Accelerating Winograd Convolutions using Symbolic Computation and Meta-programming

A. Mazaheri[1], T. Beringer[1], M. Moskewicz[2], F. Wolf[1], A. Jannesari[3]

[1] TU Darmstadt,

[2] Tesla Inc.,

[3] Iowa State University

30.04.2020

EuroSys'20, Heraklion, Crete, Greece

TECHNISCHE UNIVERSITÄT DARMSTADT

TESLA

IOWA STATE UNIVERSITY

Image: Freepik.com

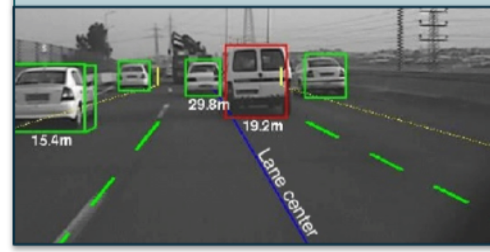
Neural networks are everywhere

TECHNISCHE UNIVERSITÄT DARMSTADT

Object detection

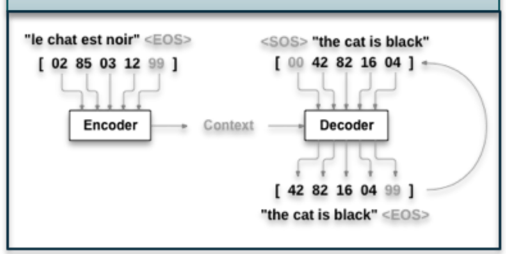
Semantic segmentation

Autonomous cars



Speech recognition

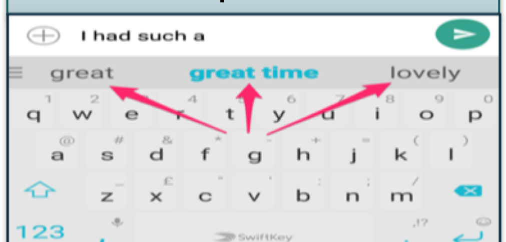
Translation



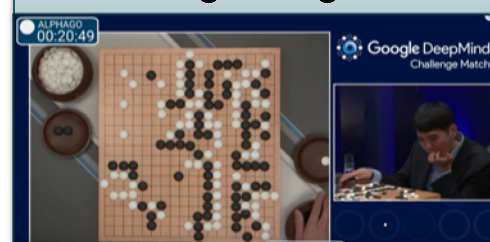
Music composition

Sentiment analysis

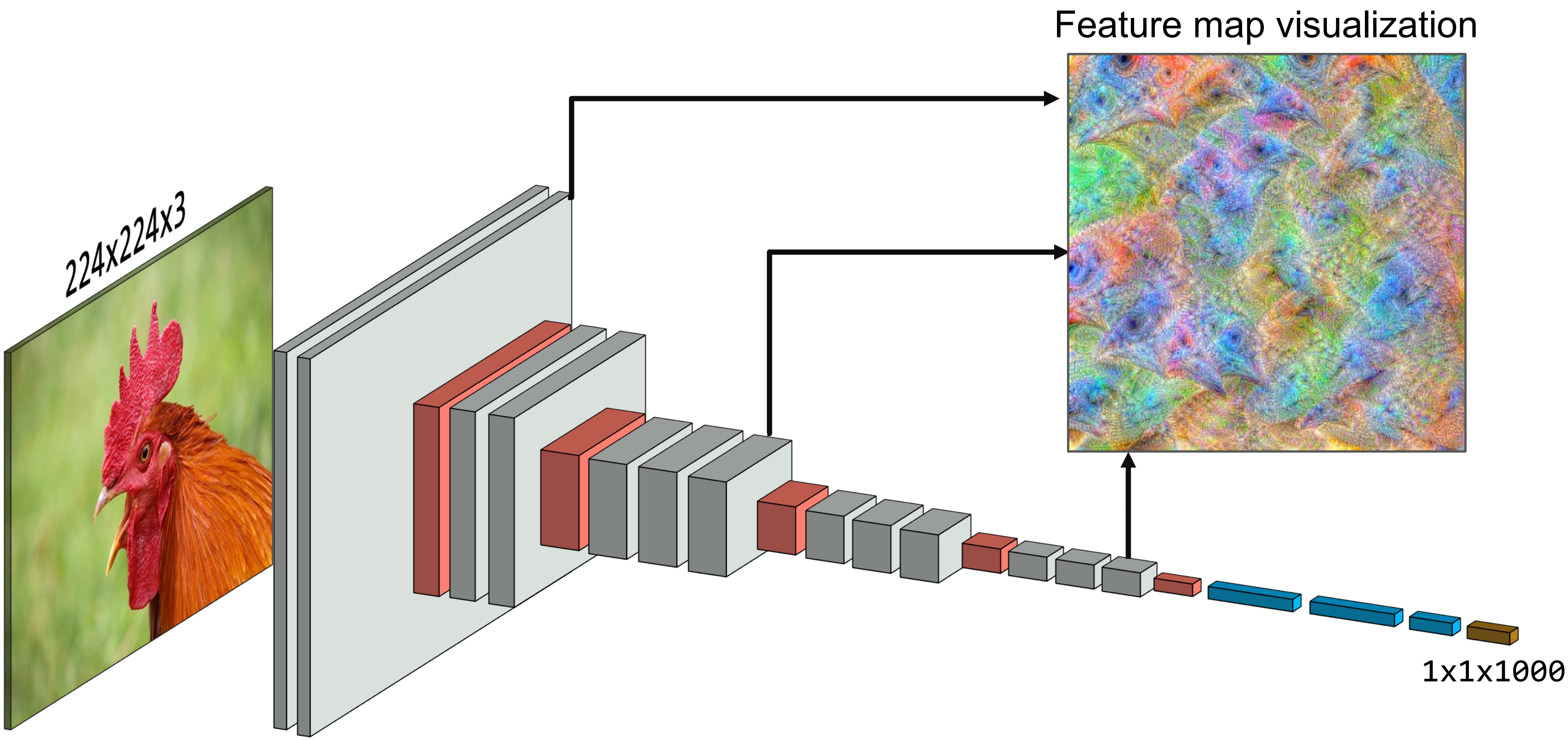
Word prediction



Intelligent agents



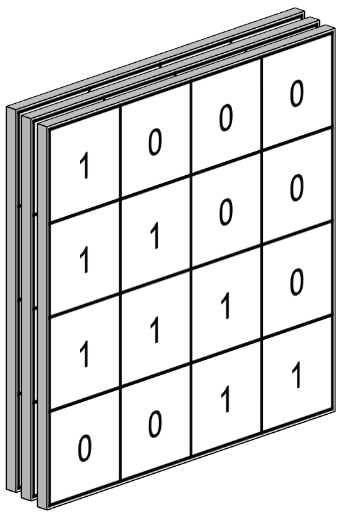
Convolutional neural networks



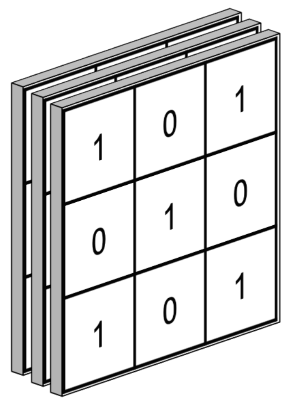
Convolution+ReLU   Max pooling   Fully connected+ReLU   Softmax

Convolution & tensors

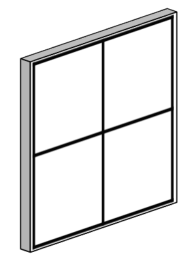
Input tensor
C × H × W



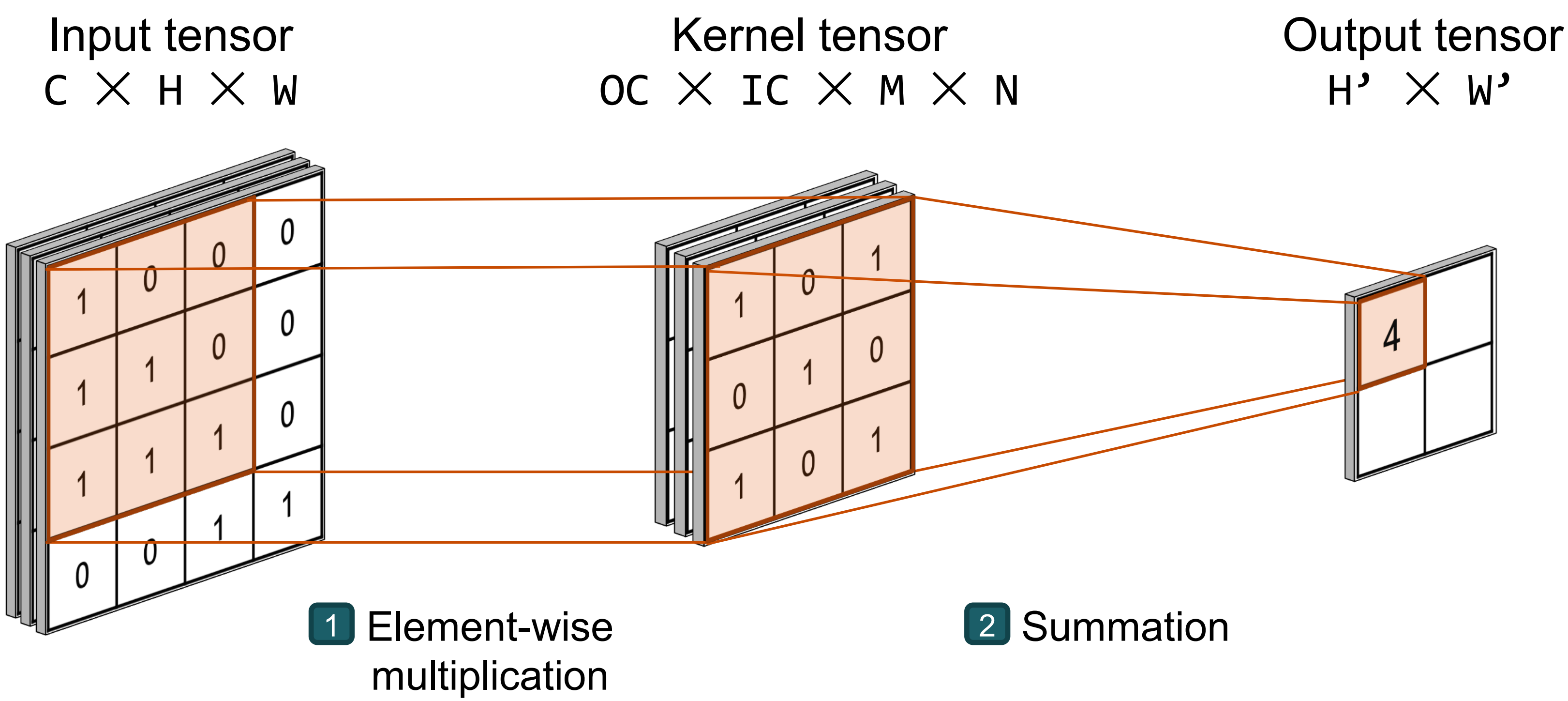
Kernel tensor
OC × IC × M × N



Output tensor
H’ × W’

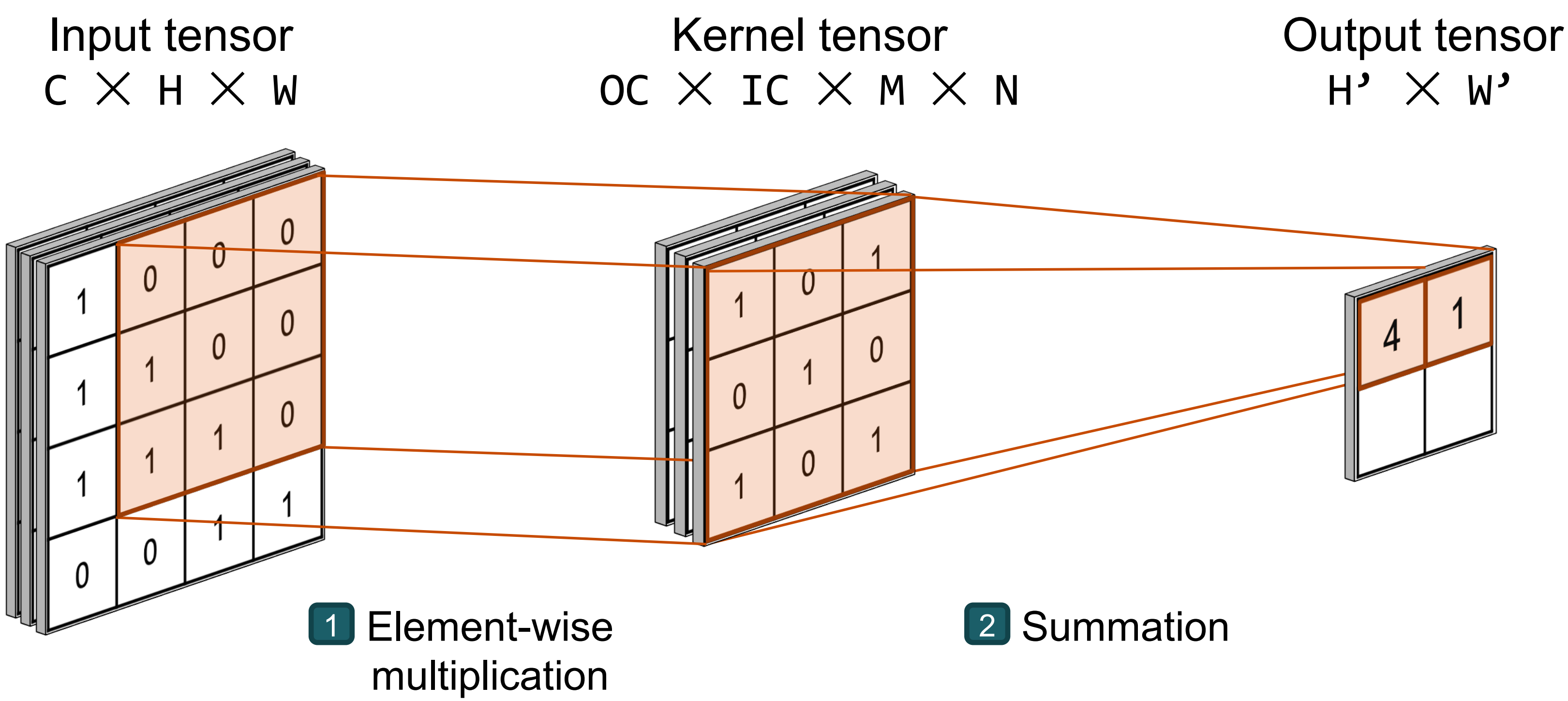


Convolution & tensors

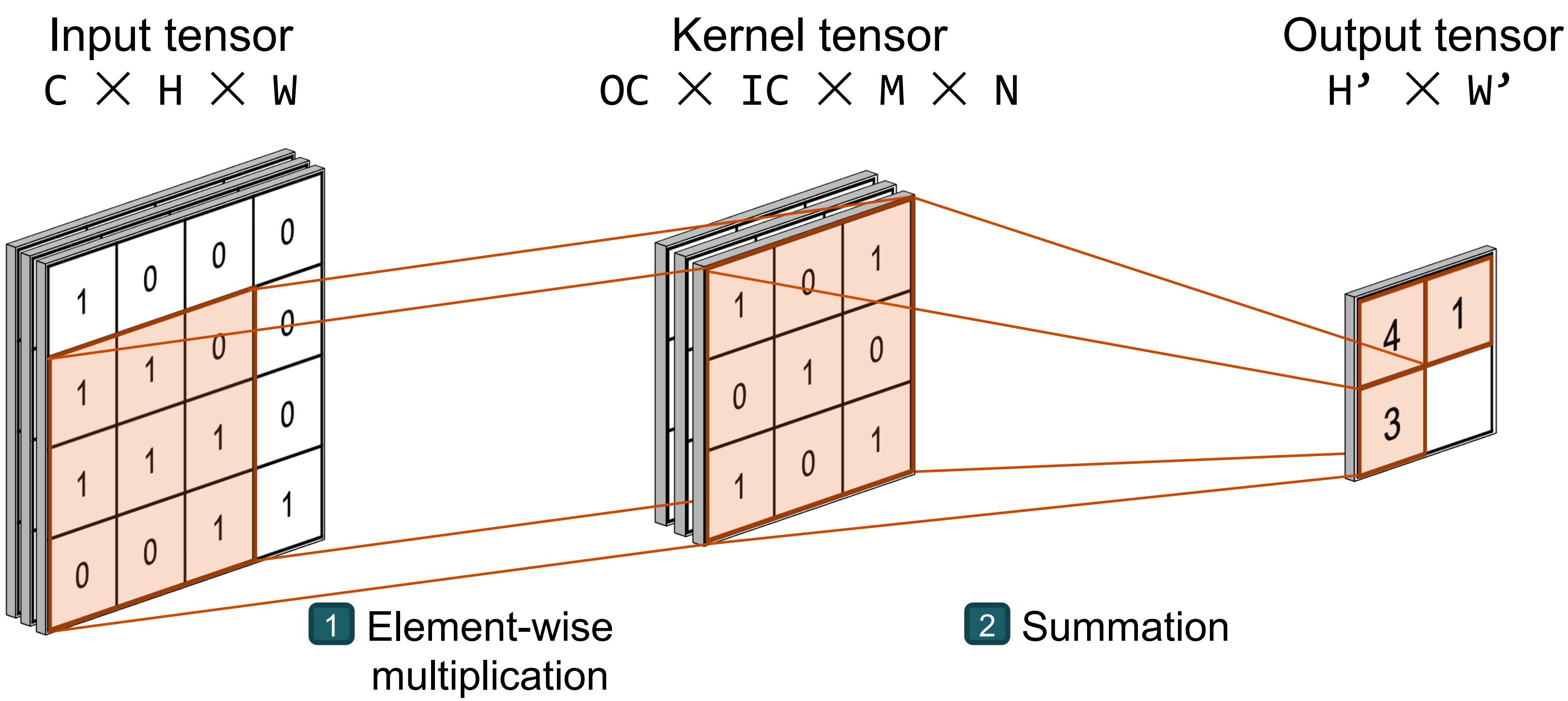


Convolution & tensors

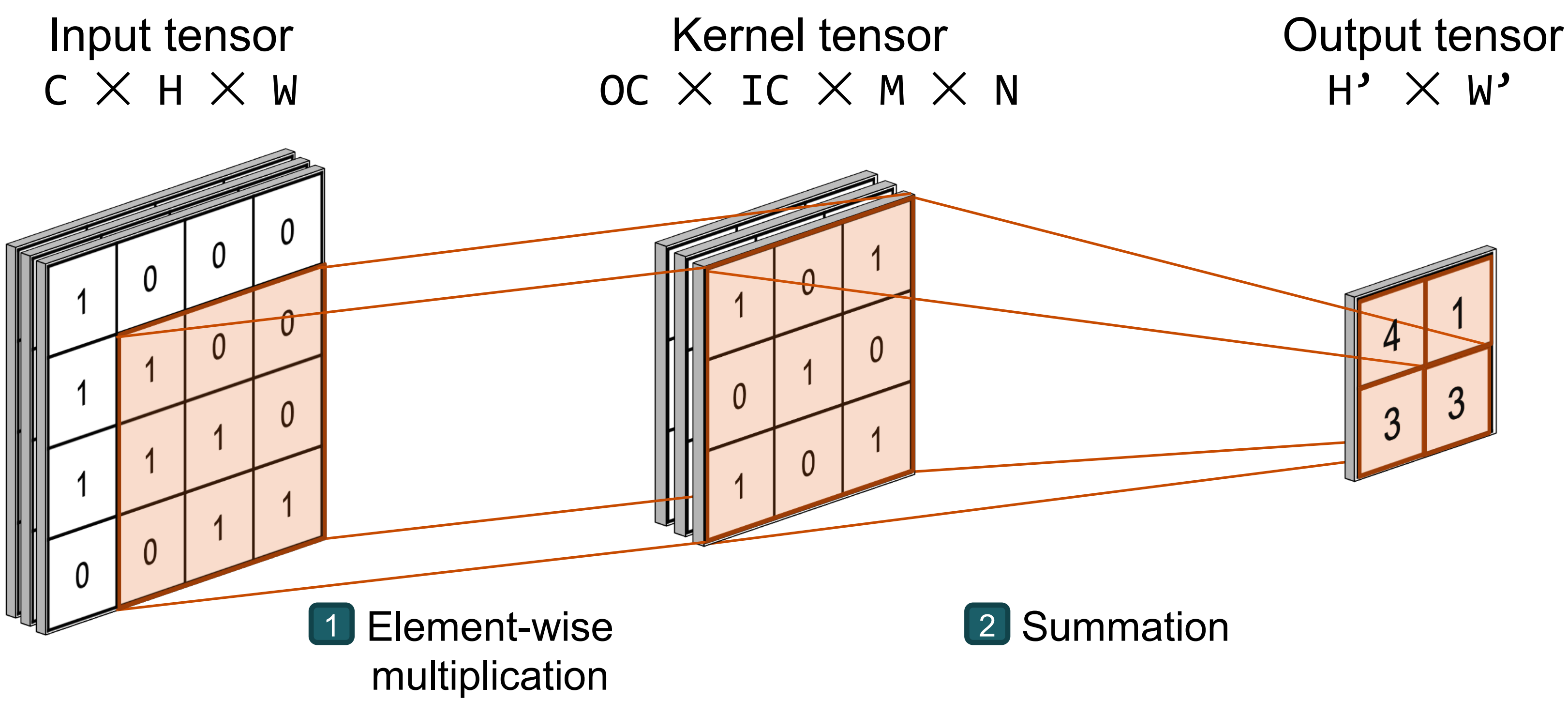
TECHNISCHE UNIVERSITÄT DARMSTADT



Convolution & tensors



Convolution & tensors

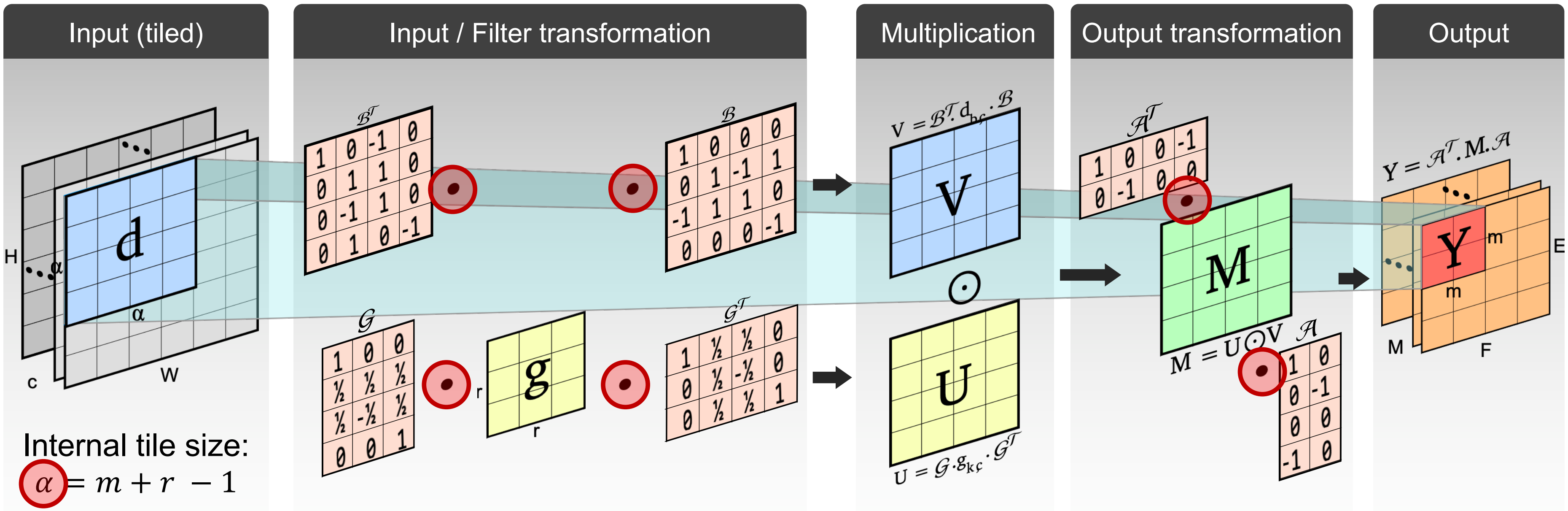


- Dominate computation (>90% of runtime)
- Similar to generalized matrix-matrix multiply → Massive GPU parallelism

Winograd convolution

$F(m, r)$

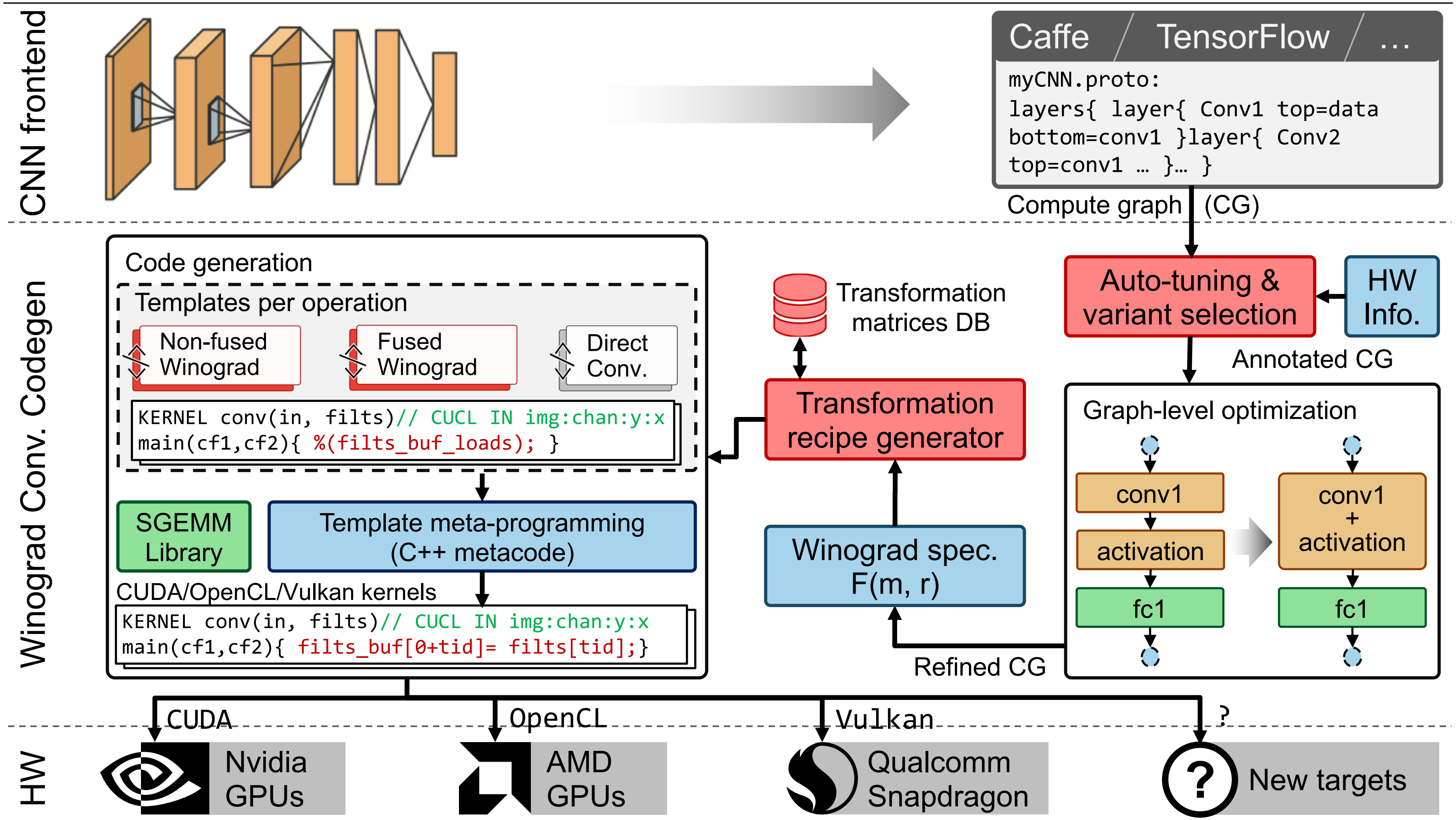
- Sample $F(m = 2\times2, r = 3\times3)$ Winograd convolution



Research questions:

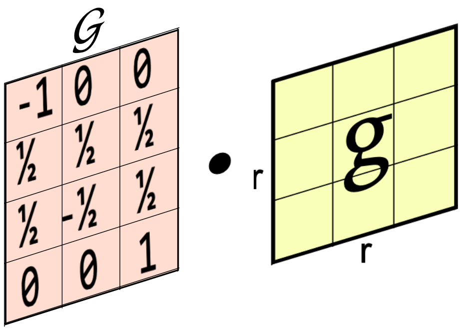
- Can we reduce the overhead of Winograd transformations?
- How to properly choose the right α?
- How to run Winograd efficiently on a wide range of GPU platforms?

Winograd code generation workflow



Optimizing Winograd transformations [Symbolic analysis]

- Represent the target matrix by symbols
- Perform multiplication and obtain the results

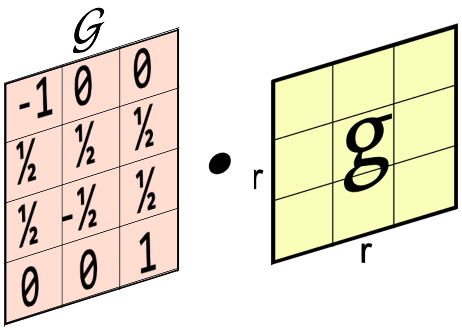


$$= \begin{bmatrix} -1\times g_{0,0}+0+0 & -1\times g_{0,1}+0+0 & -1\times g_{0,2}+0+0 \\ \frac{g_{0,0}}{2}+\frac{g_{1,0}}{2}+\frac{g_{2,0}}{2} & \frac{g_{0,1}}{2}+\frac{g_{1,1}}{2}+\frac{g_{2,1}}{2} & \frac{g_{0,2}}{2}+\frac{g_{1,2}}{2}+\frac{g_{2,2}}{2} \\ \frac{g_{0,0}}{2}-\frac{g_{1,0}}{2}+\frac{g_{2,0}}{2} & \frac{g_{0,1}}{2}-\frac{g_{1,1}}{2}+\frac{g_{2,1}}{2} & \frac{g_{0,2}}{2}-\frac{g_{1,2}}{2}+\frac{g_{2,2}}{2} \\ 0+0+1\times g_{2,0} & 0+0+1\times g_{2,1} & 0+0+1\times g_{2,2} \end{bmatrix}$$

Matrix multiplication code before optimization

```
for (i = 0; i < alpha; i++) {
    for (j = 0; j < r; j++) {
        res[i][j] = 0;
        for (k = 0; k < r; k++)
            res[i][j] += G[i][k] * g[k][j];
    }
}
```

Optimizing Winograd transformations [Remove 1,0s]



$$= \begin{bmatrix} -1\times g_{0,0} + 0 + 0 & -1\times g_{0,1} + 0 + 0 & -1\times g_{0,2} + 0 + 0 \\ \frac{g_{0,0}}{2} + \frac{g_{1,0}}{2} + \frac{g_{2,0}}{2} & \frac{g_{0,1}}{2} + \frac{g_{1,1}}{2} + \frac{g_{2,1}}{2} & \frac{g_{0,2}}{2} + \frac{g_{1,2}}{2} + \frac{g_{2,2}}{2} \\ \frac{g_{0,0}}{2} - \frac{g_{1,0}}{2} + \frac{g_{2,0}}{2} & \frac{g_{0,1}}{2} - \frac{g_{1,1}}{2} + \frac{g_{2,1}}{2} & \frac{g_{0,2}}{2} - \frac{g_{1,2}}{2} + \frac{g_{2,2}}{2} \\ 0 + 0 + 1\times g_{2,0} & 0 + 0 + 1\times g_{2,1} & 0 + 0 + 1\times g_{2,2} \end{bmatrix}$$

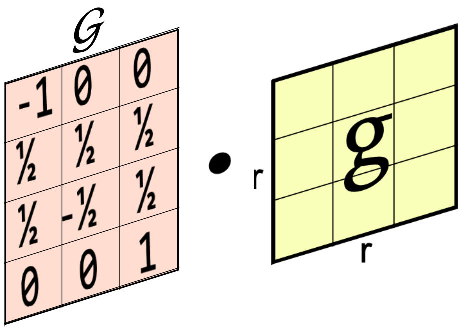
$$= \begin{bmatrix} -g_{0,0} & -g_{0,1} & -g_{0,2} \\ \frac{g_{0,0}}{2} + \frac{g_{1,0}}{2} + \frac{g_{2,0}}{2} & \frac{g_{0,1}}{2} + \frac{g_{1,1}}{2} + \frac{g_{2,1}}{2} & \frac{g_{0,2}}{2} + \frac{g_{1,2}}{2} + \frac{g_{2,2}}{2} \\ \frac{g_{0,0}}{2} - \frac{g_{1,0}}{2} + \frac{g_{2,0}}{2} & \frac{g_{0,1}}{2} - \frac{g_{1,1}}{2} + \frac{g_{2,1}}{2} & \frac{g_{0,2}}{2} - \frac{g_{1,2}}{2} + \frac{g_{2,2}}{2} \\ g_{2,0} & g_{2,1} & g_{2,2} \end{bmatrix}$$

Optimizing Winograd transformations [Index representation]

$$\mathcal{G} \bullet g = \begin{bmatrix} -1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix} \bullet \underbrace{g}_{r \times r} = \begin{bmatrix} -g_{0,0} & -g_{0,1} & -g_{0,2} \\ \frac{g_{0,0}}{2}+\frac{g_{1,0}}{2}+\frac{g_{2,0}}{2} & \frac{g_{0,1}}{2}+\frac{g_{1,1}}{2}+\frac{g_{2,1}}{2} & \frac{g_{0,2}}{2}+\frac{g_{1,2}}{2}+\frac{g_{2,2}}{2} \\ \frac{g_{0,0}}{2}-\frac{g_{1,0}}{2}+\frac{g_{2,0}}{2} & \frac{g_{0,1}}{2}-\frac{g_{1,1}}{2}+\frac{g_{2,1}}{2} & \frac{g_{0,2}}{2}-\frac{g_{1,2}}{2}+\frac{g_{2,2}}{2} \\ g_{2,0} & g_{2,1} & g_{2,2} \end{bmatrix}$$

$$= \begin{bmatrix} -g_{0,j} \\ \frac{g_{0,j}}{2}+\frac{g_{2,j}}{2}+\frac{g_{1,j}}{2} \\ \frac{g_{0,j}}{2}+\frac{g_{2,j}}{2}-\frac{g_{1,j}}{2} \\ g_{2,j} \end{bmatrix}$$

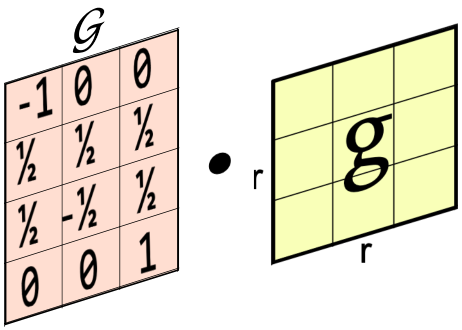
Optimizing Winograd transformations [Factorization]



$$= \begin{bmatrix} -g_{0,j} \\ \frac{g_{0,j}}{2} + \frac{g_{2,j}}{2} + \frac{g_{1,j}}{2} \\ \frac{g_{0,j}}{2} + \frac{g_{2,j}}{2} - \frac{g_{1,j}}{2} \\ g_{2,j} \end{bmatrix}$$

$$= \begin{bmatrix} -g_{0,j} \\ \frac{1}{2}(g_{0,j} + g_{2,j} + g_{1,j}) \\ \frac{1}{2}(g_{0,j} + g_{2,j} - g_{1,j}) \\ g_{2,j} \end{bmatrix}$$

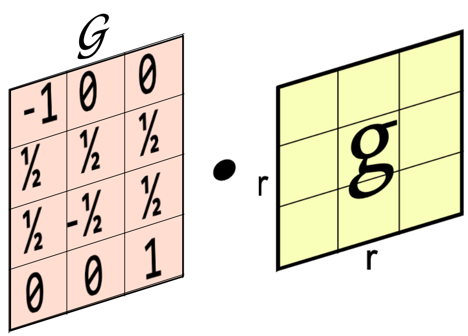
Optimizing Winograd transformations [Common subexpression elimination]



$$= \begin{bmatrix} -g_{0,j} \\ \frac{1}{2}(g_{0,j} + g_{2,j} + g_{1,j}) \\ \frac{1}{2}(g_{0,j} + g_{2,j} - g_{1,j}) \\ g_{2,j} \end{bmatrix}$$

$$= \begin{bmatrix} -g_{0,j} \\ \frac{1}{2}(cse1 + g_{1,j}) \\ \frac{1}{2}(cse1 - g_{1,j}) \\ g_{2,j} \end{bmatrix}, \quad cse1 = g_{0,j} + g_{2,j}$$

Optimizing Winograd transformations [Code generation]



$$= \begin{bmatrix} -g_{0,j} \\ \frac{1}{2}(cse1 + g_{1,j}) \\ \frac{1}{2}(cse1 - g_{1,j}) \\ g_{2,j} \end{bmatrix}, \quad cse1 = g_{0,j} + g_{2,j}$$

Before optimizations

```
for (i = 0; i < alpha; i++) {
    for (j = 0; j < r; j++) {
        Gg[i][j] = 0;
        for (k = 0; k < r; k++)
            Gg[i][j] += G[i][k] *
                            g[k][j];
    }
}
```

After optimizations

```
for(j=0, j<4, j++){
    cse1 = g[0][j] + g[2][j];
    Gg[0][j] = -g[0][j];
    Gg[1][j] = 0.5*(cse1 + g[1][j]);
    Gg[2][j] = 0.5*(cse1 - g[1][j]);
    Gg[3][j] = g[2][j];
}
```

Performance auto-tuning

TECHNISCHE UNIVERSITÄT DARMSTADT

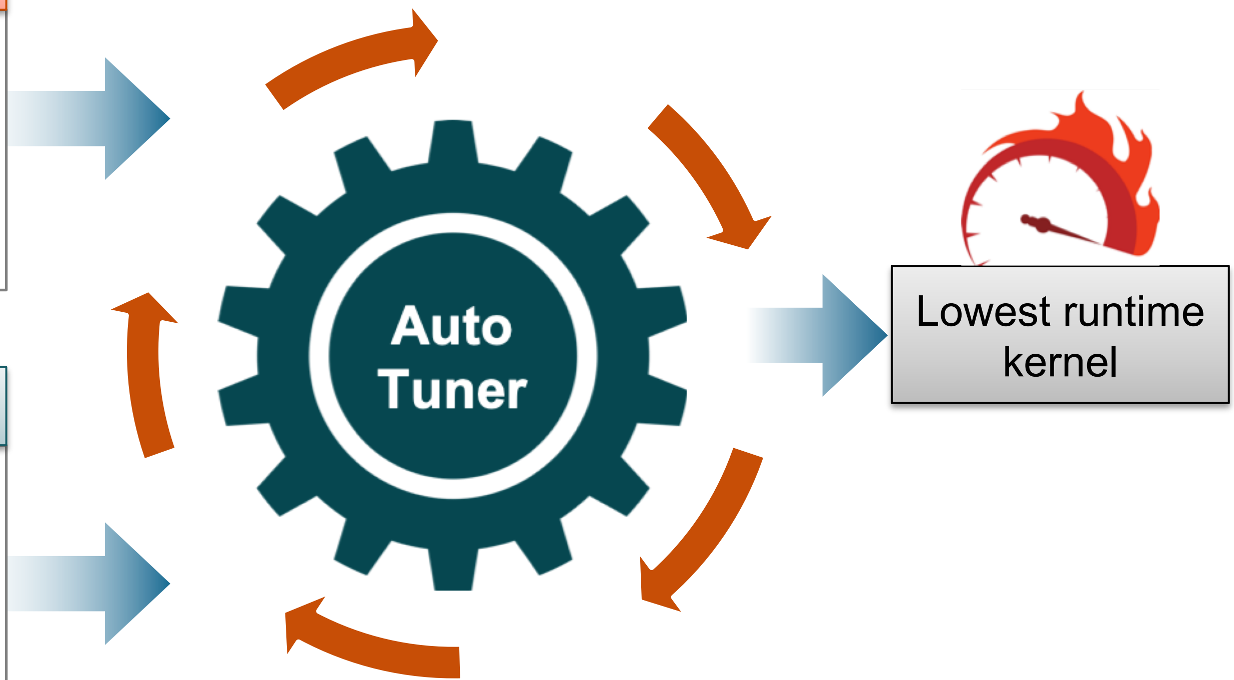
Tuning knobs

- Winograd variant
- Thread blocking
- Register blocking
- Loop unrolling factor
- Winograd output tile size

Tensor operation kernel

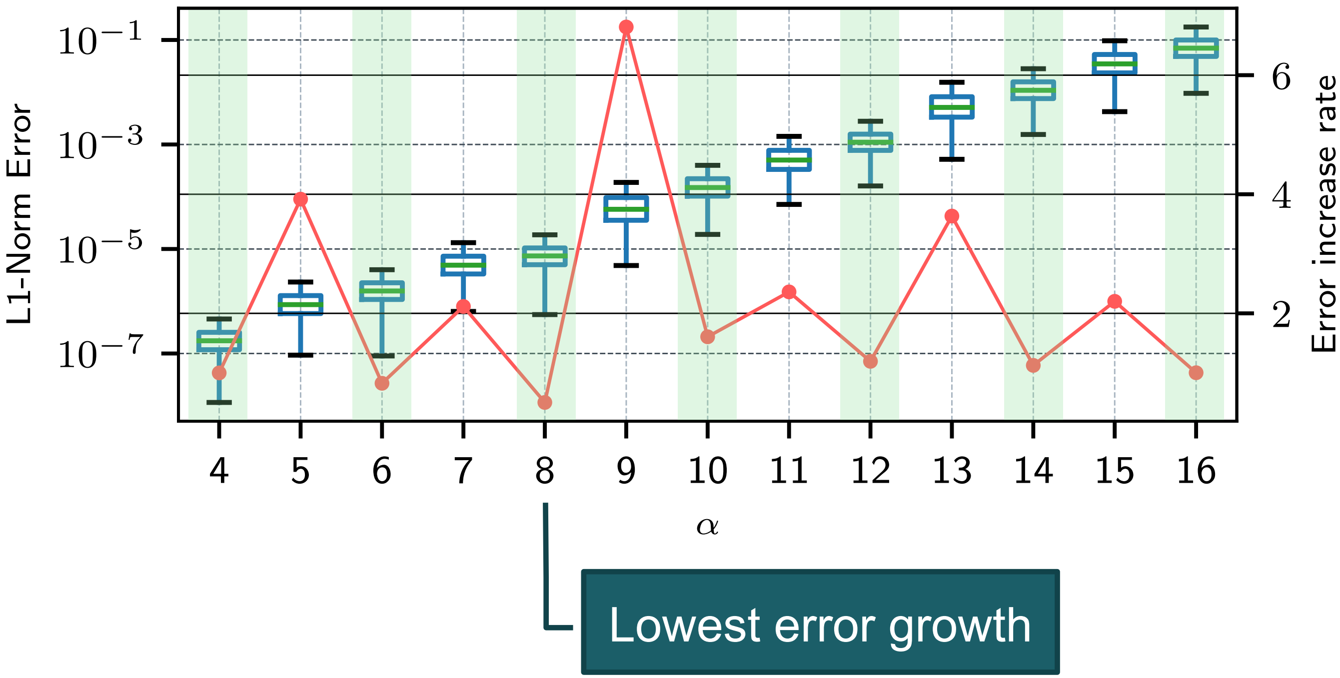
```
float g[7][7];
float Gg[8][7];
float tmp[8][8];

const GASQ float *B = filts_ref + (k * 3 + c)
* 7 * 7;
for (int i = 0; i < 7; ++i) {
  for (int j = 0; j < 7; ++j) {
    g[i][j] = B[7*j + i];
  }
}
```



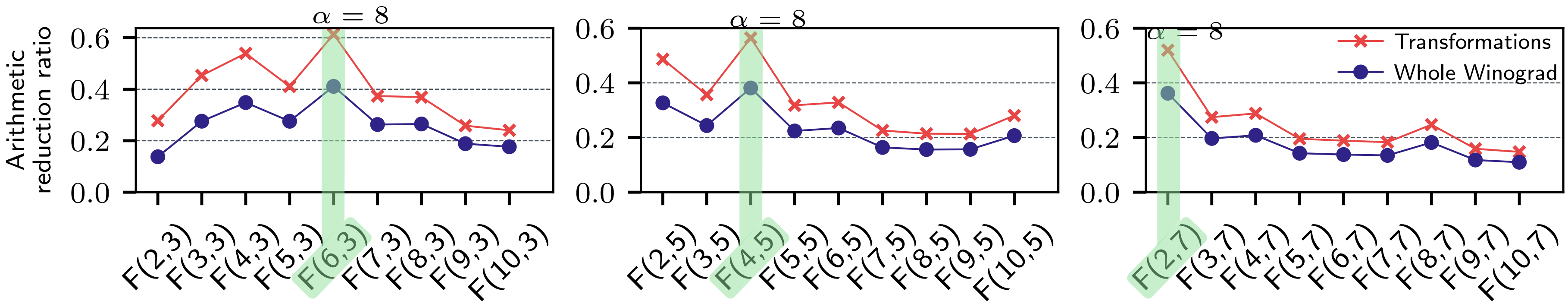
Winograd convolution accuracy

- L1-norm error analysis for various Winograd internal tile sizes

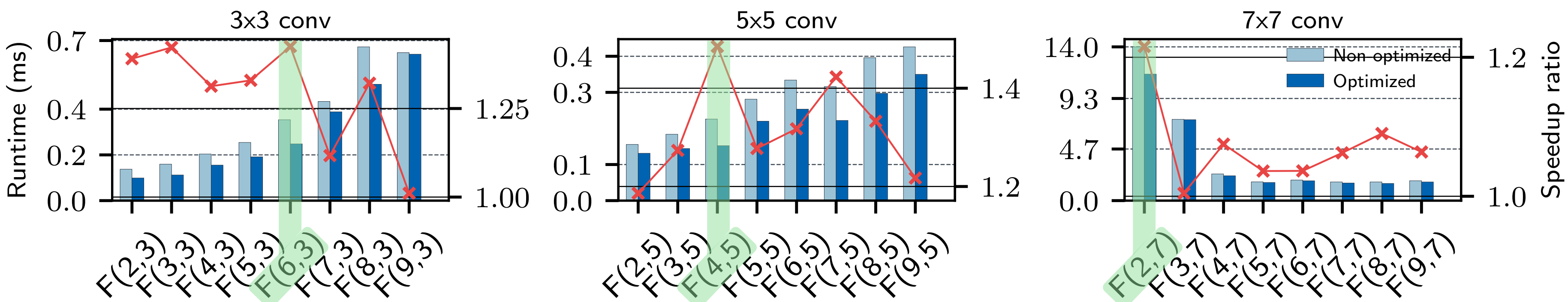


Winograd transformation optimization results

- Overall arithmetic reduction ratios related to transformation steps and the whole Winograd algorithm for a single tile

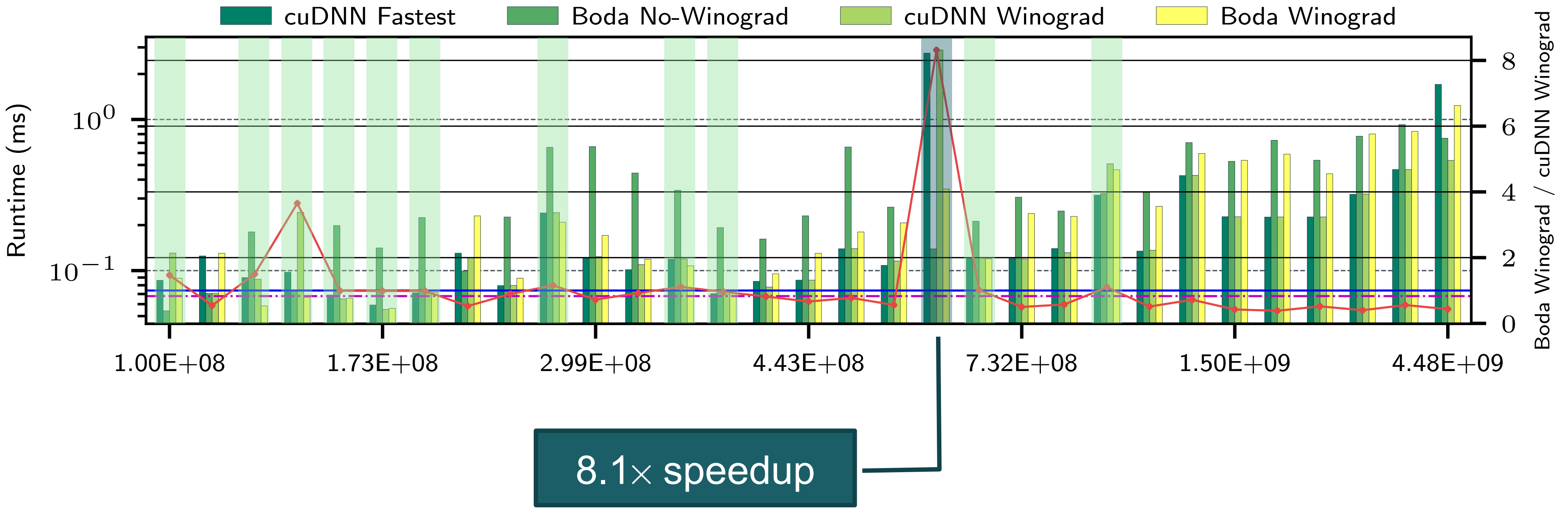


- Runtime comparison on Nvidia 1080 Ti



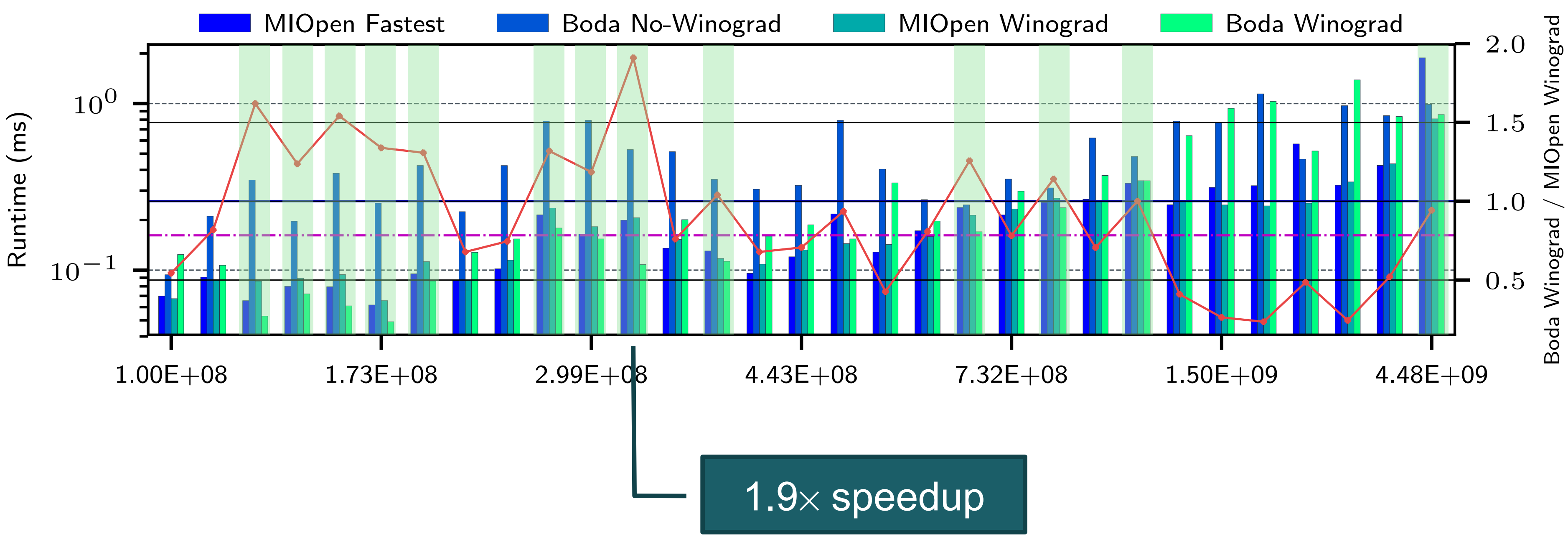
Performance portability [Nvidia GPU]

- Runtime comparison of Winograd kernels generated by our method with cuDNN vendor library on Nvidia GTX 1080 Ti.



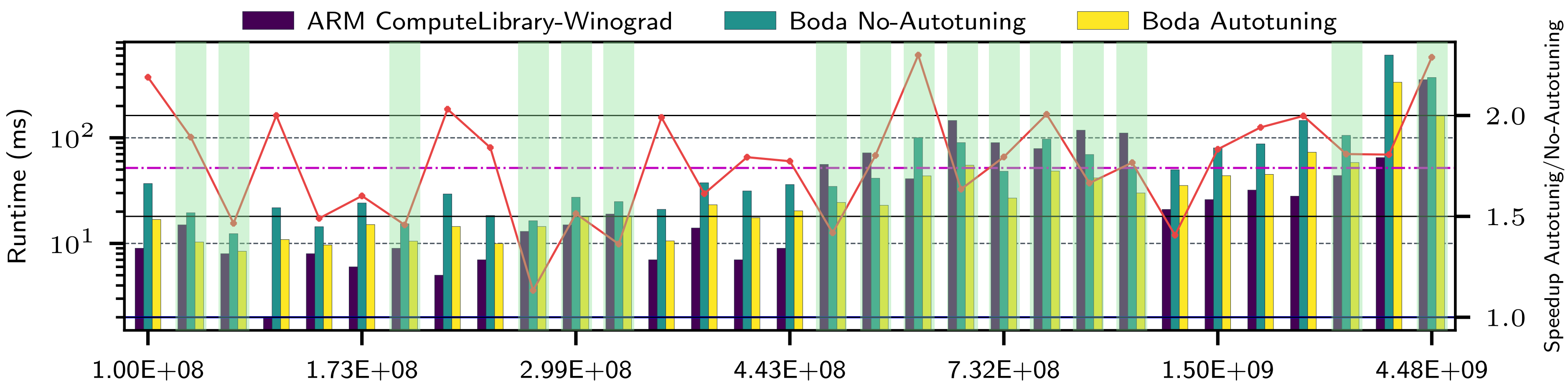
Performance portability [AMD GPU]

- Runtime comparison of Winograd kernels generated by our method with the MIOpen vendor library on AMD Radeon RX 580.



Performance portability [Mobile GPU]

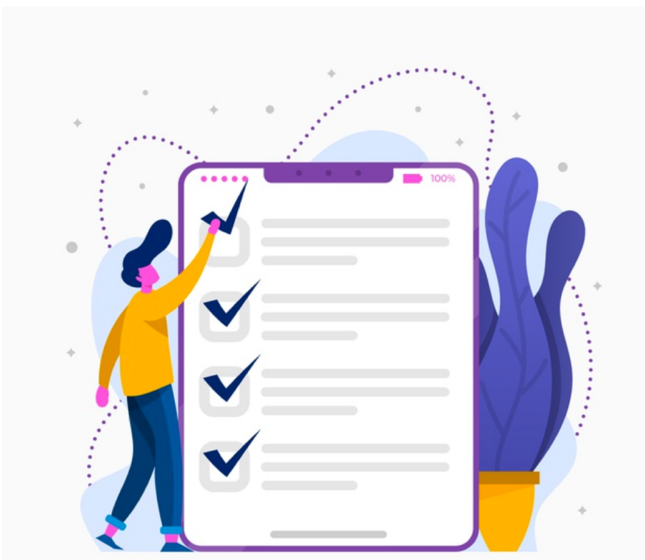
- Validated the effect of auto-tuning on Mali G71 GPU
- ARM Compute library as a baseline



1.74× speedup

Conclusion

- Symbolic analysis → Optimizing Winograd transformation steps
- Meta-programming → Enhancing the performance portability of Winograd convolutions



1. Efficient Winograd convolution is tricky to implement
2. When $\alpha = 8$; largest arithmetic reduction, acceptable accuracy
3. Performance portability on three different architectures

Questions?

Contact me if you are interested:

Arya Mazaheri
<mazaheri@cs.tu-darmstadt.de>

TECHNISCHE UNIVERSITÄT DARMSTADT

TESLA

IOWA STATE UNIVERSITY

SF 4.0

LOEWE

Exzellente Forschung für Hessens Zukunft

Image designed by vectorpouch / Freepik