facebook

TVM @ FB

Andrew Tulloch
Research Scientist

Background

- Excited to be here!
- Lots of FB folks in the audience
- Working in TVM since ~June
- Focusing on apply TVM to accelerate ML inference on CPUs/GPUs across mobile and server environments

Server ML Workloads @ FB

https://arxiv.org/abs/1811.09886 for more detail

- Rapidly growing in terms of capacity requirements
- Two key workloads are:
  - ranking/recommendation (feed and ads ranking)
  - computer vision (classification, detection, OCR, video, etc)
- For various reasons, mostly leverage various generations of Intel CPUs

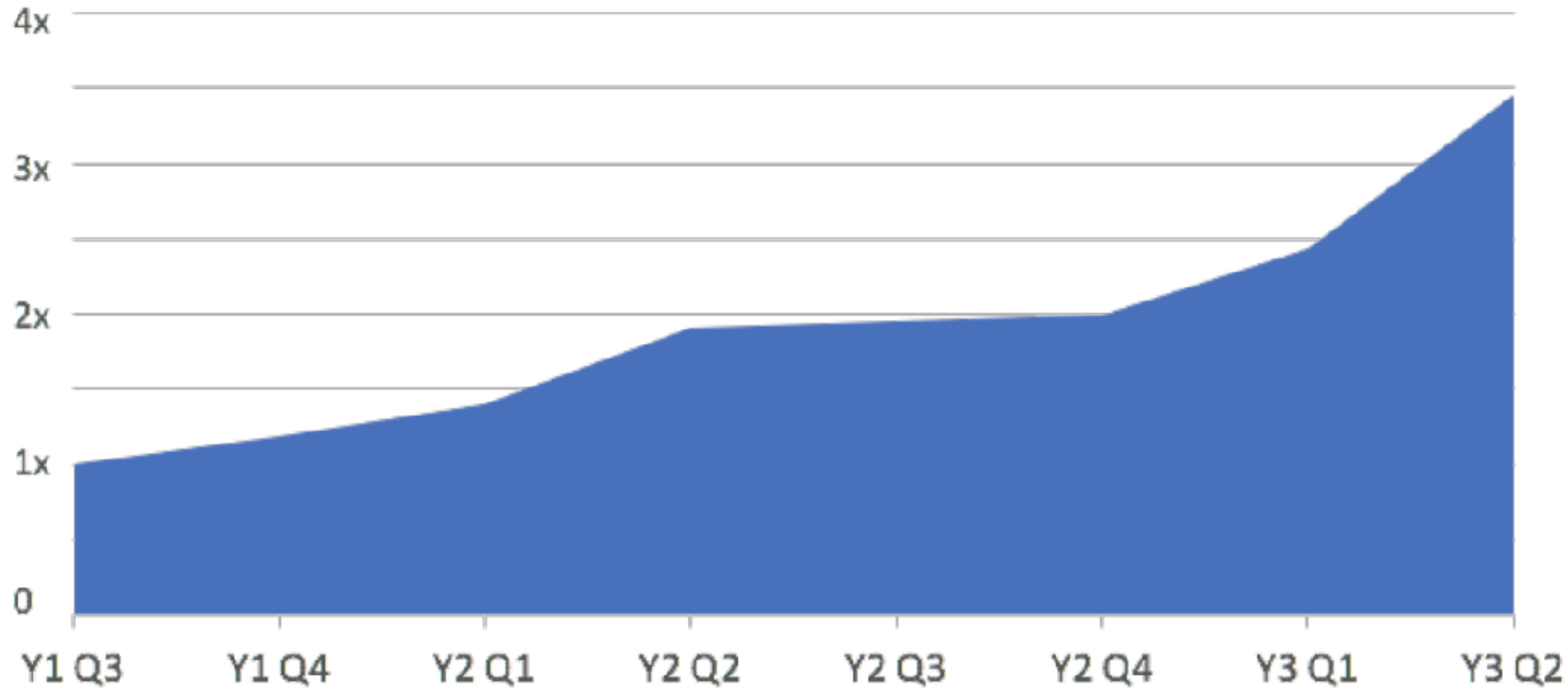


Figure 1: Server demand for DL inference across data centers

Source: https://arxiv.org/abs/1811.09886

Mobile ML Workloads @ FB

See upcoming HPCA-2019 publication

- Main workloads are real-time computer vision workloads (object detection, tracking, segmentation, etc.)
- Huge variety of computational platforms to target (ARMv7/Aarch64 CPUs, Metal/OpenGL GPUs, Hexagon DSPs, ...)
- Introduces new constraints (esp: code size)

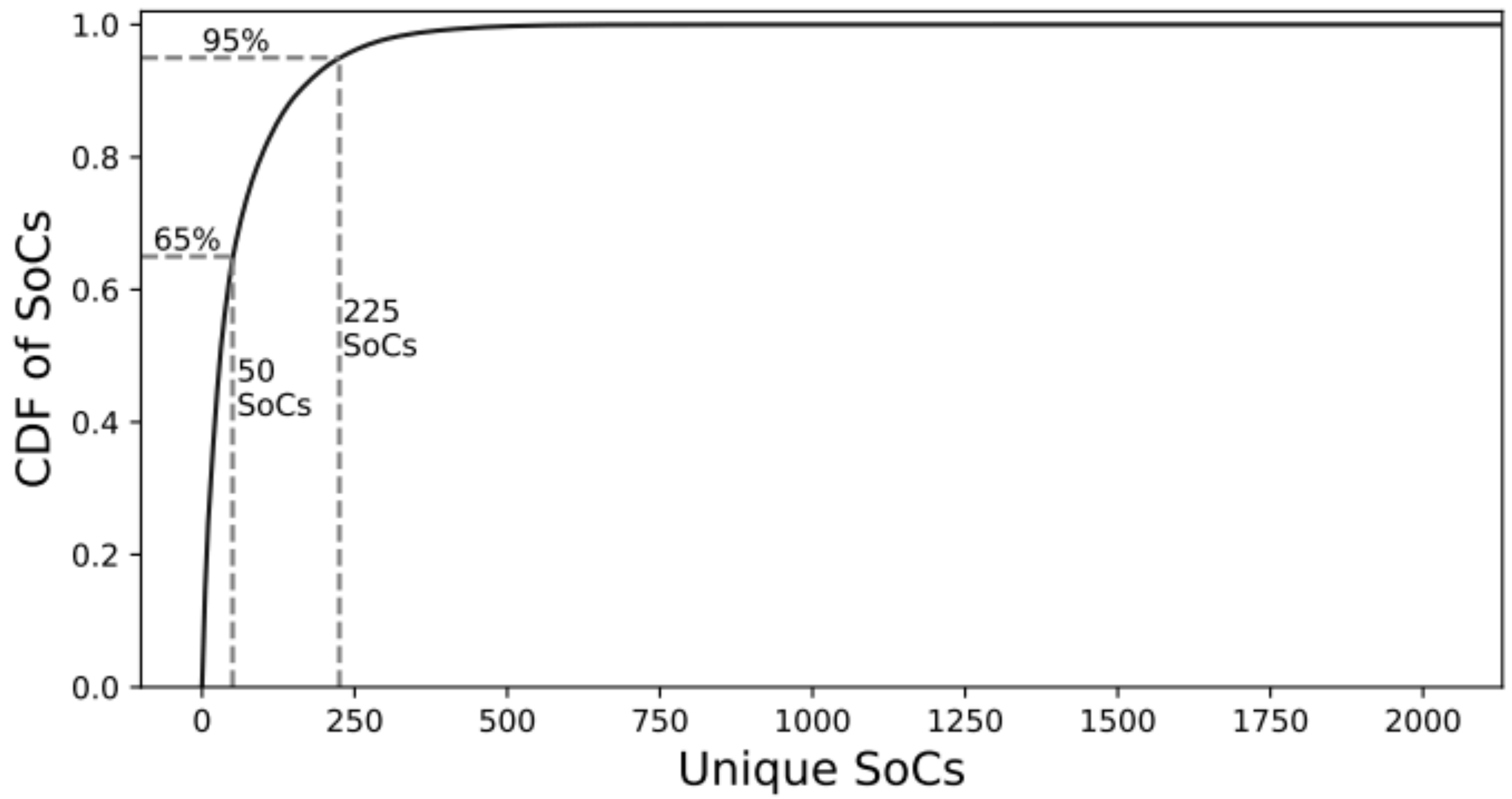


Figure 2: There is no standard mobile SoC to optimize for. The top 50 most common SoCs account for only 65% of the smartphone market.

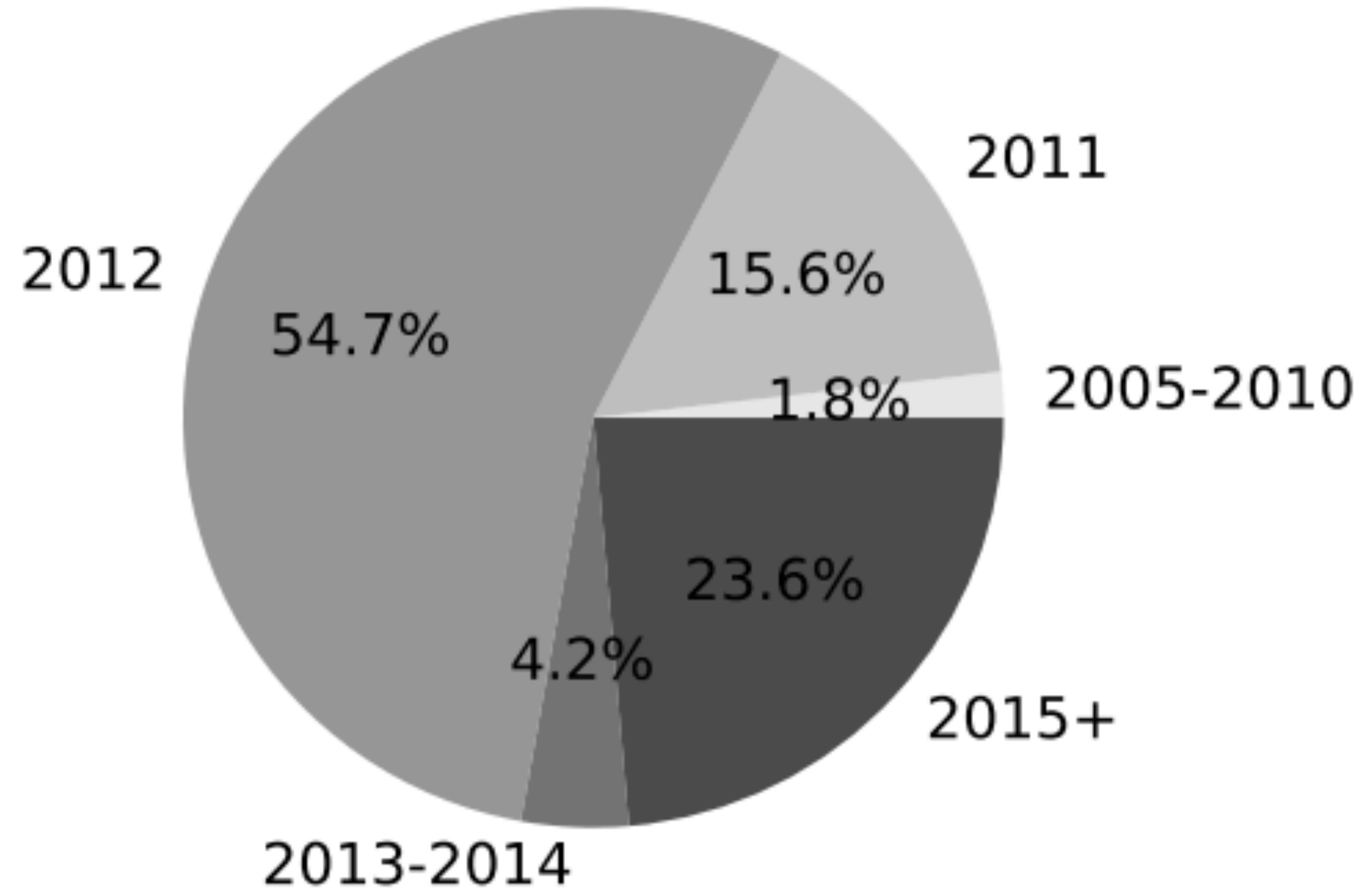


Figure 3: The most commonly-used mobile processors, Cortex A53, are at least six years old. In 2018, only a fourth of smartphones implemented CPU cores designed in 2013 or later.

Source: sed ut unde omnis

Mask-RCNN

Mask-RCNN

Object Detection

Object Detection

Why TVM (for us)?

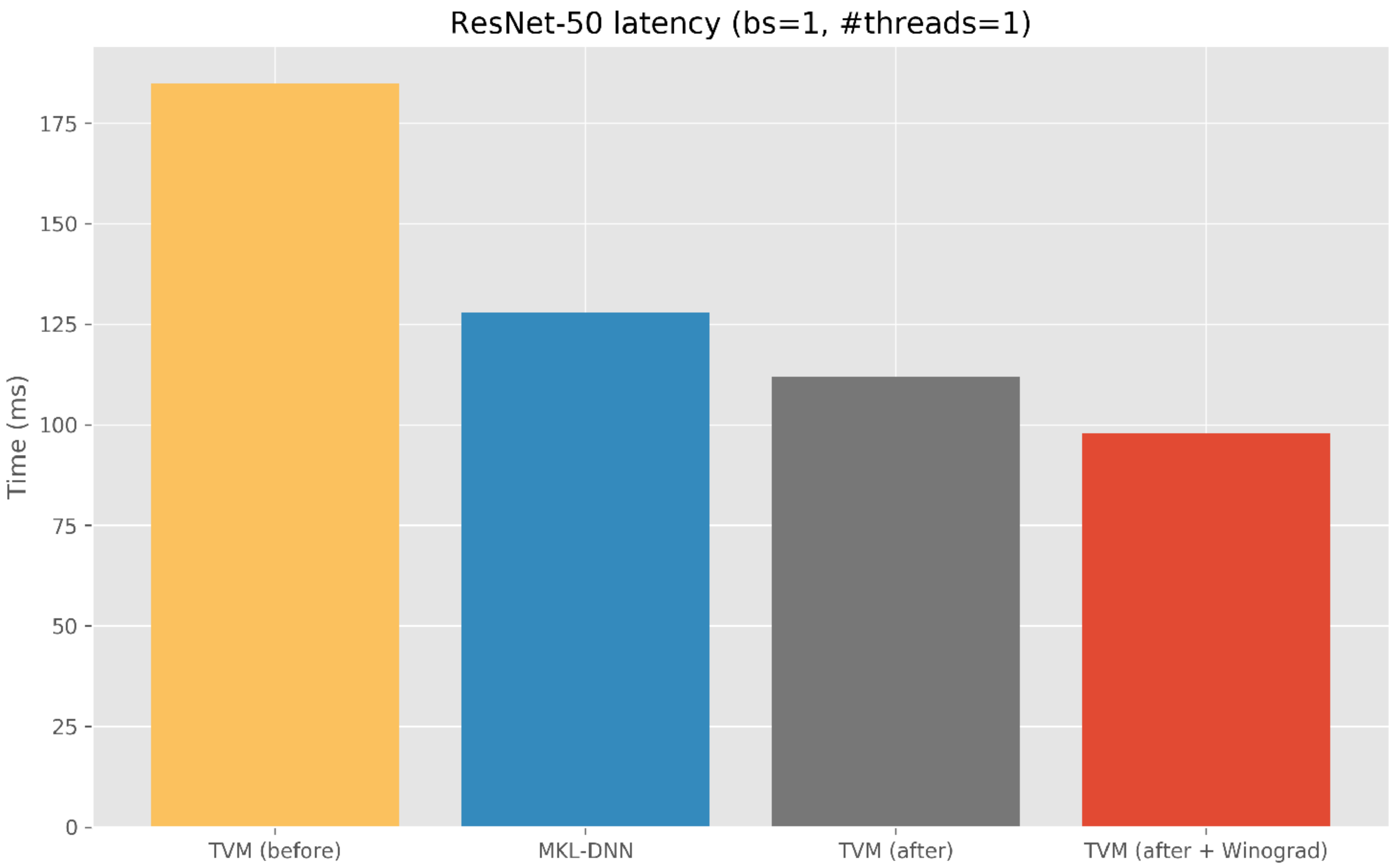
- More hardware (NPUs, TPUs, GPUs, DSPs, ...)
- More numerics (fp32, fp16/bfloat16, int8, int1, ...)
- FLOPs/BW ratio increasing, exposing inefficiencies
- Existing approaches (manual fusion, etc) unsustainable

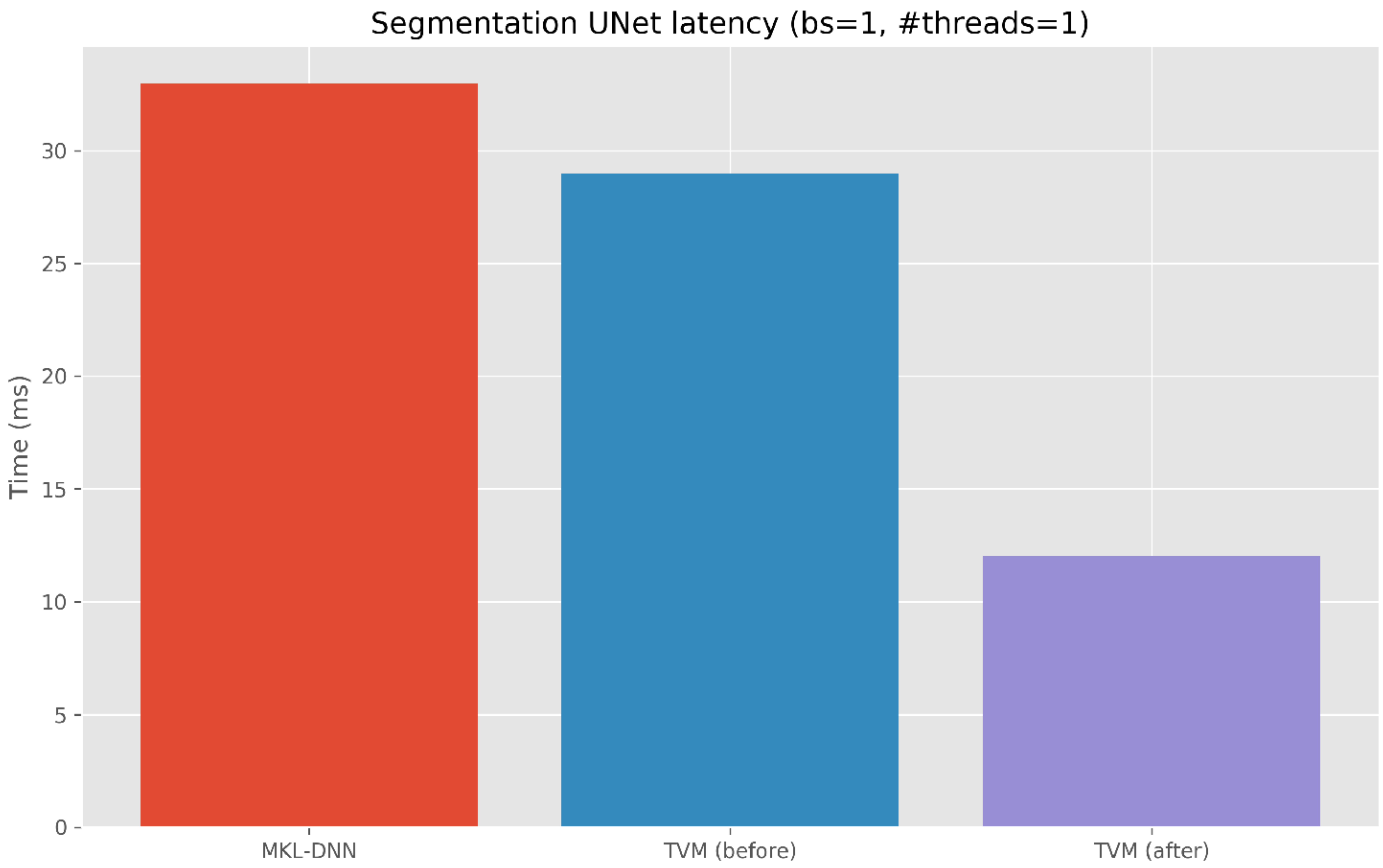
Improving TVM @ FB

TVM for Server CV

https://discuss.tvm.ai/t/improved-direct-winograd-nchwc-cpu-implementation-with-resnet-50-results/

- First workload we targeted, great fit
- Goal was to beat current FP32 production baselines (MKL-DNN)
- Key improvements:
  - Entire graph in NCHWc (no graph tuner)
  - Implement efficient NCHWc Winograd (https://github.com/dmlc/tvm/pull/2111)


ResNet-50 latency (bs=1, #threads=1)
Time (ms)
0
25
50
75
100
125
150
175
TVM (before)
MKL-DNN
TVM (after)
TVM (after + Winograd)

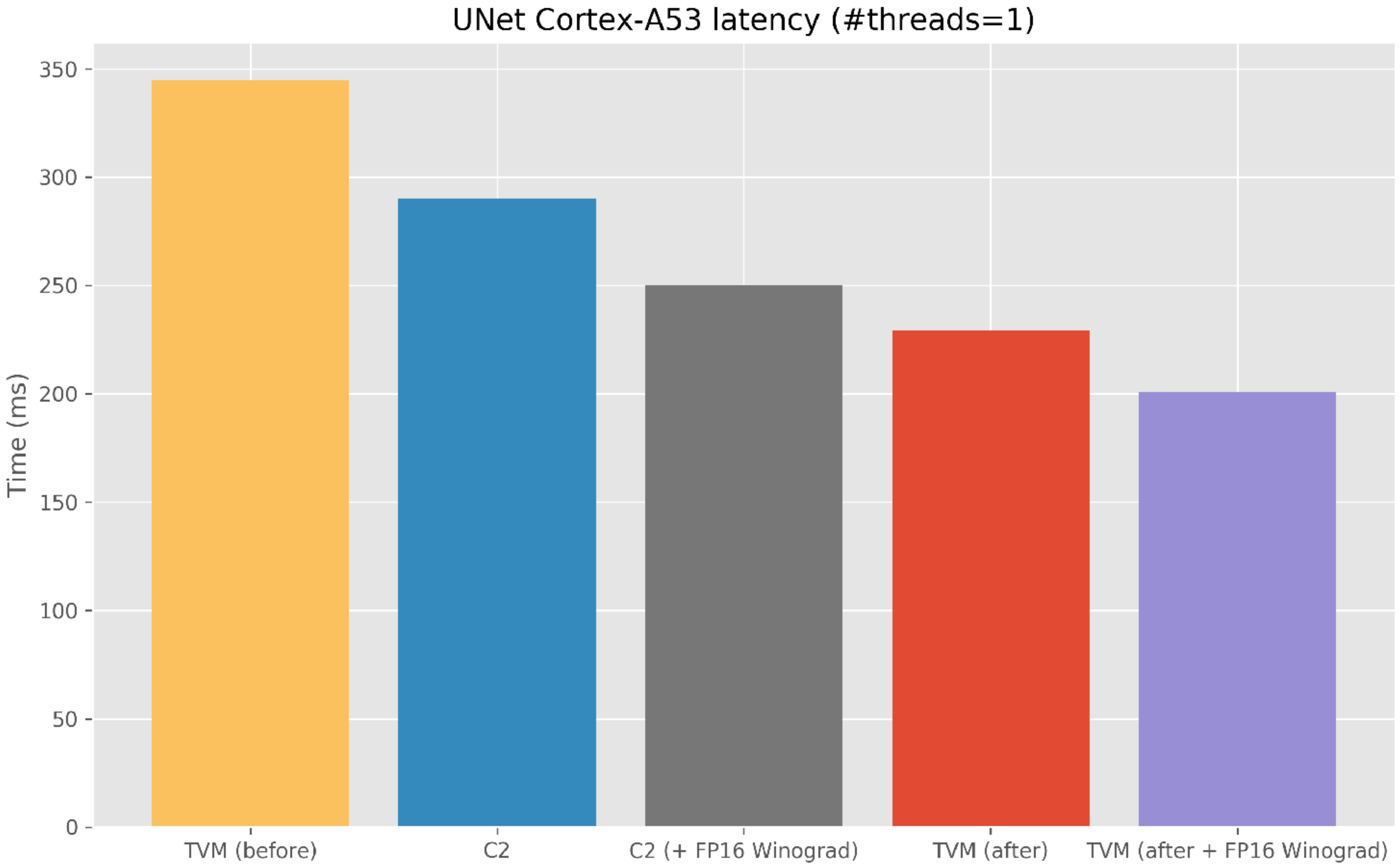


Segmentation UNet latency (bs=1, #threads=1)
Time (ms)
0
5
10
15
20
25
30
MKL-DNN
TVM (before)
TVM (after)

TVM for Mobile CV

https://discuss.tvm.ai/t/tvm-nnpack-performance-on-unet-armv7/1134

- Next, targeted proving we could beat our mobile CV models - highly optimized baseline
- Tensorization + custom layout to compete with NNPACK FP16 WT
- Leverage TVM for pointwise fusion, certain convolutions, fall back to baseline for other ops
- Replace runtime::ThreadPool with custom

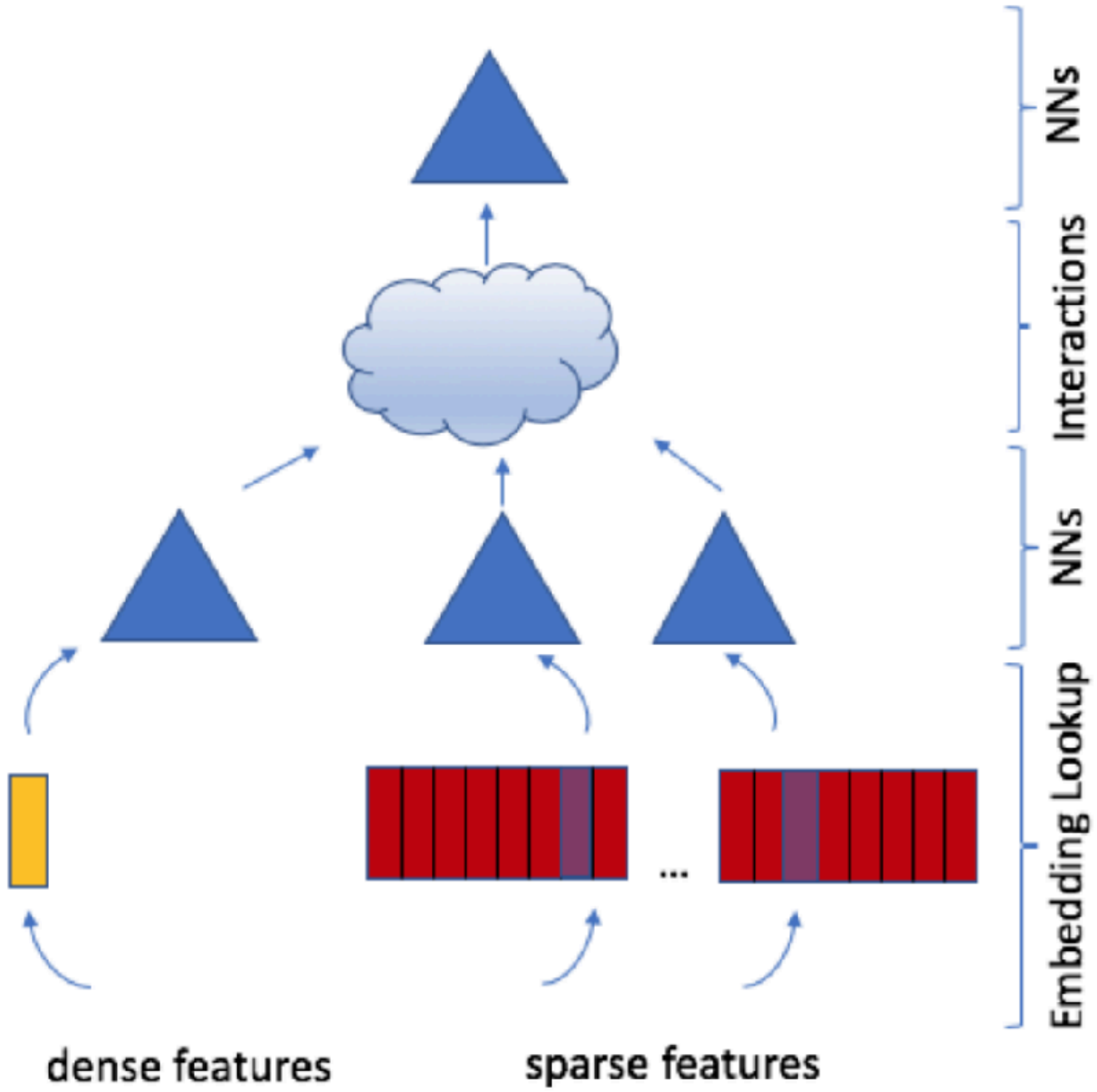


UNet Cortex-A53 latency (#threads=1)
Time (ms)
0
50
100
150
200
250
300
350
TVM (before)
C2
C2 (+ FP16 Winograd)
TVM (after)
TVM (after + FP16 Winograd)

TVM for Server Ranking

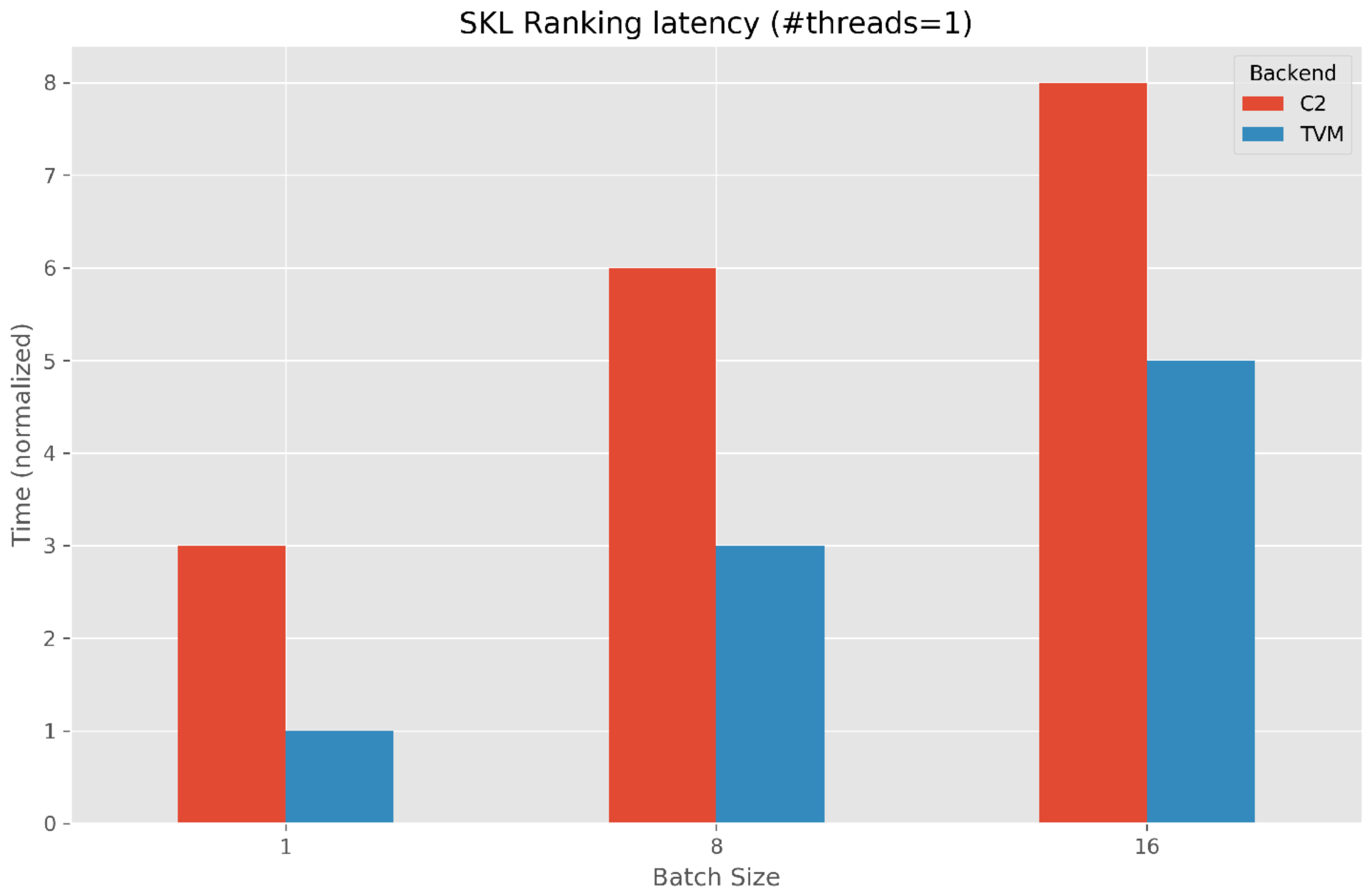
https://github.com/ajtulloch/tvm/tree/sparse-ops

- Architectures similar to e.g. Wide and Deep Networks, Deep Factorization Machines, etc.
- O(many trillions) of inferences/day.
- Mixture of sparse subgraphs (embedding lookups, pooling, pairwise products, etc), and dense subgraphs (fully-connected)
- New NNVM ops: sparse_lengths_sum,



Source: sed ut unc

Figure 2: A deep learning recommendation model


SKL Ranking latency (#threads=1)
Time (normalized)
Batch Size
Backend
C2
TVM
0
1
2
3
4
5
6
7
8
1
8
16

Some incremental ideas

TVM Core

For discussion with community

- Quantization (int8 and lower)
  - Highly tuned ukernels in FBGEMM (AVX2/AVX512) and QNNPACK (ARM NEON) could be useful.
- Constrained dynamism for shapes (codegen, runtime).
  - batch size in ranking
  - sentence length in NLP

TVM Mobile

For discussion with community

- OpenGL ES 3.2+ backend for mid/high-end Android GPUs
- Hexagon backend
- "Interpreter bundling" for highly code-size-constrained applications
- Ultra-low-precision backend (1/2/4 bit W/A)
  - Lots of exciting new research in mixed precision