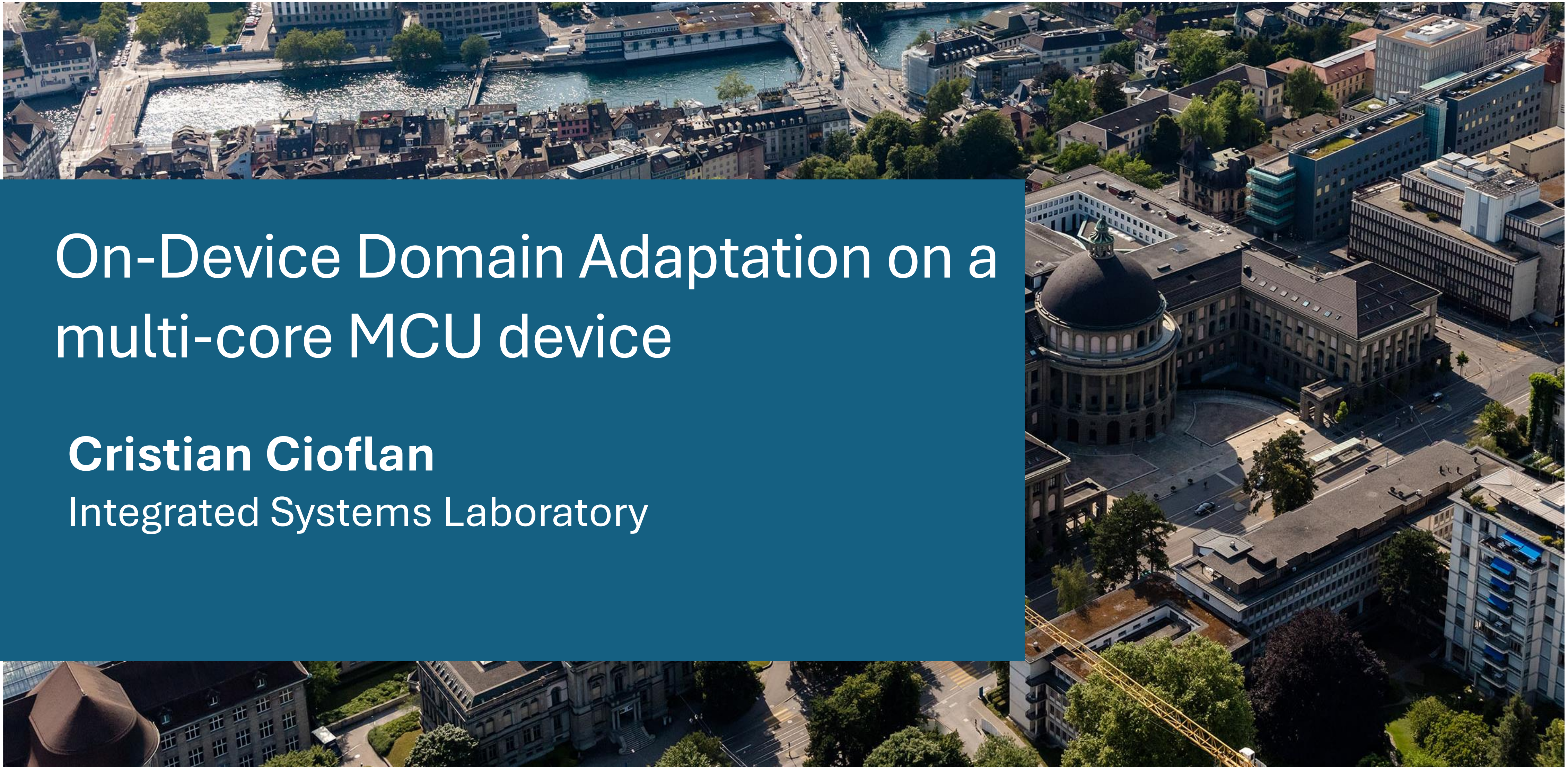
ETH zürich

PULP
Parallel Ultra Low Power

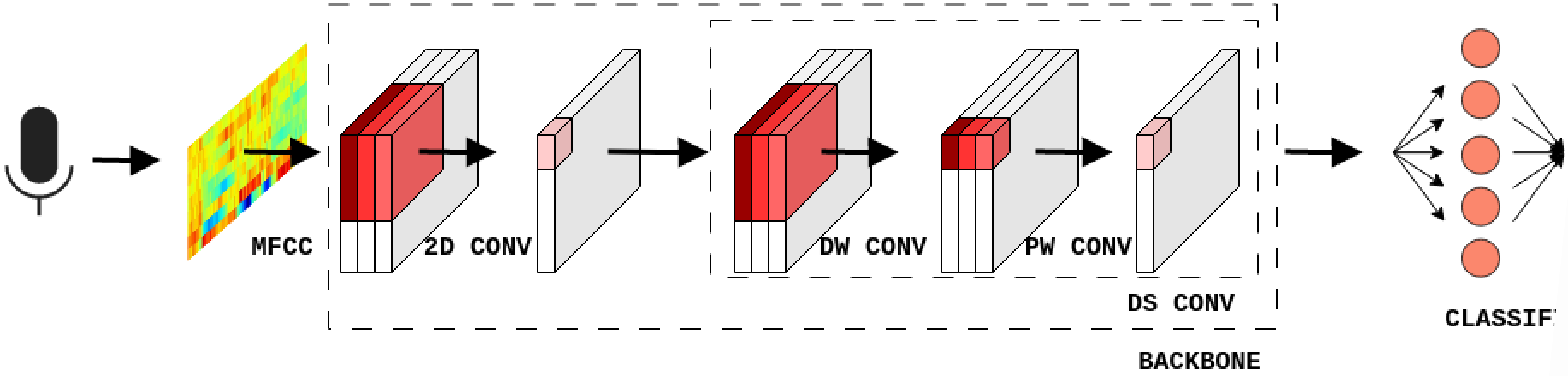


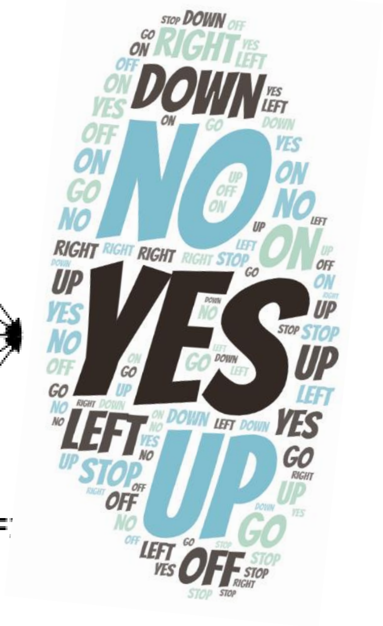
On-Device Domain Adaptation on a multi-core MCU device

Cristian Cioflan
Integrated Systems Laboratory

Keyword spotting – a case study for ODCL

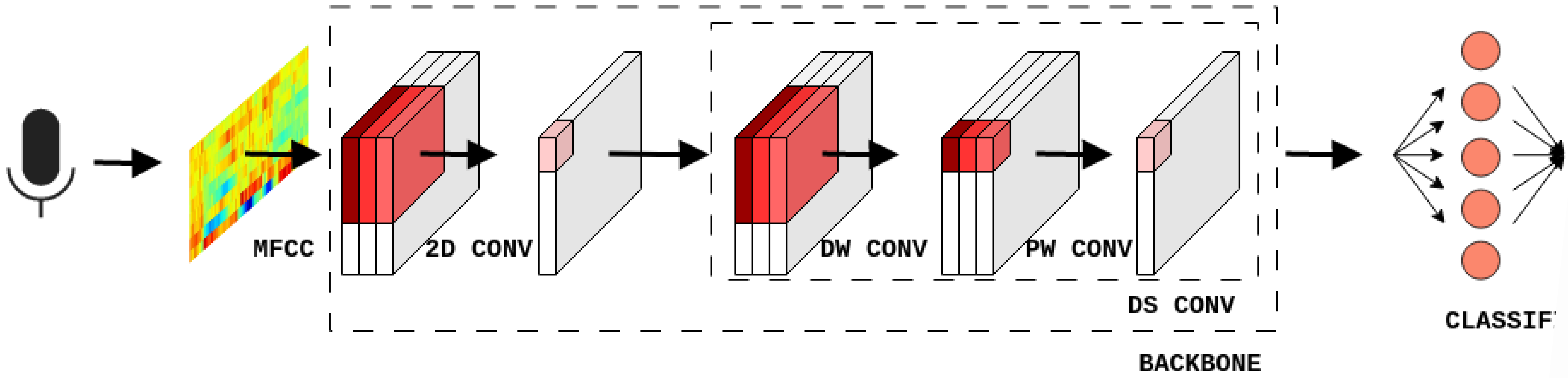
- **Process** an audio signal
- **Recognize** a **target** word from a **predefined set**

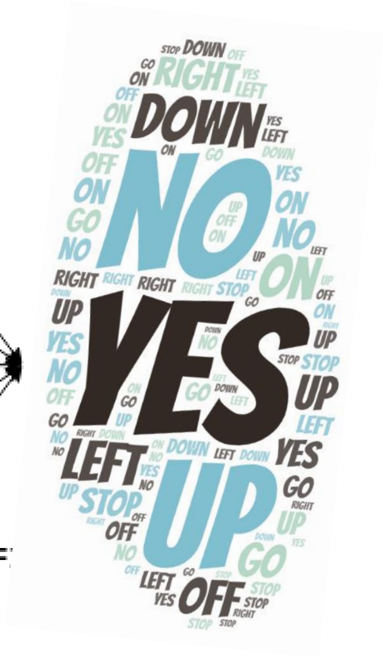




Keyword spotting – **92% in clean conditions**

- **Process** an audio signal
- **Recognize** a **target** word from a **predefined set**

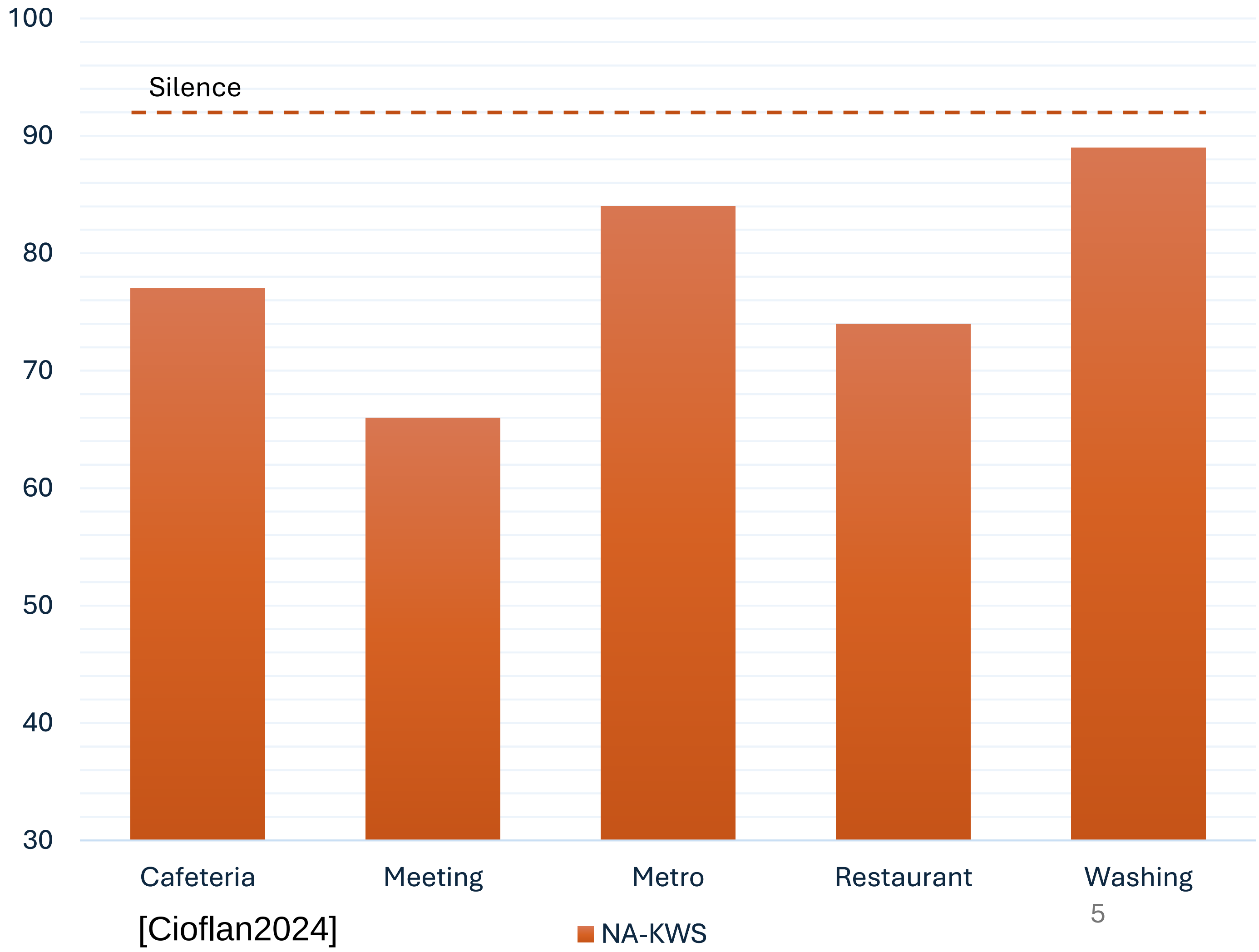




Quiet rooms are not the norm...

Can a KWS system still recognize the words?

- **Noise-Aware KWS**
  - Noise-augmented KWS at (pre)training time



[Cioflan2024]

ETH zürich

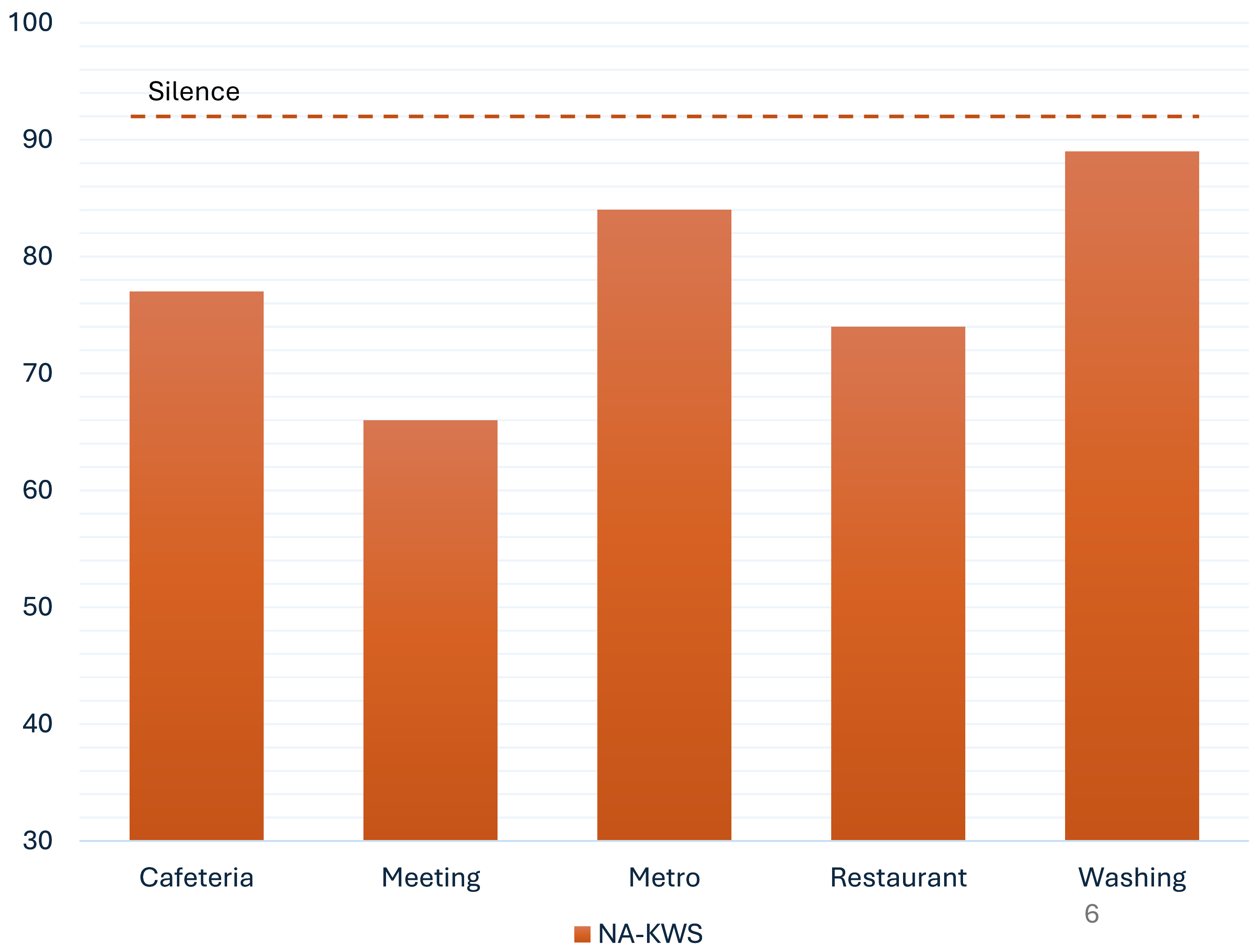
PULP Parallel Ultra Low Power

Can a KWS system still recognize the words?

- **Noise-Aware KWS**
  - Noise-augmented KWS at (pre)training time

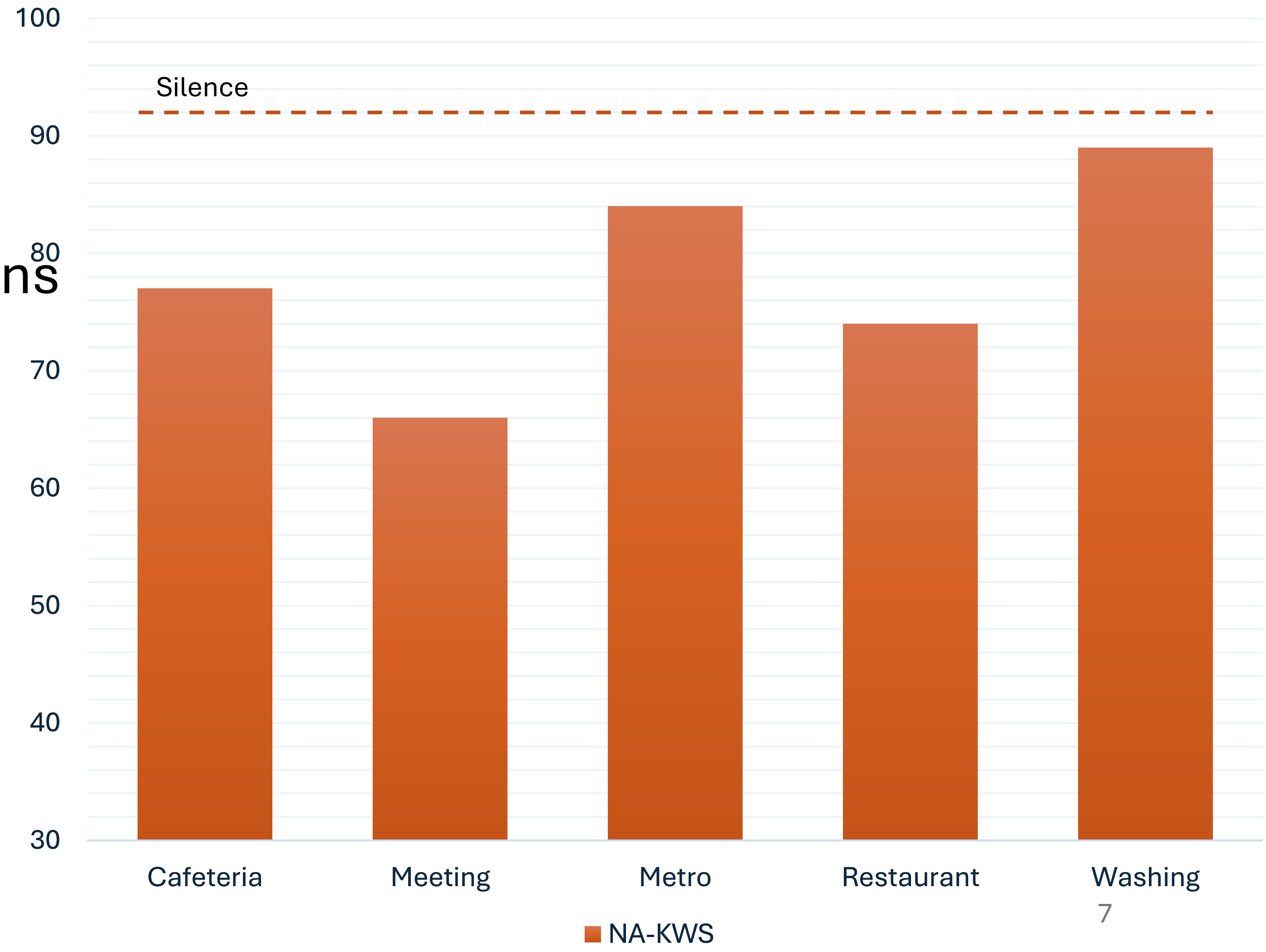
Terminology

pretraining = training (on the server)
pretraining ≠ OD(C)L
OD(C)L = adaptation = fine-tuning (at the edge)



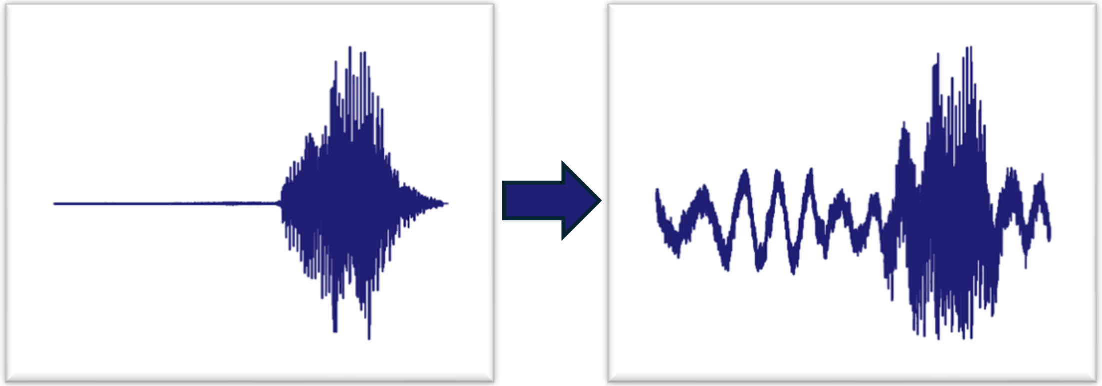
Can a KWS system still recognize the words?

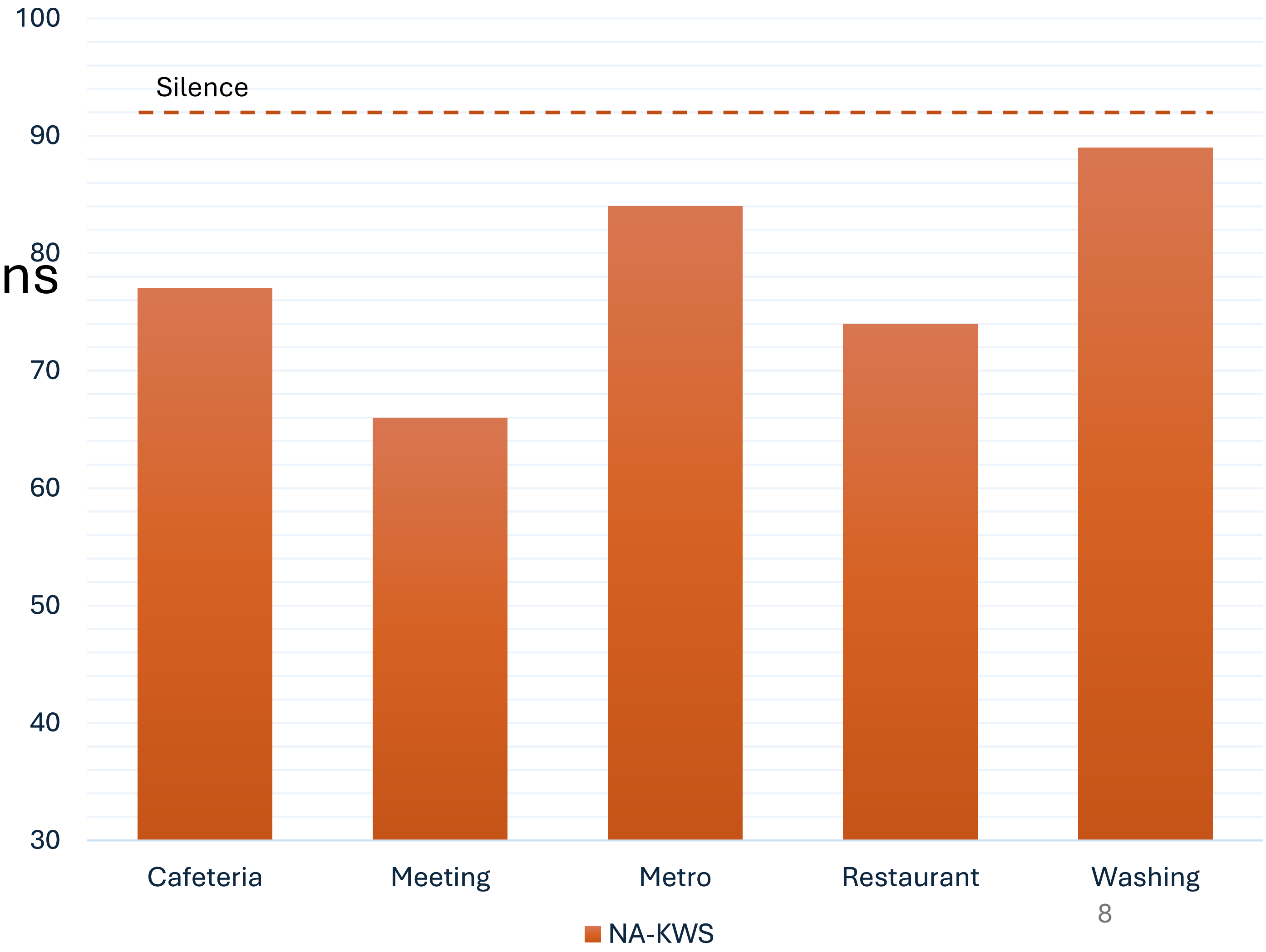
- **Noise-Aware KWS**
  - Noise-augmented KWS at (pre)training time
  - 3% to 26% accuracy reductions over silent environments



Can a KWS system still recognize the words?

- **Noise-Aware KWS**
  - Noise-augmented KWS at (pre)training time
  - 3% to 26% accuracy reductions over silent environments
  - Difficult to separate the target from the noise



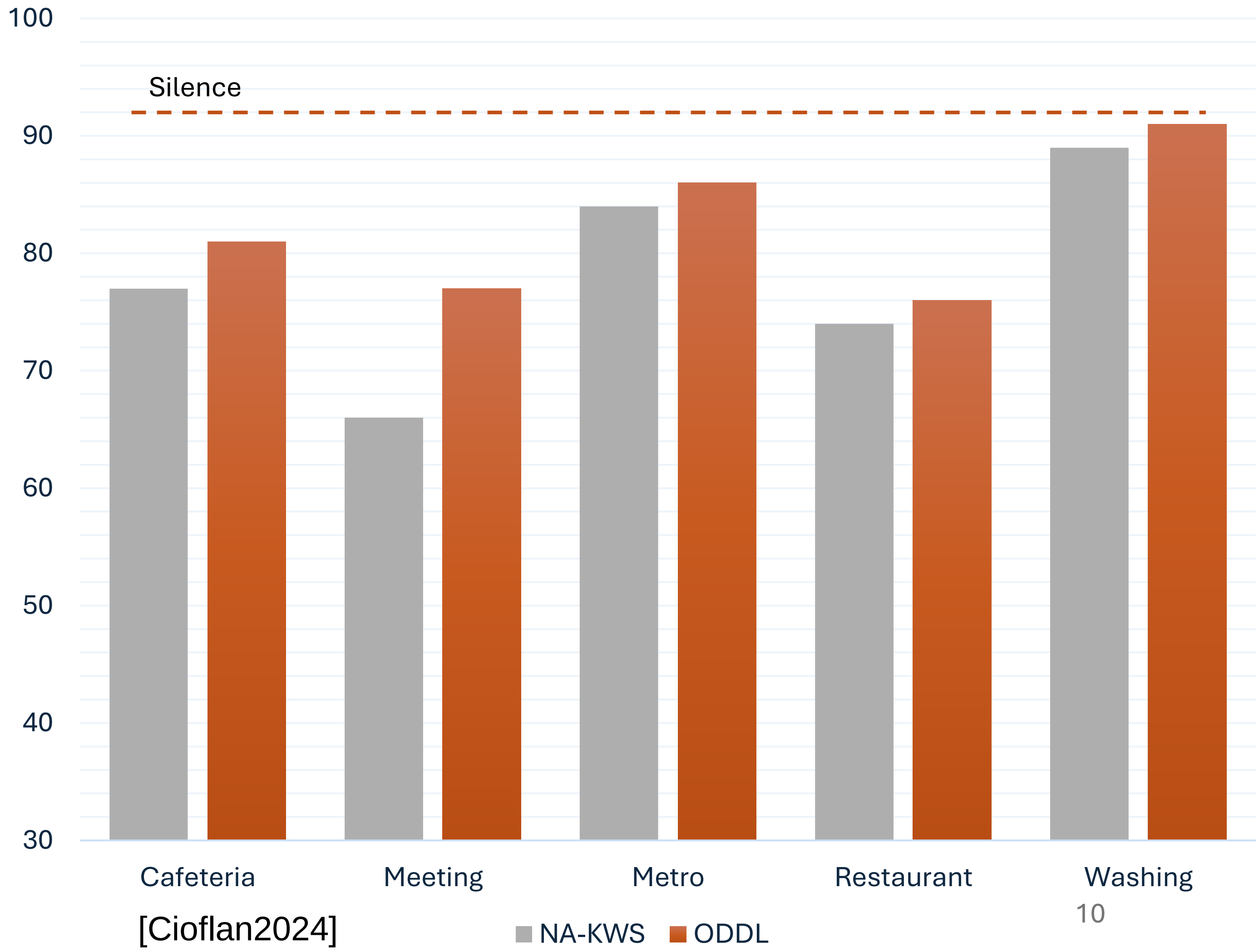


Can we improve KWS accuracy
through direct noise exposure?

On-Device Domain Learning

On-Device Domain Learning
Improving the KWS accuracy in noisy environments

- **Accuracy increments** by 4% on average over **NA-KWS**
- 13% on **speech noise**



[Cioflan2024]

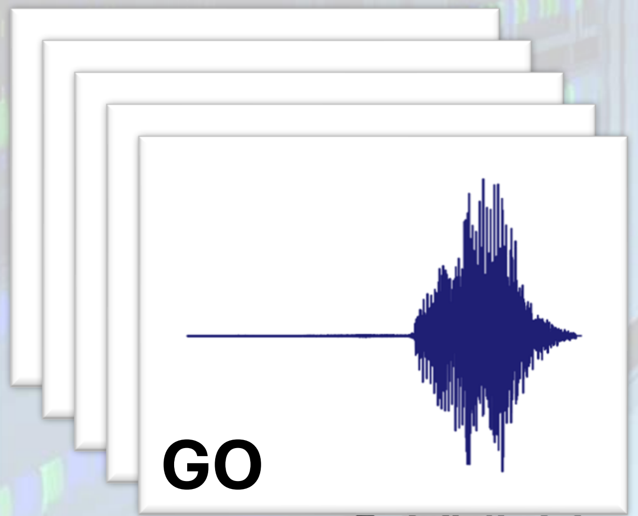
The Methodology

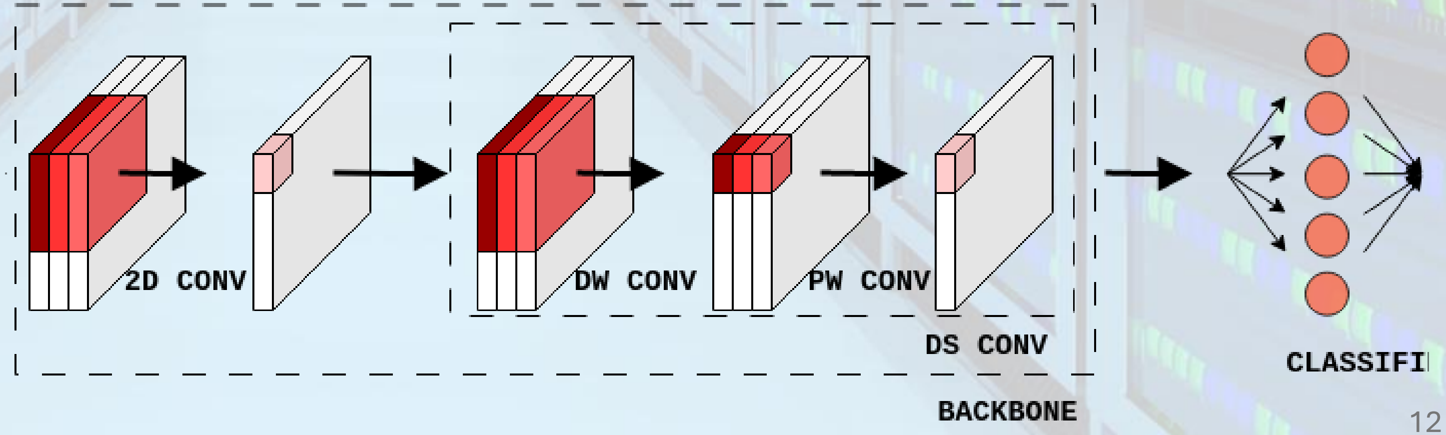
- Enable on-device keyword spotting
  - Train (and quantize) NA-KWS model – on the server

ETH zürich PULP Parallel Ultra Low Power

The Methodology

- Enable on-device keyword spotting
  - Train (and quantize) NA-KWS model – on the server
  - Deploy KWS model
  - Store pre-recorded utterances and labels



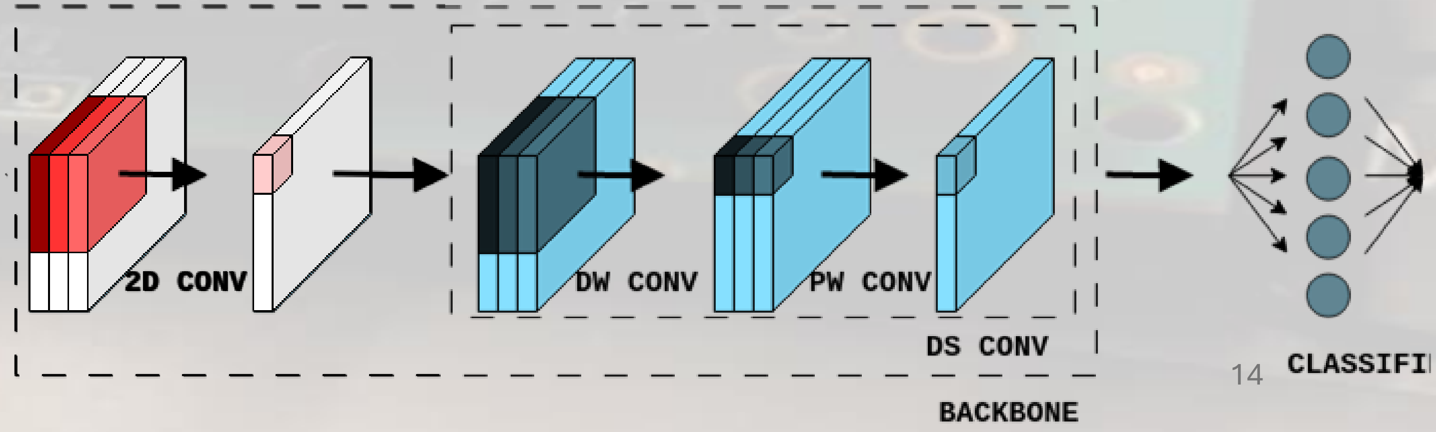


The Methodology

- Enable on-device keyword spotting
  - Train (and quantize) NA-KWS model – on the server
  - Deploy KWS model
  - Store pre-recorded utterances and labels
- Adapt to new environments
  - Record noise from the environment
  - Augment pre-recorded utterances

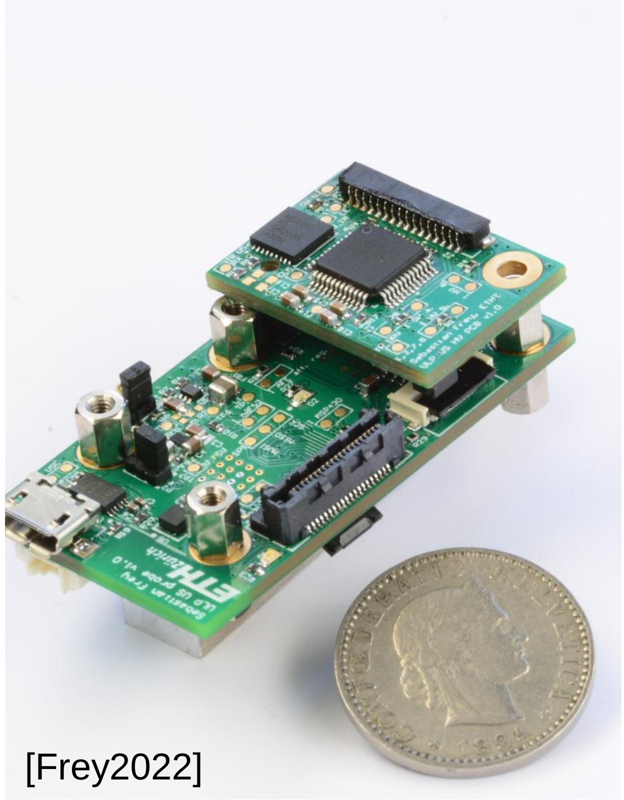
The Methodology

- Enable on-device keyword spotting
  - Train (and quantize) NA-KWS model – on the server
  - Deploy KWS model
  - Store pre-recorded utterances and labels
- Adapt to new environments
  - Record noise from the environment
  - Augment pre-recorded utterances
  - On-device learning

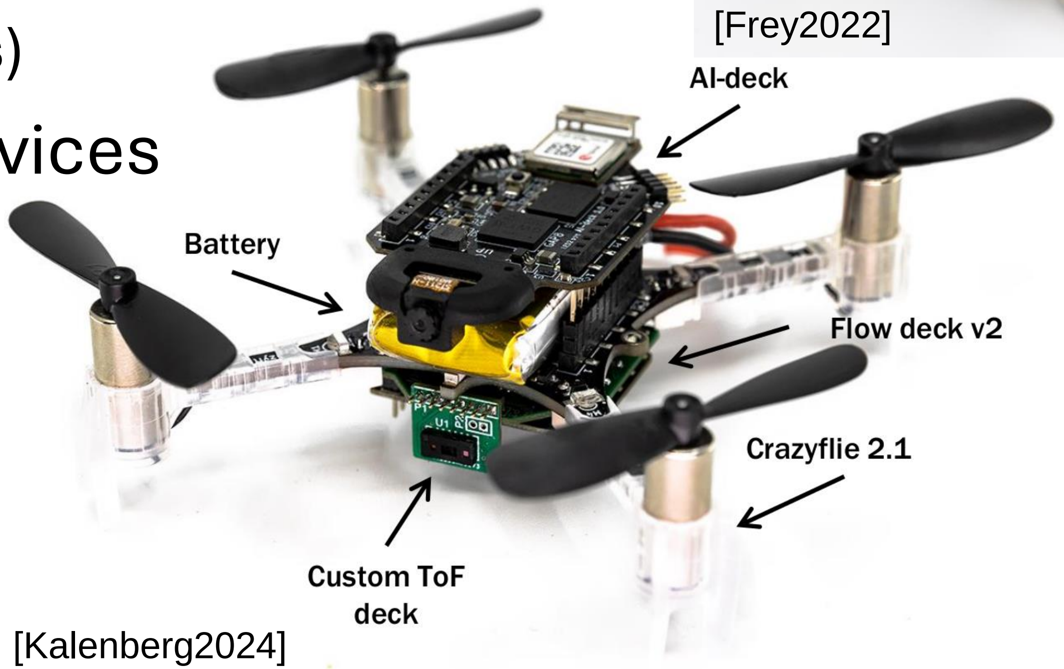


The embedded perspective

- Embedded, miniaturized devices
  - Limited **storage** (e.g., data, model parameters)
  - Limited **memory** (e.g., activations, gradients)
- Real-time operation
  - Minimize **latency** ($\propto$ **#operations**)
- Always-on, battery operated devices
  - Minimize **energy consumption**

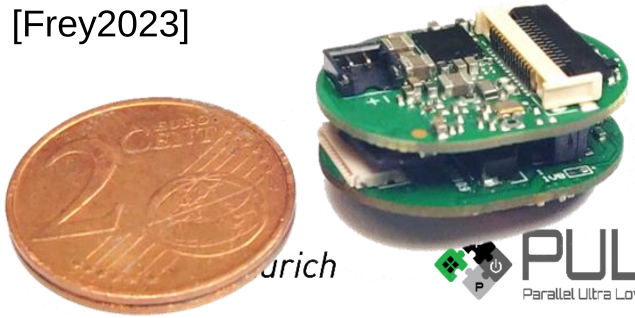


[Frey2022]



[Kalenberg2024]

[Frey2023]



On-Device Learning has costs

Storage

Memory

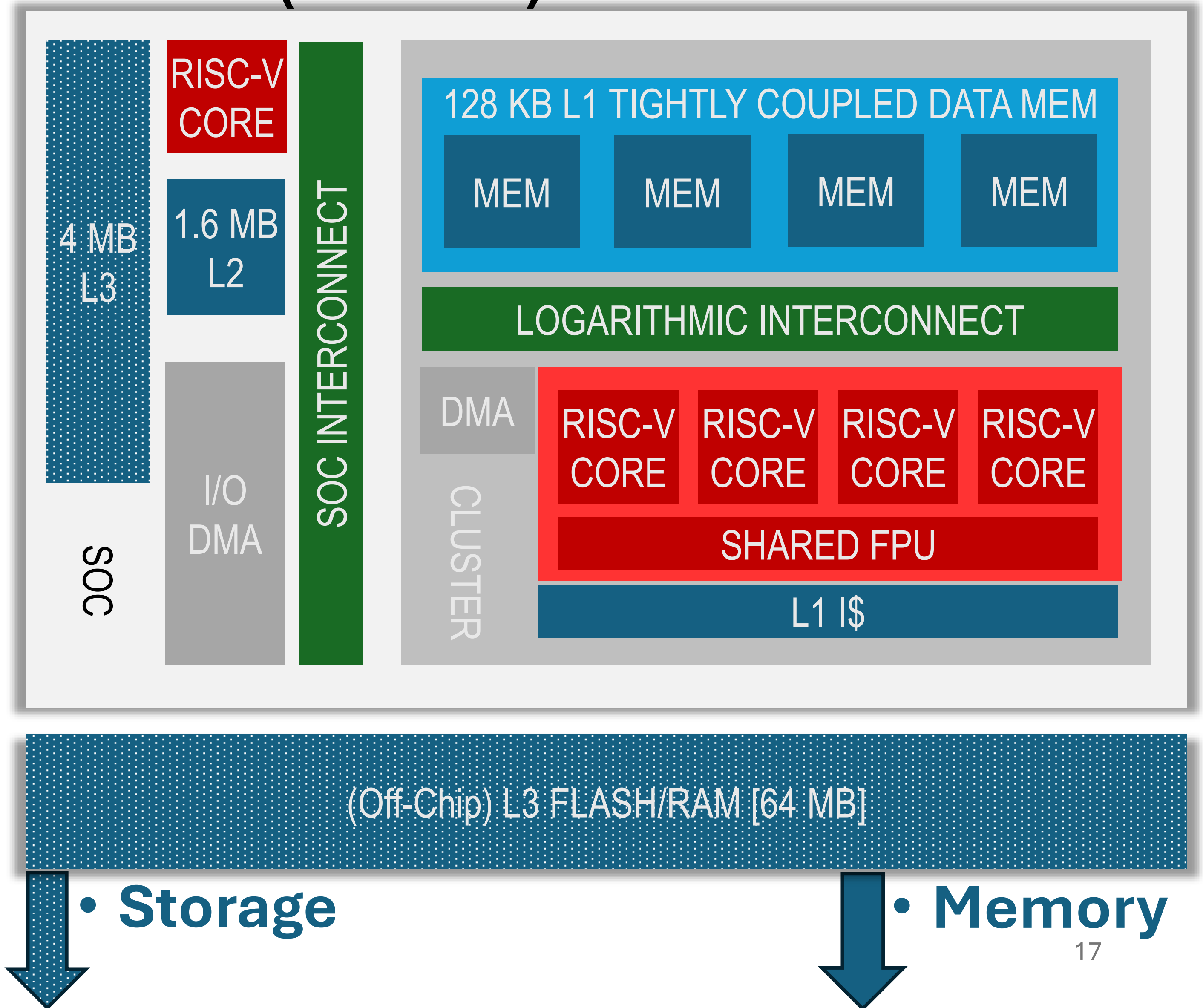
Operations
(latency)

Accuracy

Energy

Parallel Ultra-Low Power (PULP) Platform

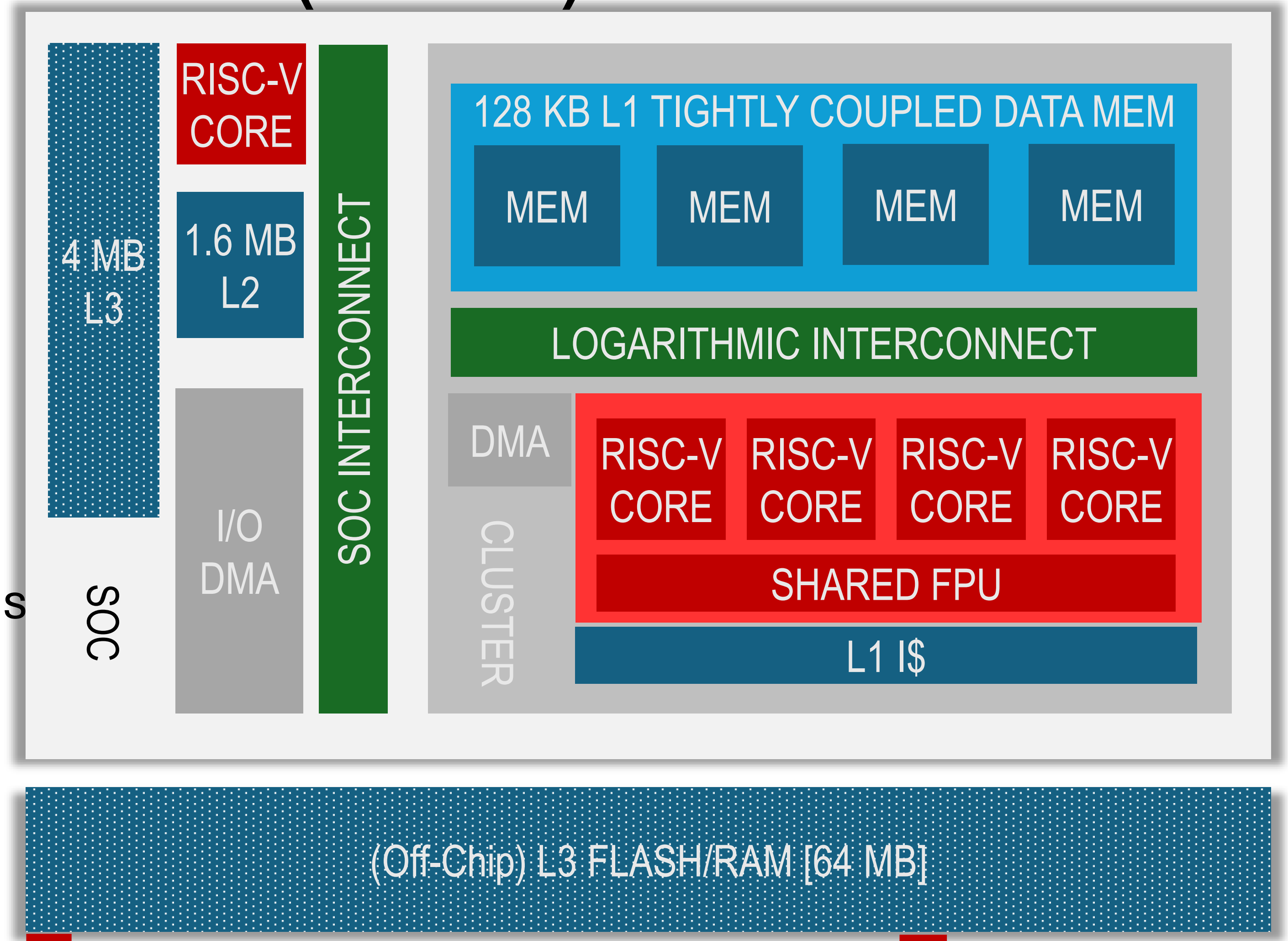
- Hierarchical memory architecture
  - L1 – single-cycle access
    - TCDM – cluster domain
  - L2 – SRAM – SoC domain
  - L3 – Non-volatile
    - On-chip MRAM
    - Off-chip mem. through SPI



- Storage
- Memory

Parallel Ultra-Low Power (PULP) Platform

- Hierarchical memory architecture
  - L1 – single-cycle access
  - L1 → L2 → L3
- Heterogeneous compute units
  - General purpose RISC-V cores
    - Control core (SoC) & cluster
    - Execution parallelization
    - SIMD extensions
  - Shared FPUs



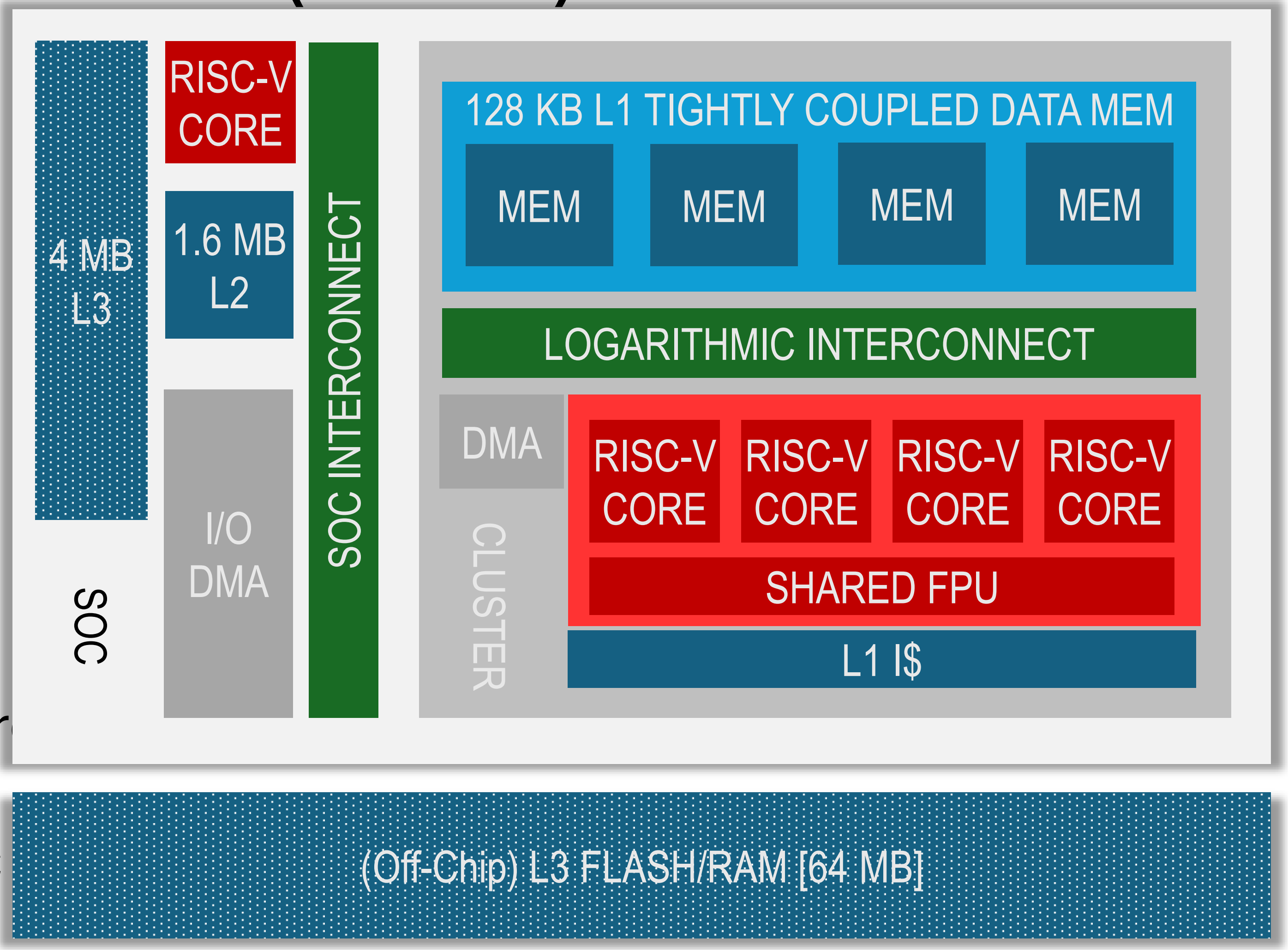
- **Operations (latency)**
- **Energy**

ETH zürich

PULP Parallel Ultra Low Power

Parallel Ultra-Low Power (PULP) Platform

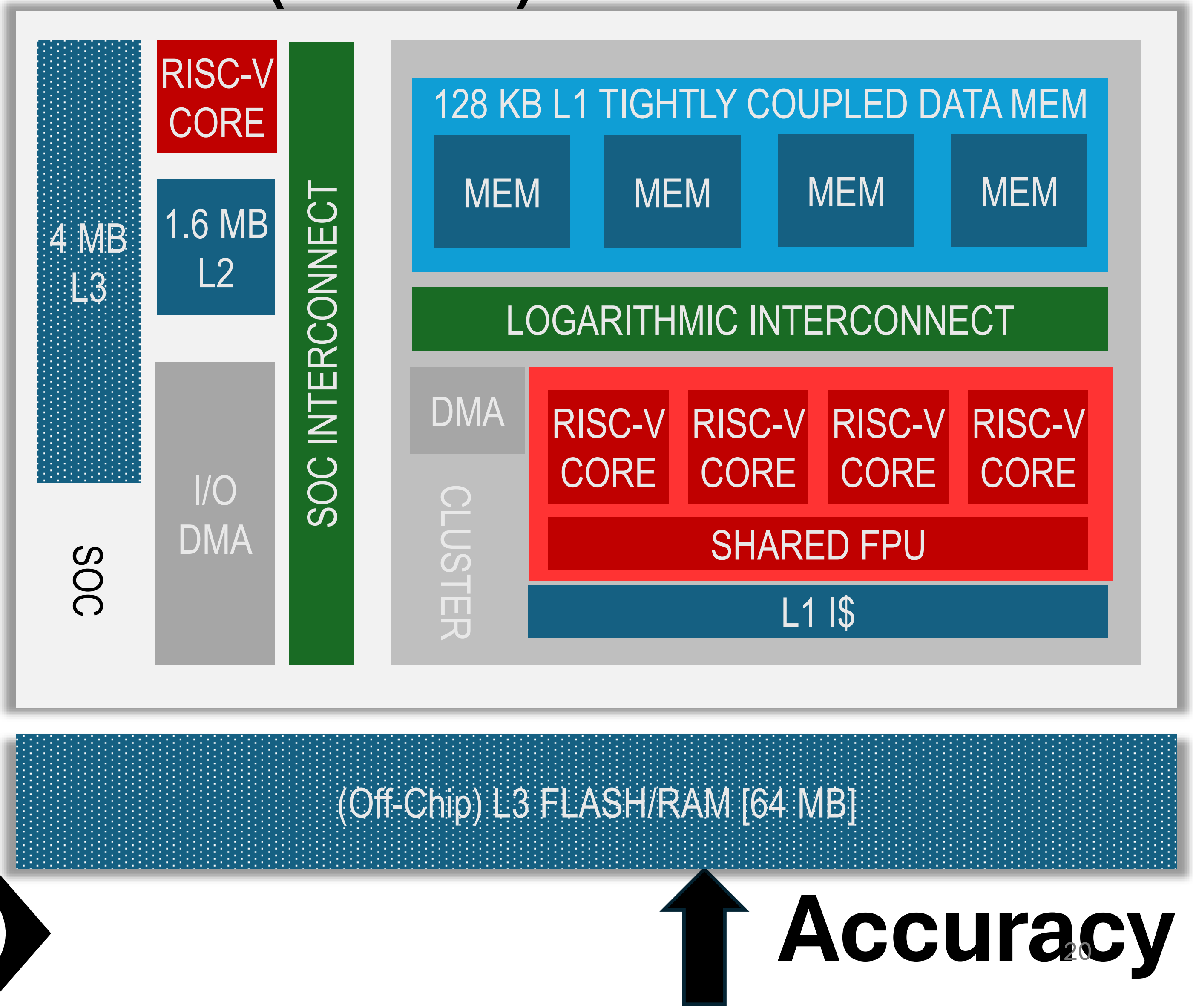
- Hierarchical memory architecture
- Heterogeneous compute units
- PULP SDK
  - PULP toolchain – compile and exploit features
  - PMSIS – PULP MCU Software Interface Standard
  - Targets boards/RTL/GVSOC



github.com/pulp-platform/pulp-sdk

Parallel Ultra-Low Power (PULP) Platform

- Hierarchical memory architecture
- Heterogeneous compute units
- PULP SDK & GVSOC
  - Instruction set simulator
    - Timing model
    - Virtual models of devices



github.com/pulp-platform/gvsoc

On-Device Domain Learning
A case study

Deep dive into hardware-aware learning

- Enable on-device keyword spotting
  - Train (and quantize) NA-KWS model – on the server
  - Deploy KWS model
  - Store pre-recorded utterances and labels
- Adapt to new environments
  - Record noise from the environment
  - Augment pre-recorded utterances
  - On-device learning

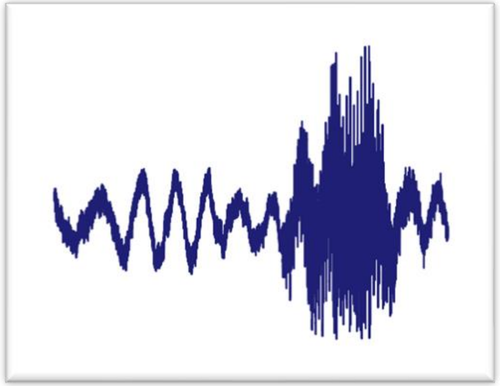
Deep dive into hardware-aware learning

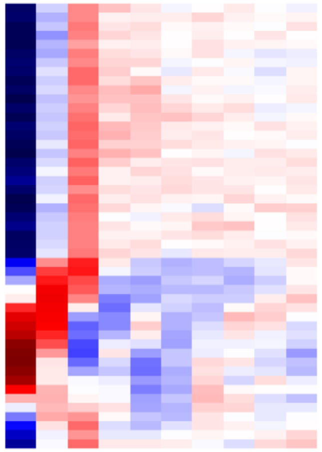
- Enable on-device keyword spotting
  - Train (and quantize) NA-KWS model – on the server
  - Deploy KWS model
  - Store pre-recorded utterances and labels
- Adapt to new environments
  - Record noise from the environment
  - Augment pre-recorded utterances
  - **On-device learning**

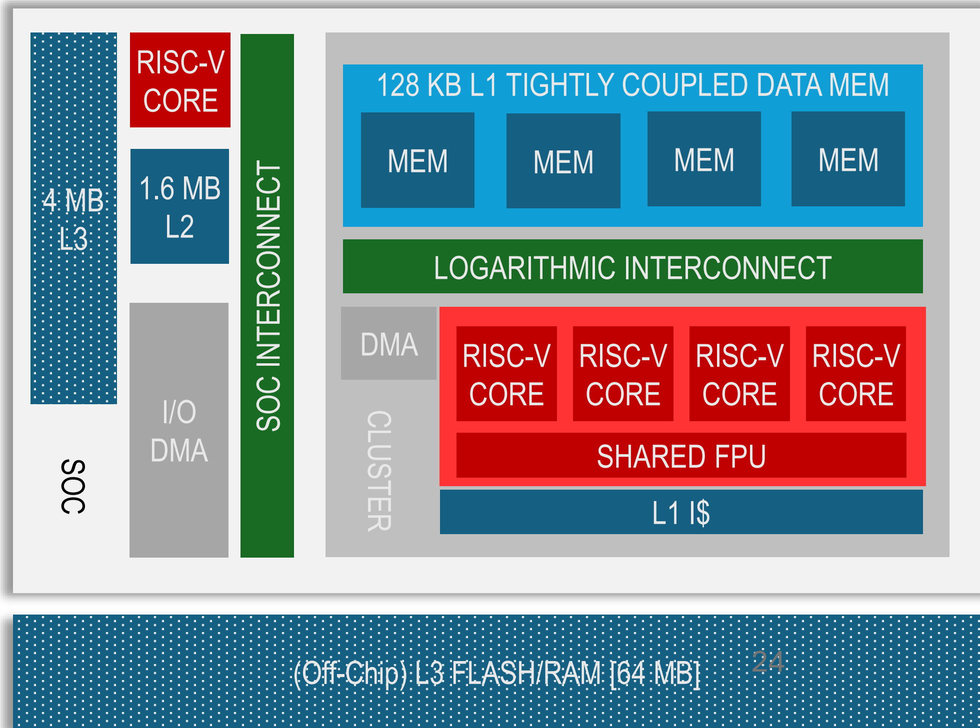
1. **Forward pass – compute the activations**
2. **Backward pass**
   1. **Compute the loss considering the ground truth (pre-recorded)**
   2. **Compute the gradients through *backpropagation***
   3. **Update the parameters**

Forward pass
Backbone features

- Usually preceded by a preprocessing step



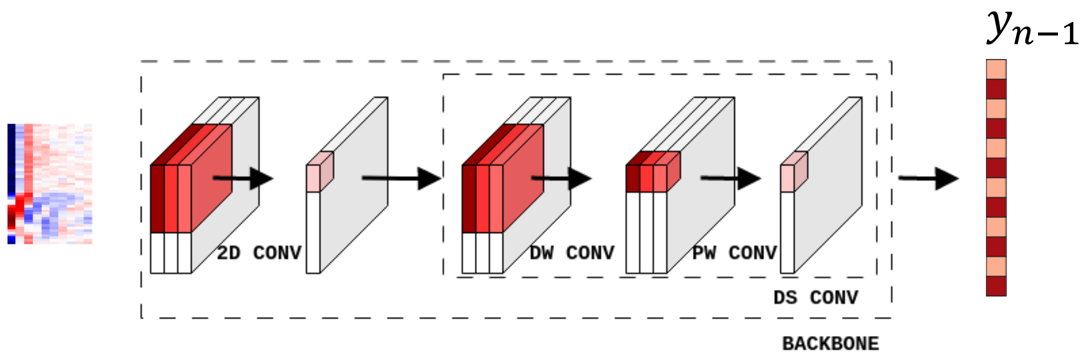




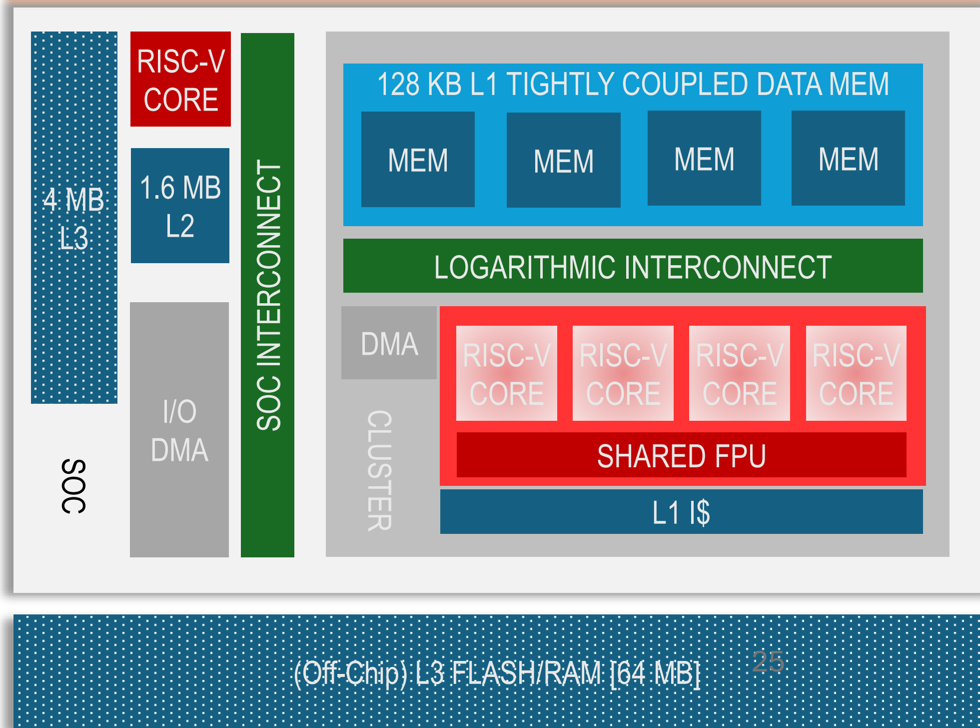
Forward pass
Backbone features

- Forward pass - the neural network approximates a **mapping function**

$$y_{n-1} = f_{n-1}\,(input)$$




```
pulp_backbone_fp32_fw_cl(&args);
```

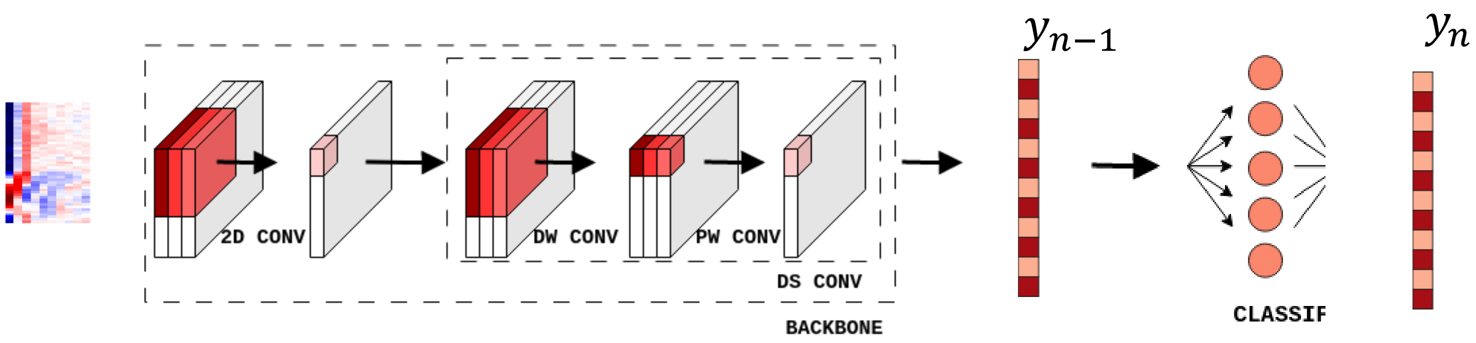


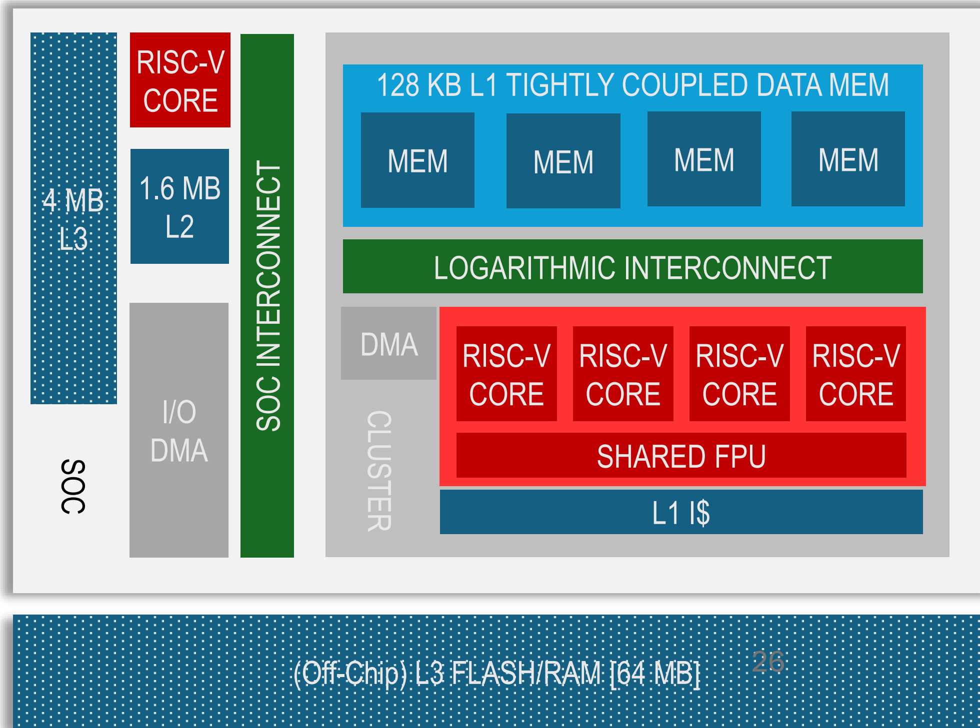
ETH zürich
PULP Parallel Ultra Low Power

Forward pass
Classifier features

- Forward pass - the neural network approximates a **mapping function**

$$y_{n-1} = f_{n-1}\,(input)$$
$$y_n = f_n\,(z_n) = f_n(W_n \cdot x_n + b_n) = f_n\,(W_n \cdot y_{n-1} + b_n)$$



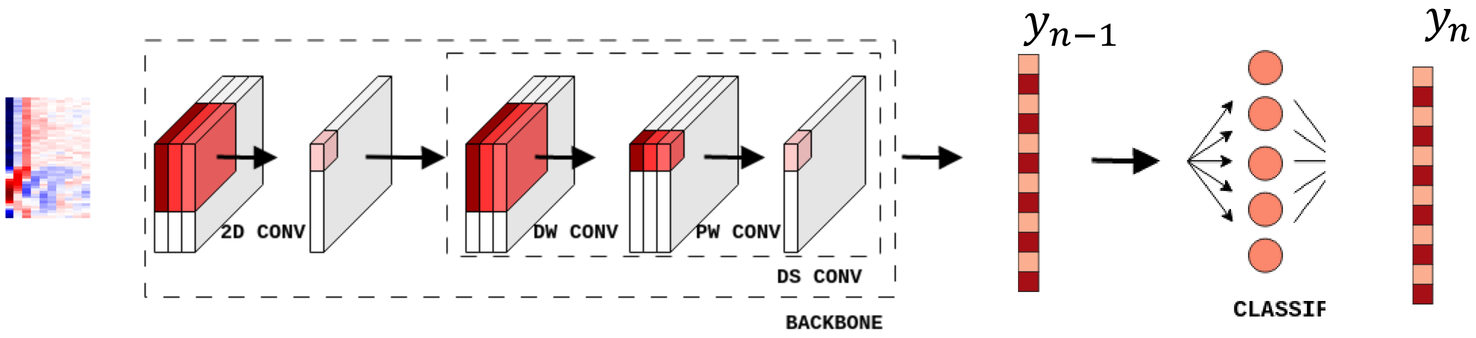


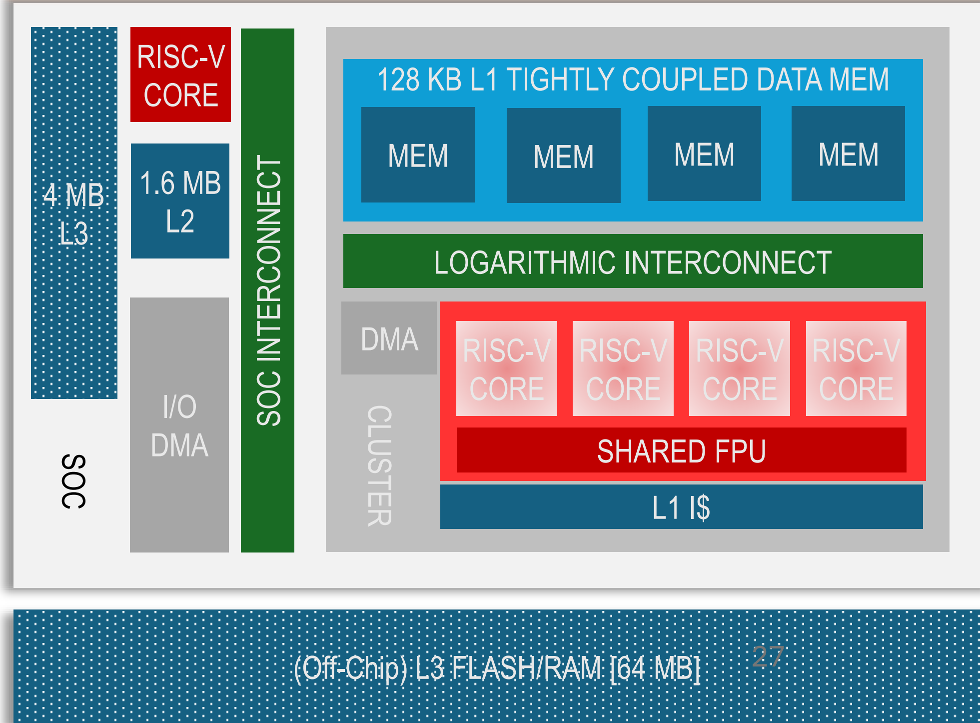
Forward pass
Classifier features

- Forward pass - the neural network approximates a **mapping function**

$$y_{n-1} = f_{n-1}\,(input)$$
$$y_n = f_n\,(z_n) = f_n(W_n \cdot x_n + b_n) = f_n\,(W_n \cdot y_{n-1} + b_n)$$

```
pulp_backbone_fp32_fw_cl(&args);
pulp_linear_fp32_fw_cl(&args);
```



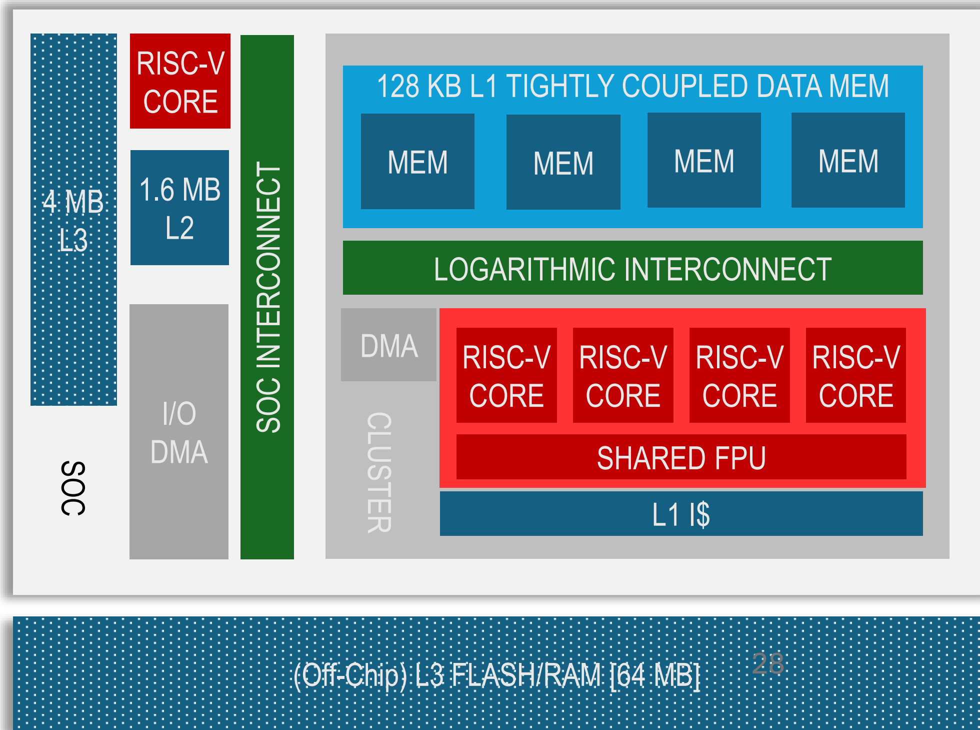


Backward pass
Compute the loss

- Backward pass via backpropagation (**training**) – **learning** the model parameters

$$\min_{input} L\left(y_n(input), y_{gt}\right)$$

  - where y_{gt} represents the label



ETH zürich PULP Parallel Ultra Low Power

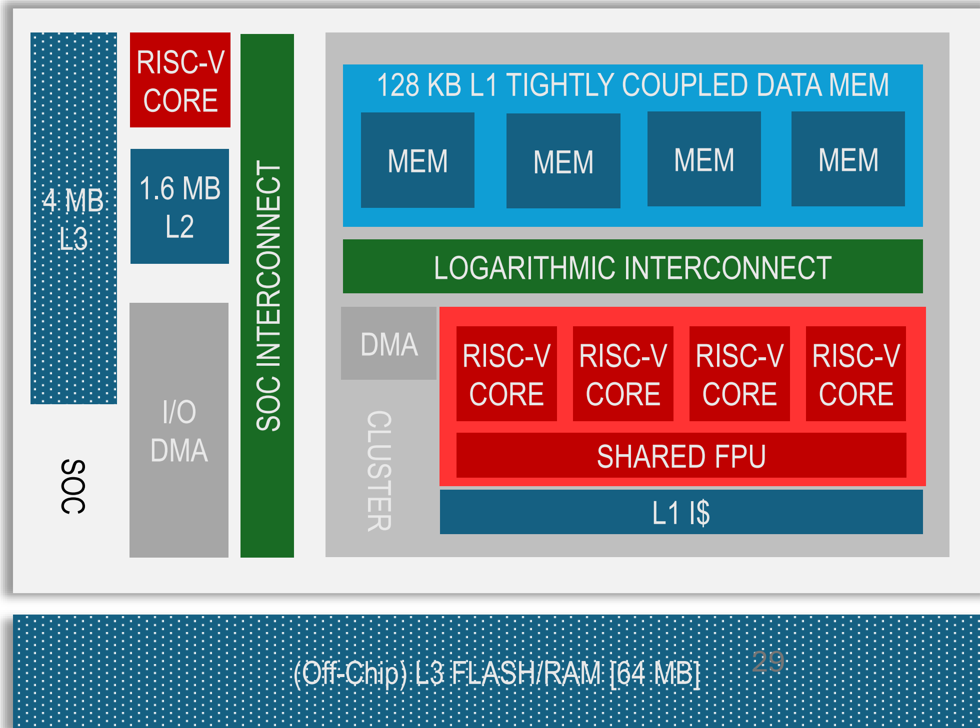
Backward pass
Compute the loss

- Backward pass via backpropagation (**training**) – **learning** the model parameters

$$\min_{input} L\left(y_n(input), y_{gt}\right)$$

$$let\ L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

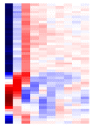
  - where y_{gt} represents the label
  - averaged over S samples

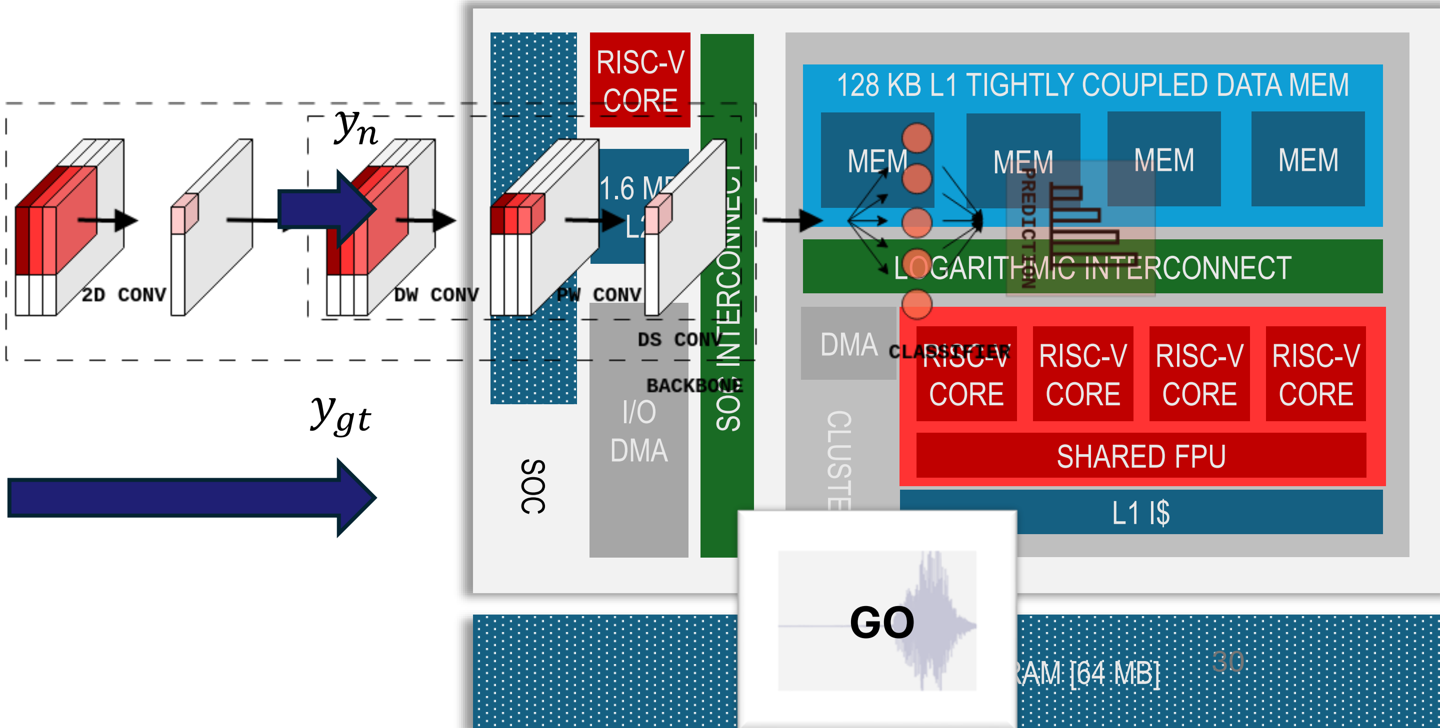


Backward pass
Compute the loss

- Backward pass via backpropagation (**training**) – **learning** the model parameters

$$L_{MSE} = \frac{1}{S}\,(y_n(input) - y_{gt})^2$$



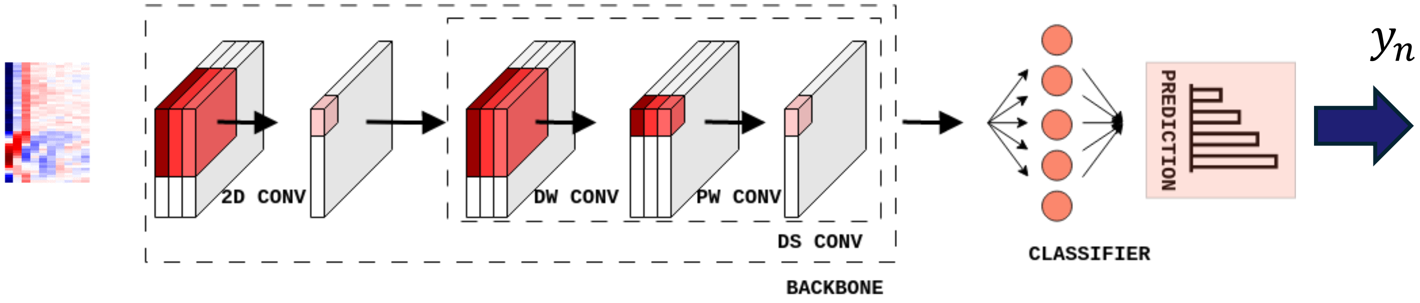


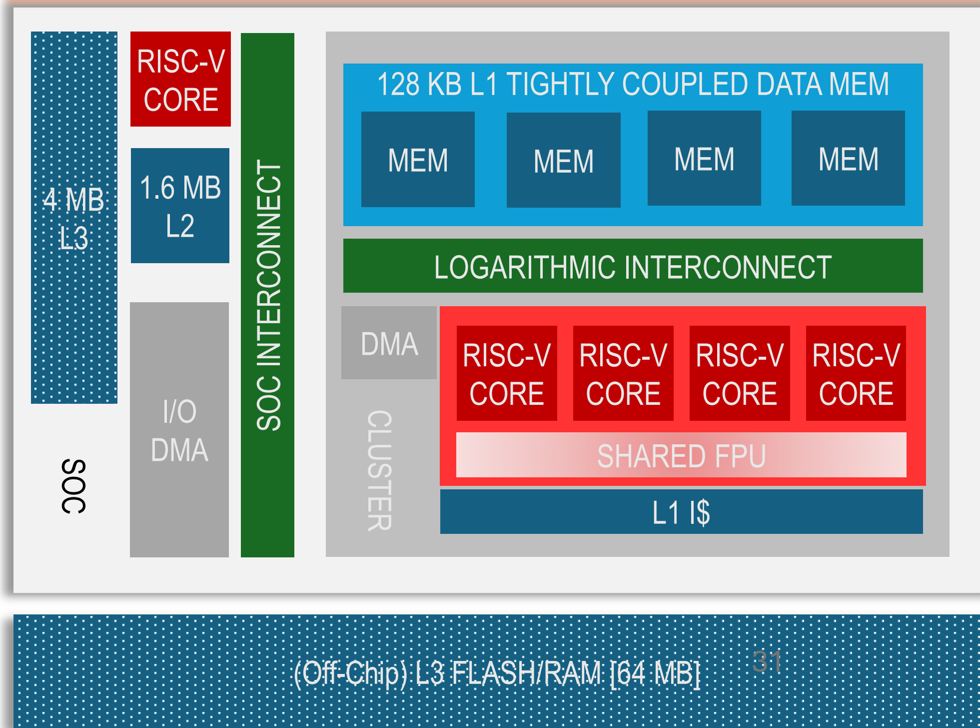
Backward pass
Compute the loss

- Backward pass via backpropagation (**train**
  model parameters

$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

```
pulp_backbone_fp32_fw_cl(&args);
pulp_linear_fp32_fw_cl(&args);
pulp_MSELoss(&loss_args);
```





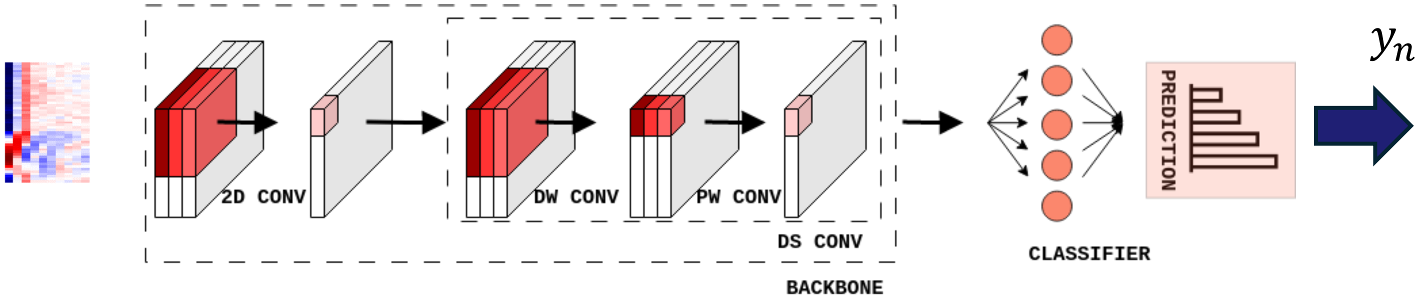
Backward pass
Compute the loss

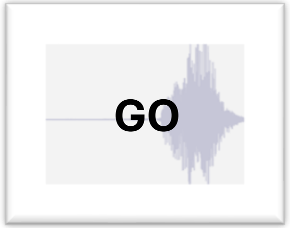
Change
weights & biases

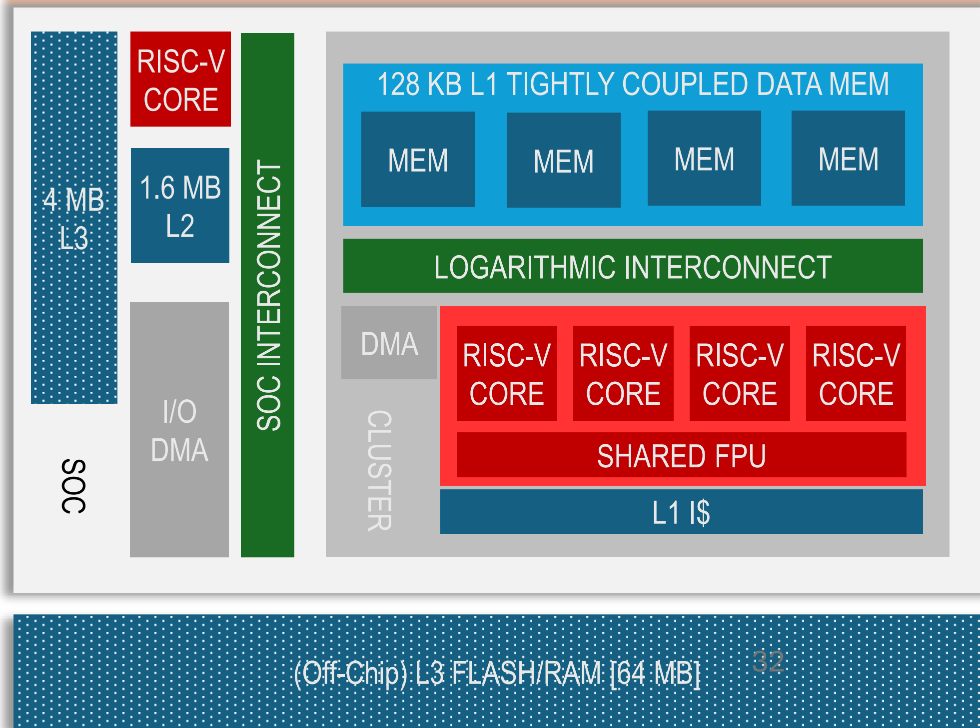
- backpropagation (**train**

$$L_{MSE} = \frac{1}{S}\,(y_n(input) - y_{gt})^2$$

```
pulp_backbone_fp32_fw_cl(&args);
pulp_linear_fp32_fw_cl(&args);
pulp_MSELoss(&loss_args);
```



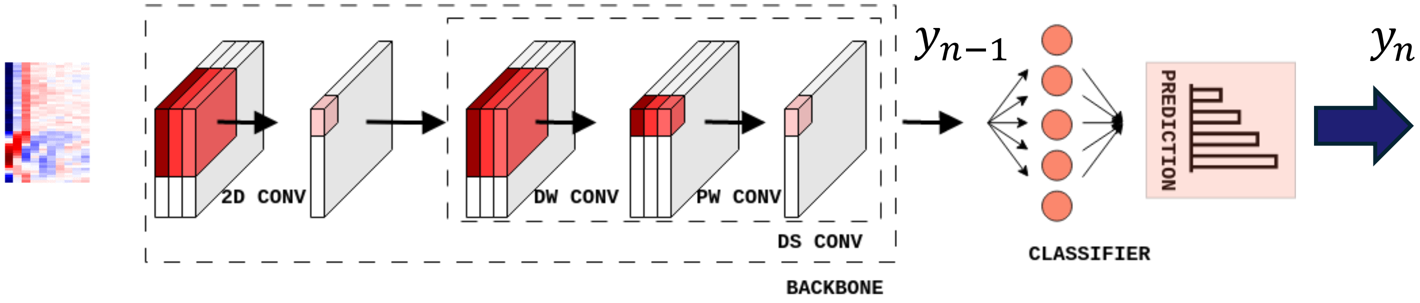


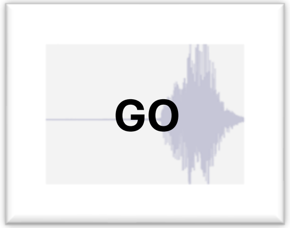


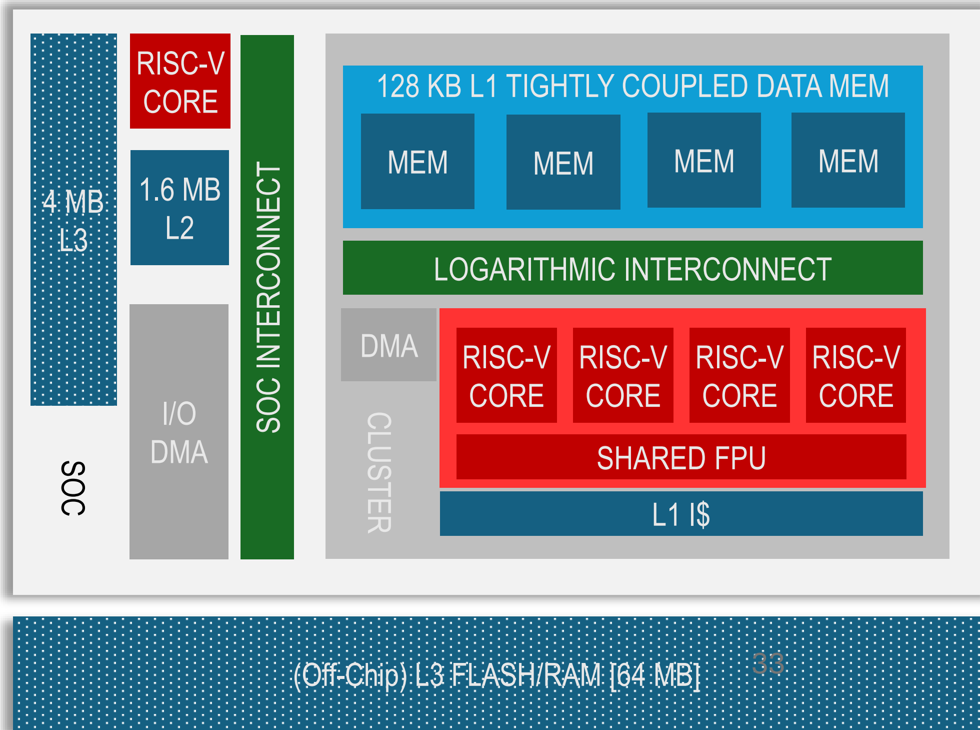
Backward pass
Compute the gradients (backpropagation)

- Backward pass via backpropagation (**training**) – **learning** the model parameters

$$y_n = f_n\,(W_n \cdot y_{n-1} + b_n)$$
$$L_{MSE} = \frac{1}{S}\,(y_n(input) - y_{gt})^2$$







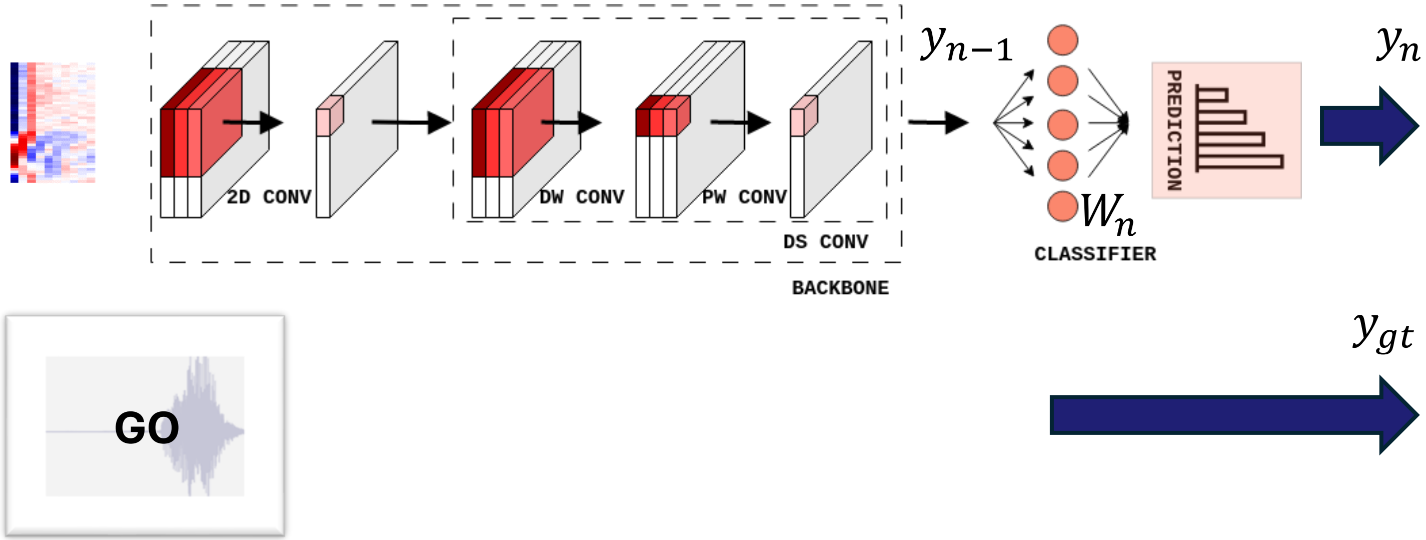
ETH zürich
PULP Parallel Ultra Low Power

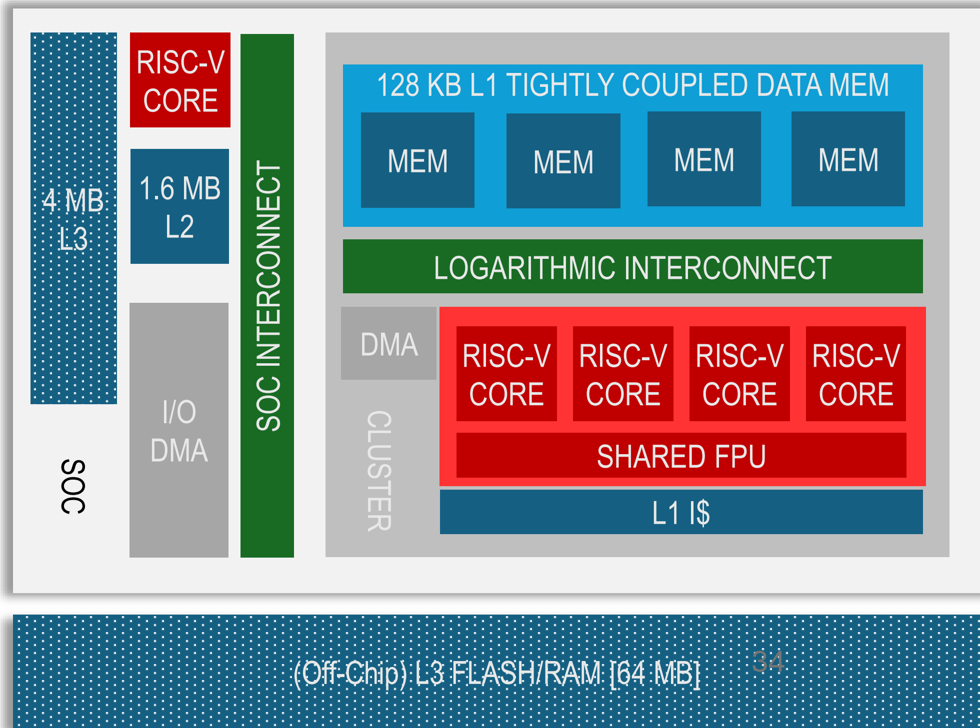
Backward pass
Compu nts (backpropagation)

Let us assume no bias

- Backwa ropagation (**training**) – **learning** the model parameters

$$y_n = f_n\,(W_n \cdot y_{n-1} + 0)$$
$$L_{MSE} = \frac{1}{S}\,(y_n(input) - y_{gt})^2$$





ETH zürich

PULP Parallel Ultra Low Power

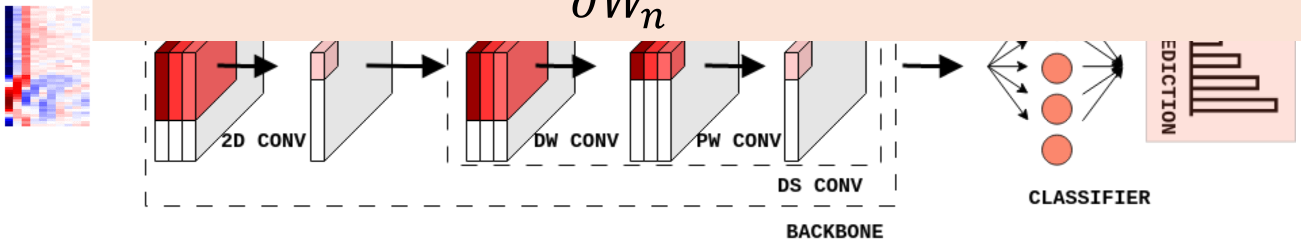
Backward pass
Compute the gradients (backpropagation)

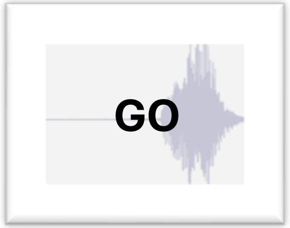
- Backward pass via backpropagation (**training**) – **learning** the model parameters

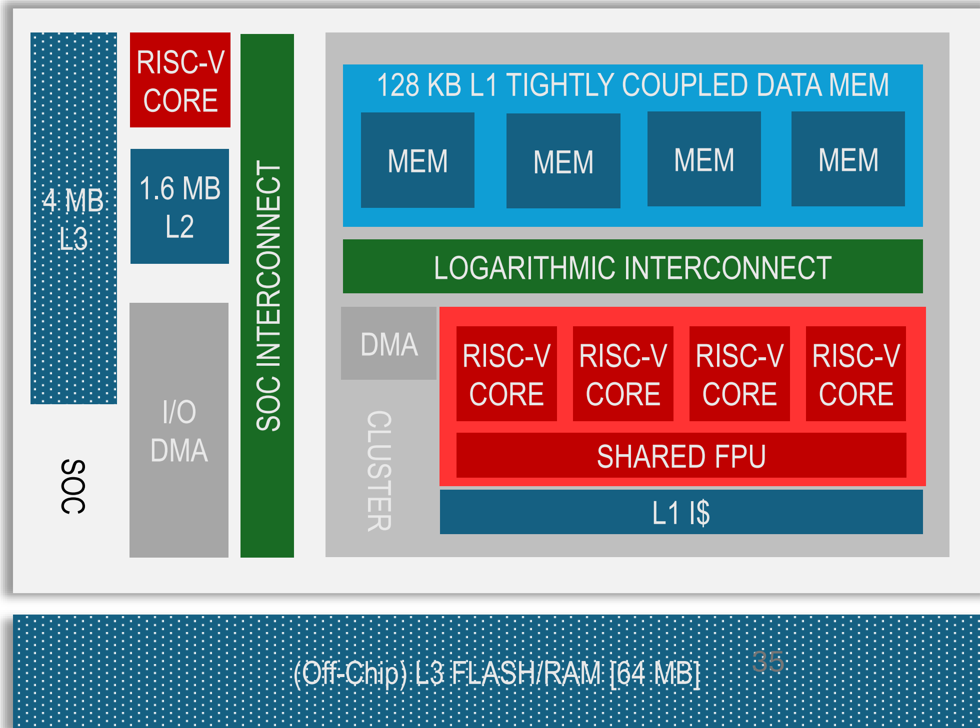
$$y_n = f_n\left(W_n \cdot y_{n-1}\right)$$

$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n}$$







ETH zürich

PULP Parallel Ultra Low Power

Backward pass
Compute the gradients (backpropagation)

- Backward pass via backpropagation (**training**) – **learning** the model parameters

$$y_n = f_n\,(W_n \cdot y_{n-1})$$

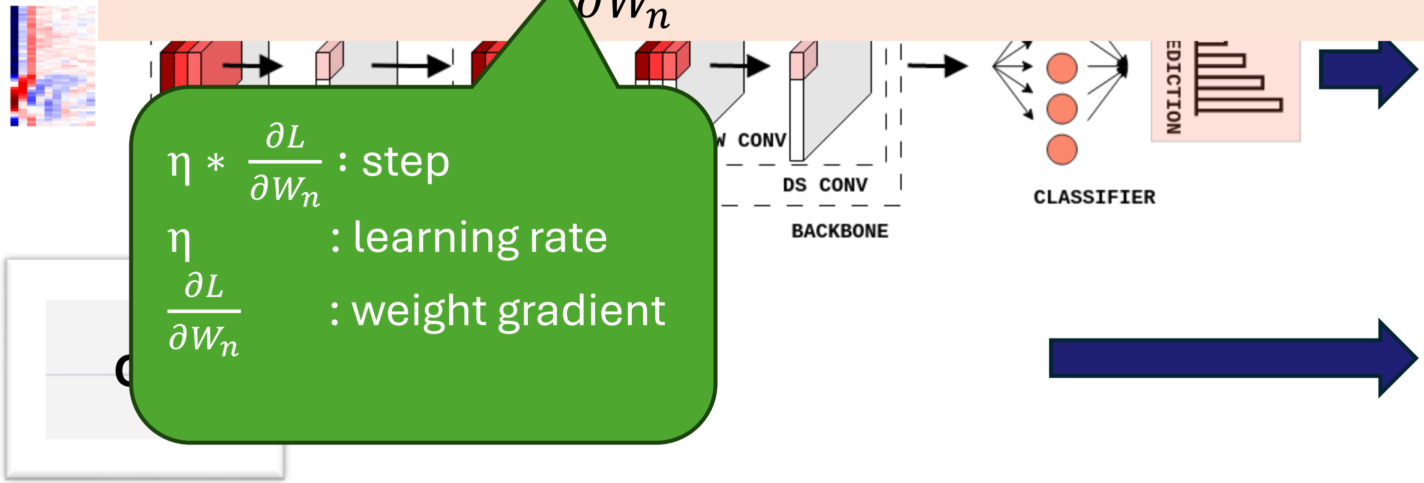
$$L_{MSE} = \frac{1}{S}\,(y_n(input) - y_{gt})^2$$

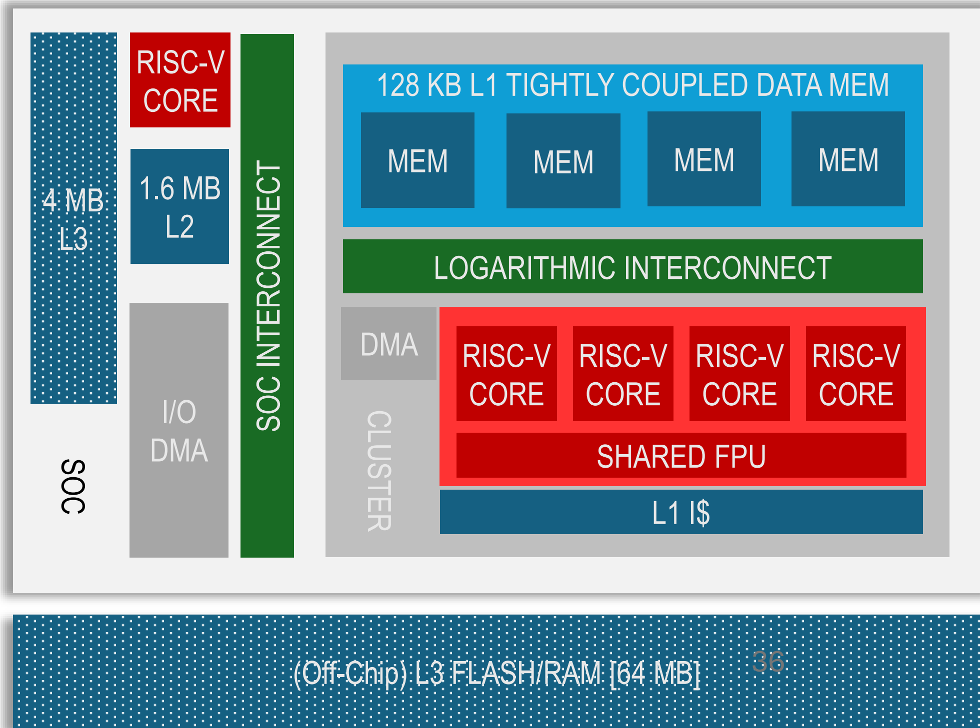
$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n}$$

$\eta * \frac{\partial L}{\partial W_n}$: step

η : learning rate

$\frac{\partial L}{\partial W_n}$: weight gradient





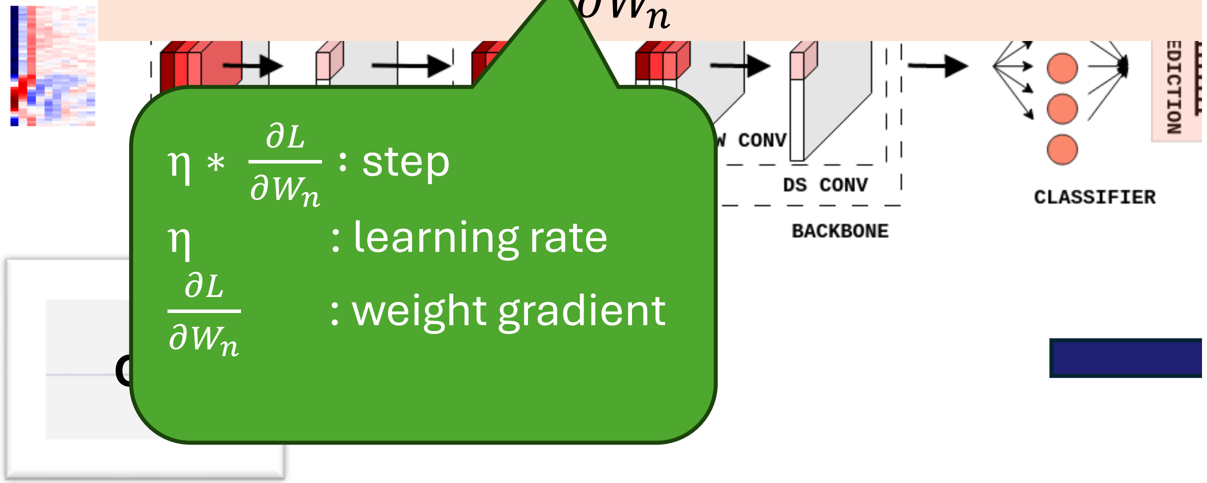
Backward pass
Compute the gradients (backpropagation)

- Backward pass via backpropagation (**training**) – **learning** the model parameters

$$y_n = f_n\,(W_n \cdot y_{n-1})$$

$$L_{MSE} = \frac{1}{S}\,(y_n(input) - y_{gt})^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n}$$

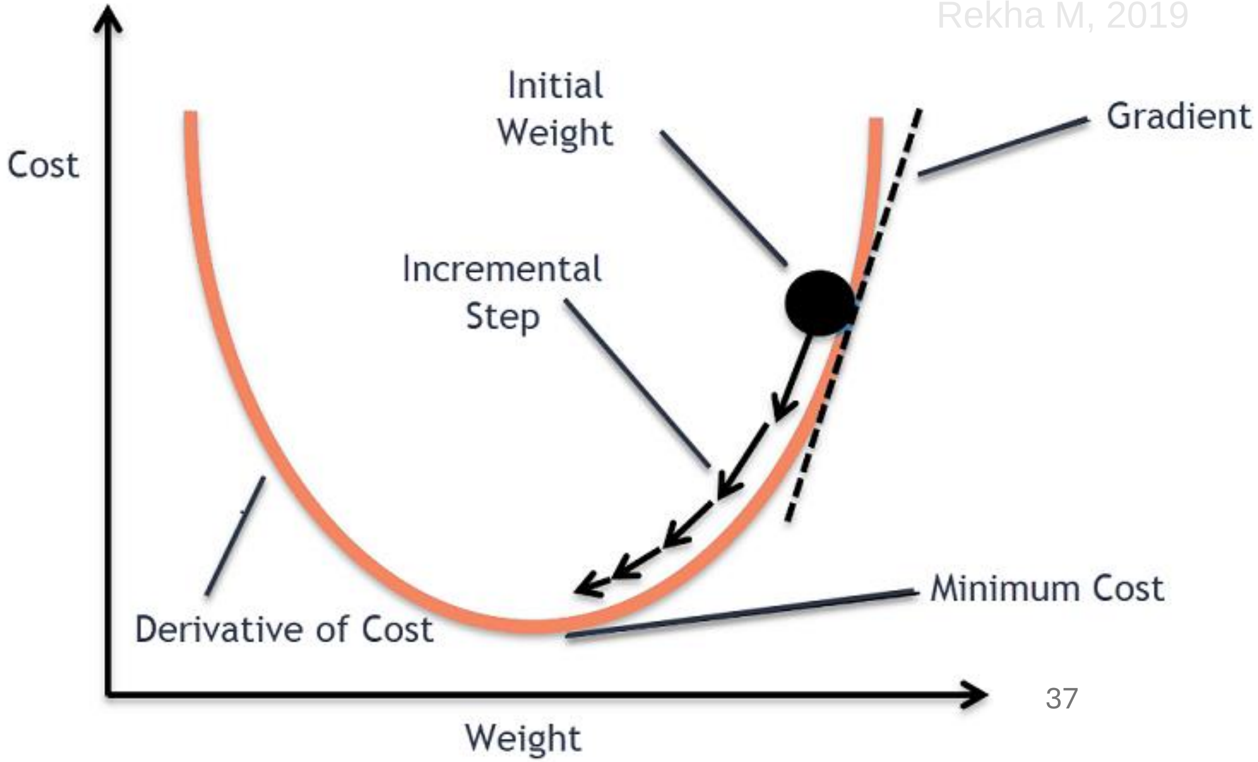


$\eta * \frac{\partial L}{\partial W_n}$: step

η : learning rate

$\frac{\partial L}{\partial W_n}$: weight gradient

Rekha M, 2019



ETH zürich

PULP Parallel Ultra Low Power

Backward pass
Compute the gradients (backpropagation)

- Backward pass via back… g the model parameters

$$y_n = f_n(z_n) = f_n(W_n \cdot y_{n-1})$$

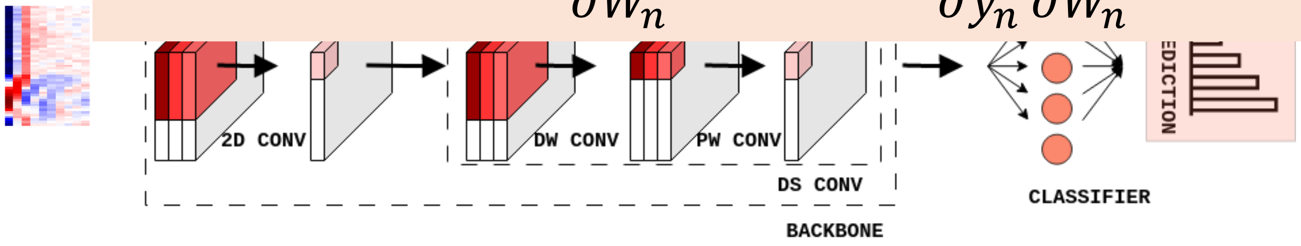
$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

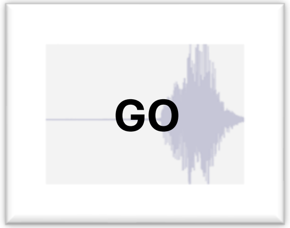
$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n}$$

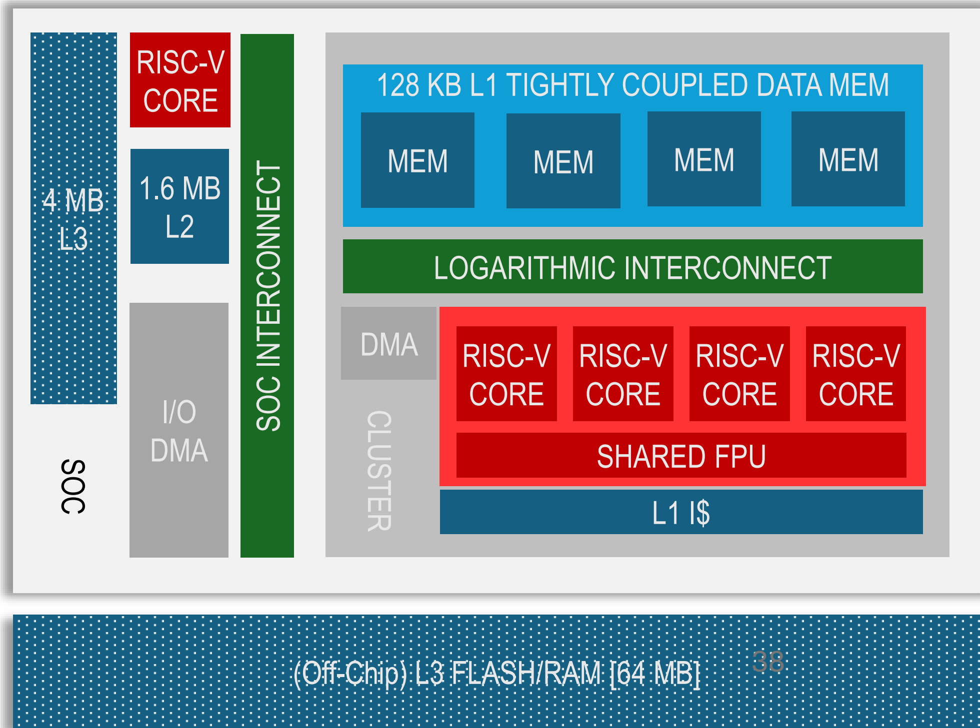
Chain rule

Let $y = f(g(x)), u = g(x)$. Then $y = f(u)$ and

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$







ETH zürich

PULP Parallel Ultra Low Power

Backward pass
Compute the gradients (backpropagation)

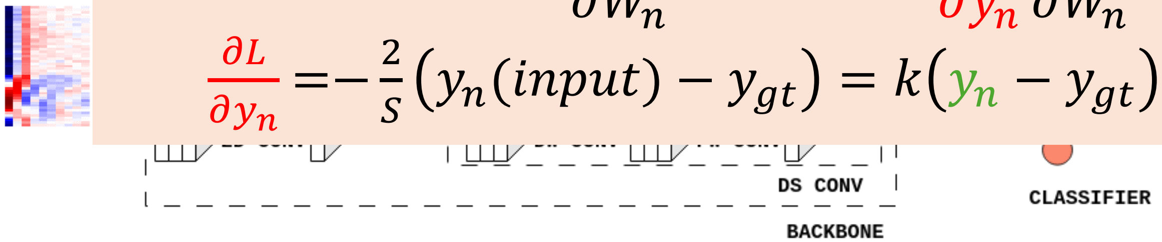
- Backward pass via backpropagation (**training**) – **learning** the model parameters

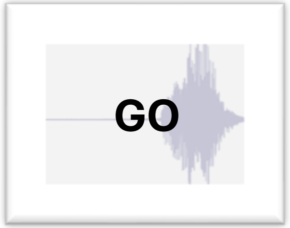
$$y_n = f_n(z_n) = f_n(W_n \cdot y_{n-1})$$

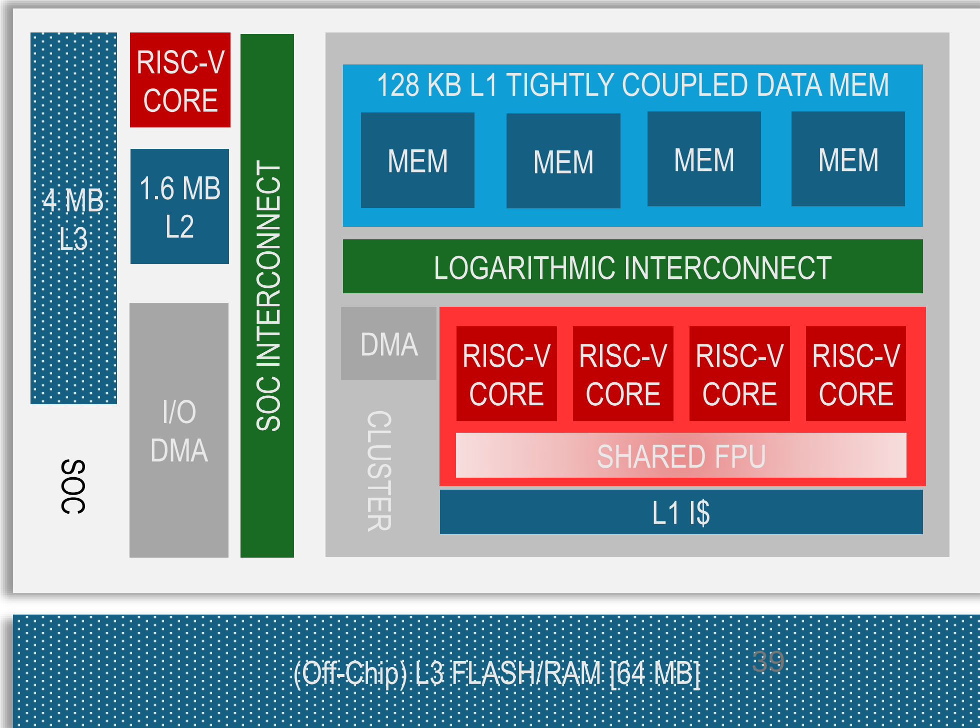
$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n}$$

$$\frac{\partial L}{\partial y_n} = -\frac{2}{S}\left(y_n(input) - y_{gt}\right) = k\left(y_n - y_{gt}\right)$$







ETH zürich

PULP Parallel Ultra Low Power

Backward pass
Compute the gradients (backpropagation)

- Backward pass via backpropagation (**training**) – **learning** the model parameters

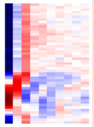
$$y_n = f_n(z_n) = f_n(W_n \cdot y_{n-1})$$

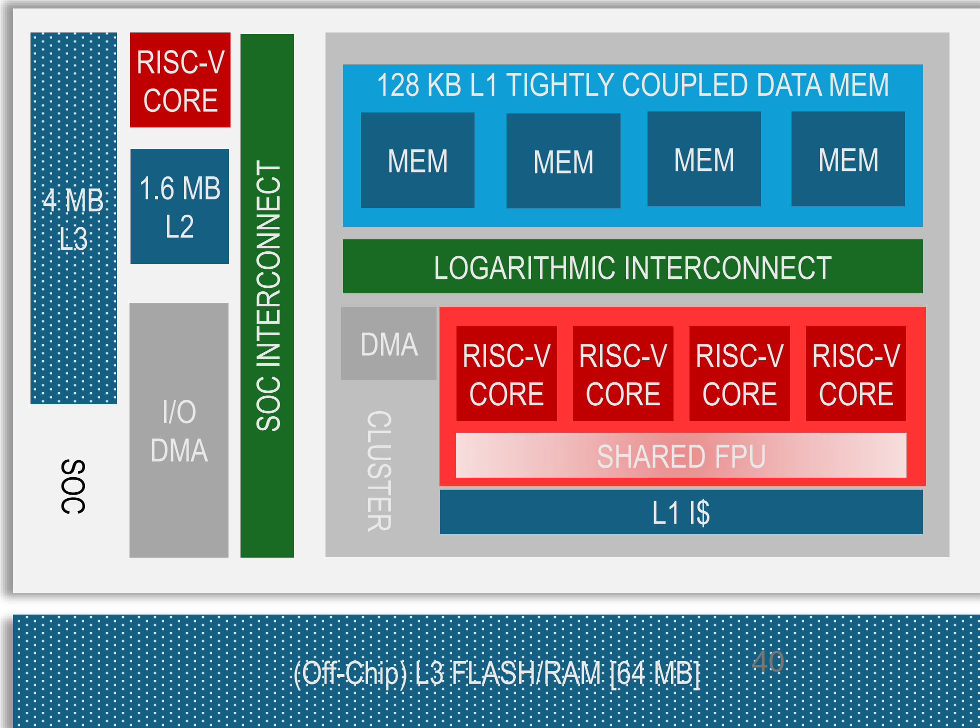
$$L_{MSE} = \frac{1}{S}(y_n(input) - y_{gt})^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n}$$

$$\frac{\partial L}{\partial y_n} = -\frac{2}{S}\left(y_n(input) - y_{gt}\right) = k\left(y_n - y_{gt}\right)$$

$$\frac{\partial z_n}{\partial W_n} = y_{n-1}$$





ETH zürich

PULP Parallel Ultra Low Power

Backward pass
Compute the gradients (backpropagation)

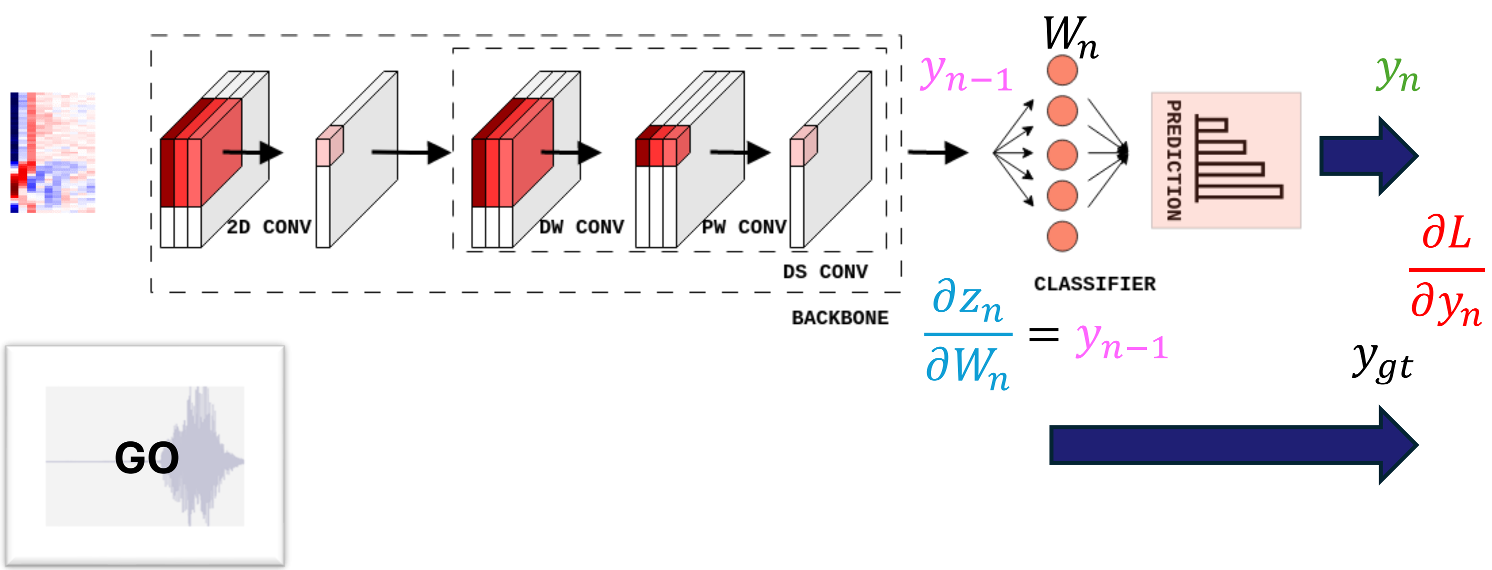
$$y_n = f_n(z_n) = f_n(W_n \cdot y_{n-1})$$

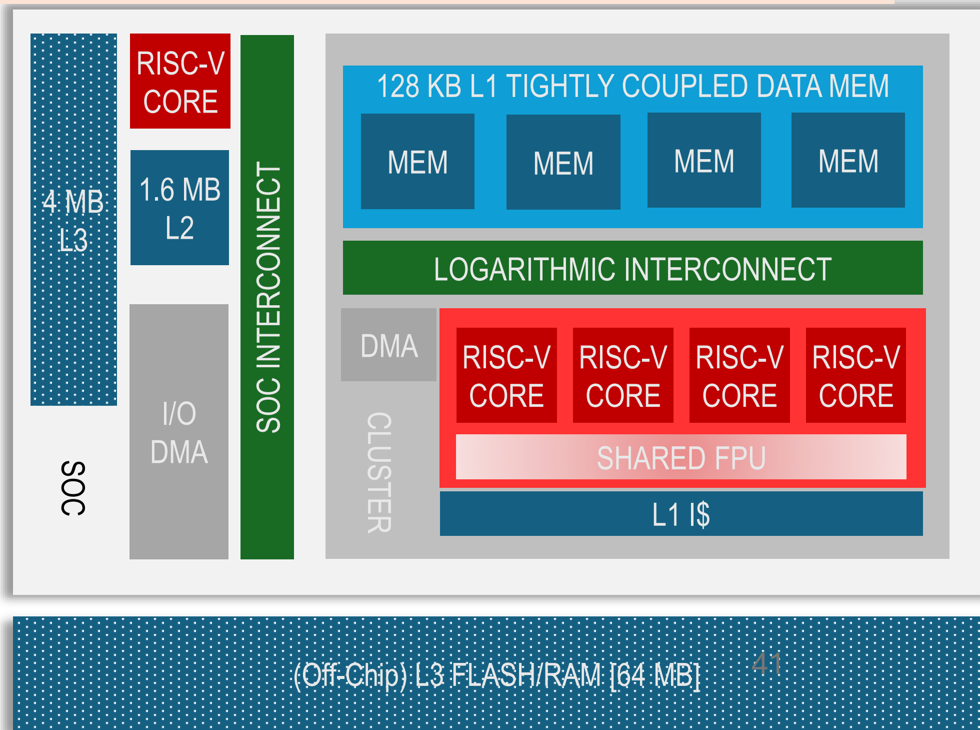
$$L_{MSE} = \frac{1}{S}(y_n(input) - y_{gt})^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n}$$

$$\frac{\partial L}{\partial y_n} = -\frac{2}{S}(y_n(input) - y_{gt}) = k(y_n - y_{gt})$$

$$\frac{\partial z_n}{\partial W_n} = y_{n-1}$$





Backward pass
Compute the gradients (backpropagation)

- Backward pass via backpropagation (**training**) – **learning** the model parameters

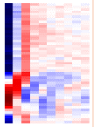
$$y_n = f_n(z_n) = f_n(W_n \cdot y_{n-1})$$

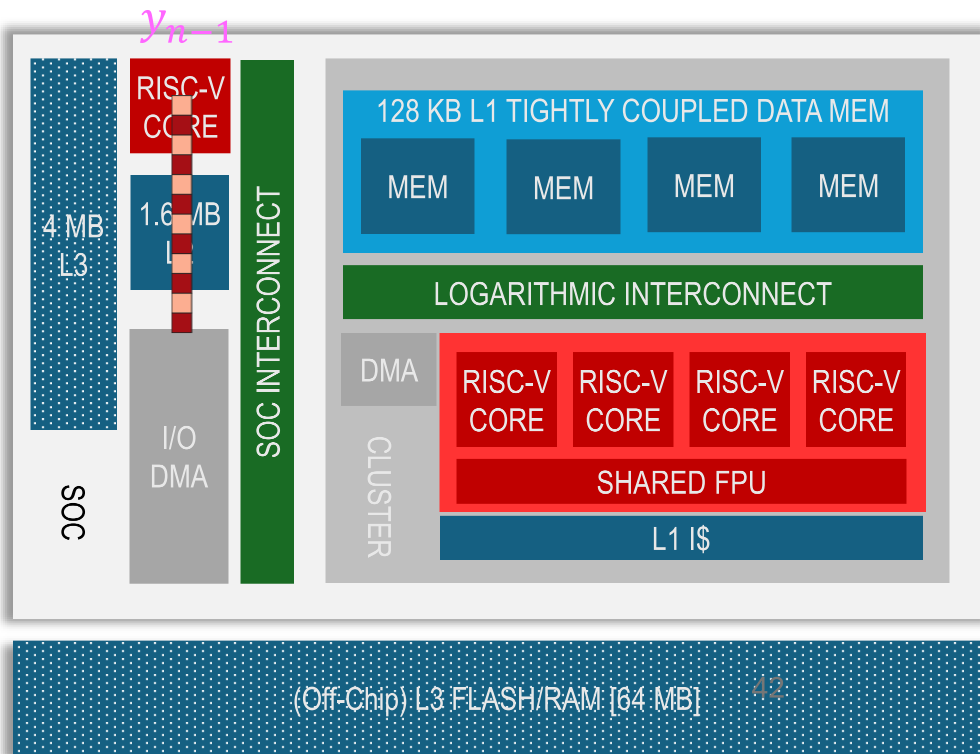
$$L_{MSE} = \frac{1}{2}(y_n(input) - y_{gt})^2$$

$$= W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n}$$

Keep the activations in the memory

$$\frac{\partial z_n}{\partial W_n} = y_{n-1}$$





Backward pass
Compute the gradients (backpropagation)

- Backward pass via backpropagation (**train** model parameters

```
pulp_backbone_fp32_fw_cl(&args);
pulp_linear_fp32_fw_cl(&args);
pulp_MSELoss(&loss_args);
pulp_linear_fp32_bw_cl(&l1_args);
```

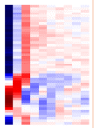
$$y_n = f_n(z_n) = f_n(W_n \cdot y_{n-1})$$

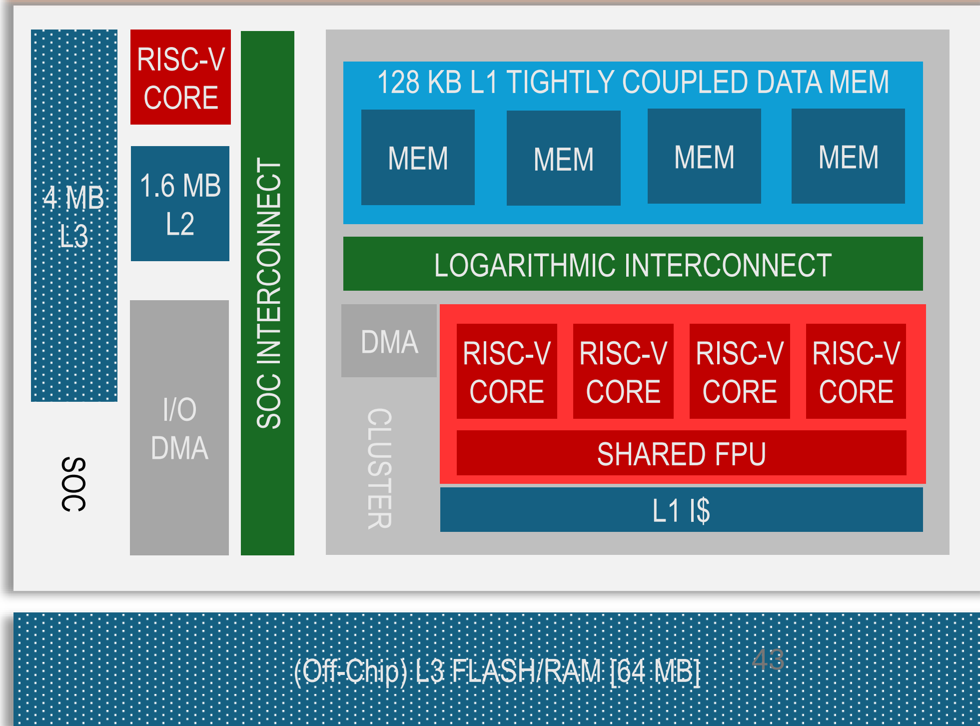
$$L_{MSE} = \frac{1}{S}(y_n(input) - y_{gt})^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n}$$

$$\frac{\partial L}{\partial y_n} = -\frac{2}{S}\left(y_n(input) - y_{gt}\right) = k\left(y_n - y_{gt}\right)$$

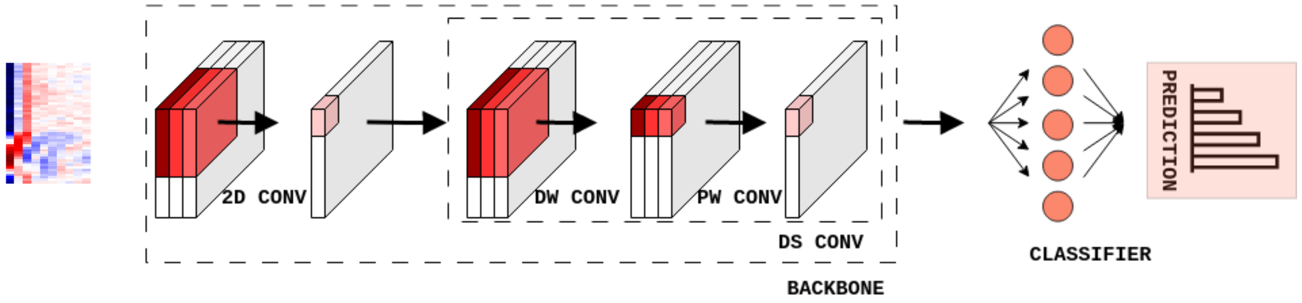
$$\frac{\partial z_n}{\partial W_n} = y_{n-1}$$





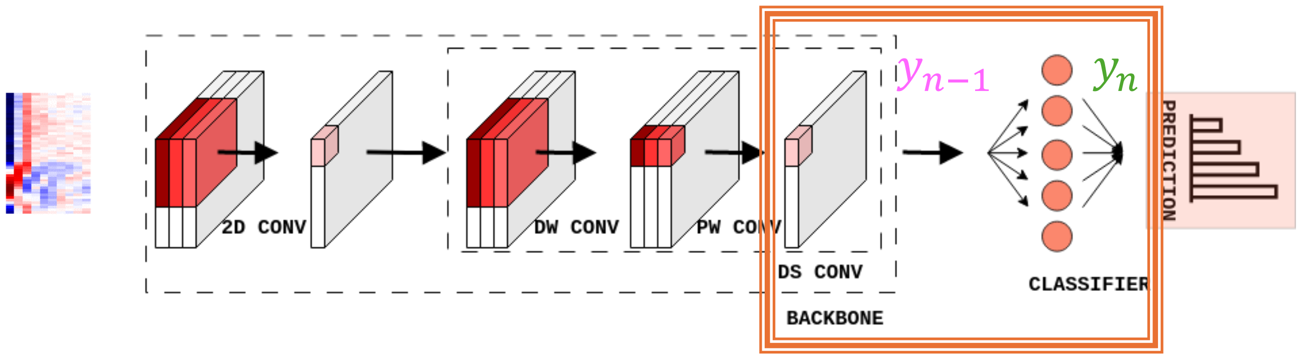
What if we increase the update depth?

$$y_n = f_n\left(W_n \cdot y_{n-1}\right)$$



What if we increase the update depth?

$$y_n = f_n\left(W_n \cdot y_{n-1}\right) = f_n\Big(W_n \cdot f_n\left(W_{n-1} * y_{n-2}\right)\Big)$$

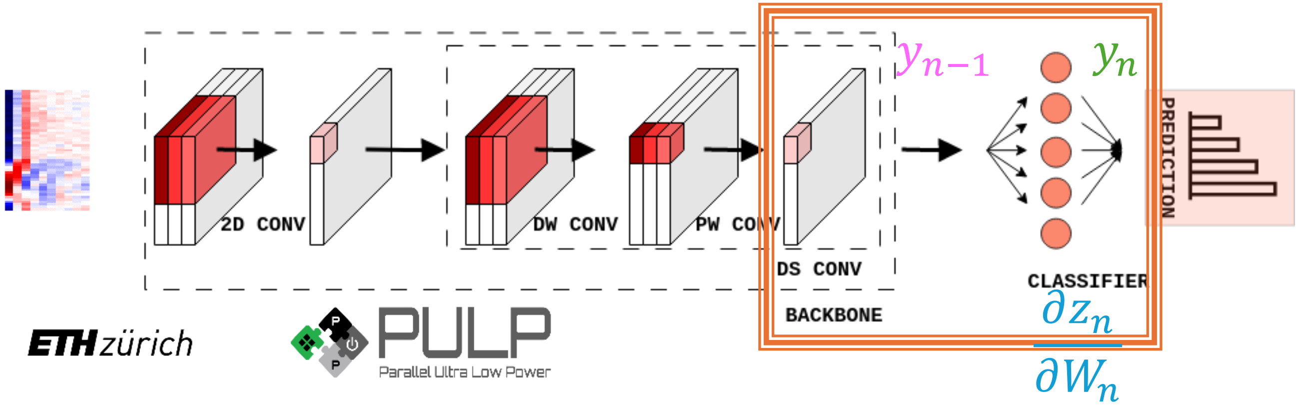


What if we increase the update depth?

$$y_n = f_n\left(W_n \cdot y_{n-1}\right) = f_n\big(W_n \cdot f_n(W_{n-1} * y_{n-2})\big)$$

$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n}$$



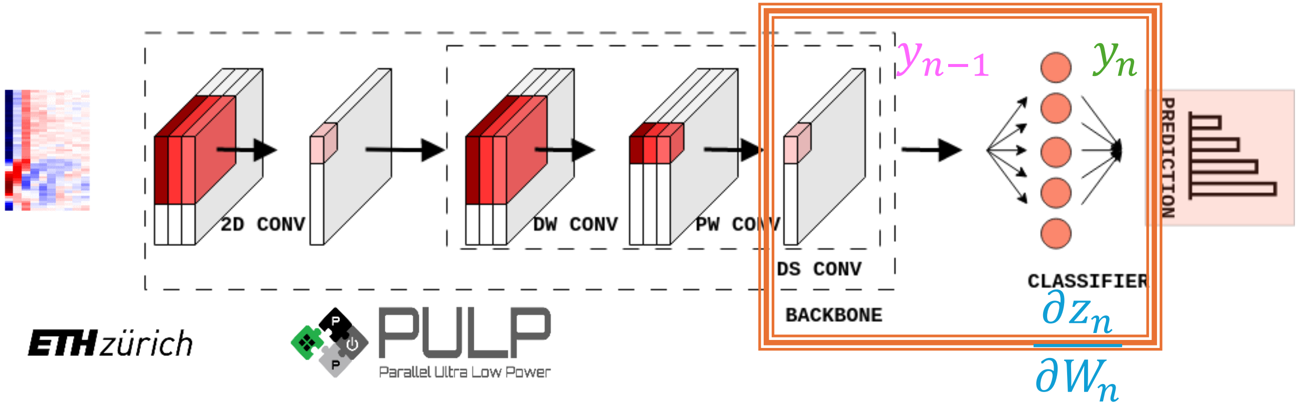
$\frac{\partial L}{\partial y_n}$

What if we increase the update depth?

$$y_n = f_n(W_n \cdot y_{n-1}) = f_n\big(W_n \cdot f_n(W_{n-1} * y_{n-2})\big)$$

$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n} = W_n - \eta \cdot k\big(y_n - y_{gt}\big)y_{n-1}$$



$\frac{\partial L}{\partial y_n}$

What if we increase the update depth?

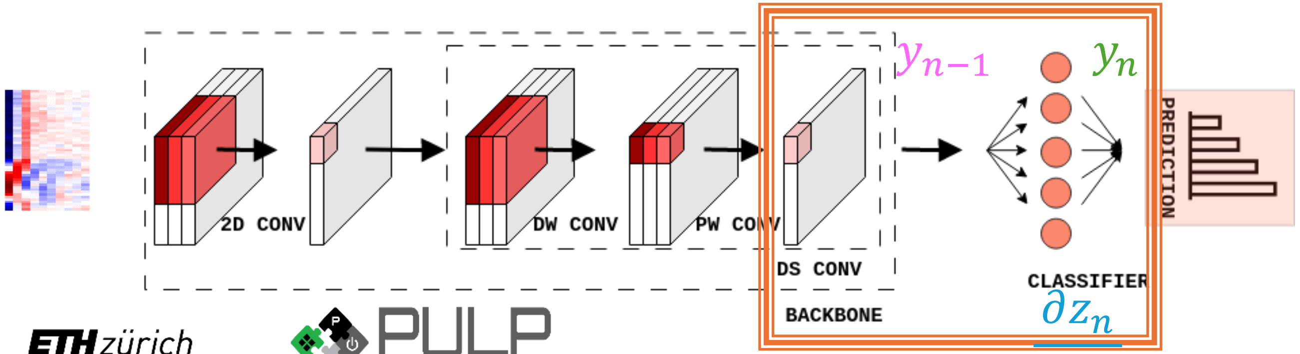
$$y_n = f_n(W_n \ldots) = f(W \ldots f(W_{n-1} * y_{n-2}))$$

$$L_{MSE} = \frac{1}{S}(\ldots$$

Weight gradients

$$W_n' = W_n - \ldots \frac{\partial z_n}{\partial W_n} = W_n - \eta \cdot k(y_n - y_{gt})y_{n-1}$$

$$W_{n-1}' = W_{n-1} - \eta \cdot \frac{\partial L}{\partial W_{n-1}}$$



$$\frac{\partial z_n}{\partial W_n}$$

$$\frac{\partial L}{\partial y_n}$$

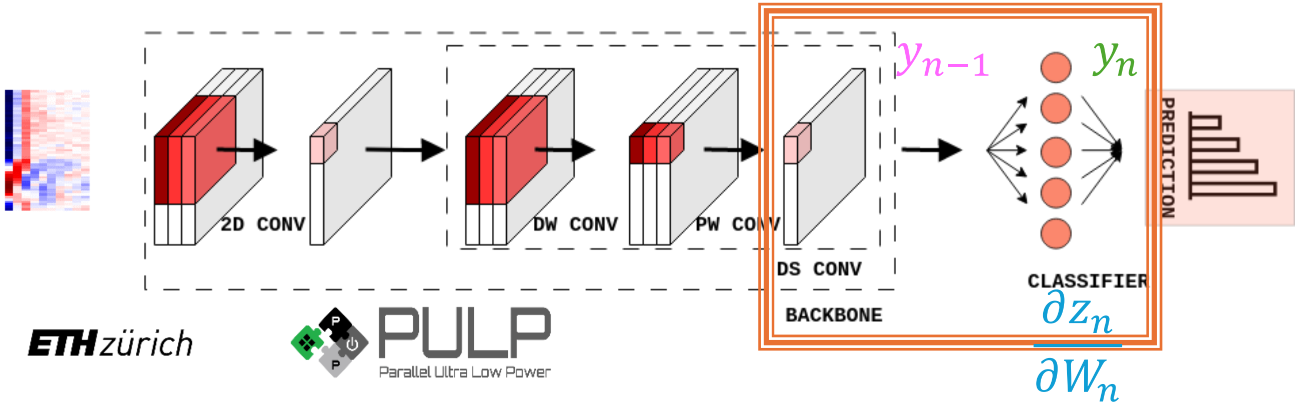
What if we increase the update depth?

$$y_n = f_n(W_n \cdot y_{n-1}) = f_n\big(W_n \cdot f_n(W_{n-1} * y_{n-2})\big)$$

$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n} = W_n - \eta \cdot k\big(y_n - y_{gt}\big)y_{n-1}$$

$$W_{n-1}' = W_{n-1} - \eta \cdot \frac{\partial L}{\partial W_{n-1}} = W_{n-1} - \eta \cdot \frac{\partial L}{\partial y_{n-1}}\frac{\partial z_{n-1}}{\partial W_{n-1}}$$



$\frac{\partial L}{\partial y_n}$

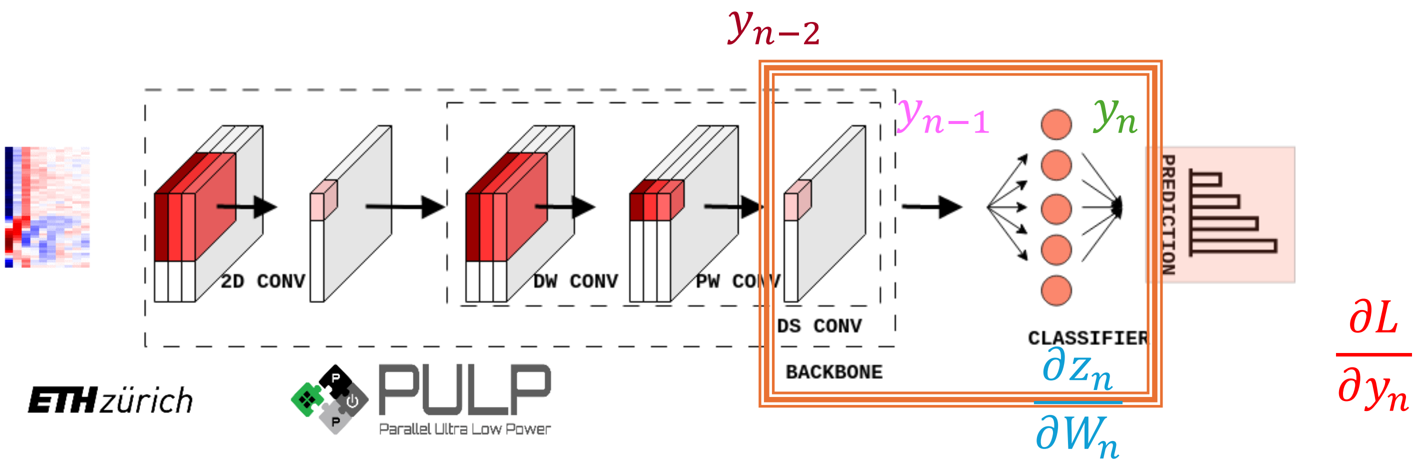
What if we increase the update depth?

$$y_n = f_n\left(W_n \cdot y_{n-1}\right) = f_n\left(W_n \cdot f_n\left(\boldsymbol{W_{n-1}} * y_{n-2}\right)\right)$$

$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n} = W_n - \eta \cdot k\left(y_n - y_{gt}\right)y_{n-1}$$

$$W_{n-1}' = W_{n-1} - \eta \cdot \frac{\partial L}{\partial W_{n-1}} = W_{n-1} - \eta \cdot \frac{\partial L}{\partial y_{n-1}}\frac{\partial z_{n-1}}{\boldsymbol{\partial W_{n-1}}}$$



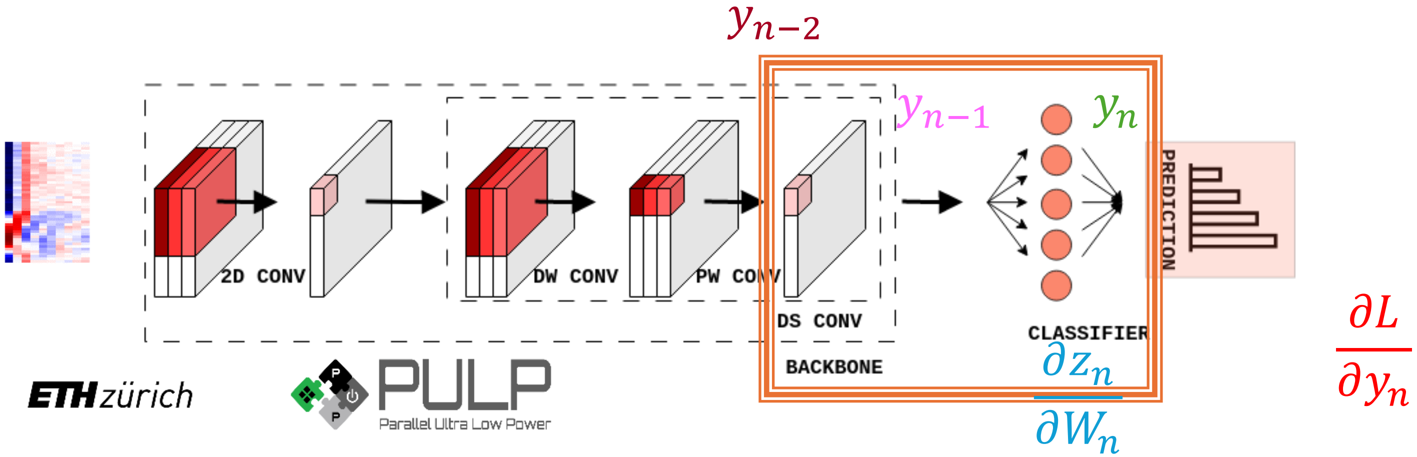
What if we increase the update depth?

$$y_n = f_n(W_n \cdot y_{n-1}) = f_n\big(W_n \cdot f_n(W_{n-1} * y_{n-2})\big)$$

$$L_{MSE} = \frac{1}{S}\,(y_n(input) - y_{gt})^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n} = W_n - \eta \cdot k\big(y_n - y_{gt}\big)y_{n-1}$$

$$W_{n-1}' = W_{n-1} - \eta \cdot \frac{\partial L}{\partial W_{n-1}} = W_{n-1} - \eta \cdot \frac{\partial L}{\partial y_{n-1}}\,y_{n-2}$$



What if we increase the update depth?

$$y_n = f_n(W_n \cdot y_{n-1}) = f_n(W_n \cdot f_n(W \ldots))$$

$$L_{MSE} = \frac{1}{S}(y_n(input) - y_{gt})^2$$

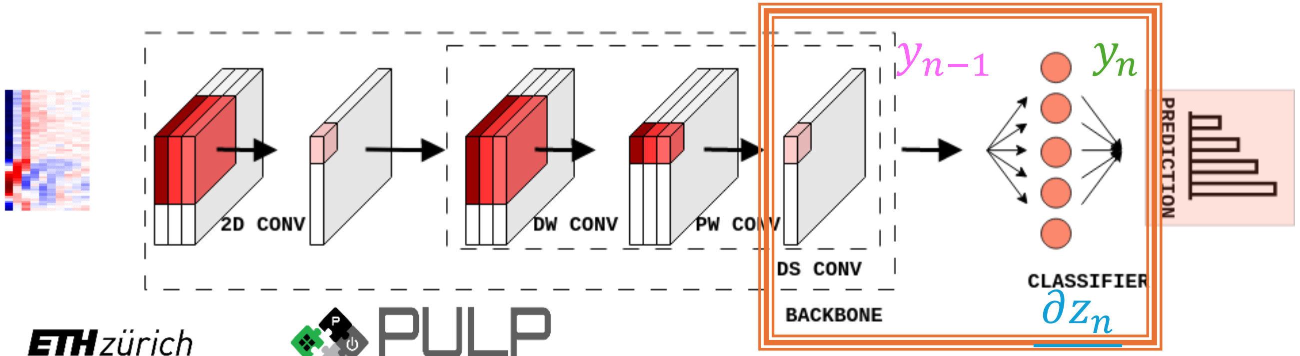
$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \ldots - y_{gt})y_{n-1}$$

$$W_{n-1}' = W_{n-1} - \eta \cdot \frac{\partial L}{\partial W_{n-1}} = W_{n-1} - \eta \cdot \frac{\partial L}{\partial y_{n-1}} y_{n-2}$$

Input gradients

y_{n-2}

y_{n-1} y_n



$\frac{\partial z_n}{\partial W_n}$

$\frac{\partial L}{\partial y_n}$

What if we increase the update depth?

$y_n = f_n(W_n \cdot y$ … $(W_{n-1} * y_{n-2}))$

$L_{MSE} = \frac{1}{S}(y_n($ …

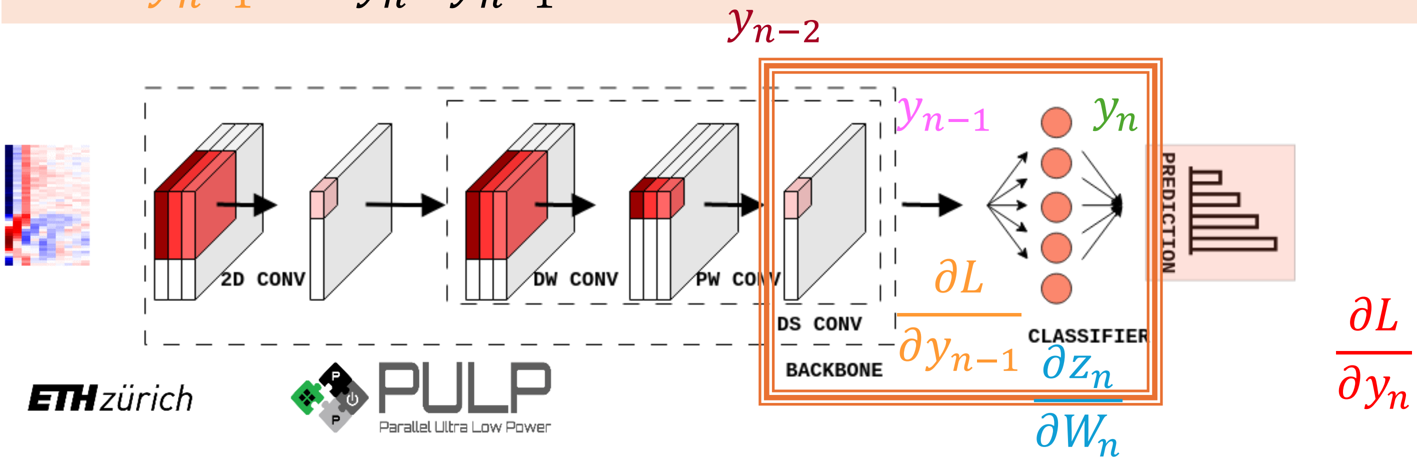
Weight gradients

Input gradients

… η … ∂W_n … $\frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n} = W_n - \eta \cdot k(y_n - y_{gt})y_{n-1}$

… $_{-1} - \eta \cdot \frac{\partial L}{\partial W_{n-1}} = W_{n-1} - \eta \cdot \frac{\partial L}{\partial y_{n-1}} y_{n-2}$

$$\frac{\partial L}{\partial y_{n-1}} = \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial y_{n-1}}$$



What if we increase the update depth?

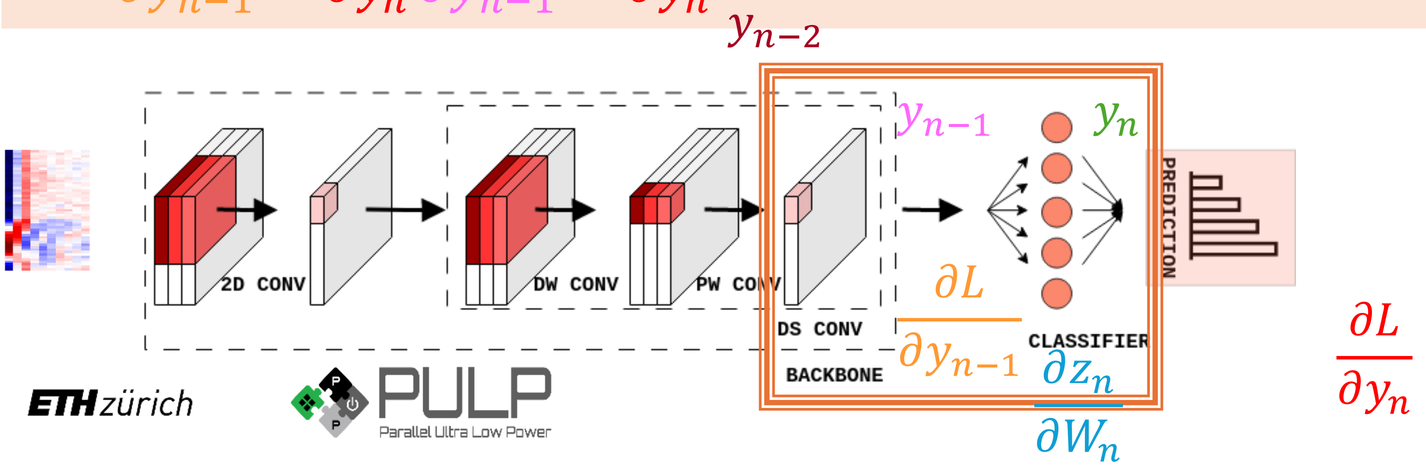
$$y_n = f_n(\boldsymbol{W_n} \cdot y_{n-1}) = f_n\big(W_n \cdot f_n(W_{n-1} * y_{n-2})\big)$$

$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n} = W_n - \eta \cdot k\big(y_n - y_{gt}\big)y_{n-1}$$

$$W_{n-1}' = W_{n-1} - \eta \cdot \frac{\partial L}{\partial W_{n-1}} = W_{n-1} - \eta \cdot \frac{\partial L}{\partial y_{n-1}} y_{n-2}$$

$$\frac{\partial L}{\partial y_{n-1}} = \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial y_{n-1}} = \frac{\partial L}{\partial y_n}\boldsymbol{W_n}$$



What if we increase the update depth?

$$y_n = f_n(\boldsymbol{W_n} \cdot y_{n-1}) = f_n(W_n \cdot f_n(W_{n-1} * y_{n-2}))$$

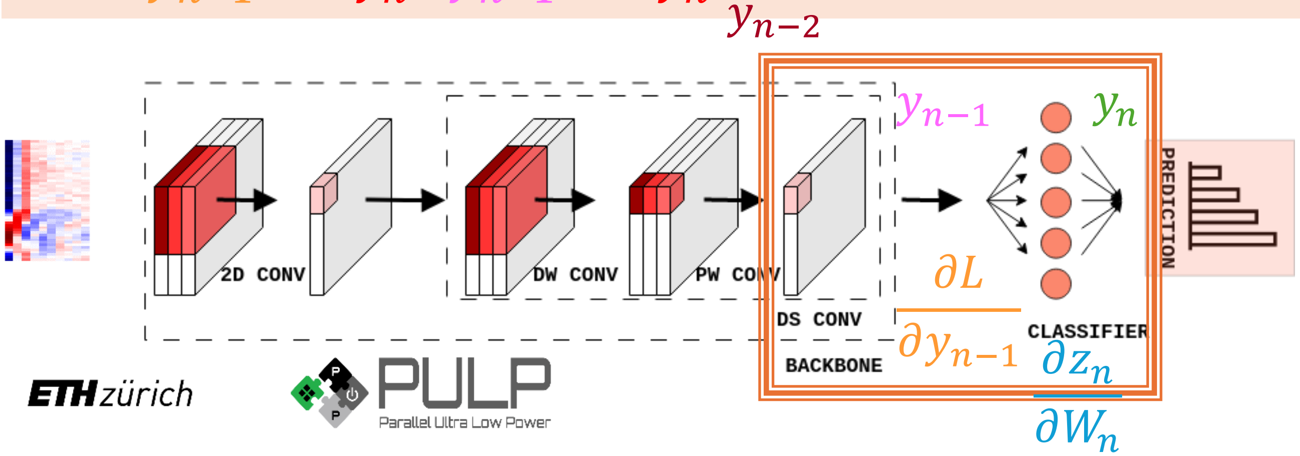
$$L \;\dots = \frac{1}{\dots}(y_n(input) - y_{gt})^2$$

Keep the gradients in the memory

$$\dots W_n - \eta \cdot \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n} = W_n - \eta \cdot k(y_n - y_{gt})y_{n-1}$$

$$\dots \frac{\partial L}{\partial W_{n-1}} = W_{n-1} - \eta \cdot \frac{\partial L}{\partial y_{n-1}} y_{n-2}$$

$$\frac{\partial L}{\partial y_{n-1}} = \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial y_{n-1}} = \frac{\partial L}{\partial y_n}\boldsymbol{W_n}$$



$\frac{\partial L}{\partial y_n}$

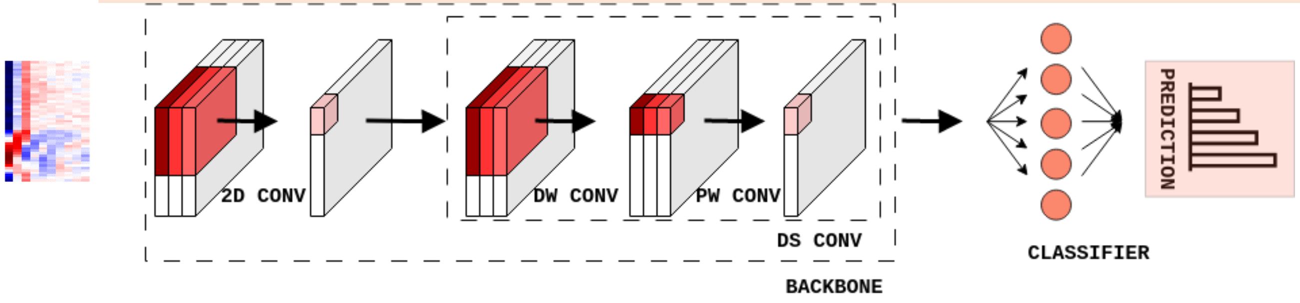
Backward pass
Update the weights

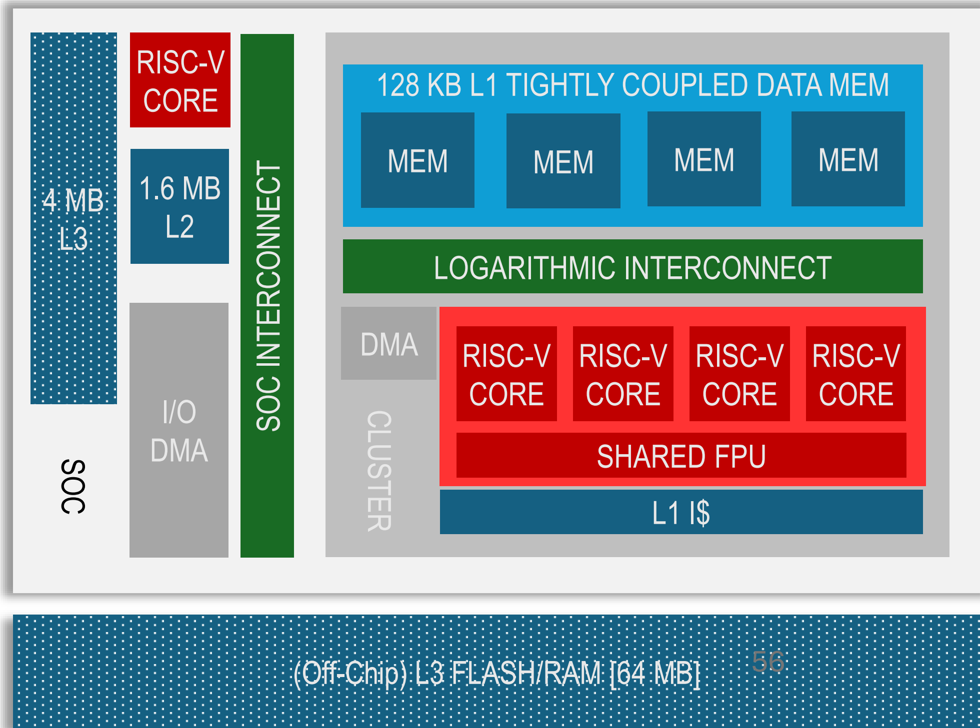
- Backward pass via backpropagation (**training**) – **learning** the model parameters

$$y_n = f_n\left(W_n \cdot y_{n-1}\right)$$

$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot k\left(y_n - y_{gt}\right)y_{n-1}$$





PULP Parallel Ultra Low Power

Backward pass
Update the weights

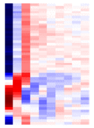
- Backward pass via backpropagation (**train**
  model parameters

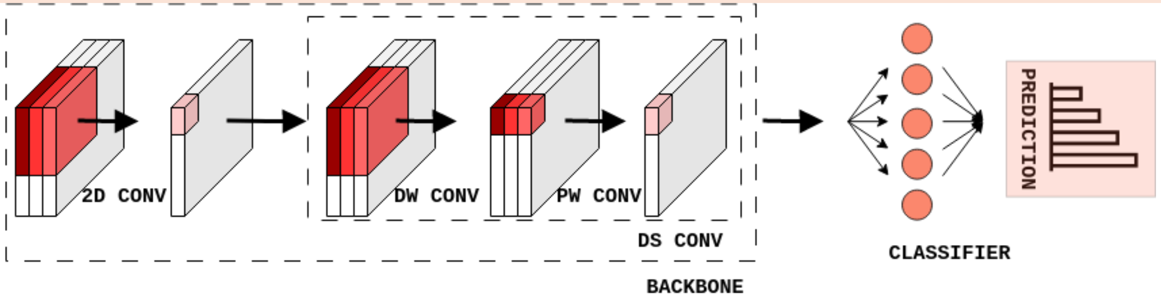
$$y_n = f_n\left(W_n \cdot y_{n-1}\right)$$

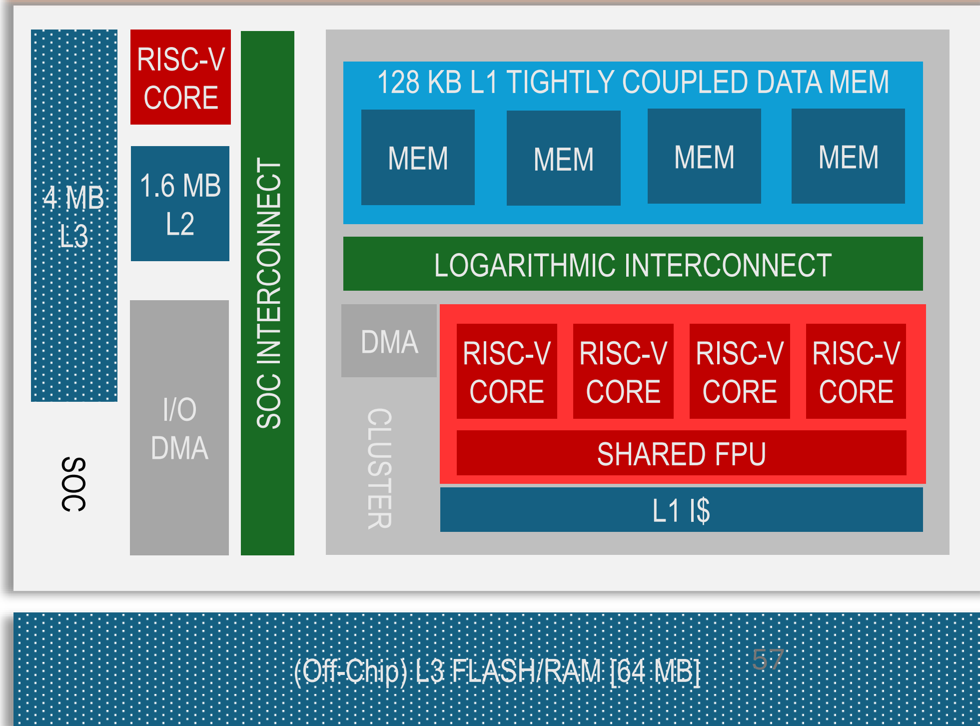
$$L_{MSE} = \frac{1}{S}\left(y_n(input) - y_{gt}\right)^2$$

$$W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n} = W_n - \eta \cdot k\left(y_n - y_{gt}\right) y_{n-1}$$

```
pulp_backbone_fp32_fw_cl(&args);
pulp_linear_fp32_fw_cl(&args);
pulp_MSELoss(&loss_args);
pulp_linear_fp32_bw_cl(&l1_args);
pulp_gradient_descent_fp32(&l1_args);
```



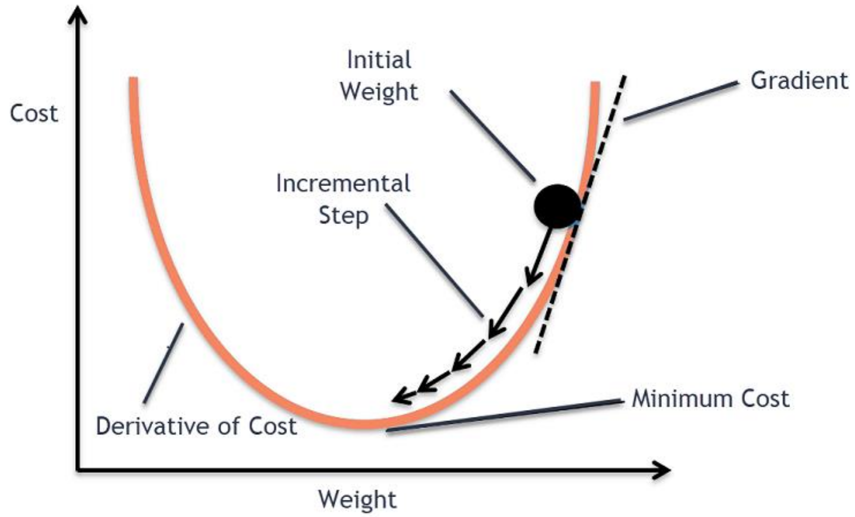




Revisiting Deep Learning – Batched update

Rekha M, 2019

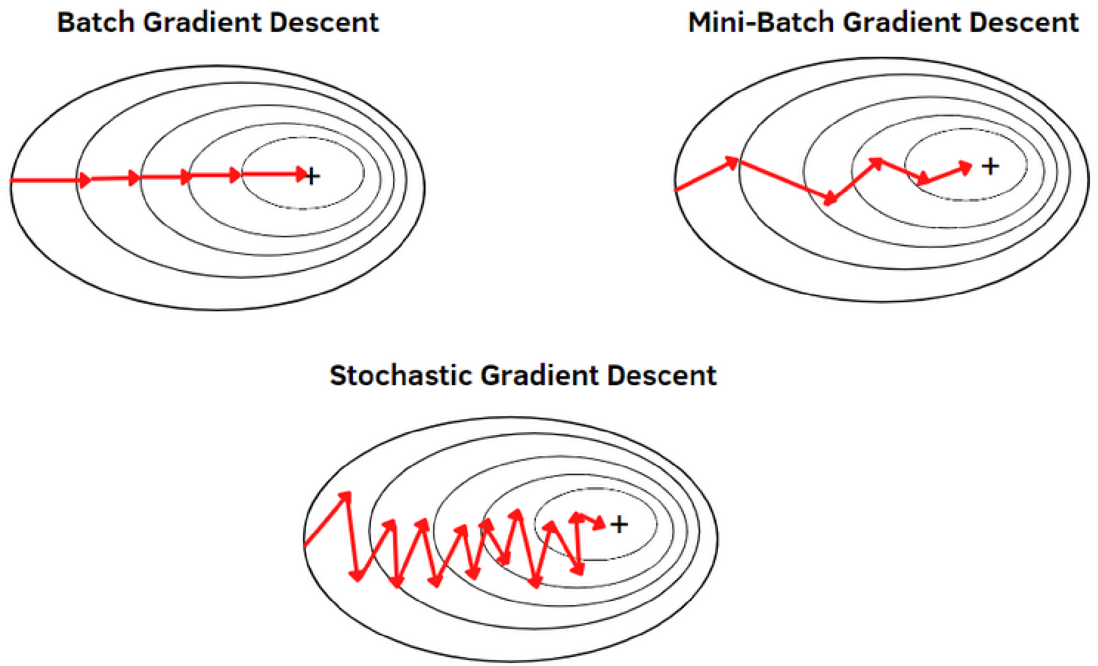
- Update formula is input-dependent
  - $W_n' = W_n - \eta \cdot k\left(y_n(input) - y_{gt}\right) y_{n-1}(input)$



Revisiting Deep Learning – Batched update

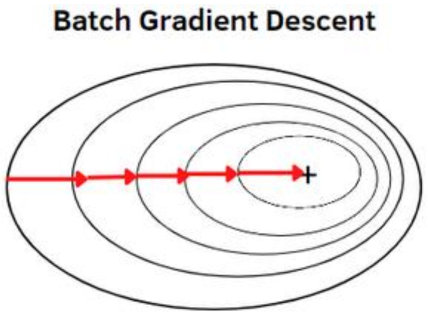
- Update formula is input-dependent
  - $W_n' = W_n - \eta \cdot k\big(y_n(input) - y_{gt}\big)y_{n-1}$

Kurtis Pykes, 2020



Batched Gradient Descent

- Update formula is input-dependent
  - $W_n' = W_n - \eta \cdot k\left(y_n(input) - y_{gt}\right) y_{n-1}(input)$

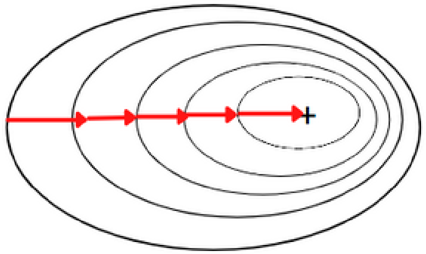


- See all the data simultaneously
- Excellent for smooth manifolds

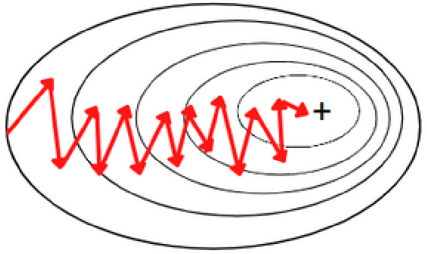
Stochastic Gradient Descent

- Update formula is input-dependent
  - $W_n' = W_n - \eta \cdot k\left(y_n(input) - y_{gt}\right)y_{n-1}(input)$

Batch Gradient Descent



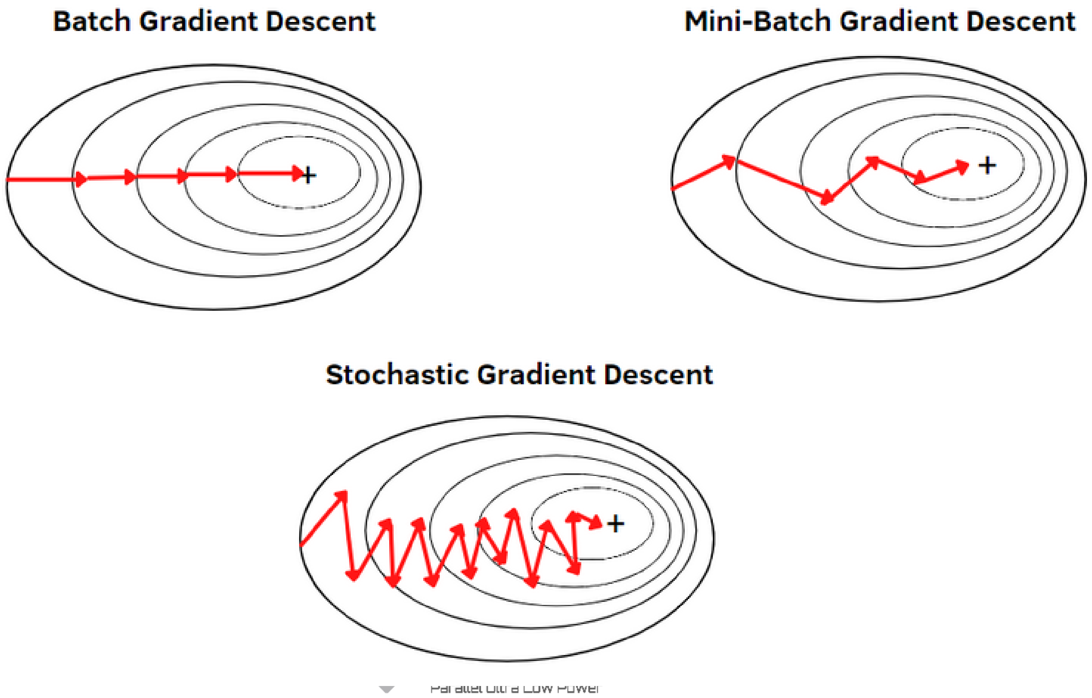
Stochastic Gradient Descent



- See one data at a time
- Minimal memory cost

Mini-Batch Gradient Descent

- Update formula is input-dependent
  - $W_n' = W_n - \eta \cdot k\left(y_n(input) - y_{gt}\right)y_{n-1}(input)$



- See **some** data at a time
- Convergence rate
- Computational and memory efficient

Backpropagation has costs

```
// forward pass
for i in 1, L
```

$$y_i = f_i\left(W_i \cdot y_{i-1} + b_i\right)$$

```
compute loss
// backward pass
for i in L, 1
```

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \cdot y_{i-1}$$

$$\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_{i+1}} \cdot W_i$$

```
update weights
```

Backpropagation has costs

```
// forward pass
for i in 1, L
```

$$y_i = f_i\left(W_i \cdot y_{i-1} + b_i\right)$$

```
compute loss
// backward pass
for i in L, 1
```

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \cdot y_{i-1}$$

$$\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_{i+1}} \cdot W_i$$

```
update weights
```

- Storage
  - Frozen weights
  - Frozen biases

Backpropagation has costs

```
// forward pass
for i in 1, L
```

$$y_i = f_i\,(W_i \cdot y_{i-1} + b_i)$$

```
compute loss
// backward pass
for i in L, 1
```

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \cdot y_{i-1}$$

$$\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_{i+1}} \cdot W_i$$

```
update weights
```

- Storage
  - Frozen weights
  - Frozen biases
- Memory (down to layer l)
  - $\sum_{i=l}^{L} W_i$ (weights)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial y_{i+1}}$ (input grads)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial W_i}$ (weight grads)
  - $\sum_{i=l-1}^{L} W_i$(activations)

Backpropagation has costs

```
// forward pass
for i in 1, L
```

$$y_i = f_i(W_i \cdot y_{i-1} + b_i)$$

```
compute loss
// backward pass
for i in L, 1
```

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \cdot y_{i-1}$$

$$\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_{i+1}} \cdot W_i$$

```
update weights
```

$$Memory_{total} = Memory_{sample=1} x \frac{\#\ samples}{\#\ batches}$$

- Memory (down to layer l)
  - $\sum_{i=l}^{L} W_i$ (weights)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial y_{i+1}}$ (input grads)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial W_i}$ (weight grads)
  - $\sum_{i=l-1}^{L} W_i$(activations)

Backpropagation has costs

```
// forward pass
for i in 1, L
```

$$y_i = f_i(W_i \cdot y_{i-1} + b_i)$$

```
compute loss
// backward pass
for i in L, 1
```

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \cdot y_{i-1}$$

$$\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_{i+1}} \cdot W_i$$

```
update weights
```

- Storage
  - Frozen weights
  - Frozen biases
- Memory (down to layer l)
  - $\sum_{i=l}^{L} W_i$ (weights)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial y_{i+1}}$ (input grads)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial W_i}$ (weight grads)
  - $\sum_{i=l-1}^{L} W_i$ (activations)
- Operations (latency)
  - forward pass
  - gradients
  - update ↓↓↓

Backpropagation has costs

```
// forward pass
for i in 1, L
```
$$y_i = f_i(W_i \cdot y_{i-1} + b_i)$$
```
compute loss
// backward pass
for i in L, 1
```
$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \cdot y_{i-1}$$
$$\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_{i+1}} \cdot W_i$$
```
update weights
```

- Storage
  - Frozen weights
  - Frozen biases
- Memory (down to layer l)
  - $\sum_{i=l}^{L} W_i$ (weights)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial y_{i+1}}$ (input grads)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial W_i}$ (weight grads)
  - $\sum_{i=l-1}^{L} W_i$(activations)
- Operations (latency)
  - forw
  - grad
  - upd

forward pass +
input gradients +
weight gradients
≅ 3 forward passes

Backpropagation has costs

```
// forward pass
for i in 1, L
```

$$y_i = f_i(W_i \cdot y_{i-1} + b_i)$$

```
compute loss
// backward pass
for i in L, 1
```

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \cdot y_{i-1}$$

$$\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_{i+1}} \cdot W_i$$

```
update weights
```

- Storage
  - Frozen weights
  - Frozen bias
- Memory (dow
  layer l)
  - $\sum_{i=l}^{L} W_i$ (weights)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial y_{i+1}}$ (input grads)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial W_i}$ (weight grads)
  - $\sum_{i=l-1}^{L} W_i$ (activations)
- Operations (latency)

forward pass +
input gradients +
weight gradients
≅ 3 forward passes x #data x #epochs

Backpropagation has costs

```
// forward pass
for i in 1, L
```
$$y_i = f_i(W_i \cdot y_{i-1} + b_i)$$
```
compute loss
// backward pass
for i in L, 1
```
$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial y_i} \cdot y_{i-1}$$
$$\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_{i+1}} \cdot W_i$$
```
update weights
```

- Storage
  - Frozen weights
  - Frozen biases
- Memory (down to layer l)
  - $\sum_{i=l}^{L} W_i$ (weights)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial y_{i+1}}$ (input grads)
  - $\sum_{i=l}^{L} \frac{\partial L}{\partial W_i}$ (weight grads)
  - $\sum_{i=l-1}^{L} W_i$ (activations)

- Operations (latency)
  - forward pass
  - gradients
  - update ↓↓↓
- Energy
  - $E =$
    $= P \cdot t =$
    $= P \cdot Eff \cdot f \cdot OPs$

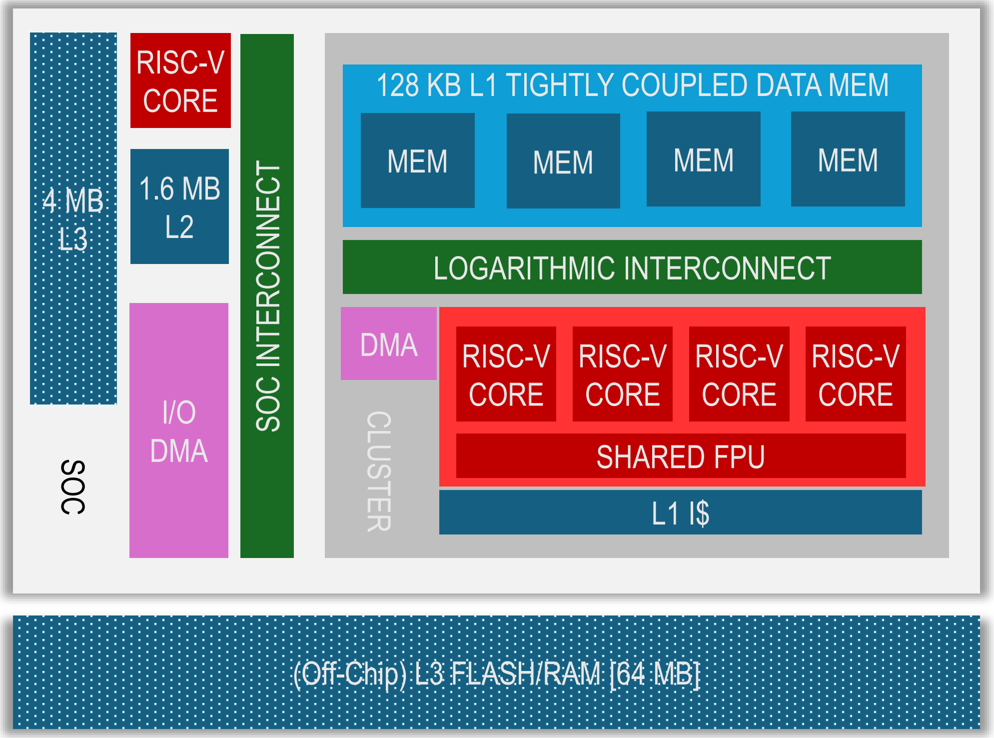
Backpropagation has costs

Storage
- Frozen weights
- Frozen biases

Memory (down to layer l)
- $\sum_{i=l}^{L} W_i$ (weights)
- $\sum_{i=l}^{L} \frac{\partial L}{\partial y_{i+1}}$ (input grads)
- $\sum_{i=l}^{L} \frac{\partial L}{\partial W_i}$ (weight grads)
- $\sum_{i=l-1}^{L} W_i$(activations)

Operations (latency)
- forward pass
- gradients
- update ↓↓↓

Energy
- $E =$
  $= P \cdot t =$
  $= P \cdot Eff \cdot f \cdot OPs$



Accuracy

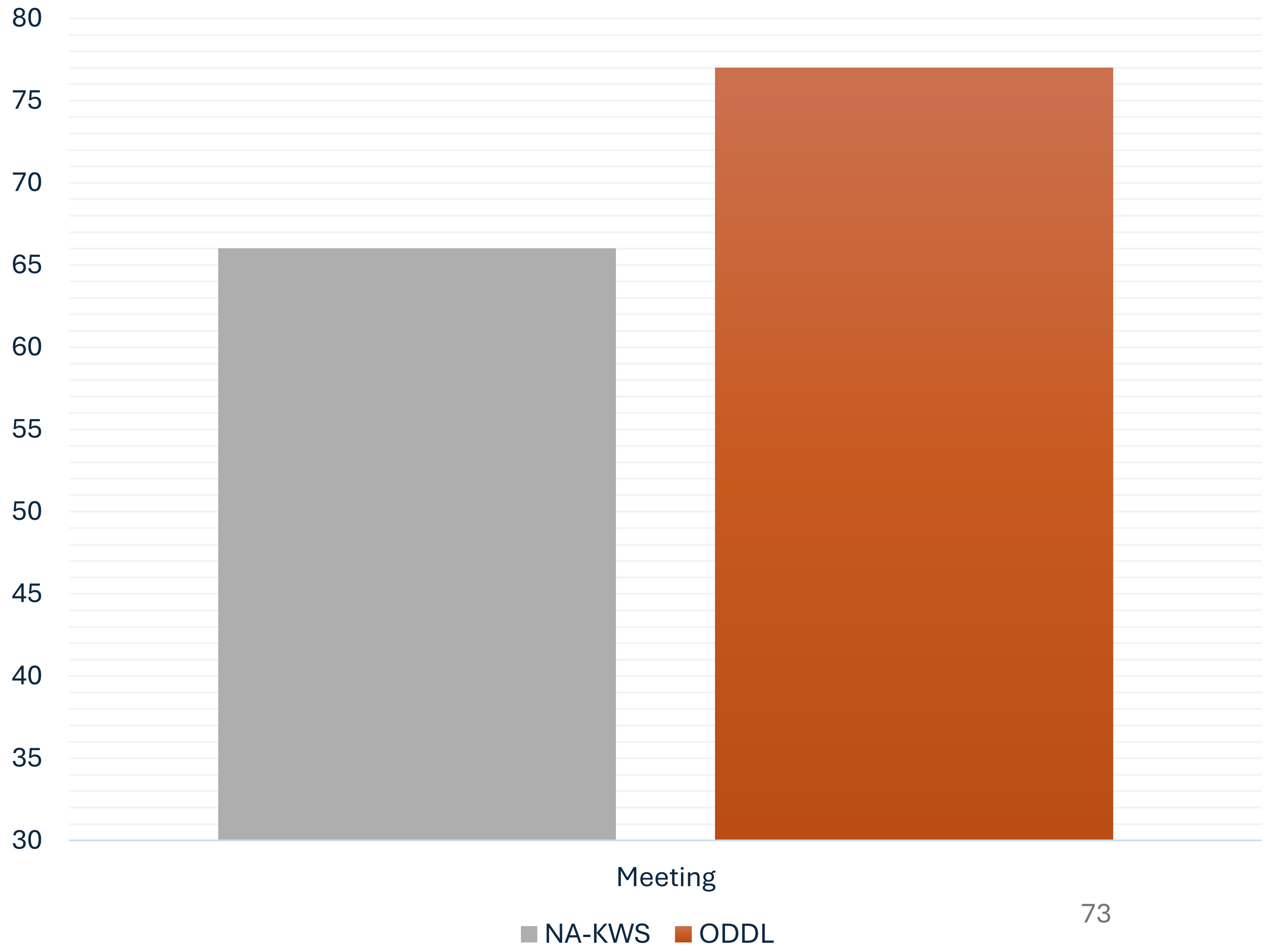
How do learning costs impact an On-Device Learning application?

Results

Improving the KWS accuracy in noisy environments

- **Accuracy increments** by 4% on average over **NA-KWS**
- 13% on **speech noise**

- Does it work well under embedded constraints?



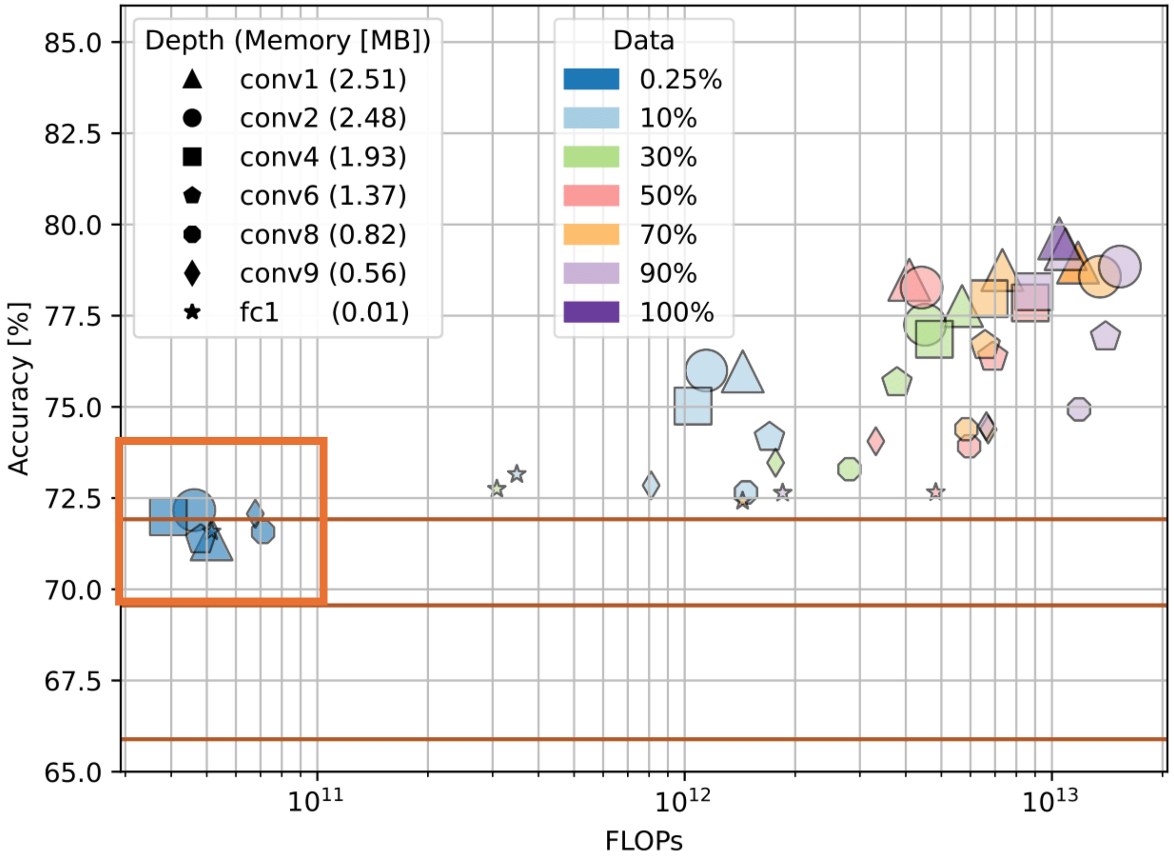
ETH zürich

PULP Parallel Ultra Low Power

Results

Memory requirements for effective learning

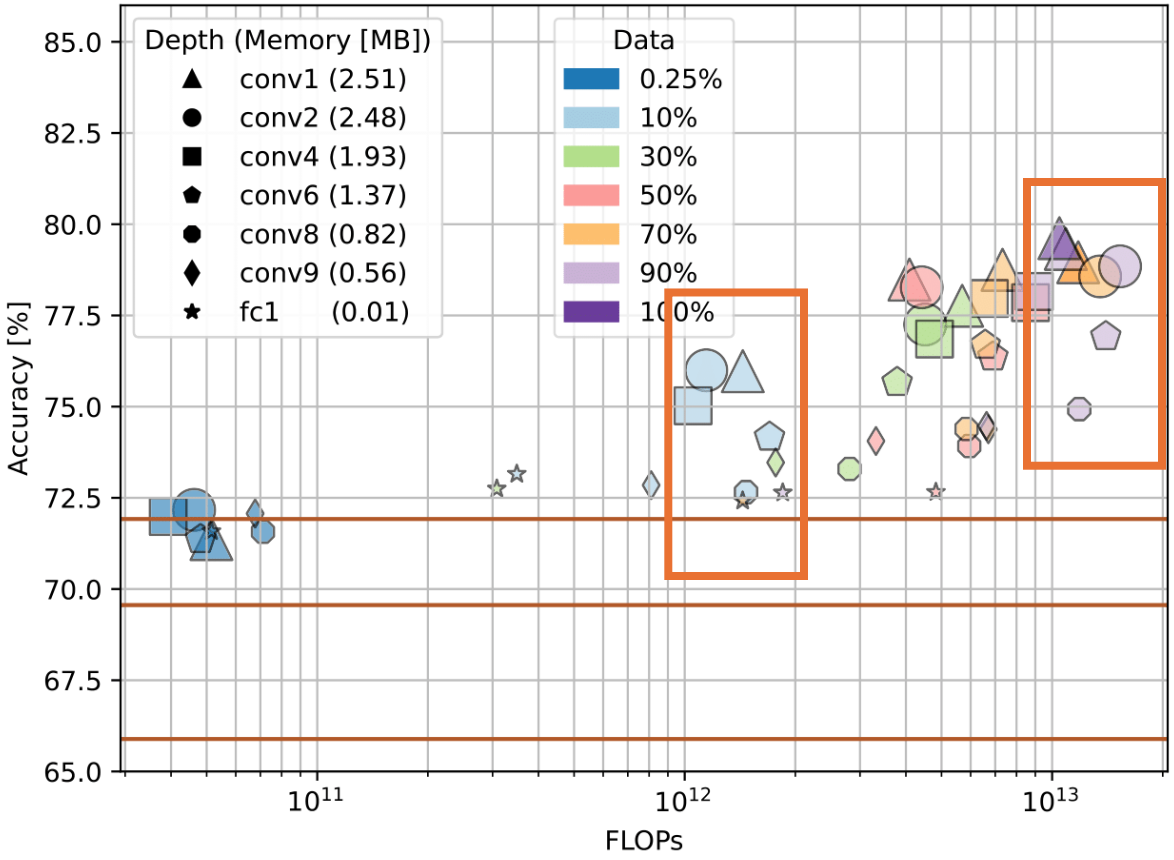
- **Refine *fc₁* layer**
  - 10 kB on-chip L1 memory
  - $S_{ODDA} = S_{NA\text{-}KWS}$ **+ 5.5%**



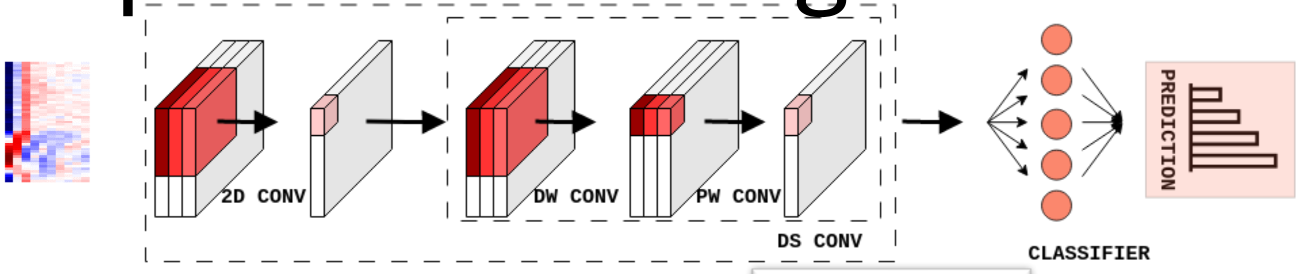
Results

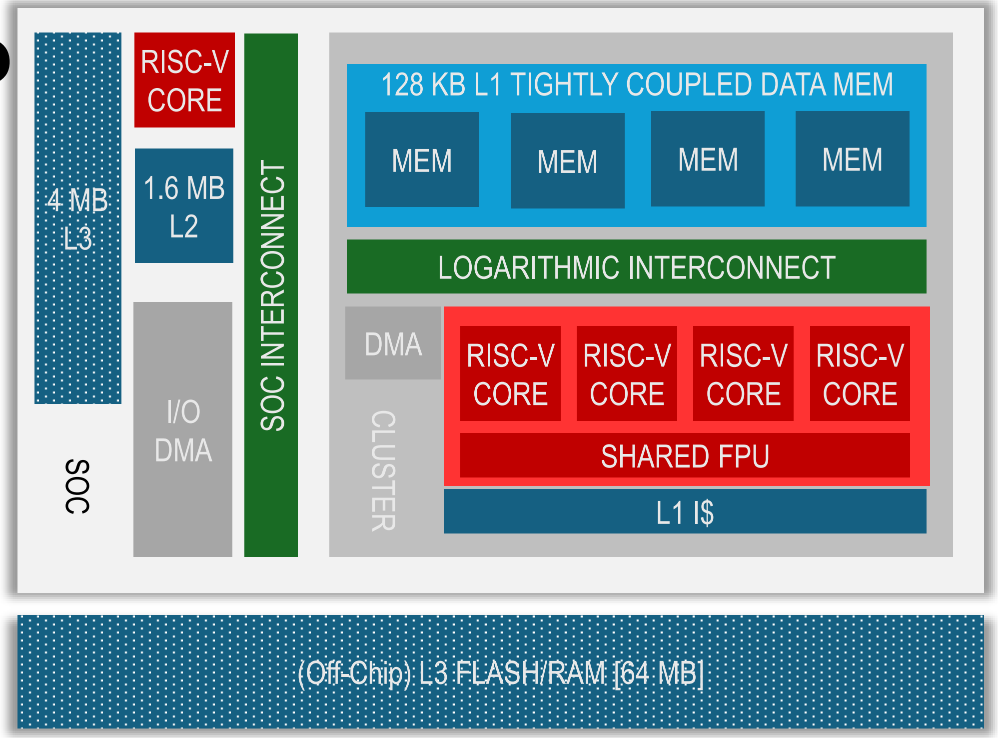
Memory requirements for effective learning

- **Refine fc_1 layer**
  - 10 kB on-chip L1 memory
  - $S_{ODDA} = S_{NA\text{-}KWS}$ **+ 5.5%**
- Refine backbone and classif.
  - +1.2% over *fc1* update using 10% of pre-recorded samples
  - +6% over *fc1* update using 100% of pre-recorded samples



On-Device Learning on PULP platforms using TrainLib





1. $y_{n-1} = f_{n-1}\,(input)$
2. $y_n = f_n\,(W_n \cdot y_{n-1} + b_n)$
3. $L = -\sum_{cls} y_{gt_{cls}} \log(y_{n_{cls}})$
4. $\frac{\partial L}{\partial W_n} = \frac{\partial L}{\partial y_n}\frac{\partial z_n}{\partial W_n}$
5. $W_n' = W_n - \eta \cdot \frac{\partial L}{\partial W_n}$

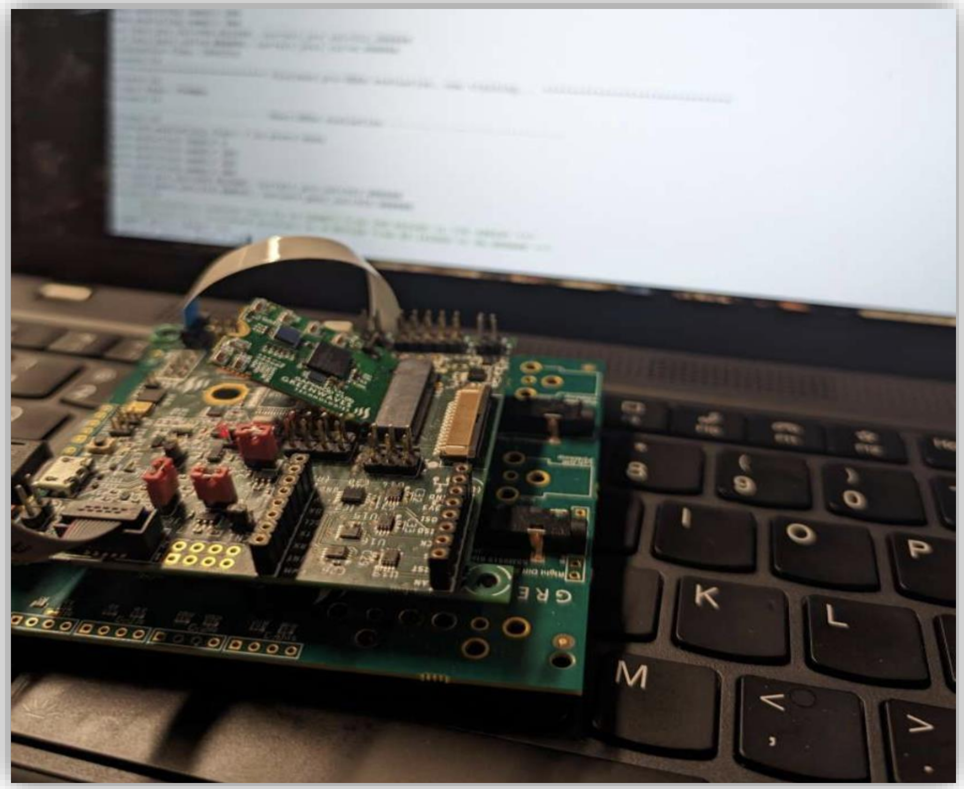
```
pulp_backbone_fp32_fw_cl(&args);
pulp_linear_fp32_fw_cl(&args);
pulp_CrossEntropyLoss(&loss_args);
pulp_linear_fp32_bw_cl(&l1_args);
pulp_gradient_descent_fp32(&l1_args);
```

Memory management

- Pre-recorded (clean) utterances and labels stored in L3 memory
- Pretrained model stored in L3 memory
- Weights, activations stored in L2 memory during forward pass
- Weights, activations, gradients stored in L1 during backward pass

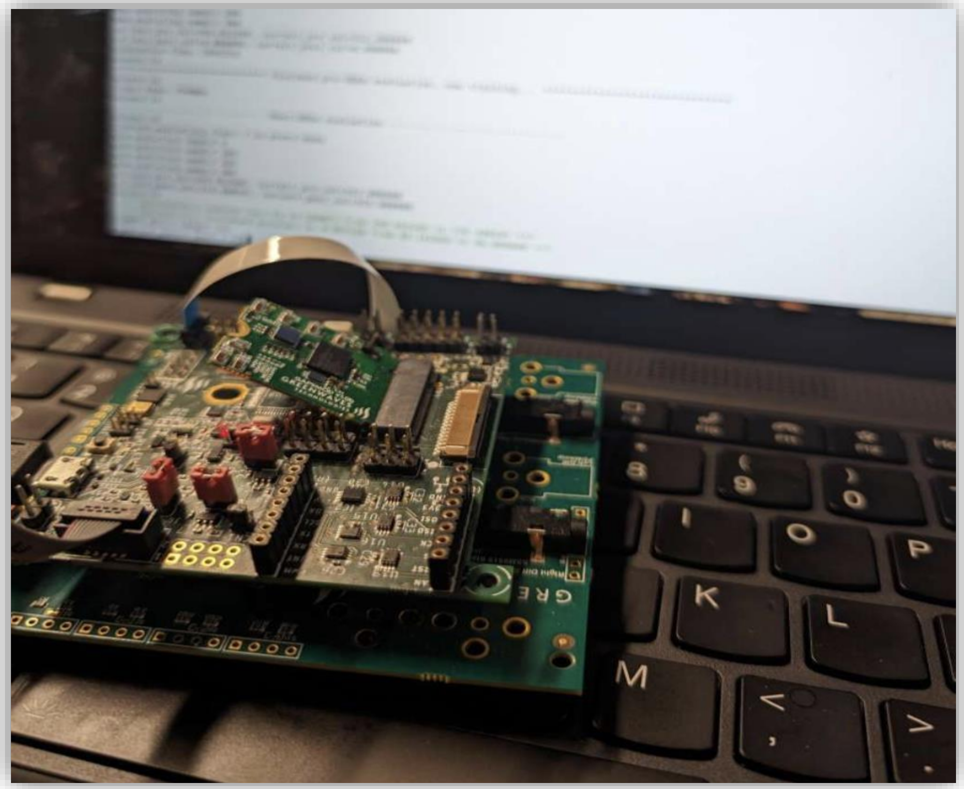
Results
Implementation on GAP 9

- Greenwaves GAP9 – based on PULP Vega [Rossi2021]
- Low-power mode: 240 MHz, 650 mV



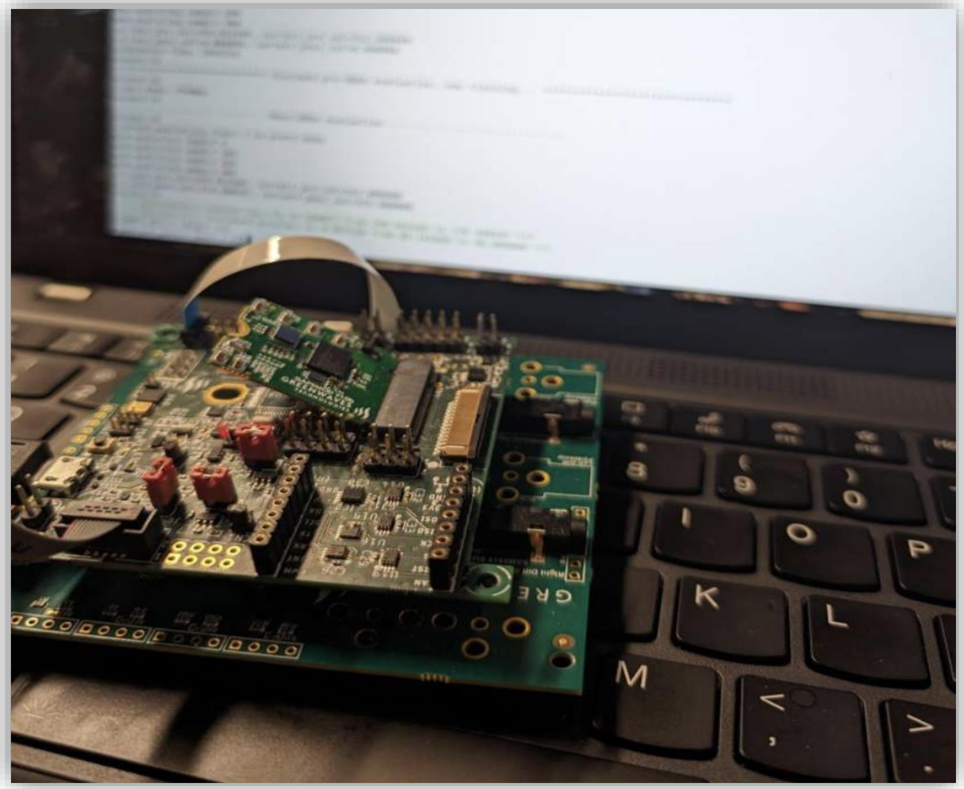
Results
Implementation on GAP 9

- Greenwaves GAP9 – based on PULP Vega [Rossi2021]
- Low-power mode: 240 MHz, 650 mV
  - On-device learning in **½ mJ**, ready in **11 ms**



DS-CNN Model	Compute [MFLOps]	Storage [kB]	Memory [kB]	Eff. [FLOPs/ cycle]	Compute time [ms]	Energy [µJ]
S	2.95	23.7	9.5	4.94	**10.89**	**424**
M	17.2	138.1	25.5	9.18	24.16	988
L	51.1	416.7	40.9	11	55.04	2313

Results
Implementation on GAP 9

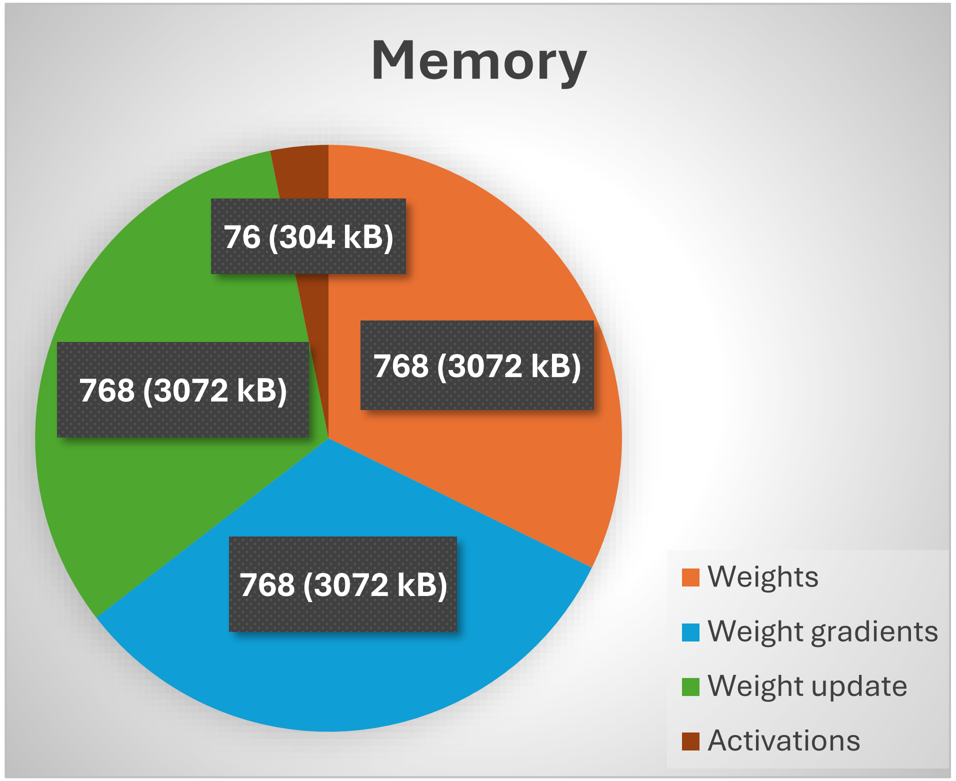


- Greenwaves GAP9 – based on PULP Vega [Rossi2021]
- Low-power mode: 240 MHz, 650 mV
  - On-device learning in ½ **mJ**, ready in **11 ms**
  - **10 kB** of L1 memory for backpropagation

DS-CNN Model	Compute [MFLOps]	Storage [kB]	Memory [kB]	Eff. [FLOPs/cycle]	Compute time [ms]	Energy [µJ]
S	2.95	23.7	**9.5**	4.94	10.89	424
M	17.2	138.1	25.5	9.18	24.16	988
L	51.1	416.7	40.9	11	55.04	2313

Results
Implementation on GAP 9

- Greenwaves GAP9 – based on PULP Vega [Rossi2021]
- Low-power mode: 240 MHz, 650 mV
  - On-device learning in **½ mJ**, ready in **11 ms**
  - **10 kB** of L1 memory for backpropagation



DS-CNN Model	Compute [MFLOps]	Storage [kB]	Memory [kB]	Eff. [FLOPs/cycle]	Compute time [ms]	Energy [μJ]
S	2.95	23.7	**9.5**	4.94	10.89	424
M	17.2	138.1	25.5	9.18	24.16	988
L	51.1	416.7	40.9	11	55.04	2313

Conclusions

- Backpropagation at the extreme edge is expensive
  - Storage, memory, operations, latency
  - Embrace accuracy-complexity trade-offs

- Demonstrated On-Device Domain Adaptation on GAP9
  - On-Device Learning for keyword spotting in *speech* noise
  - +6% over NA-KWS in extreme-edge conditions
  - 424 µJ per epoch for DS-CNN S
  - 10 kB of memory for backpropagation

References

[Cioflan2024] C. Cioflan, L. Cavigelli, M. Rusci, M. De Prado and L. Benini, "On-Device Domain Learning for Keyword Spotting on Low-Power Extreme Edge Embedded Systems," *2024 IEEE 6th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*

[Frey2022] S. Frey, S. Vostrikov, L. Benini and A. Cossettini, "WULPUS: a Wearable Ultra Low-Power Ultrasound probe for multi-day monitoring of carotid artery and muscle activity," *2022 IEEE International Ultrasonics Symposium (IUS)*, Venice, Italy, 2022, pp. 1-4

[Frey2023] S. Frey, M. Guermandi, S. Benatti, V. Kartsch, A. Cossettini and L. Benini, "BioGAP: a 10-Core FP-capable Ultra-Low Power IoT Processor, with Medical-Grade AFE and BLE Connectivity for Wearable Biosignal Processing," *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, Berlin, Germany, 2023, pp. 1-7

[Kalenberg2024] K. Kalenberg *et al*., "Stargate: Multimodal Sensor Fusion for Autonomous Navigation On Miniaturized UAVs," in *IEEE Internet of Things Journal (Early access)*

[Rossi2022] D. Rossi *et al*., "Vega: A Ten-Core SoC for IoT Endnodes With DNN Acceleration and Cognitive Wake-Up From MRAM-Based State-Retentive Sleep Mode," in *IEEE Journal of Solid-State Circuits*, vol. 57, no. 1, pp. 127-139, Jan. 2022

Cristian Cioflan

cioflanc@iis.ee.ethz.ch