Appendix - Towards an Understanding of Deep Neural Network Resiliency to Hardware Faults

Patrik Omland
Department of Informatics
Technical University Munich
Garching, Germany
patrik.omland@gmail.com

Michael Paulitsch
Intel Labs
Intel Corporation
Munich, Germany
michael.paulitsch@intel.com

Gereon Hinz
Department of Informatics
Technical University Munich
Garching, Germany
gereon.hinz@tum.de

Alois Knoll
Department of Informatics
Technical University Munich
Garching, Germany
k@tum.de

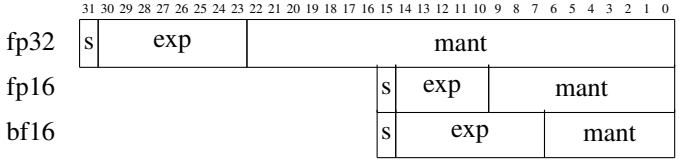


Fig. 1: Examples of floating-point bit fields.

exp	mant $\equiv 0$	mant $\neq 0$
$'1$	$\pm\infty$	NaN
0	$(-1)^s \cdot 2^{1-\text{exp}_{\text{bias}}} \cdot 0.\text{mant}$	
else	$(-1)^s \cdot 2^{\text{exp}-\text{exp}_{\text{bias}}} \cdot 1.\text{mant}$	

TABLE I: FP representation. NaN: Not a Number: Result of undefined operation. $\text{exp}_{\text{bias}} := 2^{bits(exp)-1} - 1$, eg. 127 for bf16. '1.mant' denotes the number $1 + \sum_{\text{bit}} 2^{\text{bit}-bits(mant)}$.

NOTE

This is the appendix to the publication: Omland, P., Paulitsch, M., Hinz, G., and Knoll, A. "Towards an Understanding of Deep Neural Network Resiliency to Hardware Faults". In: 2025 20th European Dependable Computing Conference (EDCC). Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2025

APPENDIX A
FLOATING-POINTS

The most common FP representations comprises three parts: Sign (s), exponent (exp) and mantissa (mant) - see figure 1. Such a FPN represents a number according to table I. A FPN is called *normal* when exp $\neq 0$ and exp $\neq$ '1, where $'1$ is shorthand for 'all bits set to 1'.

Table II lists the relative errors caused by a fault flipping a random bit of a normal FPN. We show relative errors, as we are ultimately interested in faults causing out of range errors.

Table III lists relative errors and their probability in the FPN ranges $1 \leq |x| < 2$ and $2 \leq |x| < 4$ as discussed in section IV.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 956123.

fault bit	relative error $\lvert\frac{x'-x}{x}\rvert$	
sign	2	
exp	$\pm\infty$ or NaN	if exp flipped to '1, else
	≈ 1	if exp flipped to 0, else
	$1 - 2^{-2^{\text{bit}}}$	if bit set, else
	$2^{2^{\text{bit}}} - 1$	
mant	$1 - 2^{\text{bit}-bits(\text{mant})}/1.\text{mant}$	if bit set, else
	$1 + 2^{\text{bit}-bits(\text{mant})}/1.\text{mant}$	

TABLE II: Relative error due to single bit-flip corruption on normal FPNs. $bits(\text{part})$ refers to the number of bits comprising that part. 'bit' denotes the relative position inside the resp. FPN part (little-endian) and '1 setting all of its bits to 1.

Pr	relative error $1 \leq \lvert x\rvert < 2$	relative error $2 \leq \lvert x\rvert < 4$
$\frac{1}{bits(exp)}$	NaN or $\pm\infty$	≈ 1
$1 - \frac{1}{bits(exp)}$	< 1	$\approx 2^{2^{\text{bit}}}$

TABLE III: Relative error of exponent bit-flip.

APPENDIX B
TOOLCHAIN

In this section we describe the numerical application we developed to carry out the experiments described in section VII. Given a discrete random variable X, we chose to represent the *Probability Mass Function* (PMF), p_X, by an array of triplets as

$$p_X = \{x_1, \ \dots\} = \left\{\{x_1^{\text{begin}}, \ x_1^{\text{end}}, \ x_1^{\text{probability}}\}, \ \dots\right\} \quad (1)$$

so that

$$p_X(x) = \sum_{\{i \ \mid \ x_i^{\text{begin}} \leq x \leq x_i^{\text{end}}\}} x_i^{\text{probability}} \quad (2)$$

Now, given eg. a continuous function $Z = g(X, Y)$, taking two random variables X and Y, we generate the output distribution by creating a p_Z element z_k for each $(x_i, \ y_j)$-pair:

$$z_k^{\text{begin}} = \min\left\{g(x_i^{\text{begin}}, y_i^{\text{begin}}), g(x_i^{\text{begin}}, y_i^{\text{end}}), \dots\right\} \quad (3)$$

$$z_k^{\text{end}} = \max\left\{g(x_i^{\text{begin}}, y_i^{\text{begin}}), g(x_i^{\text{begin}}, y_i^{\text{end}}), \dots\right\} \quad (4)$$

$$z_k^{\text{probability}} = x_i^{\text{probability}} \cdot y_j^{\text{probability}} \quad (5)$$

For the discontinuous bit-flipping function, we perform the operation on the mid-value of each PMF interval, as stretching/contracting an interval does not reflect the behavior of the operation on the actual data in this case. For the other discontinuous functions considered, we list the algorithms in section VI.

For an operation involving two random variables X and Y, with n PMF elements each, notice that the result Z has n^2 elements. To practically perform multiple operations on a PMF, its number of elements can't increase exponentially with the number of operations. We reduce the number of elements after each operation in a two-fold approach: First, we create new elements / reduce existing element's intervals (and their probability according to its density) such that no two elements in p_Z overlap. Then, we set a minimal probability p^{min} and starting from the expected value $E(Z)$ move outward in the decreasing / increasing direction, respectively: We collect elements $\{z_i\}$ until their accumulated probability reaches p^{min}, at which point we combine these elements into a new element with the interval beginning (ending) at the smallest (largest) interval beginning (ending) from the collection - that is to say we take the smallest contiguous interval containing the collection's intervals. Then we continue outward with a new collection until the remaining elements in increasing / decreasing direction are combined, respectively, into elements z_k with $z_k^{\text{probability}} < p^{\text{min}}$.

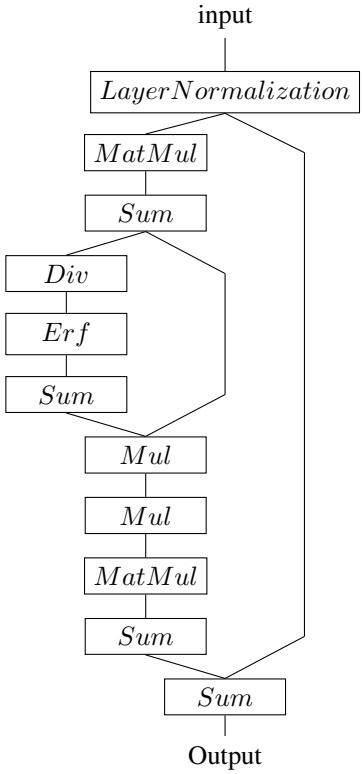


Fig. 2: BERT building block A

APPENDIX C
APPLICABILITY TO ROBERTA

RoBERTa (Robustly Optimized BERT Pretraining Approach), is a refinement of the popular BERT natural language model.[1] Its building blocks are shown in figure 2 and 3. These architectural details were taken from the ONNX Model Zoo.[2]

Analogously to section VI, we go through those operations inside RoBERTa's building blocks which have not been covered there and discuss their stochastic behavior. The operations will not be discussed generally (some come with many options in ONNX), but specifically to their use in RoBERTa.

A. Layer Normalization

For input $i \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times N}$, the *LayerNormalization* applied by RoBERTa produces output $o \in \mathbb{R}^{M_1 \times M_2 \times \cdots \times N}$ according to

$$\bar{i}_{m_0 m_1 \ldots} = \frac{1}{N} \sum_n i_{m_0 m_1 \ldots n} \tag{6}$$

$$d_{m_0 m_1 \ldots n} = i_{m_0 m_1 \ldots n} - \bar{i}_{m_0 m_1 \ldots} \tag{7}$$

$$\text{dd}_{m_0 m_1 \ldots n} = d_{m_0 m_1 \ldots n} \cdot d_{m_0 m_1 \ldots n} \tag{8}$$

$$\text{var}_{m_0 m_1 \ldots n} = \frac{1}{N} \sum_n \text{dd}_{m_0 m_1 \ldots n} \tag{9}$$

$$\text{normalized}_{m_0 m_1 \ldots n} = \frac{d_{m_0 m_1 \ldots n}}{\sqrt{\text{var}_{m_0 m_1 \ldots n} + \epsilon}} \tag{10}$$

$$o_{m_0 m_1 \ldots n} = s_n \cdot \text{normalized}_{m_0 m_1 \ldots n} + b_n \tag{11}$$

Where $\epsilon \in \mathbb{R}$ and s, $b \in \mathbb{R}^N$ are constants.

While the Batch Normalization discussed in the context of ResNet did not really normalize but applied fixed constants, this Layer Normalization actually normalizes the layer's input along one axis, n. Stochastically, the only operation used in the equations above which we have not covered already is $\frac{1}{\sqrt{x}}$. We have

$$p_{\frac{1}{\sqrt{X}}}(y) = p_X\left(\frac{1}{y^2}\right) \tag{12}$$

B. Matrix Multiplication

For inputs $a \in \mathbb{R}^{M \times K}$ and $b \in \mathbb{R}^{K \times N}$, *MatMul* produces output $o \in \mathbb{R}^{M \times N}$ according to

$$o_{mn} = \sum_k a_{mk} \cdot b_{kn} \tag{13}$$

[1] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019, arXiv 1907.11692

[2] https://github.com/onnx/models/tree/main/Natural_Language_Processing/roberta_Opset18_transformers

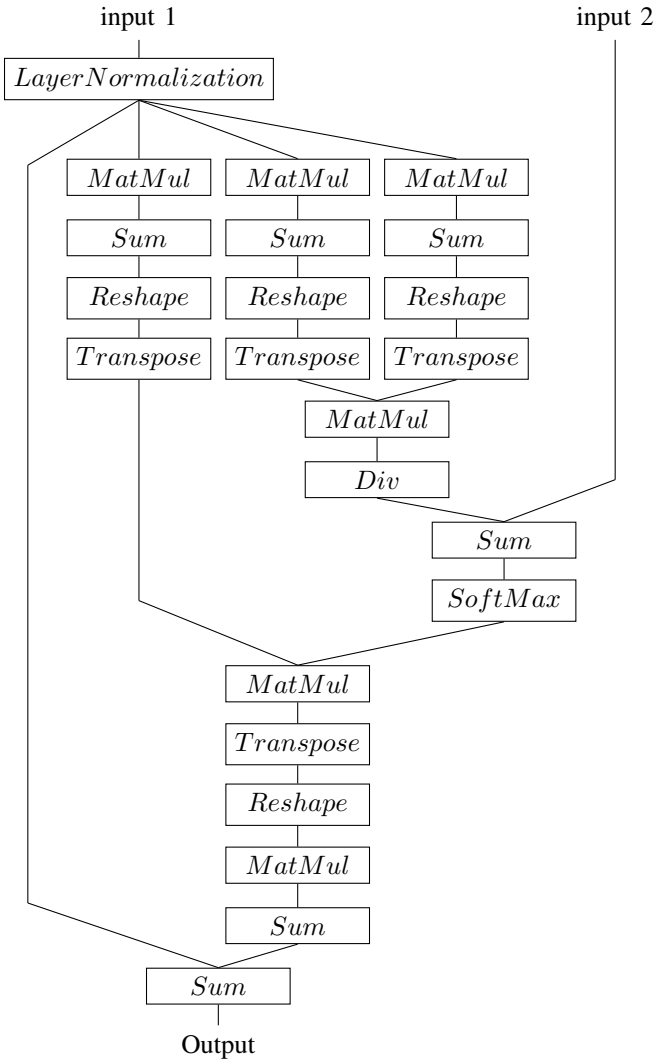


Fig. 3: BERT building block B

If MatMul is shown with only one input in the graph, b is constant. For inputs $a \in \mathbb{R}^{L\times M\times K}$ and $b \in \mathbb{R}^{L\times K\times N}$ ("stacked matrices"), output $o \in \mathbb{R}^{L\times M\times N}$ is calculated as

$$o_{lmn} = \sum_k a_{lmk} \cdot b_{lkn} \tag{14}$$

generalizing to $a \in \mathbb{R}^{L_0\times L_1\times\cdots\times M\times K}$ and $b \in \mathbb{R}^{L_0\times\cdots}$ in the obvious way (eg. in RoBERTa, a Reshape operation followed by a Transpose operation as shown in figure 3 prepare these matrix stacks). Furthermore, "broadcasting" may be applied to multiply $a \in \mathbb{R}^{L_0\times L_1\times\cdots\times M\times K}$ and $b \in \mathbb{R}^{K\times N}$ to produce $o \in \mathbb{R}^{L_0\times L_1\times\cdots\times M\times N}$ according to

$$o_{l_0 l_1 \ldots mn} = \sum_k a_{l_0 l_1 \ldots mk} \cdot b_{kn} \tag{15}$$

Whatever the dimensions may be, stochastically, these sums of products may be treated the same way as the sums of products covered for the convolution operation (section VI).

C. Transpose

Transpose rearranges ("permutes") the input tensor's dimensions. Like the *Reshape* operation discussed in section VI, Transpose does not change the value distribution of the underlying tensor (unless channels are sampled individually).

D. Division

For input $i \in \mathbb{R}^{N_0\times N_1\times\cdots}$, *Div* produces output $o \in \mathbb{R}^{N_0\times N_1\times\cdots}$ according to

$$o_{n_0 n_1 \ldots} = \frac{i_{n_0 n_1 \ldots}}{c} \tag{16}$$

for some constant c. We have $p_{I/c}(o) = p_I\,(c \cdot o)$.

E. Error Function

Erf computes the element-wise error function: For input $i \in \mathbb{R}^{N_0\times N_1\times\cdots}$ it produces output $o \in \mathbb{R}^{N_0\times N_1\times\cdots}$ according to

$$o_{n_0 n_1 \ldots} = \frac{2}{\sqrt{\pi}} \int_0^{i_{n_0 n_1 \cdots}} e^{-t^2}\, dt \tag{17}$$

The integral is normalized by the $\frac{2}{\sqrt{\pi}}$-factor, st. $Erf(\pm\infty) = \pm 1$. The inverse function Erf^{-1} required for the output distribution is well-defined on the interval $(-1, 1)$ and may be expressed as a series.[3] We get $p_{Erf(I)}(o) = p_I\left(Erf^{-1}(o)\right)$.

F. Multiplication

Mul multiplies two inputs element-wise. For inputs a, $b \in \mathbb{R}^{N_0\times N_1\times\cdots}$ the output $o \in \mathbb{R}^{N_0\times N_1\times\cdots}$ is generated as

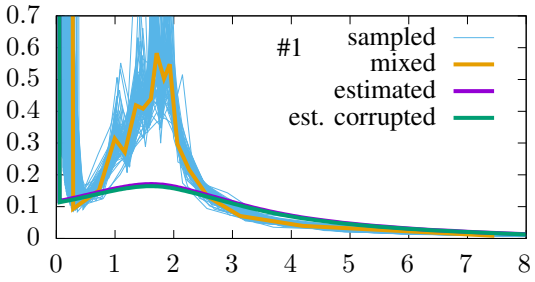
$$o_{n_0 n_1 \ldots} = a_{n_0 n_1 \ldots} \cdot b_{n_0 n_1 \ldots} \tag{18}$$

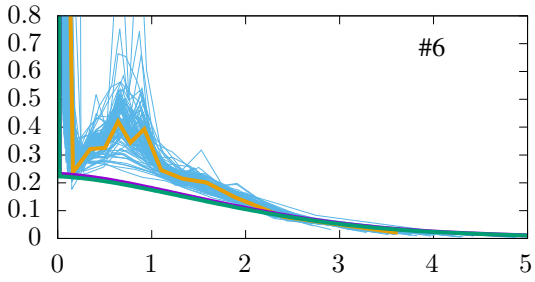
If one input has fewer but compatible dimensions, "broadcasting" is applied analogously as described for MatMul. Stochastically, we have already covered multiplication.

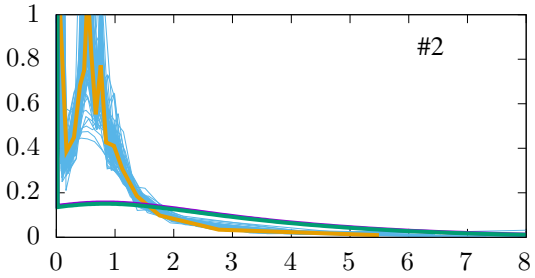
APPENDIX D
RESNET50 PLOTS

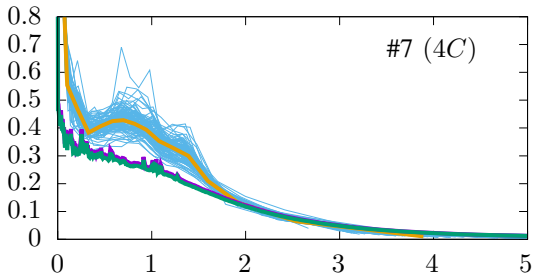
Below we show the plots for all of ResNet50's layers. For each layer, 100 random layer inputs were sampled from ImageNet to create the mixed input distribution. Using our numerical program (appendix B) and the results from section VI, we then estimated the layer's ($ReLU$) output distribution. In the plots we show the sampled output distribution for the 100 random inputs, their mixed distribution, the numerically estimated output distribution and the numerically estimated output distribution with a bit-flip fault in one of the $Conv$-summands. Most layers were simulated with $p^{\min} = 0.005$ (see appendix B). Some layers required finer sampling: $p^{\min} = 0.003$ for #5, #30, #36 and #42; $p^{\min} = 0.0025$ for #48; $p^{\min} = 0.002$ for #7 and #19. Spatial reduction are abbreviated $\frac{S}{2} := \frac{H}{2} \times \frac{W}{2}$, as above.

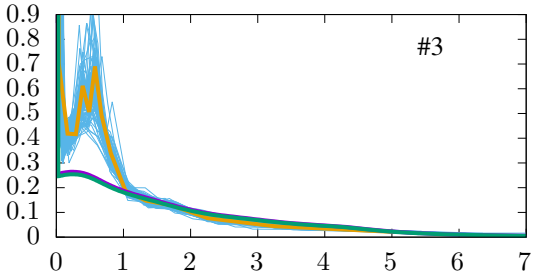
[3] See eg. E. W. Weisstein. MathWorld-A Wolfram Web Resource. [Online]. Available: http://mathworld.wolfram.com

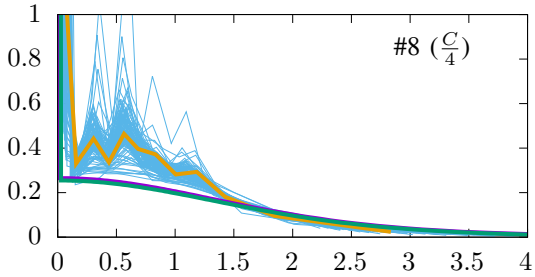

#1
sampled
mixed
estimated
est. corrupted

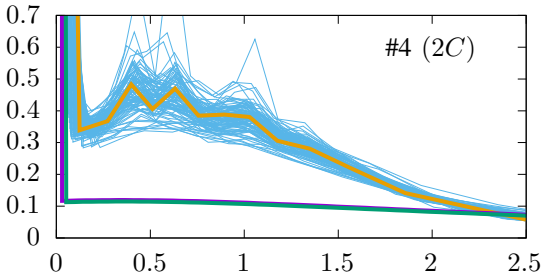

#6

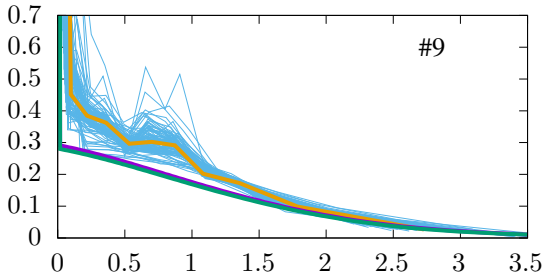

#2

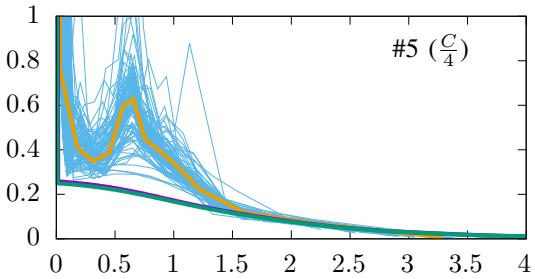

#7 (4C)

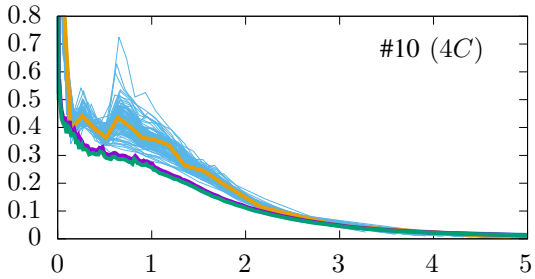

#3

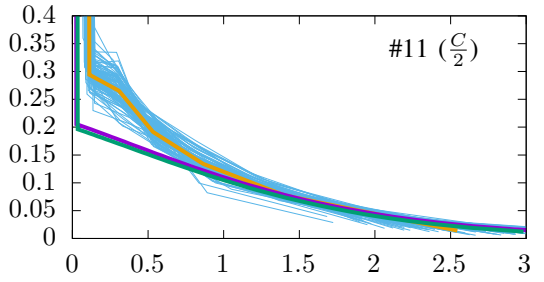

#8 (C/4)

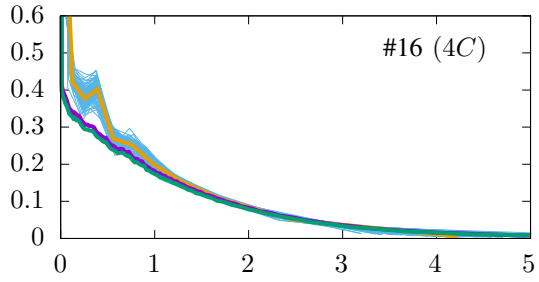

#4 (2C)

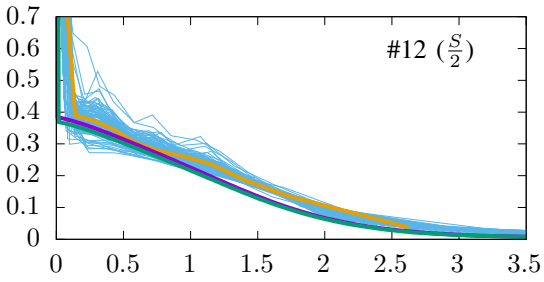

#9

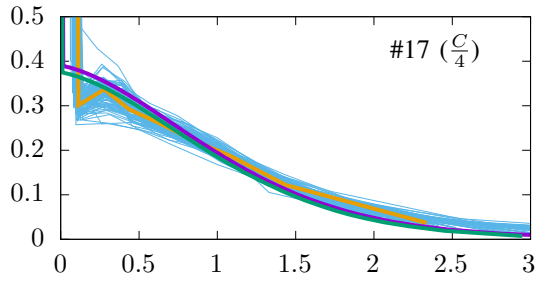

#5 (C/4)

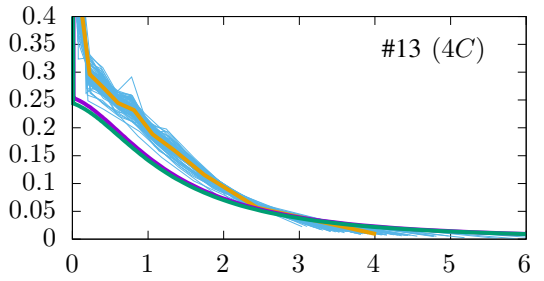

#10 (4C)

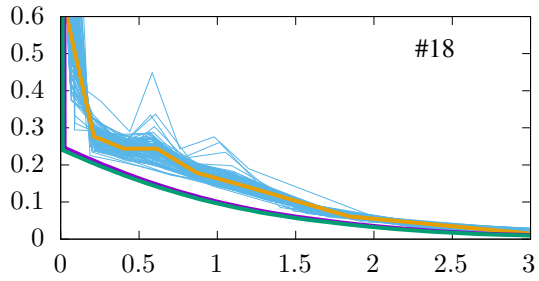

#11 ($\frac{C}{2}$)

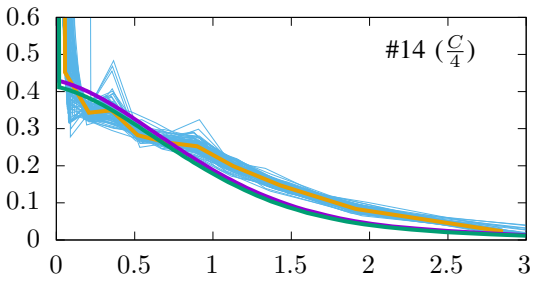

#16 ($4C$)

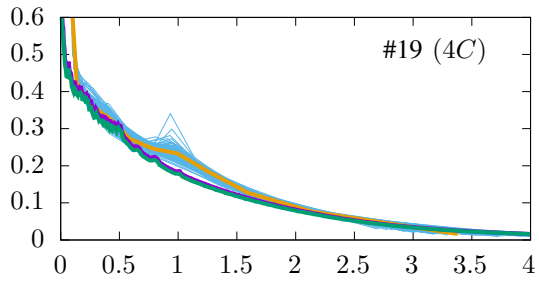

#12 ($\frac{S}{2}$)

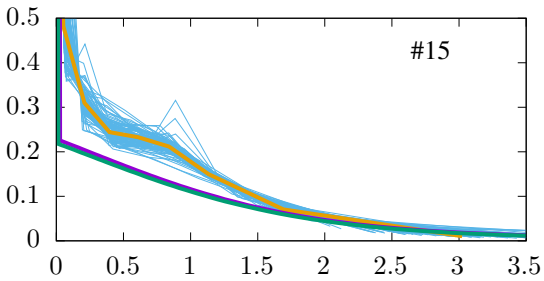

#17 ($\frac{C}{4}$)

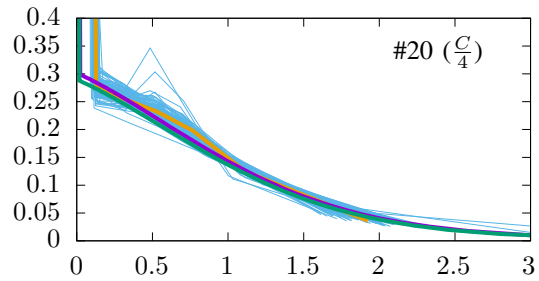

#13 ($4C$)

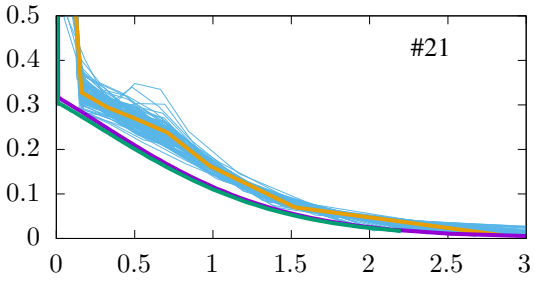

#18

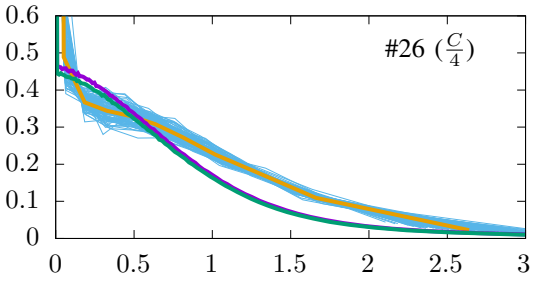

#14 ($\frac{C}{4}$)

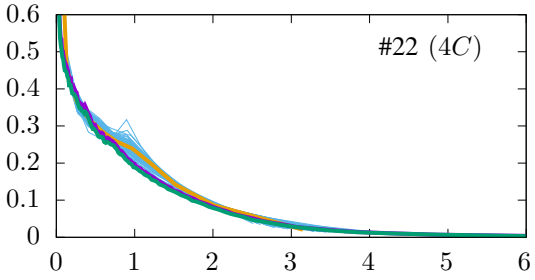

#19 ($4C$)

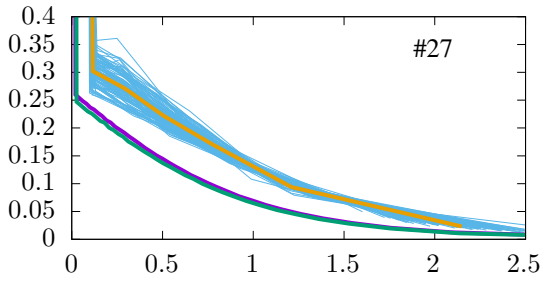

#15

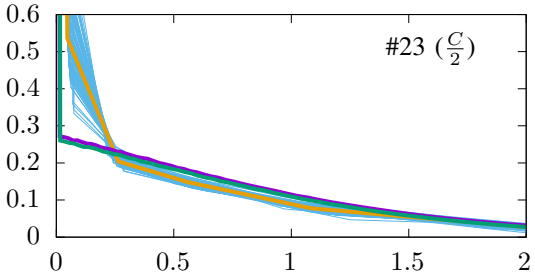

#20 ($\frac{C}{4}$)

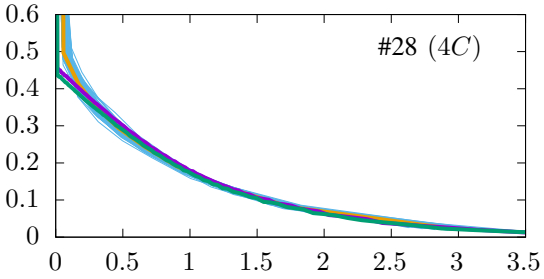

#21

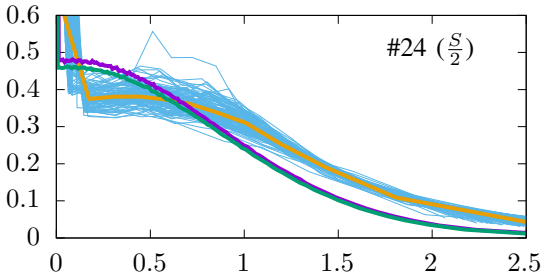

#26 ($\frac{C}{4}$)

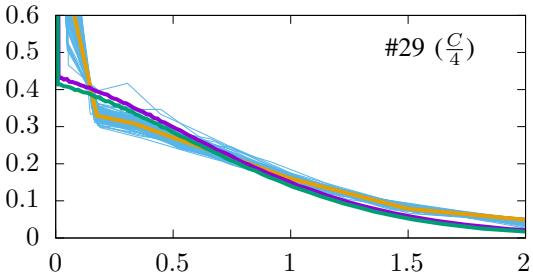

#22 ($4C$)

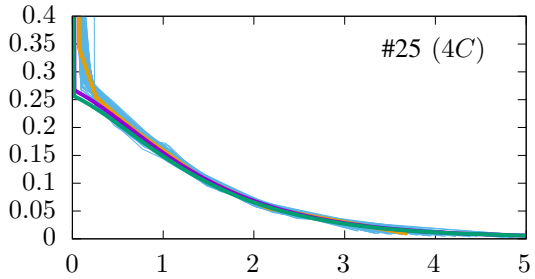

#27

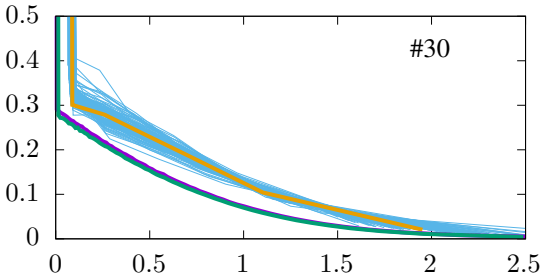

#23 ($\frac{C}{2}$)

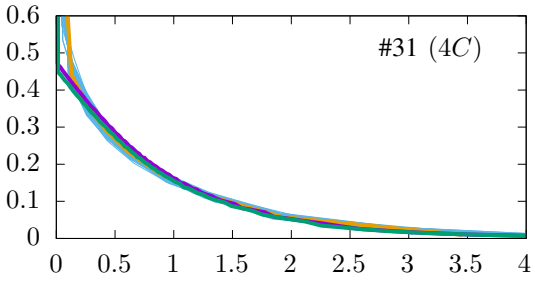

#28 ($4C$)

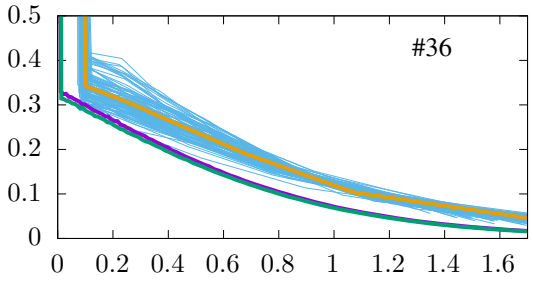

#24 ($\frac{S}{2}$)

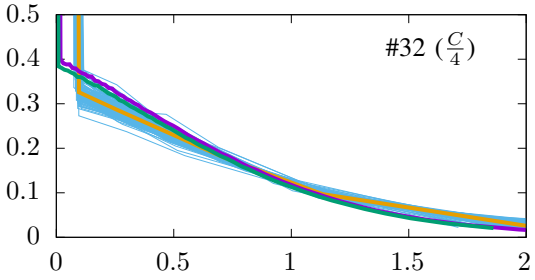

#29 ($\frac{C}{4}$)

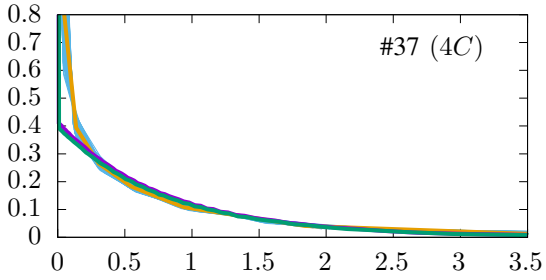

#25 ($4C$)

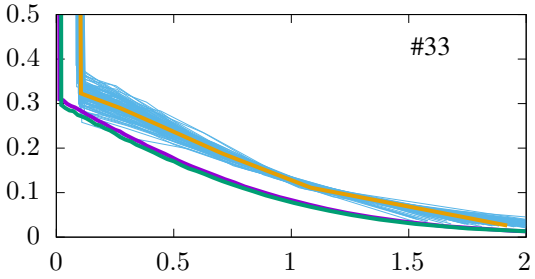

#30

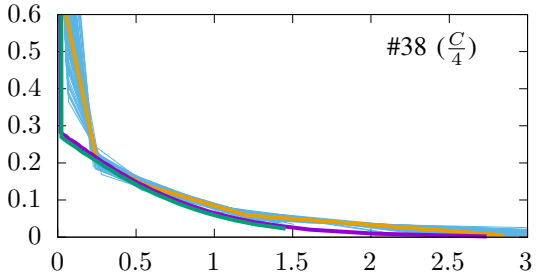

#31 (4C)

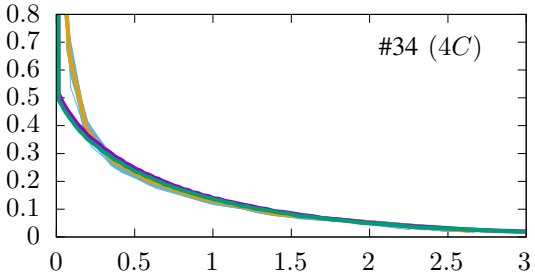

#36

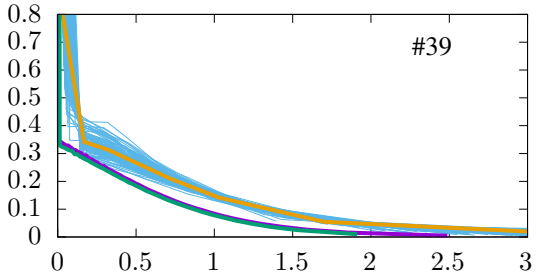

#32 (C/4)

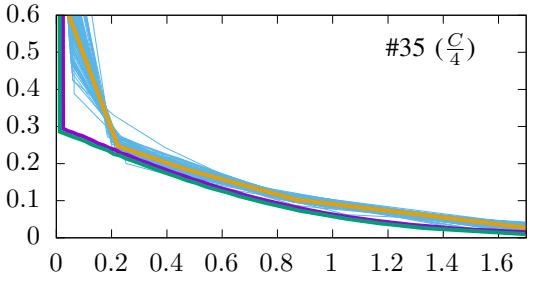

#37 (4C)

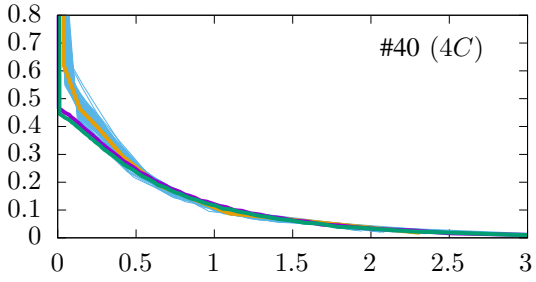

#33

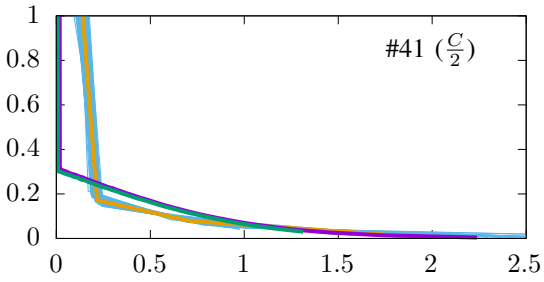

#38 (C/4)

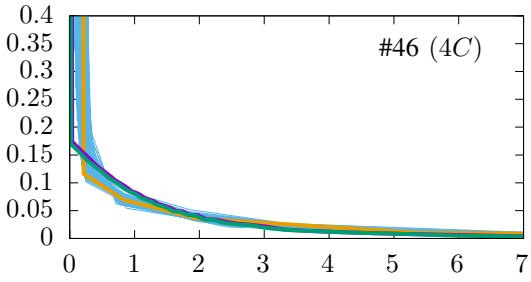

#34 (4C)

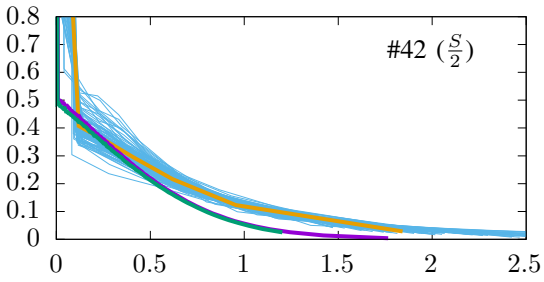

#39

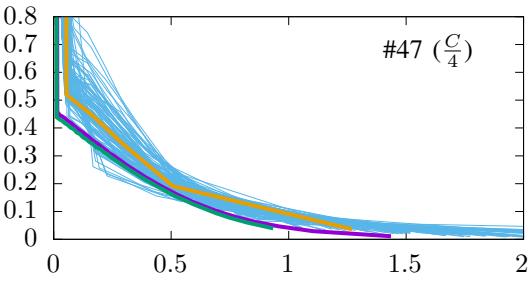

#35 (C/4)

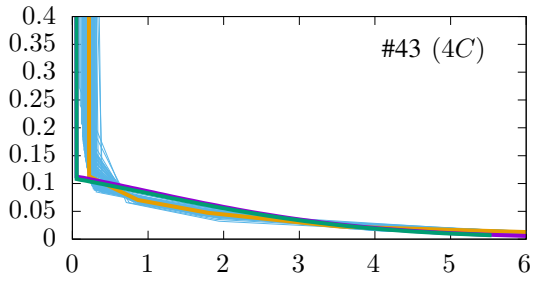

#40 (4C)

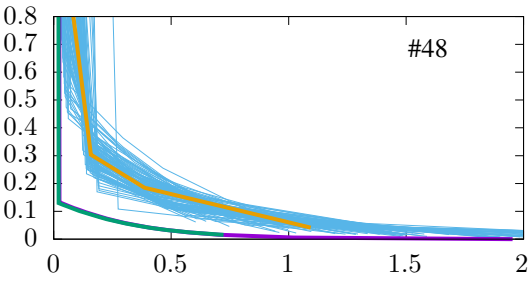

#41 ($\frac{C}{2}$)

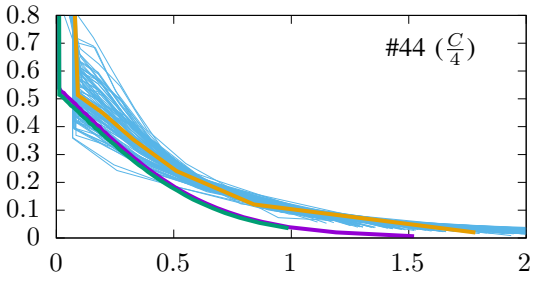

#46 ($4C$)

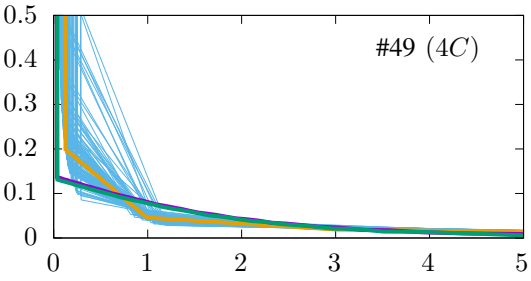

#42 ($\frac{S}{2}$)

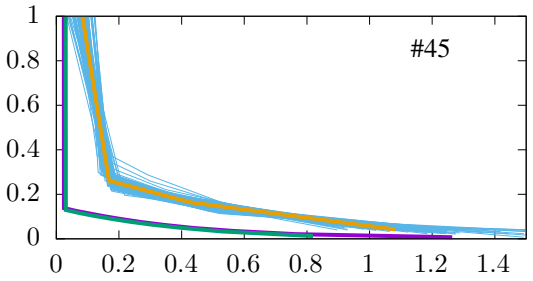

#47 ($\frac{C}{4}$)

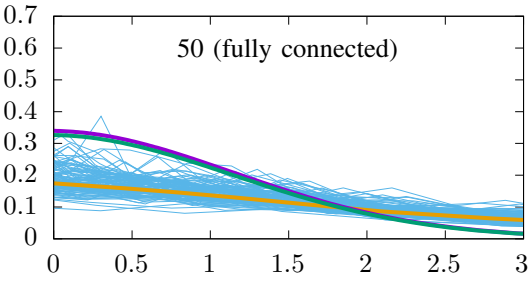

#43 ($4C$)


#48


#44 ($\frac{C}{4}$)


#49 ($4C$)


#45


50 (fully connected)