UNIVERSITY of VIRGINIA

Design Space Exploration for Compressed Deep Convolutional Neural Network on SCALE Sim

Peilin Chen, Xinyuan Fu, Hanyuan Gao, Feilian Dai

6501 Group6

2024/12/05

➢ The hardware platform we use

- Systolic CNN AcceLErator Simulator (SCALE Sim)

Parameter	Value
Array Height	32
Array Width	32
IFMAP SRAM Size	1024
Filter SRAM Size	1024
OFRAM SRAM Size	128
Dataflow	WS

Config file with architecture specs

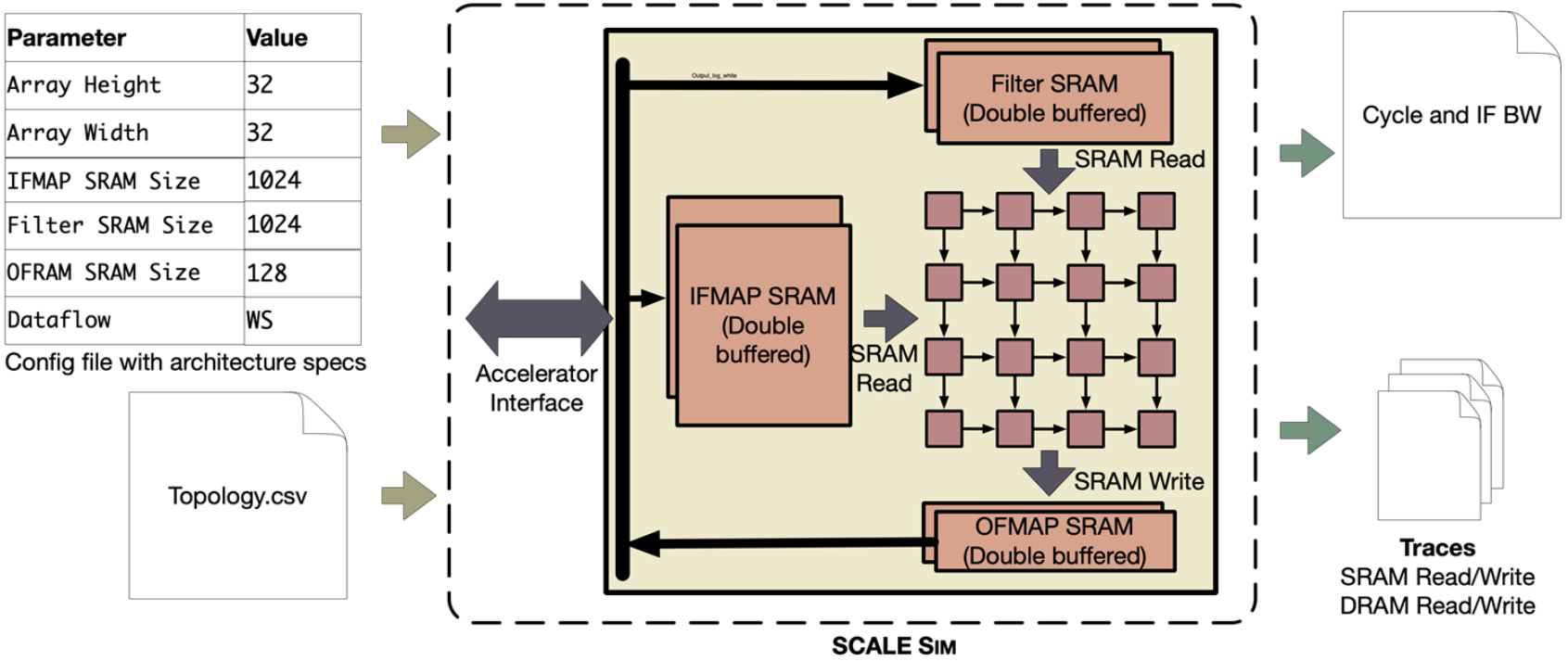


Fig. Overview of SCALE Sim

SCALE Sim takes two files as input from the user: one is the hardware configuration, the other is the neural network topology (workload).

SCALE Sim generates two types of outputs: one is the cycle accurate traces for SRAM and DRAM, the other is the metrics like cycle counts, utilization, bandwidth requirements, total data movement, etc.

https://github.com/scalesim-project/scale-sim-v2?tab=readme-ov-file

Samajdar, Ananda, et al. "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim." 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2020.

➢ NN Quantization

- LeNet-5 <-> Hand-written Digit Recognition

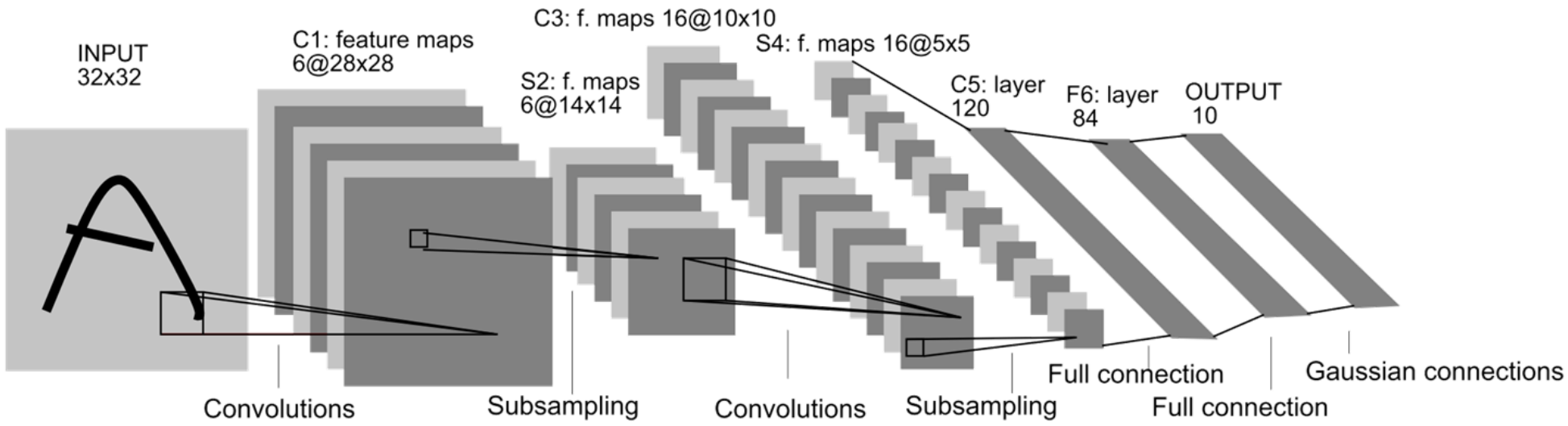


Fig. LeNet-5 Structure.

LeNet-5:
three Convolutional layers, two Pooling layers, two Fully-connected layers.

Parameter size: C1(1×5×5×6+6) + C3(6×5×5×16+16) + C5(5×5×16×120+120) + F6(120×84) + OUTPUT(84×10) = **61,686**

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

➢ NN Quantization

- Model definition

```python
import torch
from torch import nn

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()

        self.c1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, padding=2)
        self.Relu = nn.ReLU()
        self.s2 = nn.AvgPool2d(kernel_size=2, stride=2)
        self.c3 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.s4 = nn.AvgPool2d(kernel_size=2, stride=2)
        self.c5 = nn.Conv2d(in_channels=16, out_channels=120, kernel_size=5)
        self.flatten = nn.Flatten()
        self.f6 = nn.Linear(120, 84)
        self.output = nn.Linear(84, 10)

    def forward(self, x):
        x = self.Relu(self.c1(x))
        x = self.s2(x)
        x = self.Relu(self.c3(x))
        x = self.s4(x)
        x = self.c5(x)
        x = self.flatten(x)
        x = self.f6(x)
        x = self.output(x)
        return x

if __name__ == "__main__":
    x = torch.rand([1, 1, 28, 28])
    model = LeNet()
    y = model(x)
```

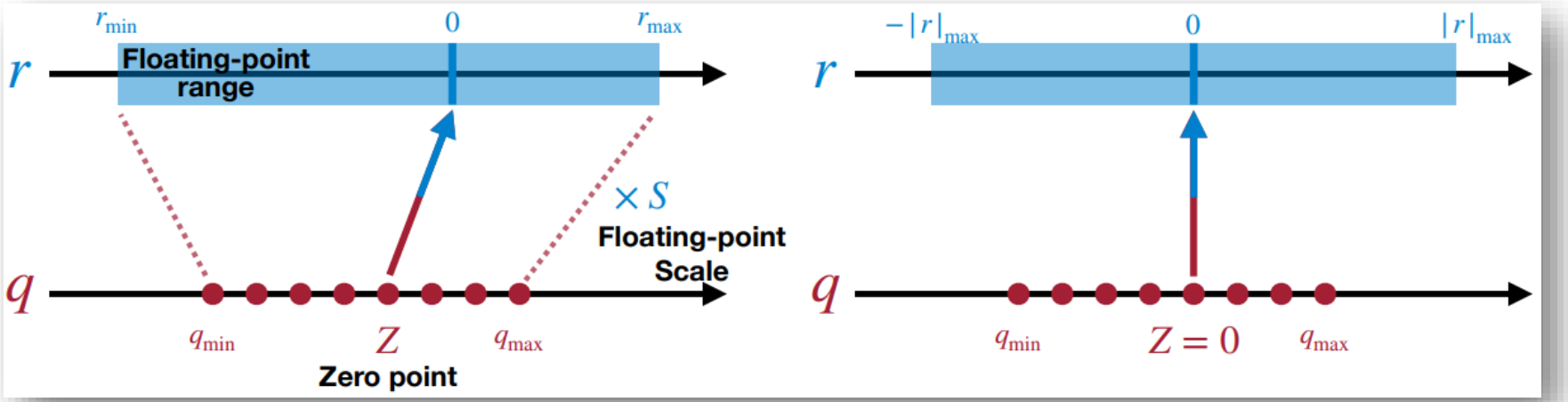

LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

➢ NN Quantization

■ Linear asymmetric quantization

Linear quantization method:

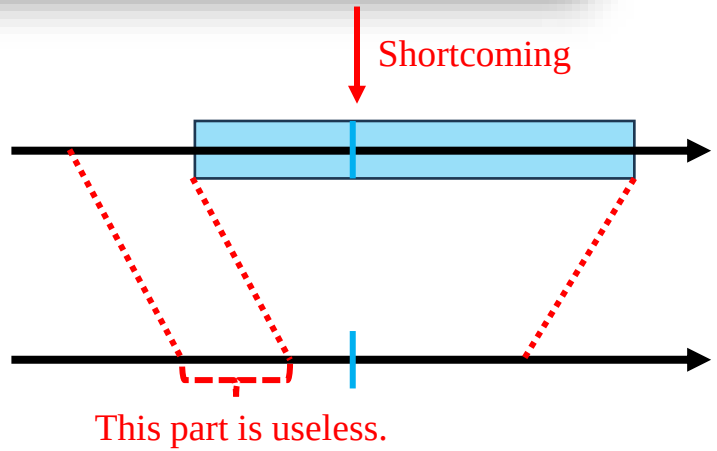
1. Symmetric Quantization
2. Asymmetric Quantization



https://hanlab.mit.edu/courses/2023-fall-65940.

Asymmetric Quantization: Better than symmetric quantization because … But more complex.

Symmetric Quantization: Simple and easy to be implemented in hardware design. But it has a shortcoming.



➢ NN Quantization

■ Training and Quantization results

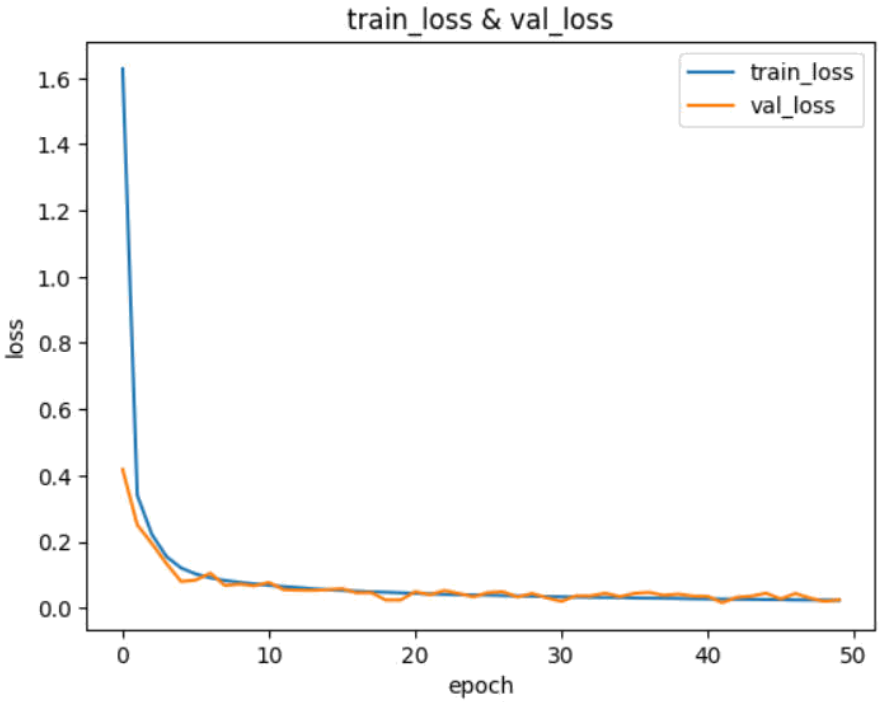


Fig. train_loss & val_loss.

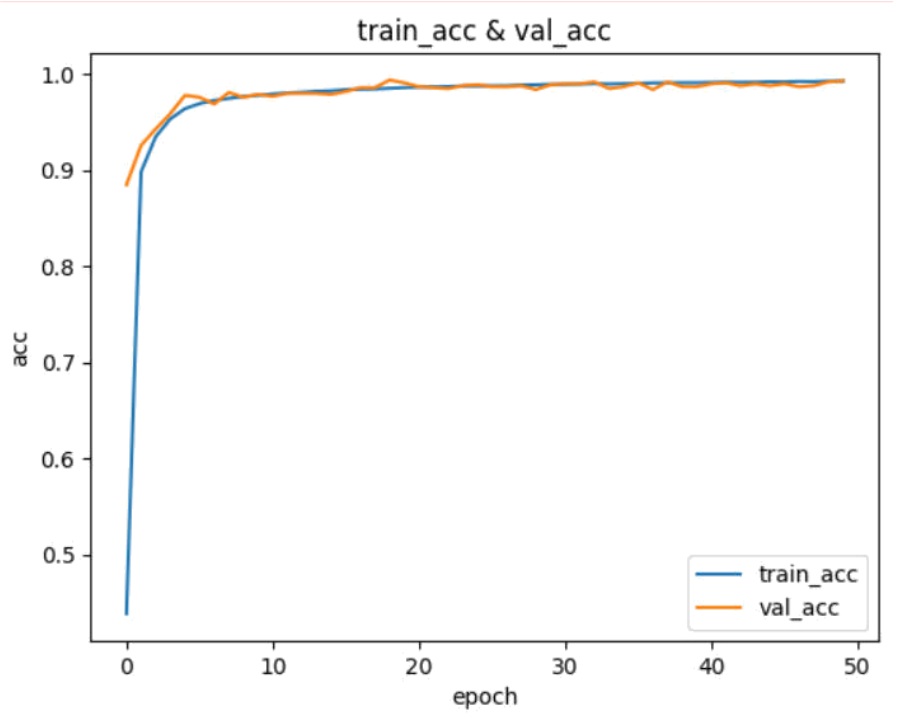


Fig. train_acc & val_acc.

```
(LLM) bash-4.4$python test_quant.py
Original model test:
Accuracy: 98.56%
####################
Quantized int8 model test:
Accuracy: 98.82%
(LLM) bash-4.4$
```

Fig. Original model vs. Quantized model.

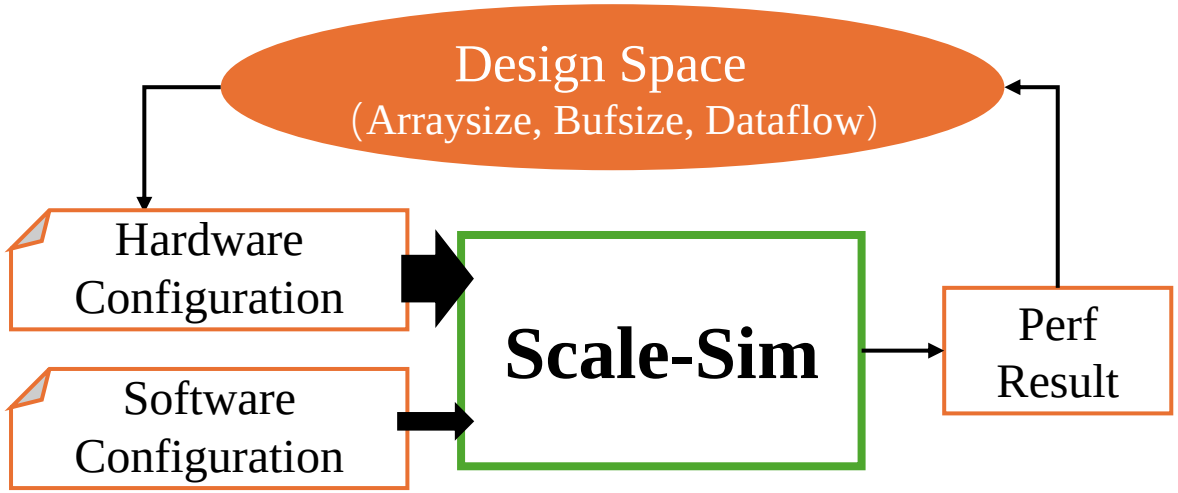
The training and test batch size are 32 and 1000, respectively.
The learning rate and momentum are 0.001 and 0.9, respectively.

Compared to the original LeNet-5 model, the quantized int8 LeNet-5 model achieves **no accuracy loss**.

■ Design Space Exploration

Design Space Exploration

- Definition: Given a network, optimize the hardware (Systolic Array) configuration (array size, dataflow)
- Object: Performance (Cycles, the sim only provides the perf result)
- Variable/Design Space: [Array Height, Array Width, Dataflow Type]



$$\boldsymbol{max\ Perf(HW; SW)}$$

$$HW = \{Height, Width, Bufsize, Dataflow\}, SW = \{layers\}$$

$$Height, Width, Bufsize \in N^{+}, Dataflow \in \{'ws', 'os', 'is'\}$$

$$Perf\ is\ from\ the\ Simulator$$

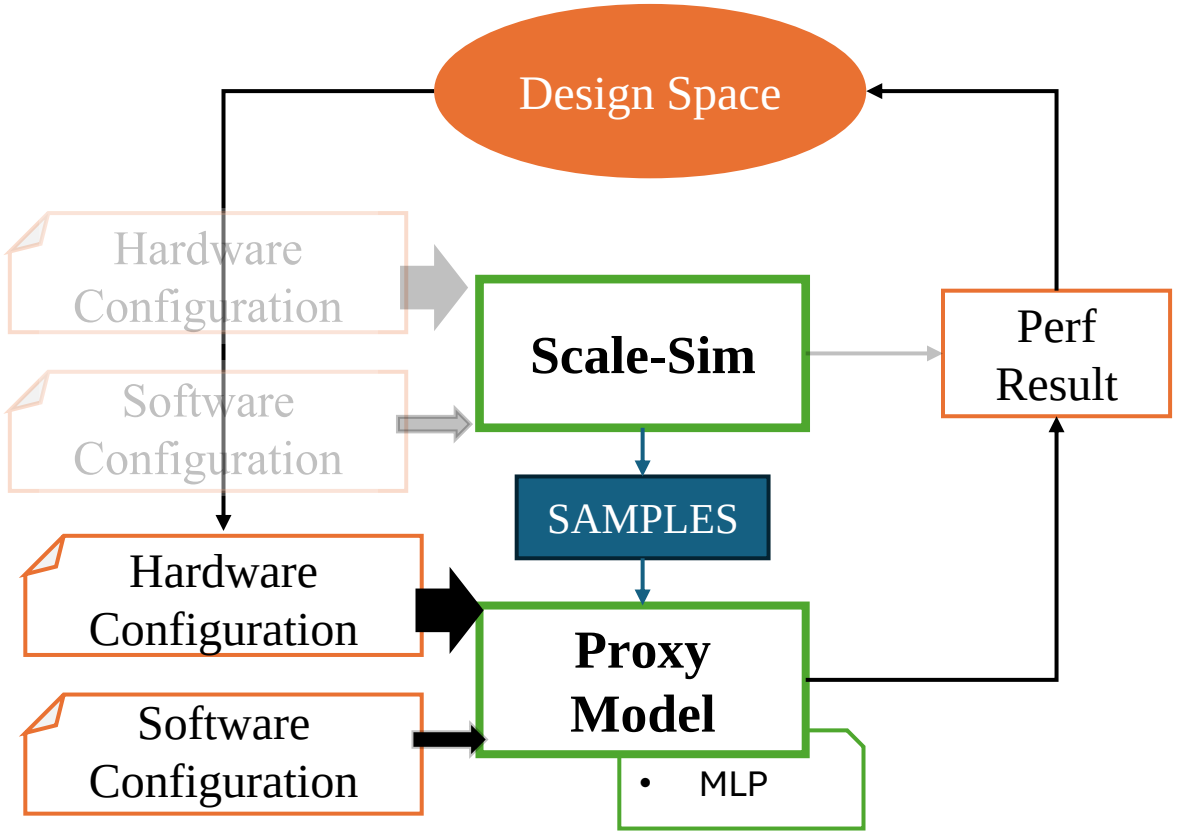
➢ Design Space Exploration

■ Design Space Exploration

For small space
- Brute Force Search √

For large design space,
the exploration time is huge (hours)

How to make it fast?
- Search fast (Genetic, Simulated annealing …).
- Estimate fast (A proxy estimator)
  - Machine Learning √
  - Analytical Model (Timeloop, Maestro)



AI FOR AI HW!

➢ Design Space Exploration Result

■ Design Space Exploration Result

- The accuracy is shown later.
- Search (Single Layer Conv, PEs=64) Result

```
Running Layer 0
100%|████████████████████████████████████████████████████████████████████████| 213/213 [00:00<00:00, 4104.07it/s]
Compute cycles: 212
Stall cycles: 0
Overall utilization: 1.08%
Mapping efficiency: 10.94%
Average IFMAP DRAM BW: 10.000 words/cycle
Average Filter DRAM BW: 10.000 words/cycle
Average OFMAP DRAM BW: 9.800 words/cycle
************ SCALE SIM Run Complete ****************
============================END============================
16
{'ArrayHeight': 8, 'ArrayWidth': 8, 'IfmapSramSzkB': 36, 'FilterSramSzkB': 36, 'OfmapSramSzkB': 36, 'IfmapOffset': 0, 'FilterOffset': 10000000, 'OfmapOffset': 2000
0000, 'Dataflow': 'os', 'Bandwidth': 10, 'MemoryBanks': 1}
Time cost: 1.6377918720245361s
```

- Time Cost Result
  - Mins to < 1sec.

➢ Proxy model

■ MLP Network

Why we need a proxy:

- **Motivation: SIM takes too much time,**
- **Example: 500k design points cost 150+hours;**

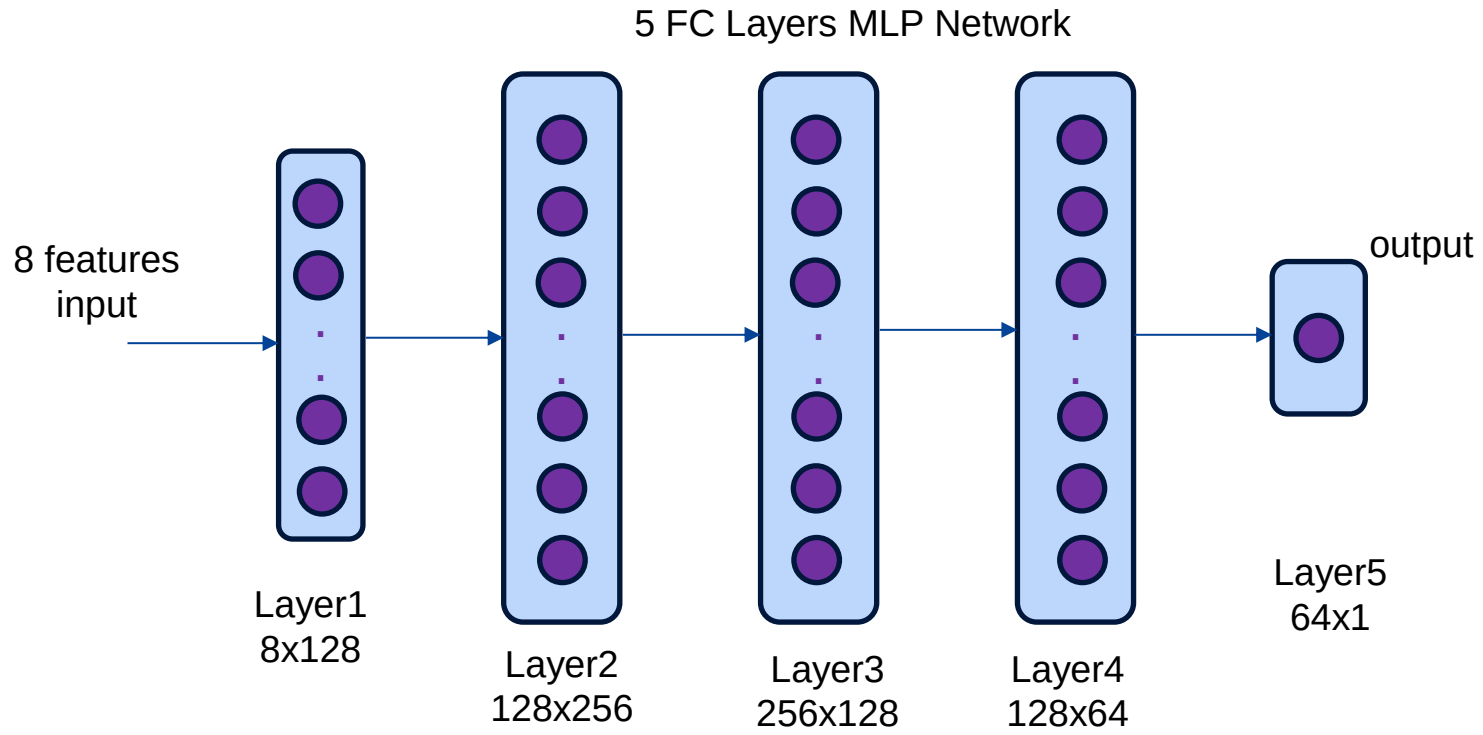
Why we choose MLP:

- **The task is a High-dimensional space function fitting task rather than a classification task;**
- **MLP is a smooth model and a good fit for later searching;**
- **MLP has a simple structure and we need a simple structure to prevent overfitting;**

Parameters:

- **Width(systolic array)**
- **Hight(systolic array)**
- **Dataflow type(ws,os,is)**
- **Buffersize**
- **Software(mat mal)**
- **M,N,K(matrix)**

Output is a number representing cycles



➢ Proxy model result

■ MLP Network

Our results:

- **Make batch size 1**
- **Relative error within 1%**

$$Prediction\ Loss = (Output - Target)^2$$

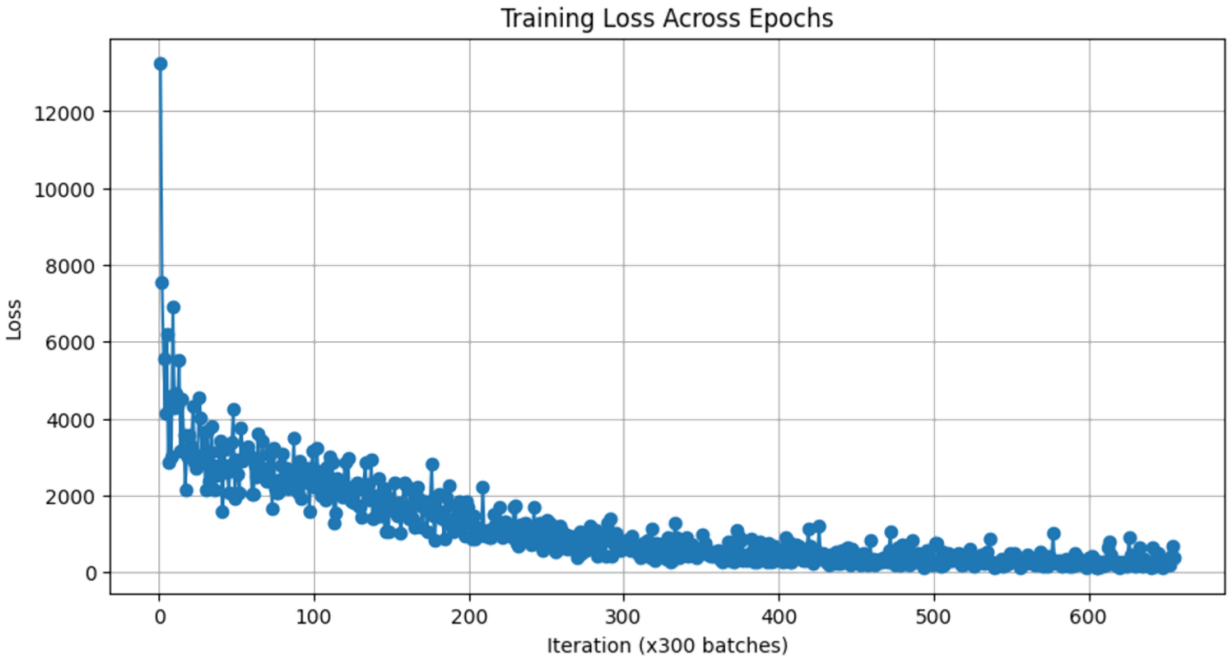


Fig. Prediction loss (MSE)

$$Relative\ Loss = (Output - Target)^2/Target$$

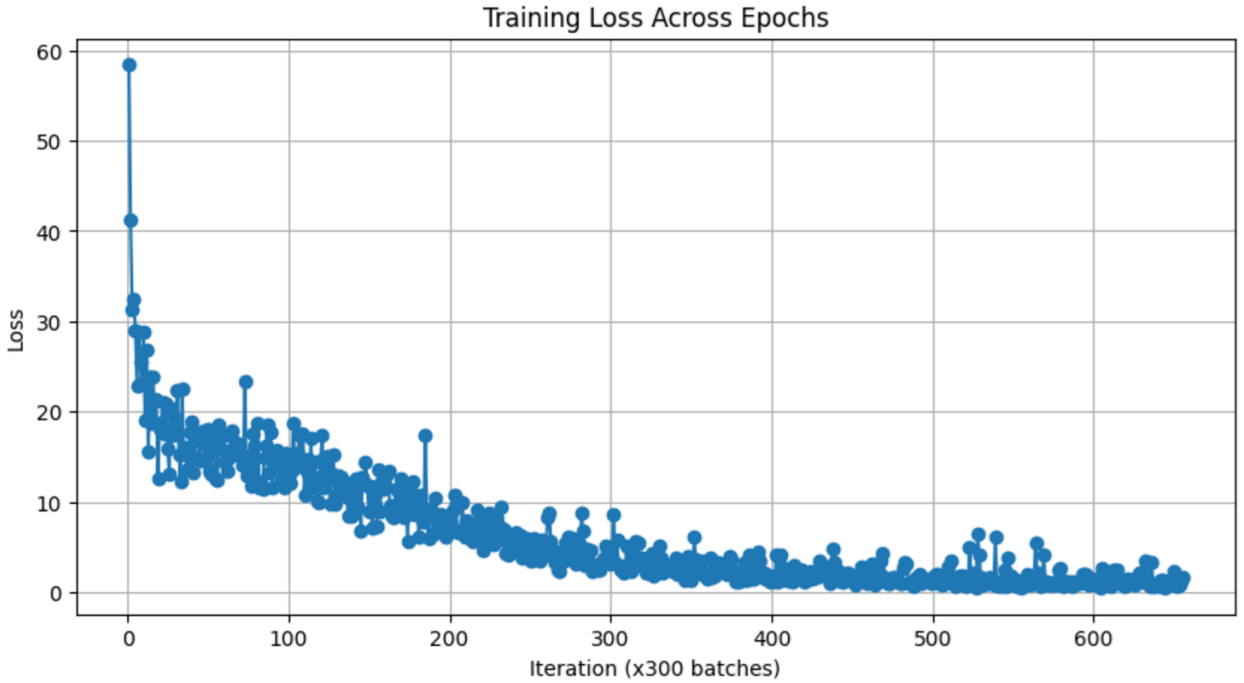


Fig. Relative prediction loss

UNIVERSITY of VIRGINIA

Thanks