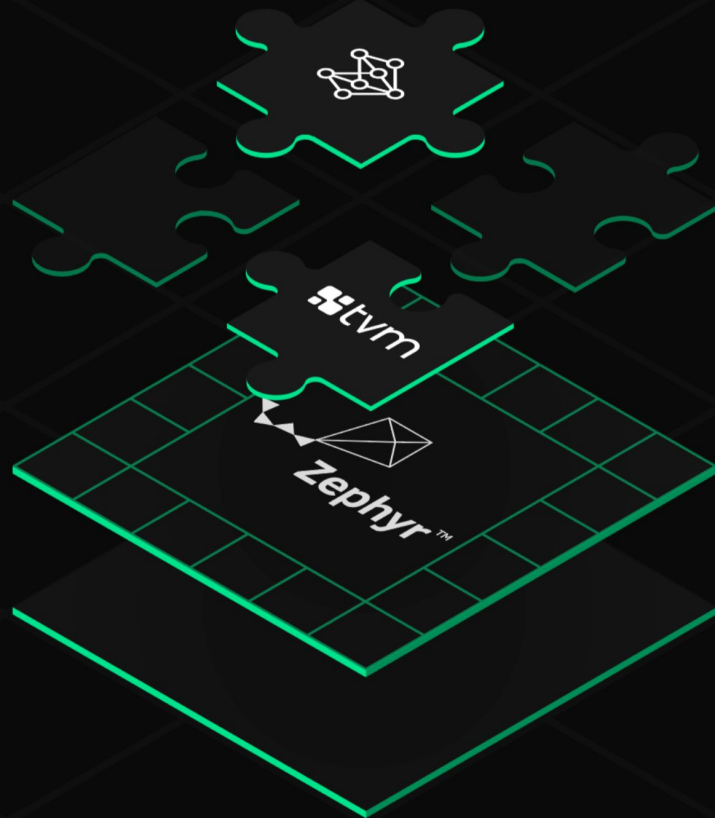
KENNING - SUPPORTED PLATFORMS

- Supported hardware running Linux (Python API):
 - **CPUs:** x86_64, ARM Cortex A, RISC-V (HiFive Unmatched), ...
 - **GPUs/eGPUs:** NVIDIA GPUs, NVIDIA Jetson platforms (Jetson Nano, Jetson AGX Xavier, Jetson AGX Orin)
 - **TPUs:** Google Coral
 - ...
- **Bare-metal CPUs:**
 - [Kenning Bare Metal IREE Runtime](#)
- **Zephyr-capable CPUs:**
 - [Kenning Zephyr Runtime](#)



KENNING + ZEPHYR

- <https://github.com/antmicro/kenning-zephyr-runtime>
- A unified API for evaluating and deploying neural networks on platforms supported by Zephyr (700+ boards)
- Provides:
 - Kenning inference library - a configurable library that lets you pick a specific implementation of the runtime and use it to load the model and run it on target device
 - Evaluation app - a Zephyr application for evaluating and benchmarking models on target device
- Supported runtimes:
 - TFLite Micro - <https://github.com/tensorflow/tflite-micro>
 - microTVM
 - IREE - <https://github.com/iree-org/iree>



SAMPLE INFERENCE LOOP

```
// ...
status_t status = STATUS_OK;
uint8_t *model_output = NULL;
size_t model_output_size = 0;

// initialize model
status = model_init();
RETURN_ON_ERROR(status, status);
// load model structure
status = model_load_struct((uint8_t *)&model_struct, sizeof(MlModel));
RETURN_ON_ERROR(status, status);
// load model weights
status = model_load_weights(model_data, model_data_len);
RETURN_ON_ERROR(status, status);
// allocate buffer for output;
model_get_output_size(&model_output_size);
model_output = malloc(model_output_size);

// inference loop
for (size_t batch_index = 0; batch_index < sizeof(data) / sizeof(data[0]); ++batch_index)
{
    status = model_load_input((uint8_t *)data[batch_index], sizeof(data[0]));
    RETURN_ON_ERROR(status, status);

    status = model_run();
    RETURN_ON_ERROR(status, status);

    status = model_get_output(model_output_size, model_output, NULL);
    RETURN_ON_ERROR(status, status);

    format_output(output_str, sizeof(output_str), model_output);
    LOG_INF("model output: %s", output_str);
}
```

SAME APP, DIFFERENT MODEL EXECUTION

```
west build -p always -b stm32f746g_disco app -- -DEXTRA_CONF_FILE=tflite.conf
```

SAME APP, DIFFERENT MODEL EXECUTION

```
west build -p always -b stm32f746g_disco app -- -DEXTRA_CONF_FILE=tvm.conf
```

SAME APP, DIFFERENT HARDWARE

```
west build -p always -b nrf52840dongle app -- -DEXTRA_CONF_FILE=tvm.conf
```

What if hardware is not available?

RENODE

- Repository: <https://github.com/renode/renode>
- Antmicro's open source emulation framework allowing software developers to build, run and test software w/o hardware
- Multinode, deterministic, built with automation and testing in mind
- Configuration-oriented platform definitions
- Wide RISC-V support, with easy prototyping of custom instructions
 - Also ARM, Power, SPARC, ...
- Support for writing simulation in Python
- Read more at about.renode.io

RE




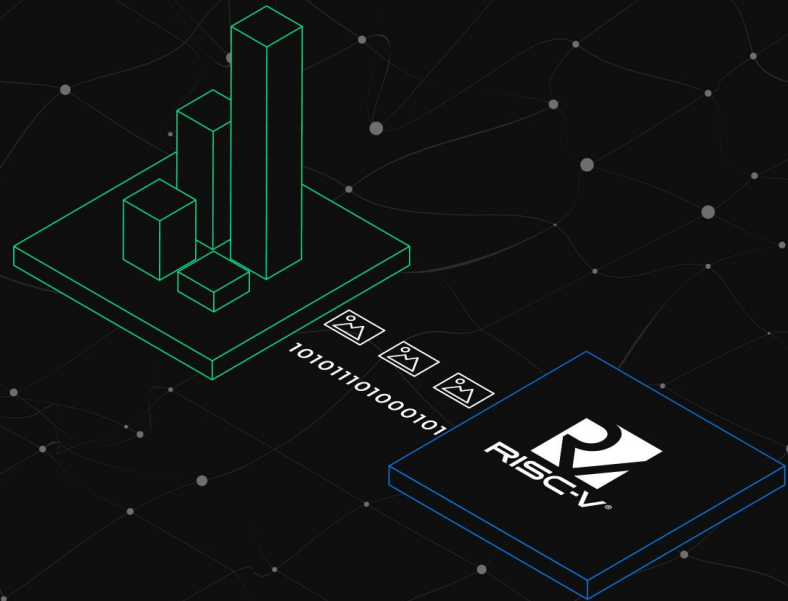
KENNING/RENODE SIMULATION

- Kenning and Renode simulation brings various benefits:
 - Debugging and profiling on simulated device
 - Testing whether AI application will run on target platform without hardware
 - Continuous Integration pipelines checking actual application execution on simulated device with SoC, sensors and other peripherals, testable with Robot framework
 - Co-development of hardware design and inference library in a simulated environment with co-simulation of accelerator design
 - Testing of runtime implementations on various platforms to check for hardware coverage

 Kenning

RENODE™

 antmicro



Nodes browser

Search

- › datasets

- › modelwrappers

- › optimizers

- › protocols

- › runtimes

MagicWandDataset
 window_size 128
 window_shift 128
 noise_level 20
 dataset_root /build/MagicWandDataset
 inference_batch_size 1
 download_dataset
 force_download_dataset
 external_calibration_dataset
 external_calibration_dataset
 split_fraction_test 0.200
 split_fraction_val 0.000
 split_seed 1234
 DataSet

ModelWrapper

< window_size 128 >

model_path:

kenning://models/classification/tr

model_name:

model_name

Dataset Model

ModelWrapper



```


iree-compile
model_framework: keras
backend: llvm-cpu
compiler_args: -iree-llvm-debug-symbols=false
compiled_model_path: ./build/tfite-magic-wand.vmlb
location:
host:
input_model: input_model.npy
compiled_model: compiled_model.npy

```

```

RenodeRuntime
runtime_binary_path:
  kenning://renode/springbok/free.
platform_resc_path:
  gh://intrinco/kenning-bare-meta
resc_dependencies:
  gh://intrinco/kenning-bare-meta
pool_start_commands:
  sysbus.vec_controlblock WriteCio.
runtime_log_uart:
  runtime_log_uart
runtime_log_init_mig:
  Runtime started
  disable_profiler
profiler_dump_path:
  build/profiler.dump
  profiler_interval_step 10.000 >
sensor:
  sensor
  batches_count 10 >
  disable_performance_measurement
Model
ModelWrapper

```

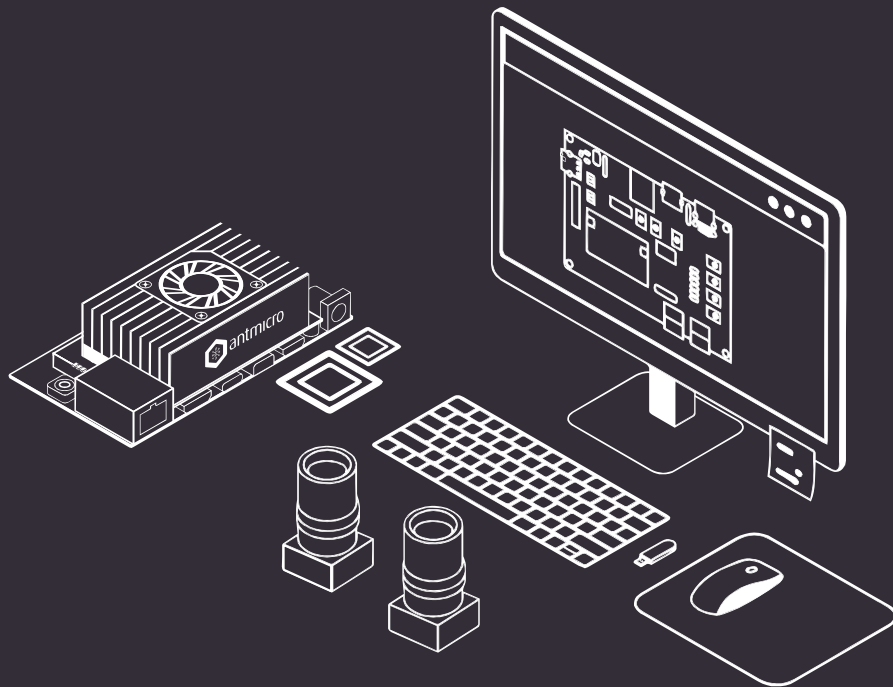


Terminal Kenning Terminal

Clear terminal

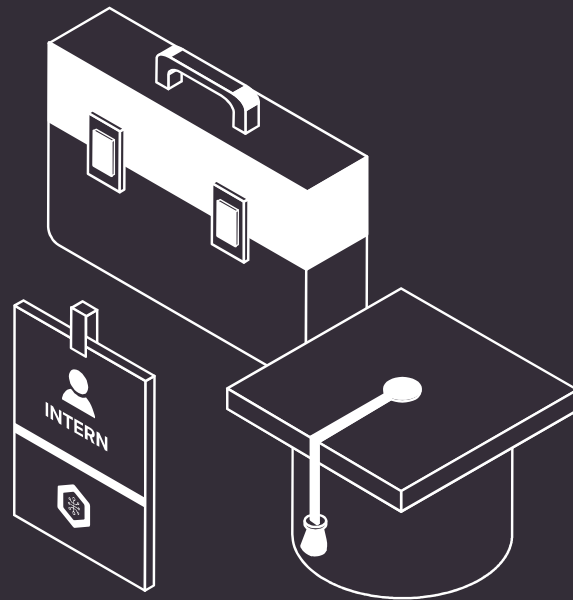
```
[2024-03-08 22:57:34 kenning resource_manager.py:288] [INFO] Using cached: /home/grzegorz/.kenning/sim/config/platforms/springbok.repl
[2024-03-08 22:57:35 kenning resource_manager.py:586] [WARNING] Cannot verify /home/grzegorz/.kenning/sim/config/infrastructure/SpringbokRiscV32.cs checksum
[2024-03-08 22:57:35 kenning resource_manager.py:288] [INFO] Using cached: /home/grzegorz/.kenning/sim/config/infrastructure/SpringbokRiscV32.cs
[2024-03-08 22:57:44 kenning resource_manager.py:288] [INFO] Using cached: /home/grzegorz/.kenning/models/classification/magic_wand.h5
[2024-03-08 22:57:44 kenning resource_manager.py:288] [INFO] Using cached: /home/grzegorz/.kenning/models/classification/magic_wand.h5.json
[2024-03-08 22:57:44 kenning resource_manager.py:288] [INFO] Using cached: /home/grzegorz/.kenning/renode/springbok/iree_runtime
[2024-03-08 22:57:44 kenning resource_manager.py:586] [WARNING] Cannot verify /home/grzegorz/.kenning/sim/config/springbok.resc checksum
[2024-03-08 22:57:44 kenning resource_manager.py:288] [INFO] Using cached: /home/grzegorz/.kenning/sim/config/springbok.resc
[2024-03-08 22:57:45 kenning resource_manager.py:586] [WARNING] Cannot verify /home/grzegorz/.kenning/sim/config/platforms/springbok.repl checksum
[2024-03-08 22:57:45 kenning resource_manager.py:288] [INFO] Using cached: /home/grzegorz/.kenning/sim/config/platforms/springbok.repl
[2024-03-08 22:57:45 kenning resource_manager.py:586] [WARNING] Cannot verify /home/grzegorz/.kenning/sim/config/infrastructure/SpringbokRiscV32.cs checksum
[2024-03-08 22:57:45 kenning resource_manager.py:288] [INFO] Using cached: /home/grzegorz/.kenning/sim/config/infrastructure/SpringbokRiscV32.cs
```

ENGINEERING INTERNSHIPS



LOOKING FOR INTERNS

- Engineering internships
 - ASIC/SoC Design
 - Digital design/FPGA
 - Hardware design
 - Software
 - AI
 - C#
 - C / Rust
 - Cloud
 - Backend
 - Frontend



ENGINEERING INTERNSHIPS

Full list on our careers website



<https://careers.antmicro.com/jobs/>



**THANK YOU
FOR YOUR ATTENTION!**

