Andrei Ivanov, Siyuan Shen, **Gioele Gottardo**, Marcin Chrapek, Afif Boudaoud, Timo Schneider, Luca Benini, Torsten Hoefler

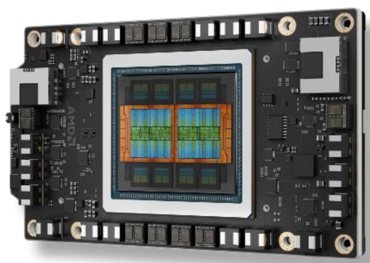
PerfDojo: Automated ML Library Generation for Heterogeneous Architectures

Introduction

Specialized High-Performance Accelerators

GPUs

nvidia.com



amd.com

TPUs / ASICs

cloud.google.com

FPGAs

amd.com

Evolving Processor Architectures

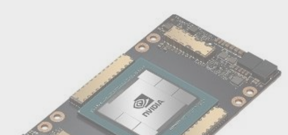
Advanced Vector Extensions (AVX)
Advanced Matrix Extensions (AMX)
ARM Neon
ARM Scalable Vector Extension (SVE)
ARM Scalable Matrix Extension (SME)
AMD Matrix Cores
NVIDIA Tensor Cores
Apple Matrix Coprocessor
RISC-V Stream Semantic Registers (SSR) [1]
RISC-V Floating-Point Repetition (FREP) [1]

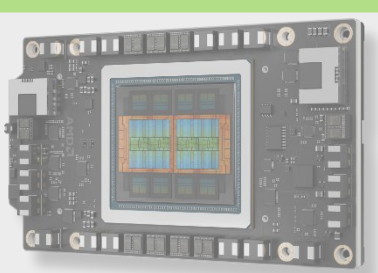
[1] F. Zaruba, F. Schuiki, T. Hoefler and L. Benini, "Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," in *IEEE Transactions on Computers*

Introduction

Specialized High-Performance Accelerators

GPUs | **TPUs / ASICs** | **FPGAs**





amd.com

cloud.google.com

amd.com

Evolving Processor Architectures

Advanced Vector Extensions (AVX)
Advanced Matrix Extensions (AMX)
ARM Neon
ARM Scalable Vector Extension (SVE)
Apple Matrix Coprocessor
RISC-V Stream Semantic Registers (SSR) [1]
RISC-V Floating-Point Repetition (FREP) [1]

Fully utilizing the peak performance is increasingly difficult

[1] F. Zaruba, F. Schuiki, T. Hoefler and L. Benini, "Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," in *IEEE Transactions on Computers*

Introduction

Language Models are Unsupervised Multitask Learners

Mistral 7B

Google DeepMind 2024-02-21

Gemma: Open Models Bas

OUTRAGEOUSLY LARGE NEURAL NETWORKS:
THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

A SIMPLE AND EFFECTIVE PRUNING APPROACH FOR LARGE LANGUAGE MODELS

QLORA: Efficient Finetuning of Quantized LLMs

T-TRAINING QUANTIZATION
RAINED TRANSFORMERS

PerfDojo

abstract **code-transformation environment** for hardware-specific optimizations

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xi

ABSTRACT

PerfLLM

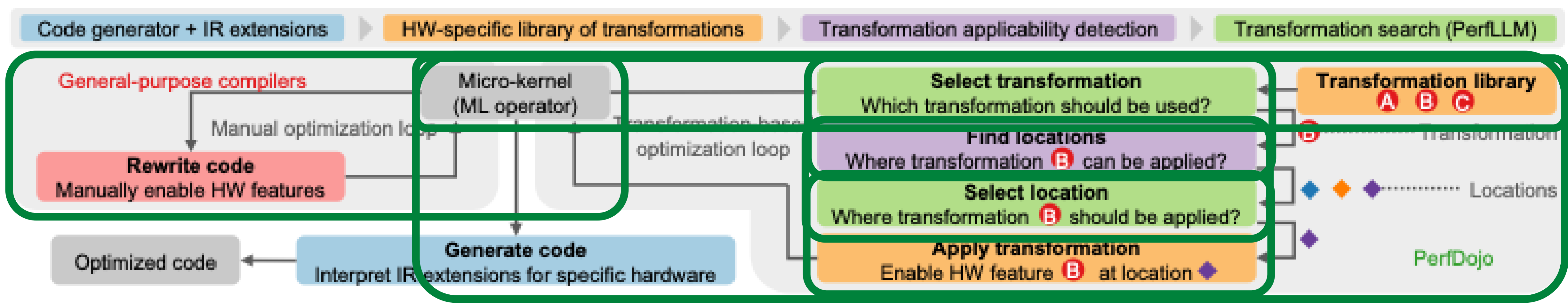
automatic optimization methodology, optimizing code **without prior architecture knowledge**

ABSTRACT

Large language models (LLMs) have revolutionized the field of artificial intelligence, enabling natural language processing tasks that were previously thought to be exclusive to humans. In this work, we introduce QWEN[1], the first installment of our large language model series. QWEN is a comprehensive language model series that encompasses distinct models with varying parameter counts. It includes QWEN, the base pretrained language models, and QWEN-CHAT, the chat models finetuned with human alignment techniques. The base language models consistently demonstrate superior performance across a multitude of downstream tasks, and the chat models, particularly those trained using Reinforcement Learning from Human Feedback (RLHF), are highly competitive. The chat models possess advanced tool-use and planning capabilities for creating agent applications, showcasing impressive performance even when compared to bigger models on complex tasks like utilizing a code interpreter. Furthermore, we have developed coding-specialized models, CODE-QWEN and CODE-QWEN-CHAT, as well as mathematics-focused models, MATH-QWEN-CHAT, which are built upon base language models. These models demonstrate significantly improved performance in comparison with open-source models, and slightly fall behind the proprietary models.

model compression, the applicability and performance of existing compression techniques is limited by the scale and complexity of GPT models. In this paper, we address this challenge, and propose GPTQ, a new one-shot weight quantization method based on approximate second-order information, that is both highly-accurate and highly-efficient. Specifically, GPTQ can quantize GPT models with 175 billion parameters in approximately four GPU hours, reducing the bitwidth down to 3 or 4 bits per weight, with negligible accuracy degradation relative to the uncompressed baseline. Our method more than doubles the compression gains relative to previously-proposed one-shot quantization methods, preserving accuracy, allowing us for the first time to execute an 175 billion-parameter model inside a single GPU for generative inference. Moreover, we also show that our method can still provide reasonable accuracy in the *extreme quantization* regime, in which weights are quantized to 2-bit or even *ternary* quantization levels. We show experimentally that these improvements can be leveraged for end-to-end inference speedups over FP16, of around 3.25x when using high-end GPUs (NVIDIA A100) and 4.5x when using more cost-effective ones (NVIDIA A6000). The implementation is available at https://github.com/IST-DASLab/gptq.

PerfDojo: Manual vs. PerfDojo's transformation-centric optimization workflows



The transformation-centric workflow enforces semantic preservation by construction

PerfDojo: Overview of the PerfLLM training pipeline

Iterate over all transformations and choose one as per the ϵ-***greedy policy***

Compute loss based on ***Max Q-learning*** and ***Double Q-learning***

Transformation represented as the concatenation of s and s'

Embedding $s = E(k)$

Kernel, k

Applicable transformation, a

Kernel after transformation, k'

Finetuned LLM

BERT DeepSeek LlaMA ...

$\begin{bmatrix} 0.9 \\ 0.7 \\ \vdots \\ 0.3 \end{bmatrix}$

$\begin{bmatrix} 0.5 \\ 0.7 \\ \vdots \\ 0.2 \end{bmatrix}$

Policy Network

Parameter θ

Update θ

Update θ^-

Target Network

Parameter θ^-

Applicable transformations at the next step

Embedding $s' = E(k')$

Sampled mini-batch

(s, s')

(a', s')

Scaling constant

Compile and execute

Reward $r_{s,a} = \frac{c}{T_{s'}}$

Store $(s, r_{s,a}, s', a')$

Replay buffer

r

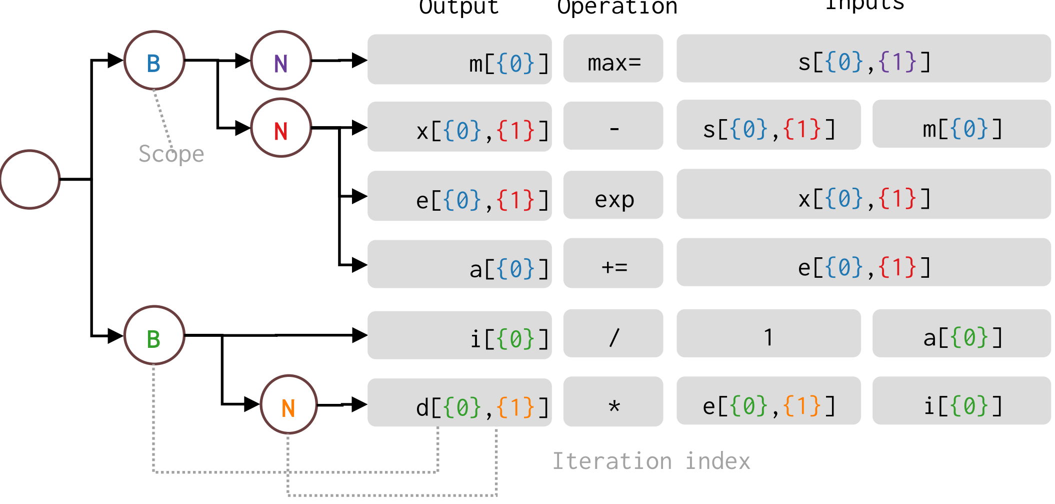
PerfDojo: Representation

Softmax kernel:

$$b \in \{1..B\}$$
$$n \in \{1..N\}$$

$$\begin{aligned} m_b &= \max_n s_{b,n} \\ x_{b,n} &= s_{b,n} - m_b \\ e_{b,n} &= \exp x_{b,n} \\ a_b &= \mathrm{sum}_n e_{b,n} \\ i_b &= 1/a_b \\ d_{b,n} &= e_{b,n} i_b \end{aligned}$$

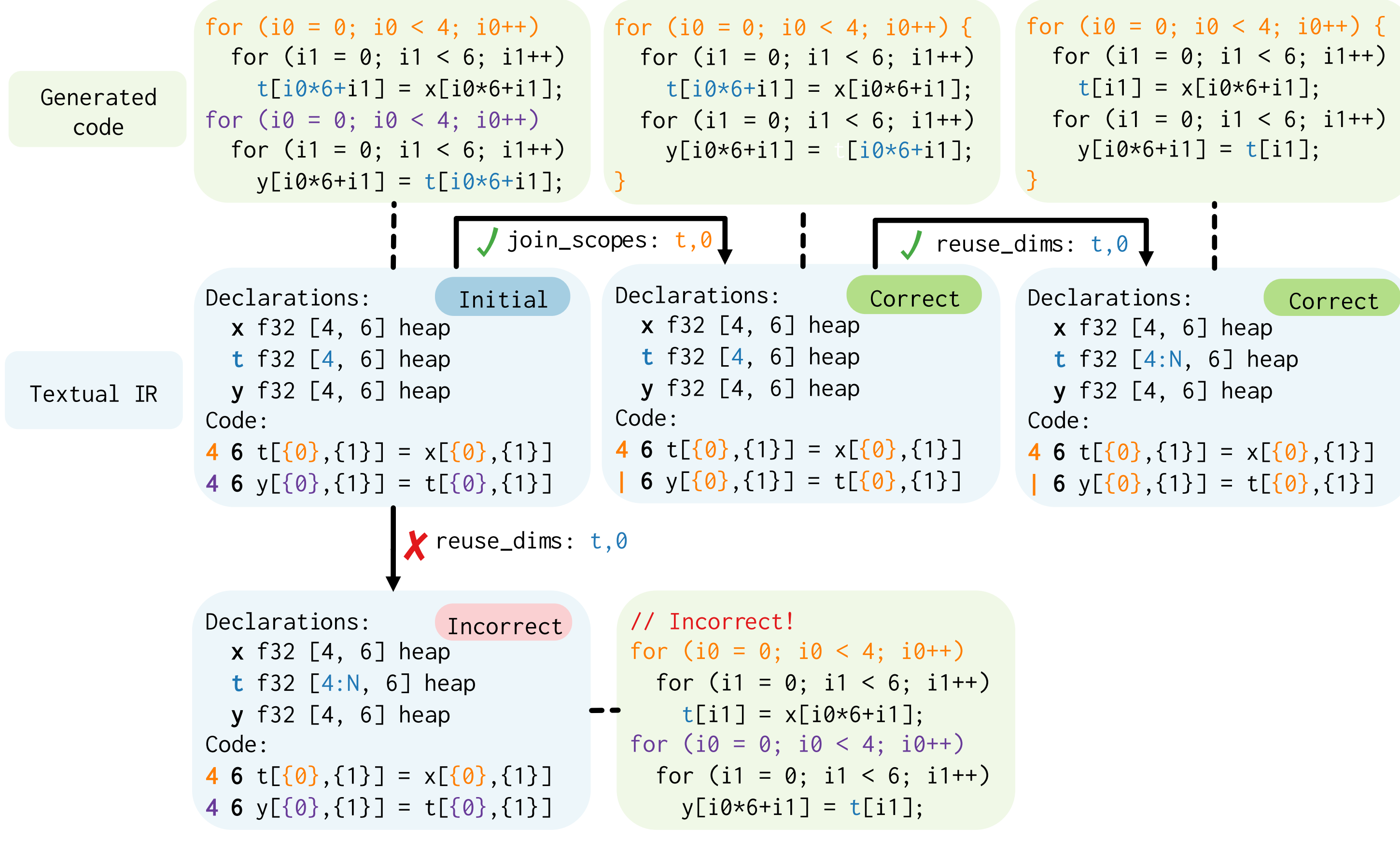
```
for (b=0;b<B;b++) {
  m[b] = -DBL_MAX; for (n=0;n<N;n++) { m[b] = max(m[b],s[b*N+n]); }
  a[b] = 0;        for (n=0;n<N;n++) { x[b*N+n] = s[b*N+n]-m[b];
                                       e[b*N+n] = exp(x[b*N+n]);
                                       a[b]    += e[b*N+n];        }}
for (b=0;b<B;b++) {                    i[b]     = 1 / a[b];
                   for (n=0;n<N;n++) { d[b*N+n] = e[b*N+n]*i[b]; }}
```




```
B N m[{0}]   max= s[{0},{1}]
| N x[{0},{1}] = s[{0},{1}] - m[{0}]
| | e[{0},{1}] = exp( x[{0},{1}] )
| | a[{0}]    += e[{0},{1}]
B   i[{0}]     = 1 / a[{0}]
| N d[{0},{1}] = e[{0},{1}] * i[{0}]
```

@spcl
@spcl_eth

PerfDojo: Transformations



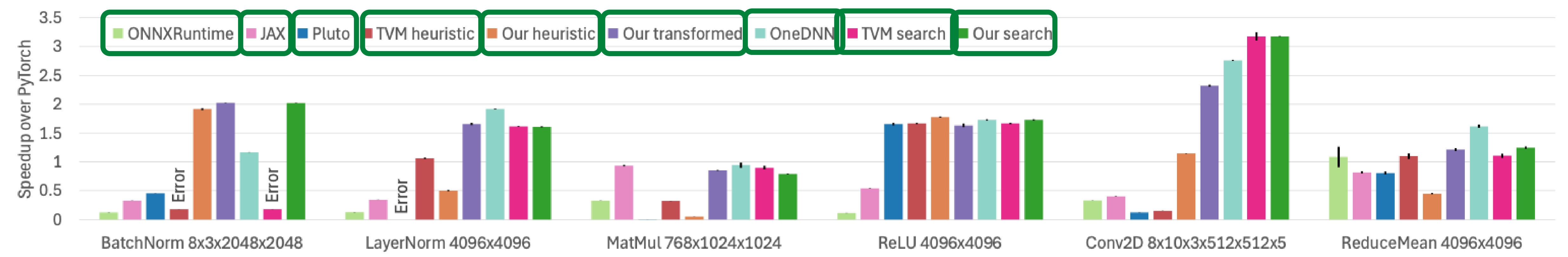
Transformation	Description	Example before	Example after
swap_ops	Swaps adjacent operations or scopes	N y[{0}]=x[{0}] z=q	z=q N y[{0}]=x[{0}]
tile_scope	Tiles a scope	10 y+=x[{0}]	5 2 y+=x[{0}*2+{1}]
tile_buffer	Tiles a buffer dimension	M N y+=x[{0}*N+{1}]	M N y+=x[{0},{1}]
swap_scopes	Reorders adjacent scopes	M N y+=x[{0},{1}]	N M y+=x[{1},{0}]
join_scopes	Joins the bodies of adjacent scopes	M y[{0}]=x[{0}] M p[{0}]=q[{0}]	M y[{0}]=x[{0}] | p[{0}]=q[{0}]
create_temporary	Introduces a temporary array copy	M y+=x[{0}]	M t[{0}]=x[{0}] M y+=t[{0}]
enable_frep	Instantiates a scope using FREP	N y+= \\ x[{0}|s:5,d:1,h:2]	N:f y+= \\ x[{0}|s:5,d:1,h:2]
parallelize	Parallelizes a scope using OpenMP	N y[{0}]=x[{0}]	N:p y[{0}]=x[{0}]
parallelize_grid	Parallelizes a scope as a CUDA grid	N y[{0}]=x[{0}]	N:g y[{0}]=x[{0}]
parallelize_block	Parallelizes a scope as a CUDA block	N y[{0}]=x[{0}]	N:b y[{0}]=x[{0}]
parallelize_warp	Parallelizes a scope as a CUDA warp	N y[{0}]=x[{0}]	N:w y[{0}]=x[{0}]
vectorize_scope	Vectorizes a scope	4 y[{0}]=x[{0}]	4:v y[{0}]=x[{0}]

Transformation	Description	Example before	Example after
reuse_dims	Avoids materializing a dimension	x f32 [M, K] heap	x f32 [M:N, K] heap
reuse_buffers	Shares storage between arrays	x f32[M,K] heap y f32[M,K] heap	x f32[M,K] heap->x,y
swap_dims	Changes the memory layout of an array	x f32 [M, N, K] heap	x f32 [M, K, N] heap
increment_padding	Increases padding	x f32 [M,K+2] heap	x f32 [M,K+3] heap
repeat_padding	Doubles the padding	x f32 [M+3,K+3] heap	x f32 [M+6,K+3] heap
move_to_stack	Moves an array to the stack	x f32 [N] heap	x f32 [N] stack
vectorize_buffer	Vectorizes storage of a buffer	x f32 [4,16] stack	x f32x16 [4,16] stack
enable_ssr	Enables SSR for an array access	M N y+= x[{0},{1}]	M N y+= x[{0},{1}|s:5,d:1]
increase_ssr_depth	Increases the SSR depth	M N y+= \\ x[{0},{1}|s:5,d:1]	M N y+= \\ x[{0},{1}|s:5,d:2]
merge_ssr	Merges multiple SSR accesses into one	M y+=x[{0}|s:5,d:1] M y+=z[{0}|s:7,d:1]	M y+=x[{0}|s:5,d:1] M y+=z[{0}|s:5,d:1]
activate_ssr	Activates an SSR configuration	N y+= \\ x[{0}|s:5,d:1]	N y+= \\ x[{0}|s:5,d:1,h:2]

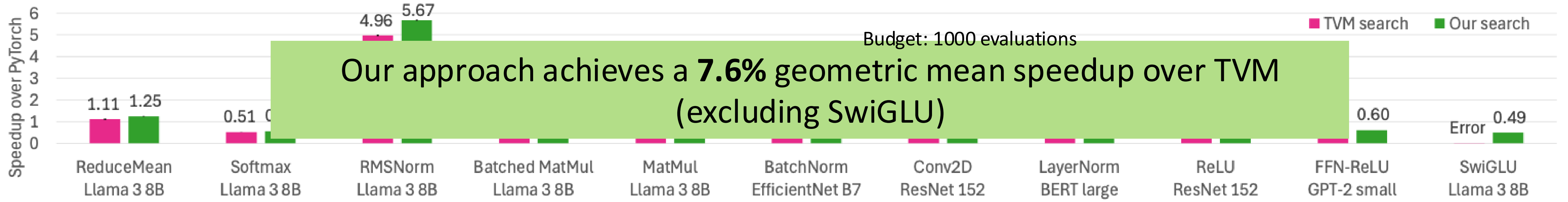
27 transformations with applicability detection

Evaluation: x86 + heuristics

Intel Xeon CPU E5-2695 v4
18 cores

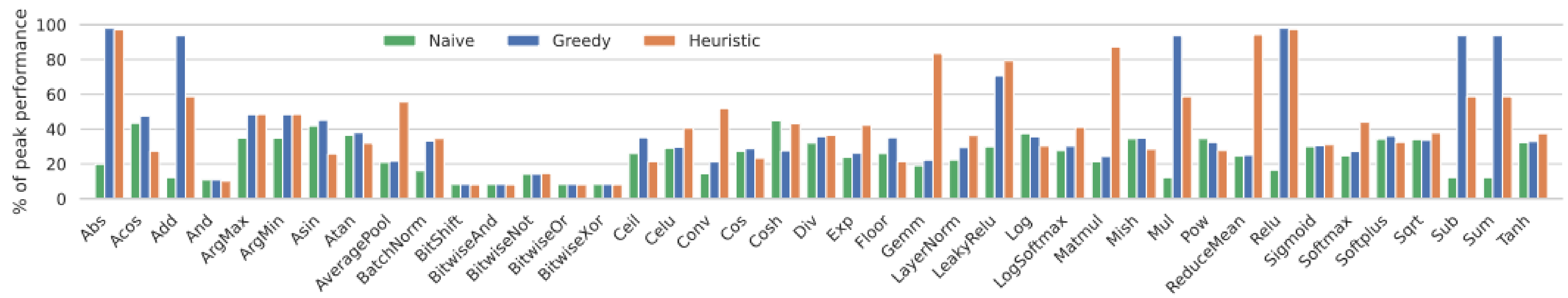


heuristic = single program evaluation pass
search = 1000 evaluations
transformed = applying transformations manually

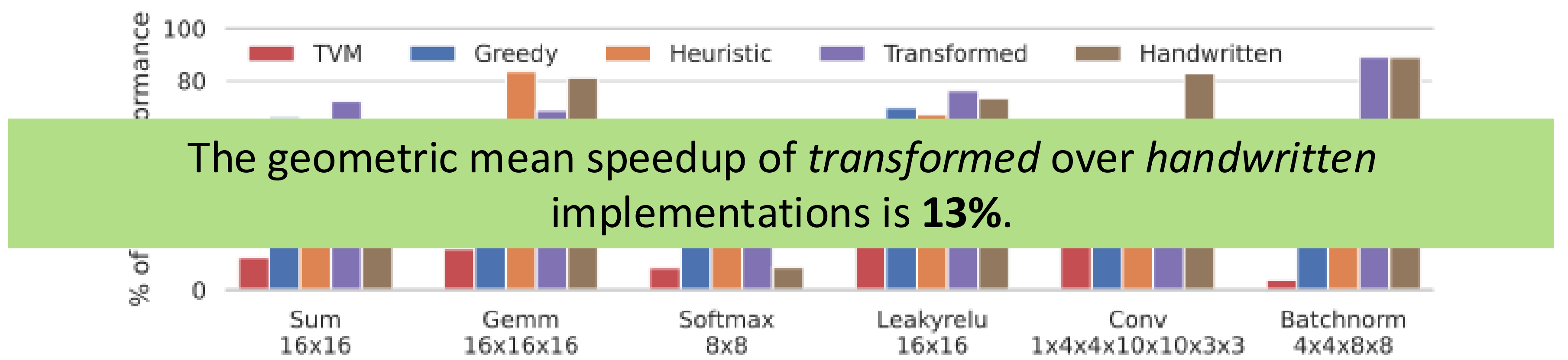


Our approach achieves a **7.6%** geometric mean speedup over TVM (excluding SwiGLU)

Evaluation: Snitch [1]



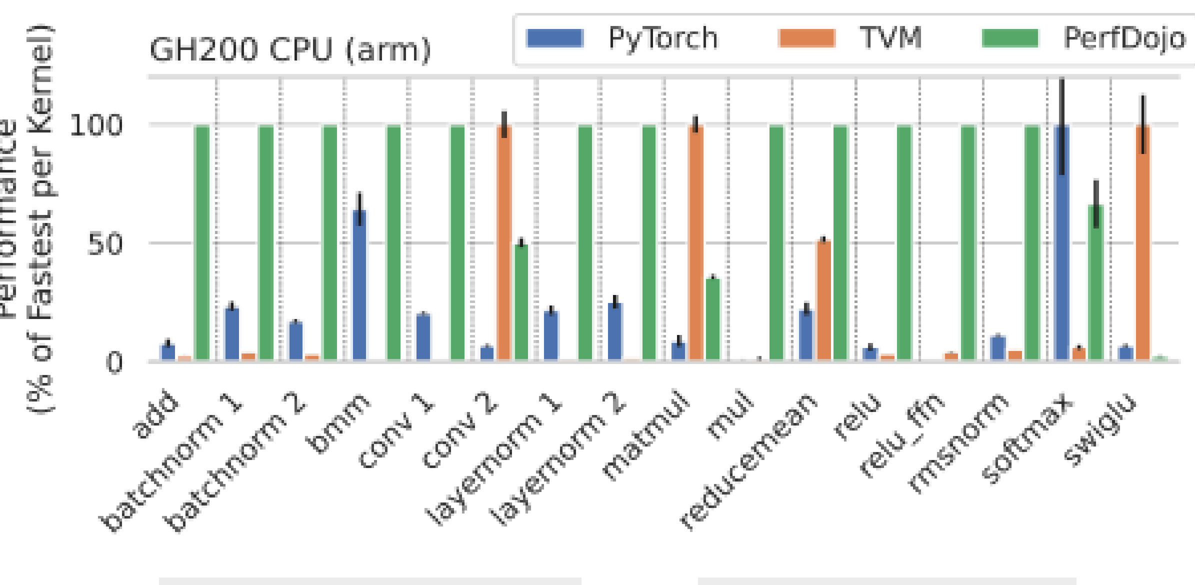
Naive = loop fusion and memory reuse until exhaustion
Greedy = *naive* + hardware-specific transformations (SSR/FREP)
Heuristic = includes analysis pass made by hardware expert to determine the order of transformations



The geometric mean speedup of *transformed* over *handwritten* implementations is **13%**.

[1] F. Zaruba, F. Schuiki, T. Hoefler and L. Benini, "Snitch: A Tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," in *IEEE Transactions on Computers*

PerfDojo: Evaluation on x86 & arm / no heuristics

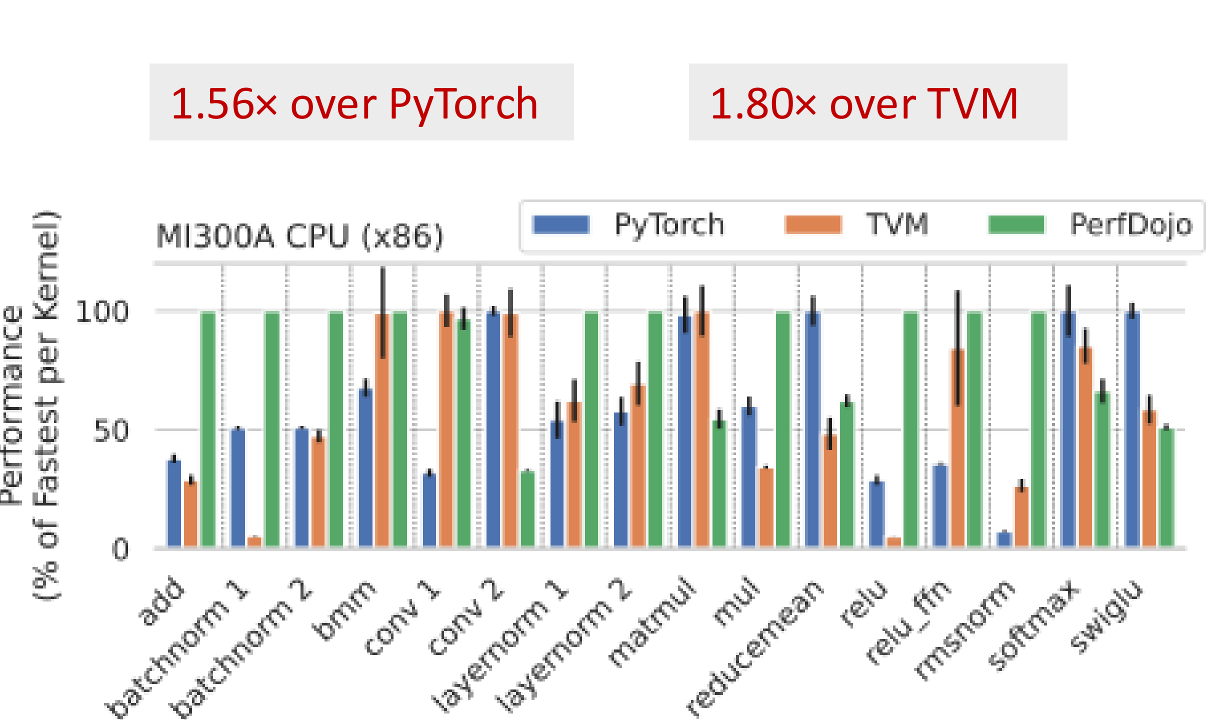


1.56× over PyTorch

1.80× over TVM

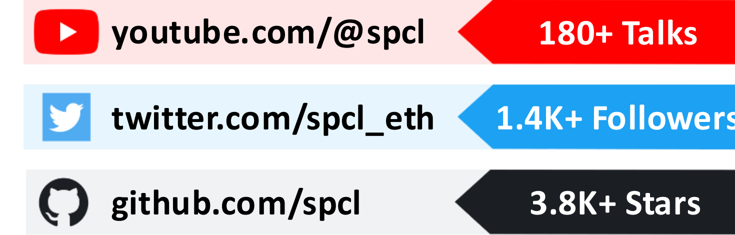
6.65× over PyTorch

13.65× over TVM



Conclusions

More of SPCL's research:



... or spcl.ethz.ch