Benanza: Automatic µBenchmark Generation to Compute "Lower-bound" Latency and Inform Optimizations of Deep Learning Models on GPUs

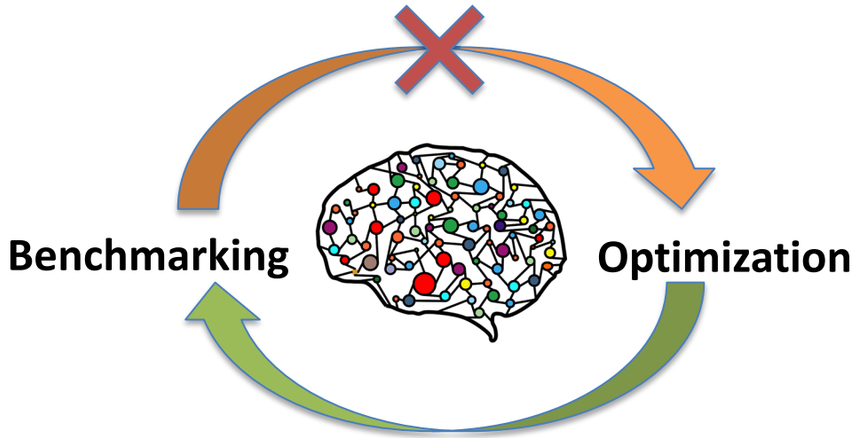
Cheng Li*[1], Abdul Dakkak*[1], Jinjun Xiong[2], Wen-mei Hwu[1]

University of Illinois Urbana-Champaign[1], IBM Research[2]

{cli99, dakkak, w-hwu}@illinois.edu, jinjun@us.ibm.com

Motivation

- The benchmarking → optimization process for Deep Learning (DL) workloads is ad-hoc and slow
- There is a need for a DL optimization advising design that can systematically guide researchers to potential optimization opportunities and assess hypothetical execution scenarios



Answers to the following are highly desired

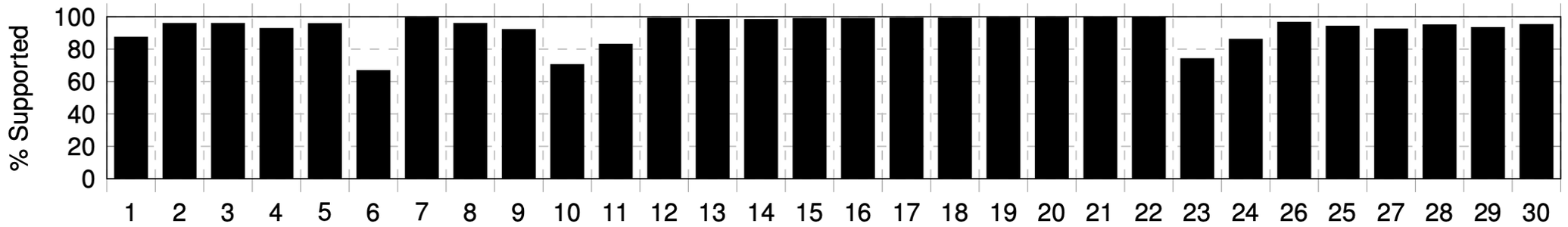
- What is the potential latency speedup if optimizations are performed?
- Are independent layers executed in parallel?
- Are the optimal algorithms used for convolution layers?
- Is there any inefficiency or unexpected behavior in frameworks?
- Does the execution fuse layers or leverage Tensor Cores? And what are the benefits?

Observations

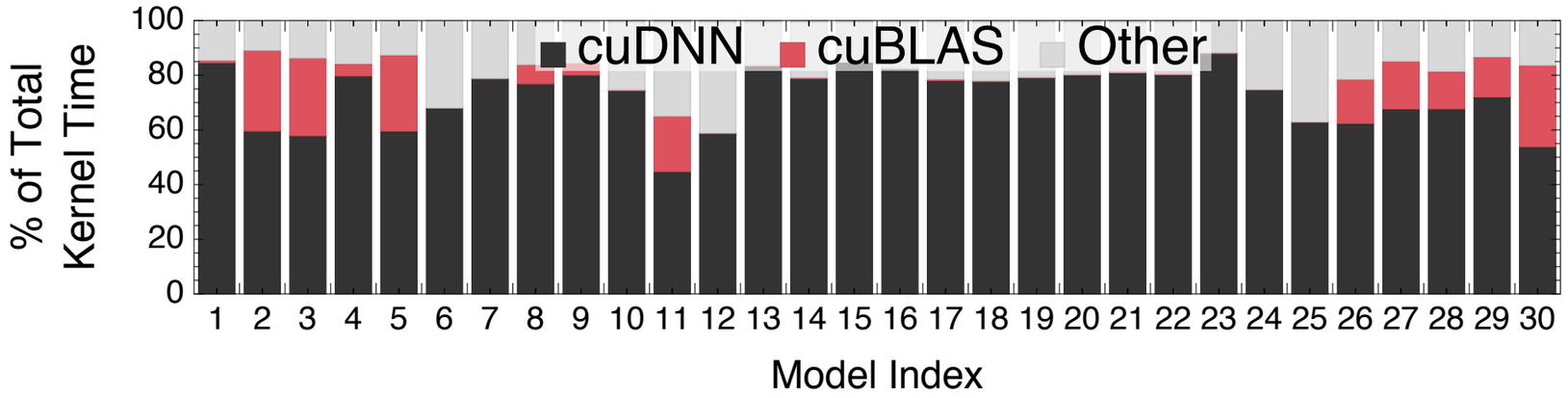
- Layers are the performance building blocks
- Frameworks use cuDNN & cuBLAS to execute layers on GPUs
  - Given a specific model/HW/SW, the cuDNN & cuBLAS functions invoked are fixed

Layer Type	cuDNN / cuBLAS API	Tensor Core Support
Convolution	`cudnnConvolutionForward`	✓
Activation	`cudnnActivationForward`	✗
BatchNorm	`cudnnBatchNormalizationForwardInference`	✗
Conv+Bias+Activation	`cudnnConvolutionBiasActivationForward`	✓
RNN	`cudnnRNNForwardInference`	✓
Dropout	`cudnnDropoutForward`	✗
Pooling	`cudnnPoolingForward`	✗
Softmax	`cudnnSoftmaxForward`	✗
Add	`cudnnAddTensor`	✗
Element-wise	`cudnnOpTensor`	✗
Rescale	`cudnnScaleTensor`	✗
GEMM	`cublas*Gemm / cublasGemmEx`	✓
GEMV	`cublasSgemv`	✗

cuDNN and cuBLAS Dominate the Compute



The percentage of layers that are supported in cuDNN and cuBLAS



GPU kernel time breakdown for all 30 models on Volta GPU

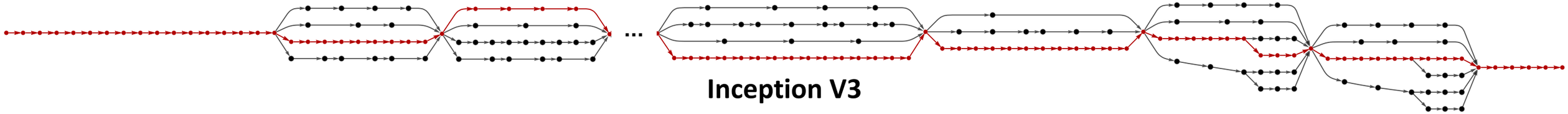
Our Approach

- Knowing the ideal helps understand how to improve the latency
- We introduce a new metric, “lower-bound” latency (LBL)
  - Defined by the latencies of the cuDNN & cuBLAS functions corresponding to the model layers
  - Estimates the ideal latency of a model given a specific GPU HW/SW
- (measured_latency - LBL) indicates optimization opportunities

“Lower-bound” Latency (LBL)

- Computed under different scenarios.
- Data-independent layers might be executed sequentially or in parallel
  - $LBL_{sequential}$ = sum of all layer latencies
  - $LBL_{parallel}$ = sum of layer latencies on the critical path

VGG16



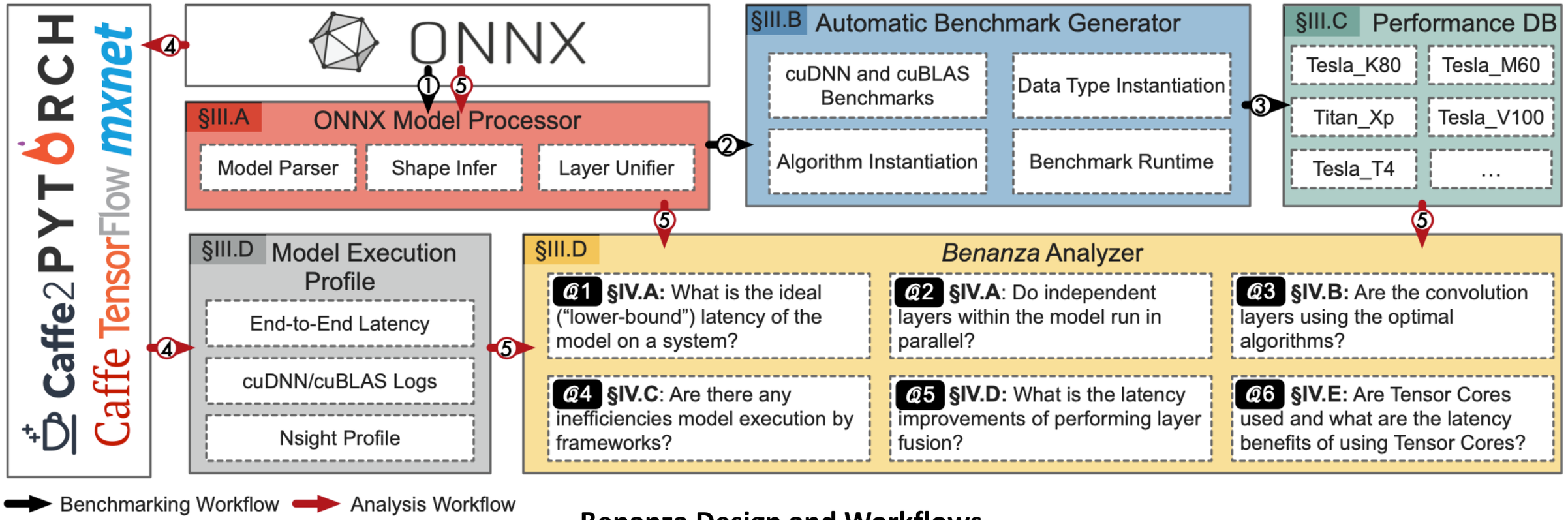
Inception V3

  - $LBL_{sequential} > LBL_{parallel}$ for models with parallel modules, otherwise equal

Benanza

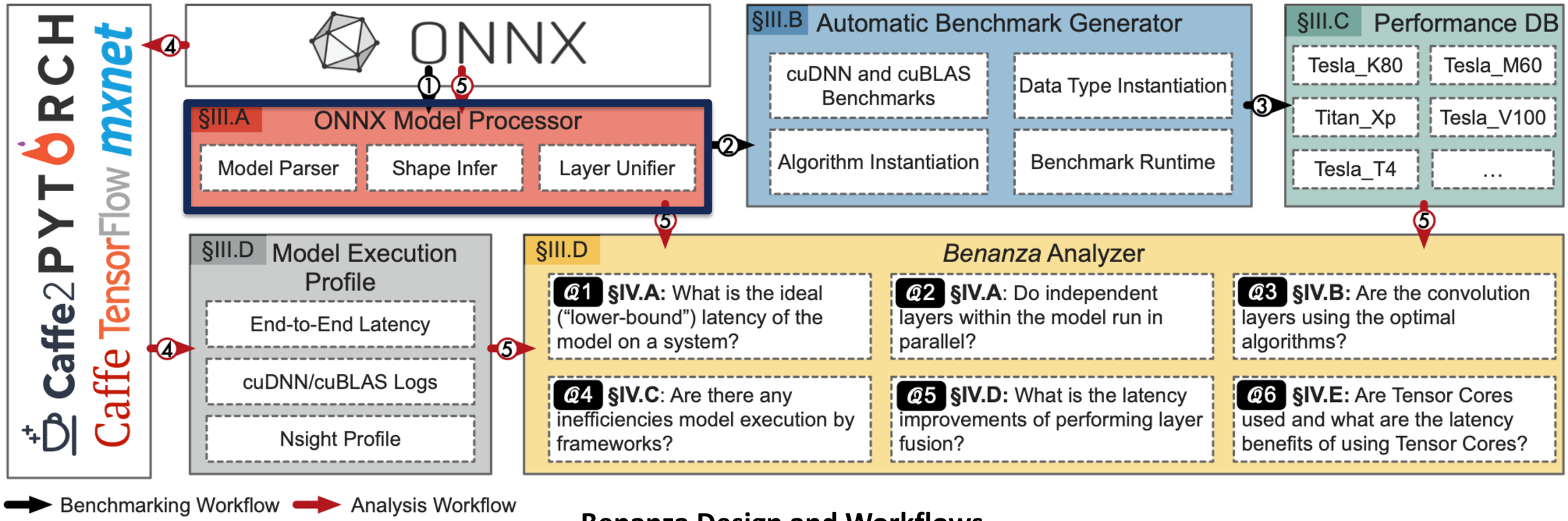
- A benchmarking and analysis design that speeds up the benchmarking/optimization cycle of DL models on GPUs
- Consists of 4 modular components:
  - Model Processor
  - Benchmark Generator
  - Performance Database
  - Analyzer

Design and Workflows



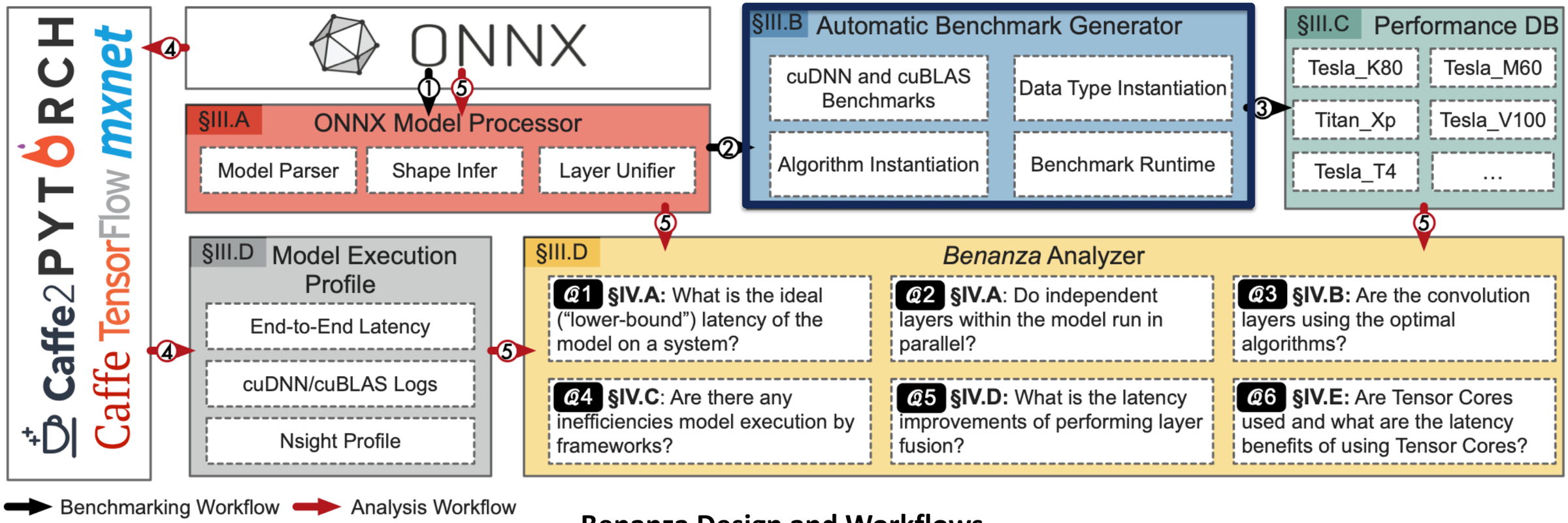
Benanza Design and Workflows

The Benchmarking Workflow



Benanza Design and Workflows

The Benchmarking Workflow



Benanza Design and Workflows

Benchmark Generator

- Generates C++ code using the layer information to measure the cuDNN or cuBLAS API corresponding to the layer
- Algorithm Instantiation
  - 8 different algorithms for the cuDNN convolution API
  - Frameworks rely on a heuristic function to select algorithm
  - Generates benchmarks for all available algorithms

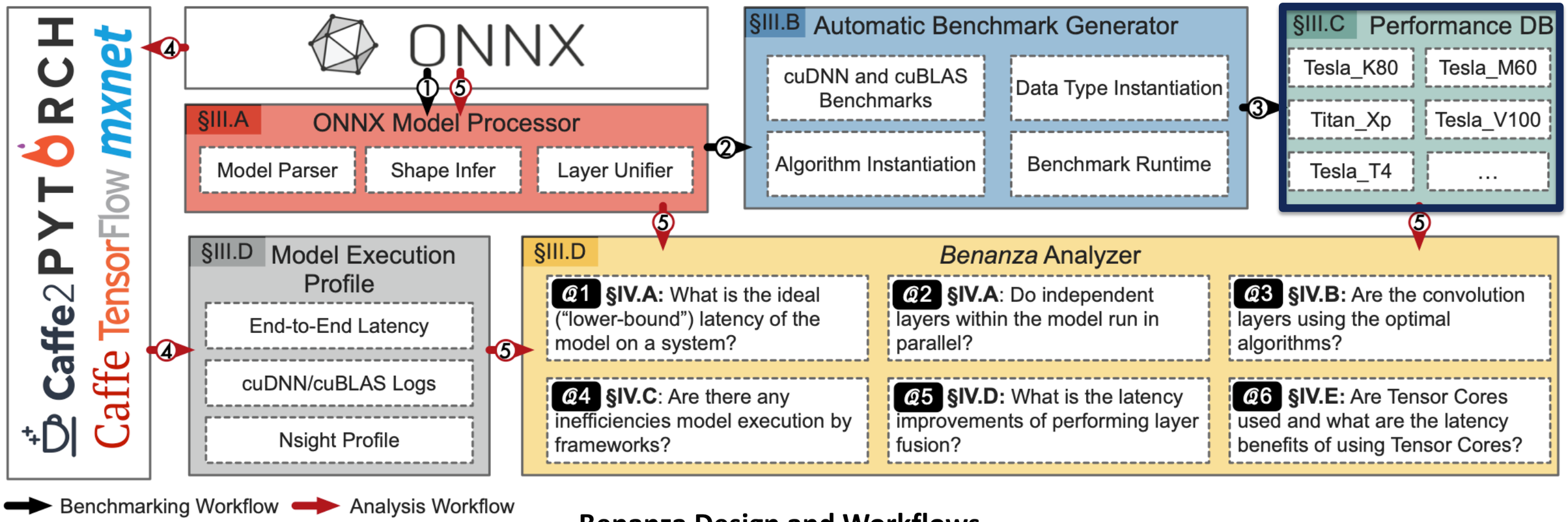
```
algorithm  = cudnnGetConvolutionForwardAlgorithm(layer)
cudnnConvolutionForward(layer, algorithm)
```

Benchmark Generator

- Layer Fusion Support
  - Generates benchmarks that target the cuDNN fused API
  - E.g. *convolution->bias->activation, convolution->bias*
- Data Type Support
  - Generates benchmarks that target different data types
  - E.g. float16 for Tensor Cores

The Benchmarking Workflow

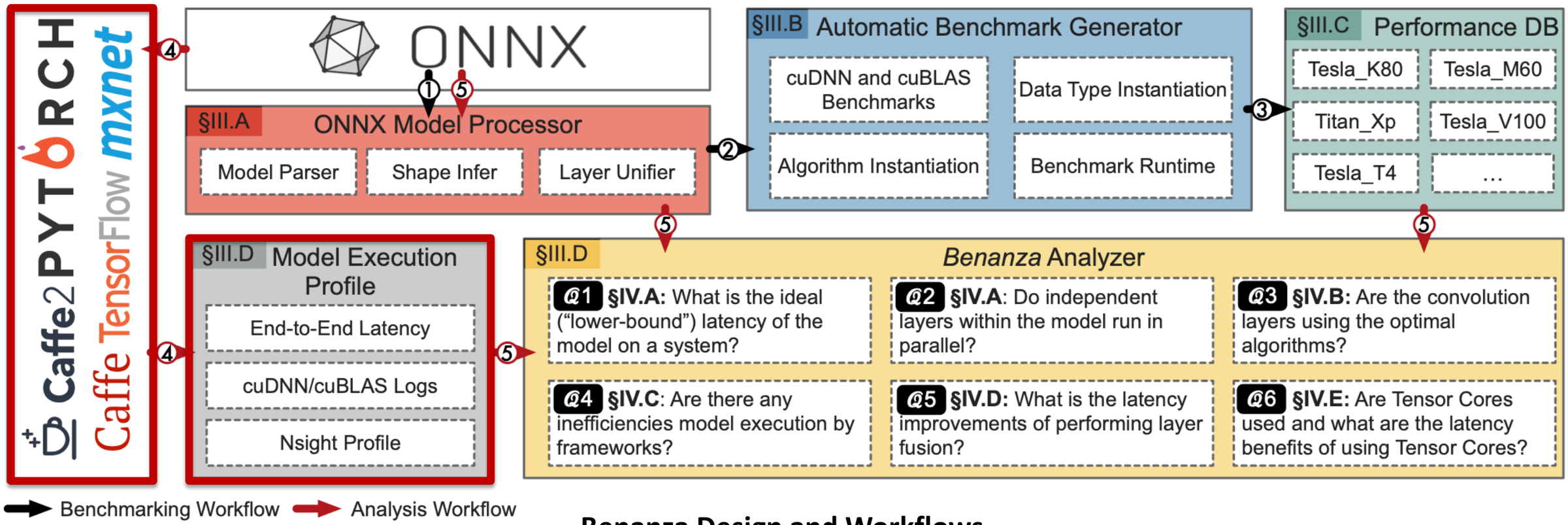
Indexed by the system, data type, and layer information



Benanza Design and Workflows

The Analysis Workflow

The user runs the target model using the SW/HW of interest to get the model execution profile

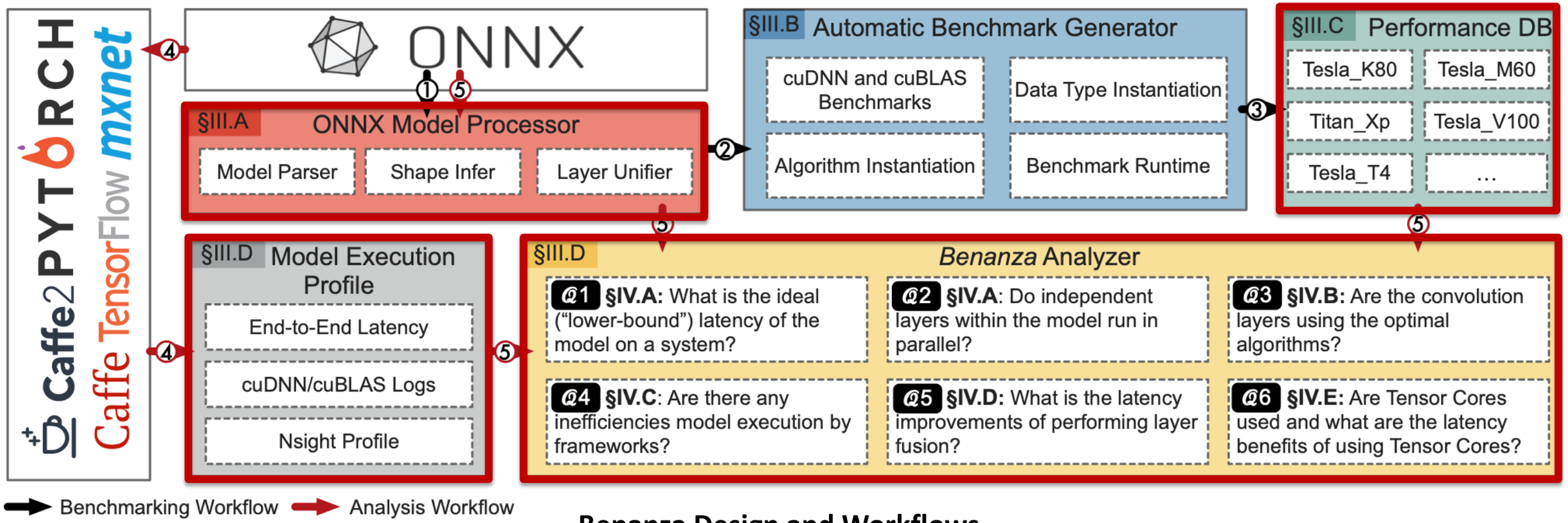


Benanza Design and Workflows

The Analysis Workflow

The user inputs the model execution profile along with the model, system, data type

The Analyzer queries the DB to inform optimizations Q1-6



Benanza Design and Workflows

Evaluation Setup

- MXNet, ONNX Runtime and PyTorch
- 7 GPU systems from Kepler to the latest Turing
- CUDA 10.1 and cuDNN7.6.3
- Unless specified, batch size = 1

Name	CPU	GPU (Release Year)	GPU Architecture	GPU Memory Capacity, Bandwidth	Theoretical FP32 TFLOPS	Theoretical Tensor TFLOPS
Tesla_K80 (AWS P2)	Intel Xeon CPU E5-2686 v4	Tesla K80 (2014)	Kepler	12 GB, 480 GB/s	5.6	✗
Tesla_M60 (AWS G3)	Intel Core i9-7900X CPU	Tesla M60 (2015)	Maxwell	7 GB, 160.4 GB/s	4.8	✗
TITAN_ Xp	Intel Xeon CPU E5-2686 v4	TITAN Xp (2017)	Pascal	12 GB, 547.6 GB/s	12.2	✗
TITAN_V	Intel Core i7-7820X CPU	TITAN V (2017)	Volta	12 GB, 672 GB/s	14.9	110.0
Tesla_V100 (AWS P3)	Intel Xeon CPU E5-2686 v4	Tesla V100 SXM2 (2018)	Volta	16 GB, 900 GB/s	15.7	125.0
Quadro_RTX	Intel Xeon CPU E5-2630 v4	Quadro RTX 6000 (2019)	Turing	24 GB, 624 GB/s	16.3	130.5
Tesla_T4 (AWS G4)	Intel Xeon Platinum 8259CL CPU	Tesla T4 (2019)	Turing	15 GB, 320 GB/s	8.1	65.0

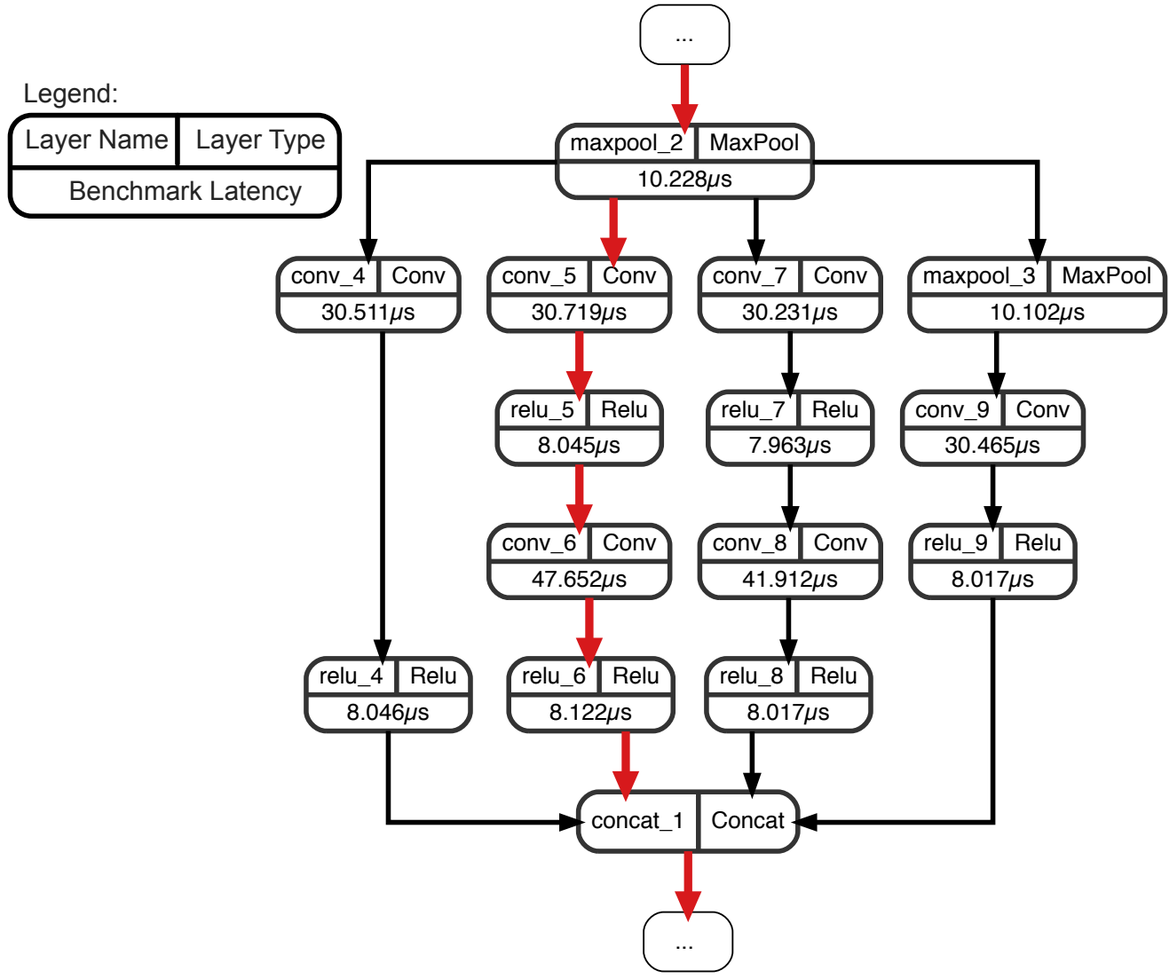
7 GPU systems

ID	Name	Task	MACs	# Layers	Year
1	Arcface [16]	FR	12.08G	412	2018
2	BVLC-Alexnet [17]	IC	656M	24	2012
3	BVLC-Caffenet [17]	IC	721M	24	2012
4	BVLC-Googlenet [18]	IC	1.59G	143	2014
5	BVLC-RCNN-ILSVRC13 [19]	IC	718M	23	2013
6	Densenet-121 [20]	IC	2.87G	910	2016
7	DUC [21]	SS	34.94G	355	2017
8	Emotion Ferplus [22]	ER	877M	52	2016
9	Inception-v1 [23]	IC	1.44G	144	2015
10	Inception-v2 [24]	IC	2.03G	509	2015
11	LeNet [25]	HR	796K	12	2010
12	MobileNet-v2 [26]	IC	437M	155	2017
13	Resnet18-v1 [27]	IC	1.82G	69	2015
14	Resnet18-v2 [28]	IC	1.82G	69	2016
15	Resnet34-v1 [27]	IC	3.67G	125	2015
16	Resnet34-v2 [28]	IC	3.67G	125	2016
17	Resnet50-v1 [27]	IC	3.87G	175	2015
18	Resnet50-v2 [28]	IC	4.10G	174	2016
19	Resnet101-v1 [27]	IC	7.58G	345	2015
20	Resnet101-v2 [28]	IC	7.81G	344	2016
21	Resnet152-v1 [27]	IC	11.30G	515	2015
22	Resnet152-v2 [28]	IC	11.53G	514	2016
23	Shufflenet [29]	IC	127M	203	2015
24	Squeezenet-v1.1 [30]	IC	352M	66	2016
25	Tiny Yolo-v2 [31]	OD	3.13G	32	2016
26	Vgg16-BN [32]	IC	15.38G	54	2014
27	Vgg16 [32]	IC	15.38G	41	2014
28	Vgg19-bn [32]	IC	19.55G	63	2014
29	Vgg19 [32]	IC	19.55G	47	2014
30	Zfnet512 [33]	IC	1.48G	22	2013

30 ONNX models

Q1&2. LBL and Parallel Execution Analysis

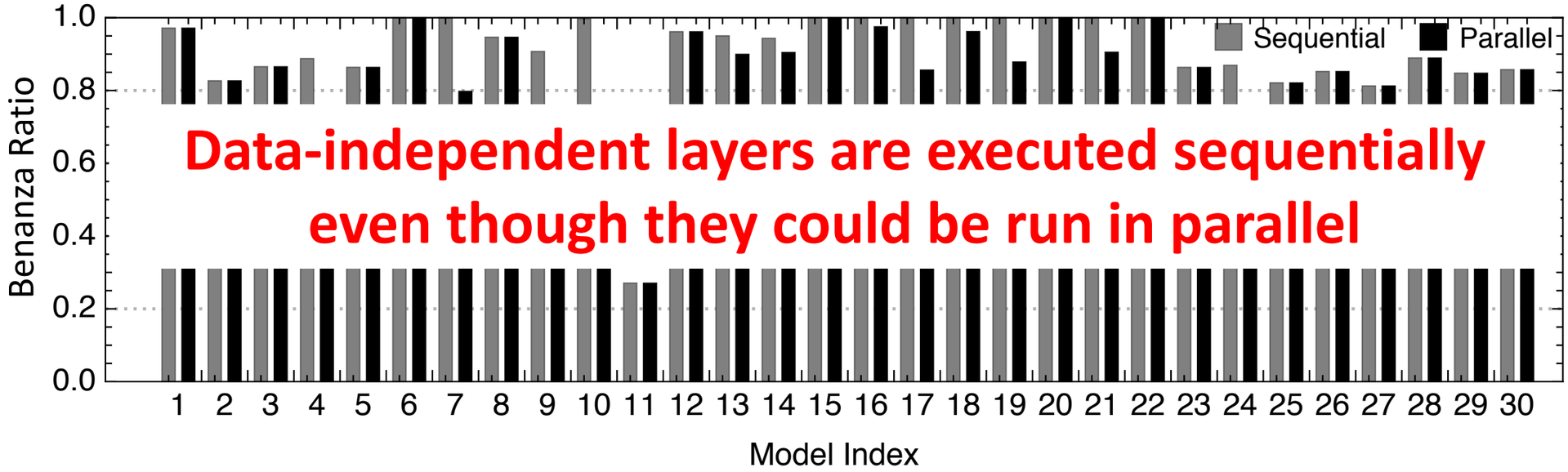
- $LBL_{sequential}$ = sum of all layer benchmark latencies
- $LBL_{parallel}$ = sum of layer benchmark latencies on the critical path



The first Parallel module of Inception-v1

Q1&2. LBL and Parallel Execution Analysis

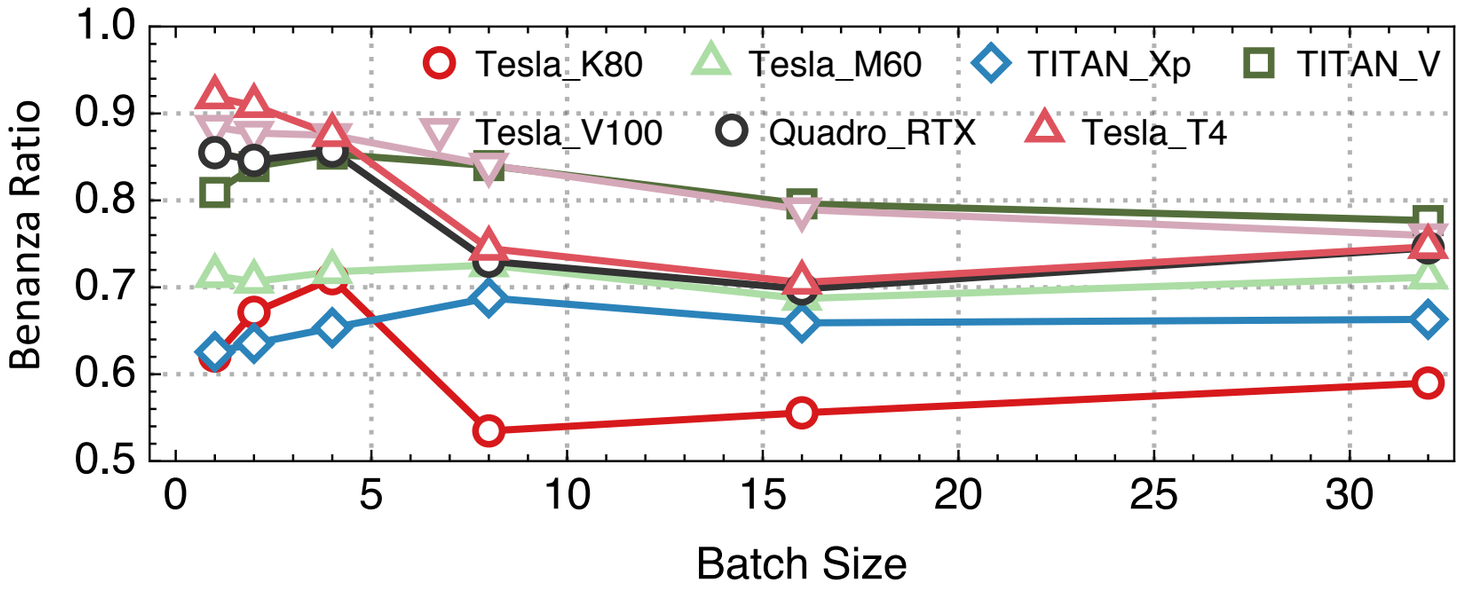
- Benanza Ratio (BR) = LBL / measured latency
  - $BR_{sequential} = BR_{parallel}$ for models without parallel modules
  - $BR_{parallel} < BR_{sequential} < 1$ for models with parallel modules



The sequential and parallel BR of 30 models using MXNet on Tesla V100

Q1&2. BR Across GPUs and Batch Sizes

- Overall, the software stack is more optimized on the recent GPUs (Turing and Volta) and for smaller batch sizes



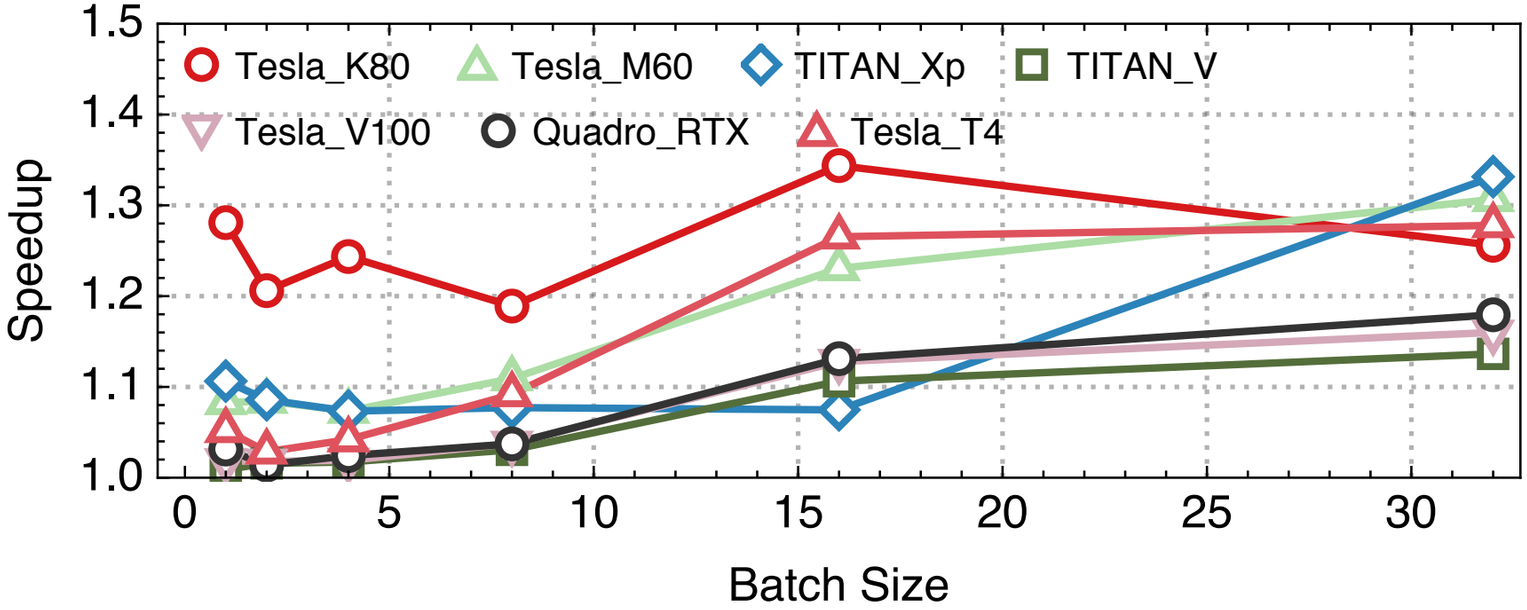
The geometric mean of the BR of all models

Q3. Convolution Algorithm Selection Analysis

- Recall that in benchmark generation the convolution API is invoked with all available algorithms
- The Analyzer parses the cuDNN log to determine if the cuDNN algorithm used by the framework is optimal
- Cases where the choice is suboptimal, and the potential latency improvement are reported

Q3. Convolution Algorithm Selection Analysis

- Both recent and older GPU architectures can benefit from better cuDNN heuristics



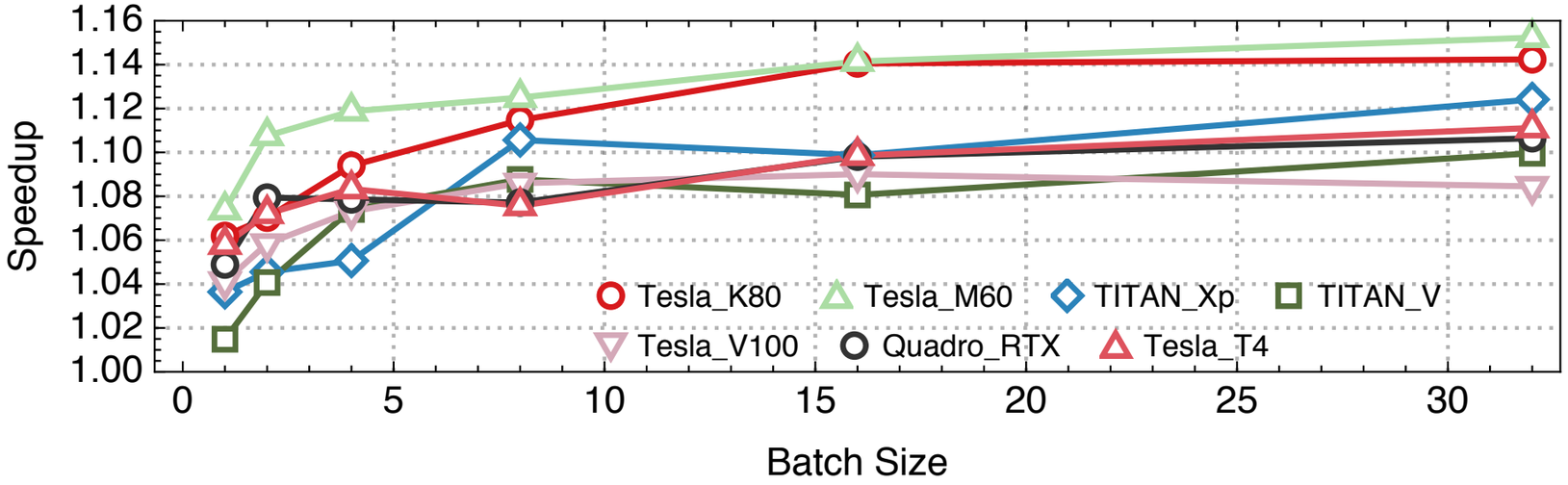
The geometric mean of the latency speedup for all models by using the optimal convolution algorithm

Q4. Framework Inefficiency Inspection

- The expected cuDNN and cuBLAS API calls are known
- The Analyzer compares the model execution profile against the expected execution to pinpoint inefficiencies within the framework
- The Analyzer presents any deviation observed in cuDNN or cuBLAS API invocation’s parameters or their execution order

Q4. An Inefficiency in MXNet

- Using Benanza we observed that MXNet ONNX model loader adds a padding layer before every convolution layer
- Unnecessary if the convolution does not use asymmetric padding



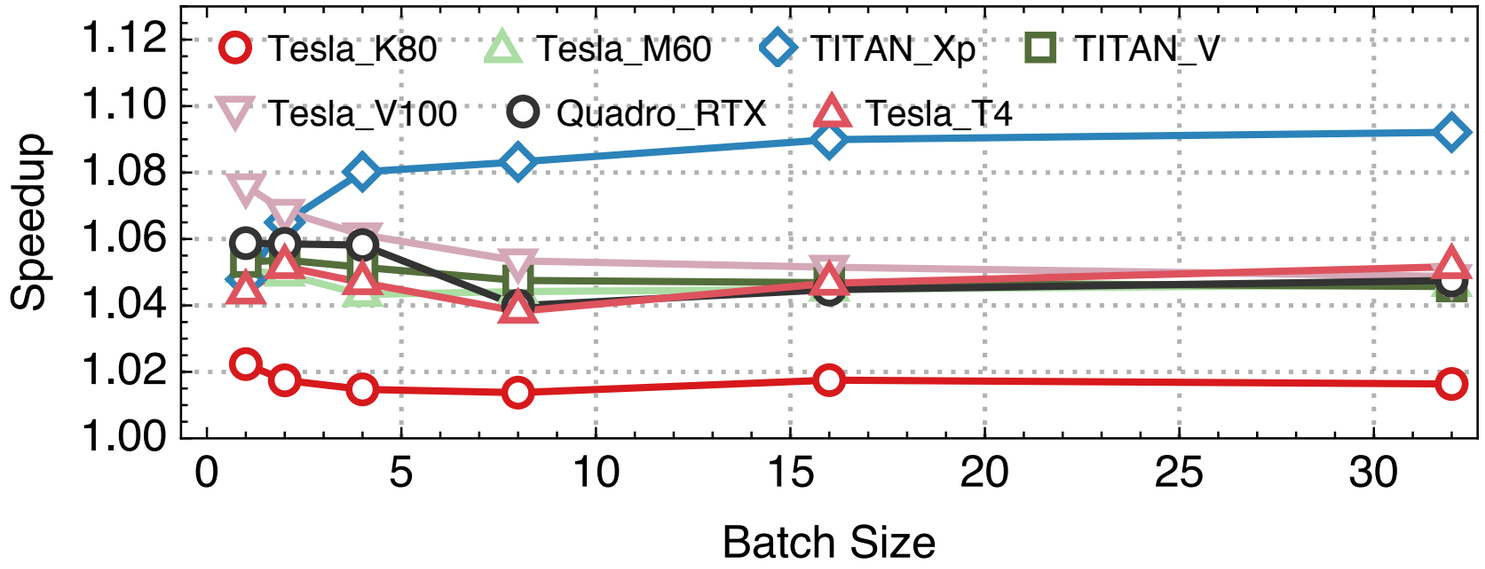
The speedup achieved for ResNet50-v1 by applying the MXNet optimization

Q5. Layer Fusion Analysis

- Recall that Benanza generates benchmarks that target the cuDNN fused API
- The Analyzer traverses the model layers and looks for the fusion patterns
  - $Profit_{layer\_fusion} = LBL_{non\text{-}fused} - LBL_{fused}$

Q5. Layer Fusion Analysis

- ResNet50-v1 has the layer sequence pattern Conv -> Bias -> BatchNorm ->Activation
- Benanza reports the Conv ->Bias can be fused for better latency



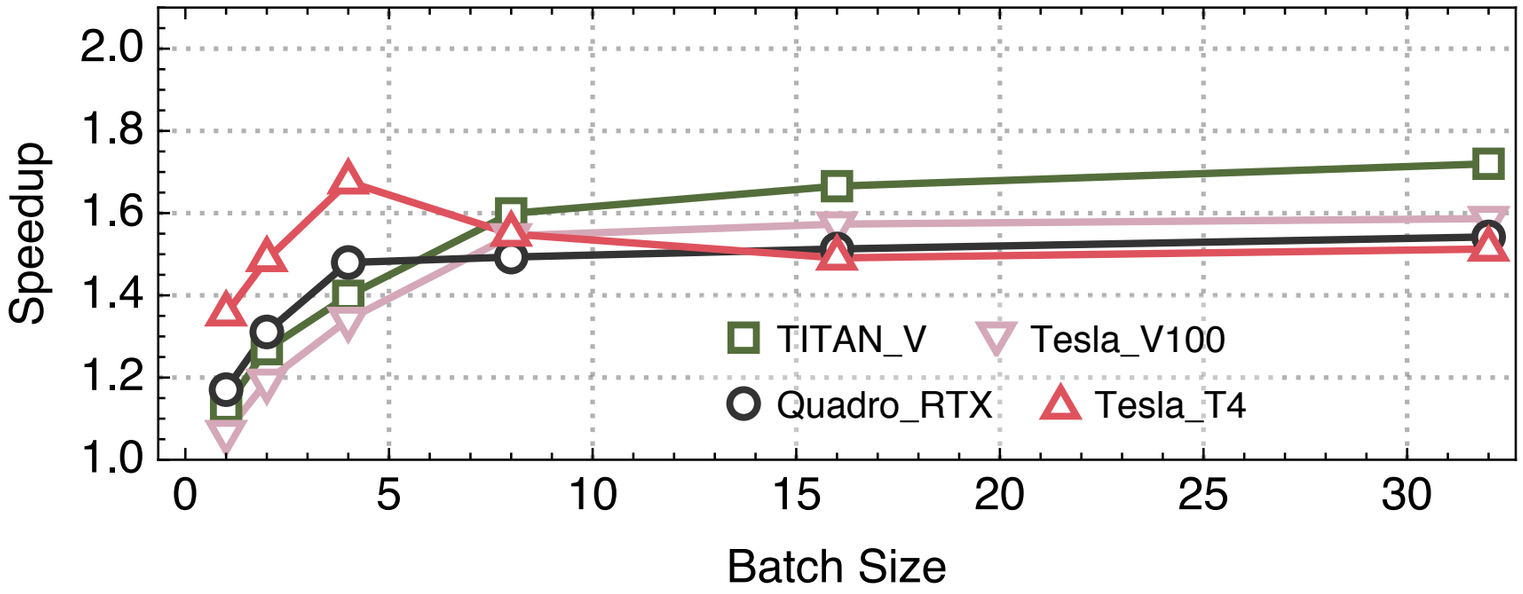
The latency speedup for ResNet50-v1 if layer fusion was performed

Q6. Tensor Core Analysis

- Recall that Benanza generates benchmarks that target Tensor Cores
- The Analyzer determines if the target model execution utilizes Tensor Cores by looking at kernel names
  - $\text{Profit}_{\text{TensorCore}} = \text{LBL}_{\text{non-TensorCore}} - \text{LBL}_{\text{TensorCore}}$

Q6. Tensor Core Analysis

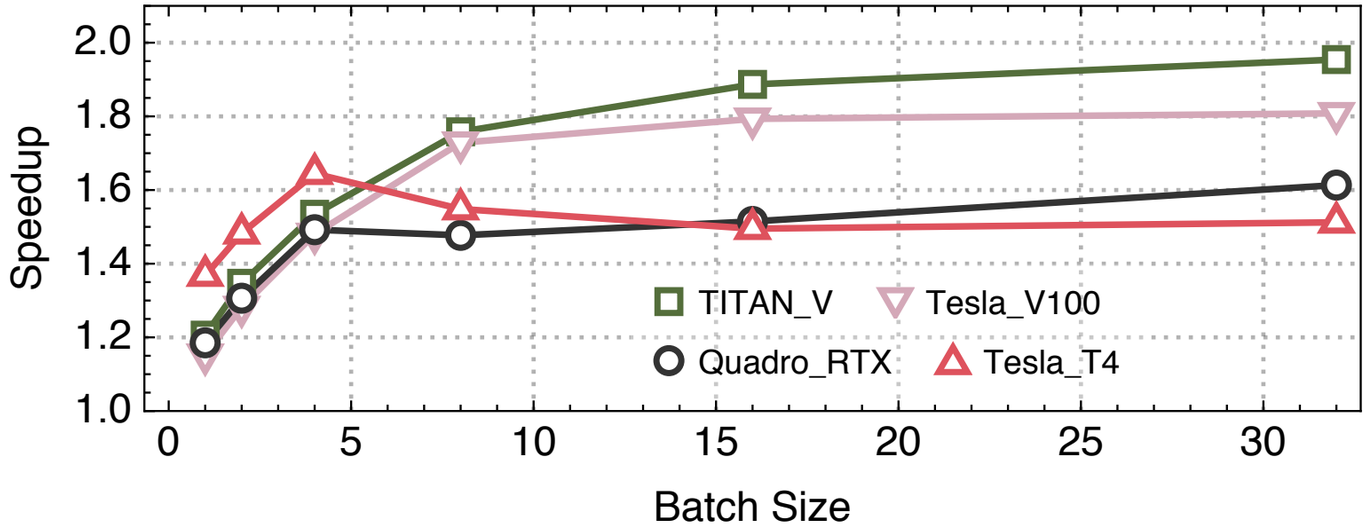
- TITAN V achieves significant speedup, up to 1.72x
- For smaller batch sizes, Tesla T4 benefits most from Tensor Cores



The latency speedup for ResNet50-v1 if Tensor Cores were used

Q1,2,3,5,6. Joint Optimizations

- Benanza can perform the analyses jointly
- Up to 1.95x and 1.8x speedup can be achieved by TITAN V and Tesla V100 respectively



The latency speedup for ResNet50-v1 if parallel execution, optimal algorithm selections, layer fusion, and Tensor Cores were used

Sustainability and Extensibility

- The workflow is automated, and the user only needs to compile and run the generated code
- The Performance Database is continuously updated
  - For new models, only the newly introduced layers are benchmarked
  - Layer repeatability keeps the number of entries in the database in check
- Components are modular and can be extended with
  - New model parsers
  - New cuDNN/cuBLAS API or algorithm
  - Other runtimes that target other SW libraries or HW

Conclusion

- Benanza automatically generates layer-wise benchmarks for DL models to compute the “lower-bound” latency and inform optimizations on GPUs
- The design is sustainable and extensible, not limited to GPUs and affords other usages, e.g.
  - Helping DL compiler optimizations
  - Improving work scheduling for DLaaS
  - Model/framework/system advising for DL tasks

Thank you

More information in the paper

Cheng Li*[1], Abdul Dakkak*[1], Jinjun Xiong[2], Wen-mei Hwu[1]

University of Illinois Urbana-Champaign[1], IBM Research[2]