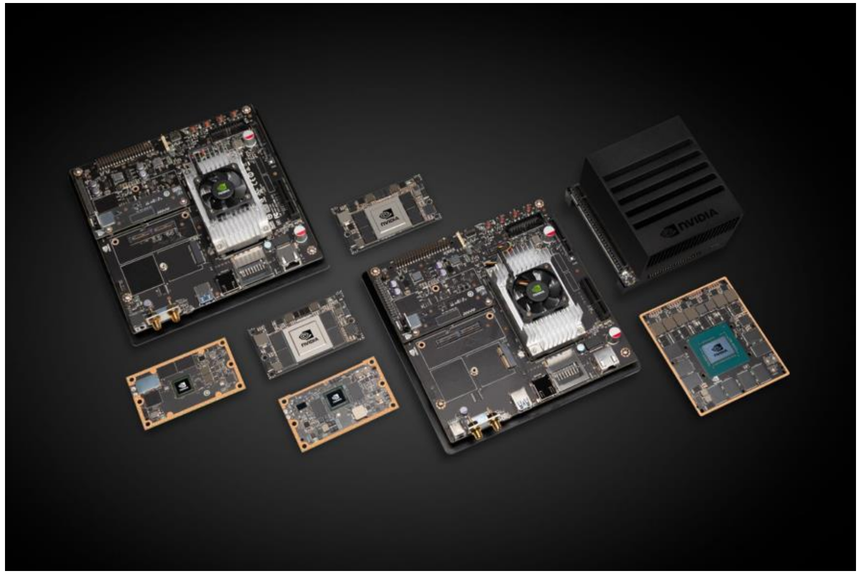
BlastNet: Exploiting Duo-Blocks for Cross-Processor Real-Time DNN Inference

Ling, Xuan Huang, Zhihe Zhao, Nan Guan, Zhenyu Yan, and Guoliang Xing

SenSys'22

Introduction

- **CPU-GPU heterogeneous architectures for edge platforms**

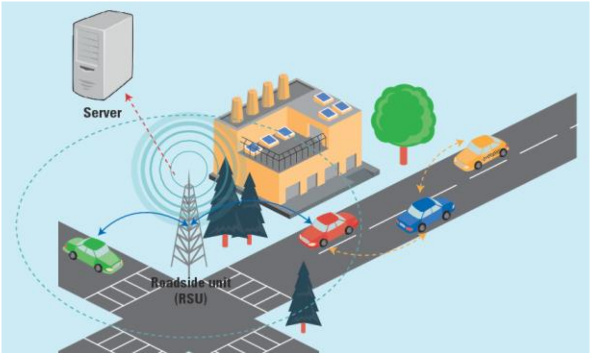


- **NVIDIA Xavier**
  - **An 8-core CPU**
  - **A Volta GPU**
- **Google Pixel 6**
  - **An 8-core CPU**
  - **A MaliG78 MP20 GPU**

Introduction

- **Cross-Processor Real-Time DNN Inference**

autonomous driving



smart roadside infrastructure

embedded computer vision

Motivation

- **Platforms**

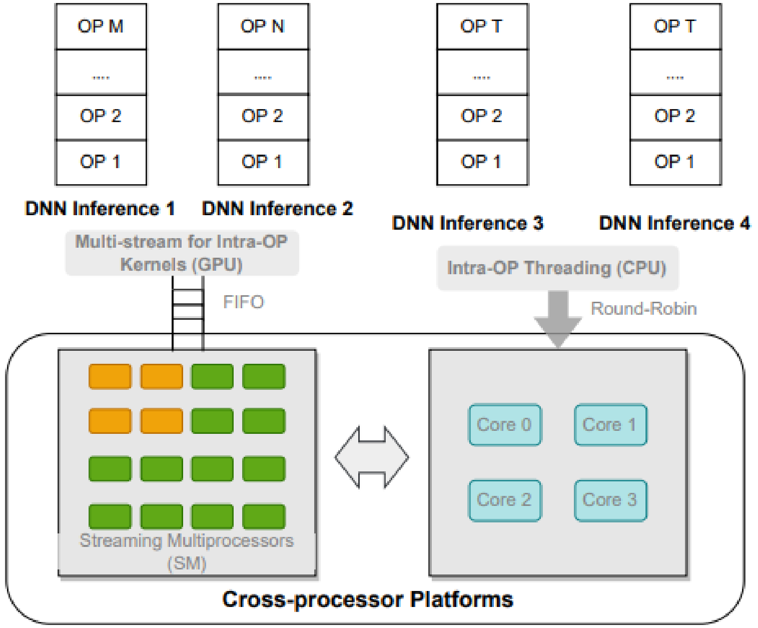


Figure 1: Concurrent DNN inference under PyTorch framework on heterogeneous CPU-GPU platforms.

Current mainstream deep learning frameworks monolithically allocate heterogeneous resources for concurrently executed DNN models..

Motivation

- **Model-level DNN Inference**

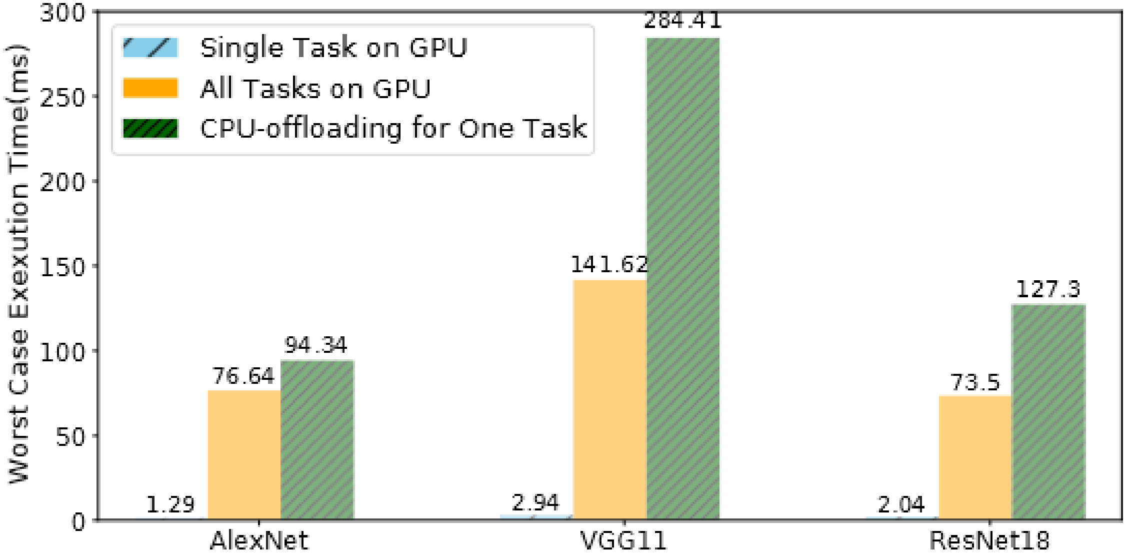
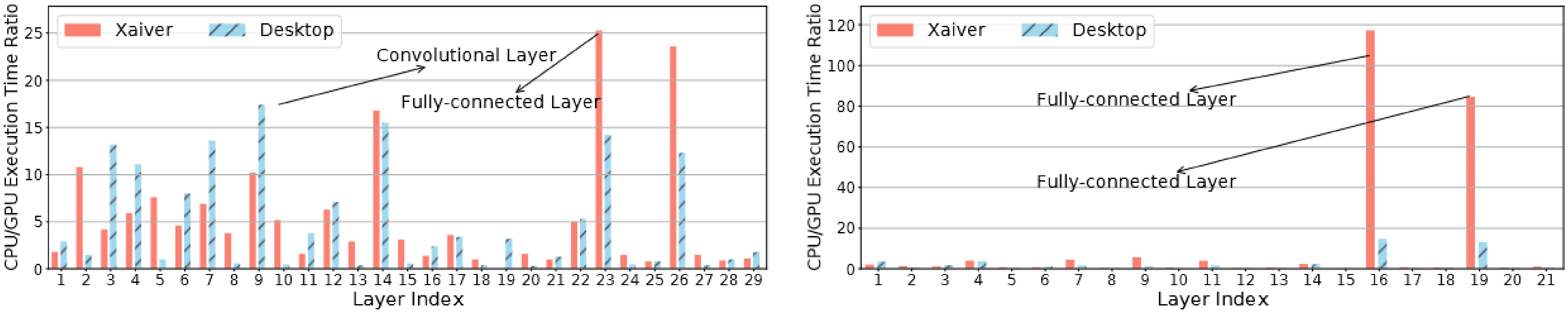


Figure 2: Worst-case execution time for concurrent DNN model inference on CPU-GPU platform under different resource allocation strategies.

Allocate DNN models to CPU and GPU in a model-level granularity leads to severe resource contention.

Motivation

- **Layer-level DNN Inference**



(a) VGG11 execution difference on CPU/GPU

(b) AlexNet execution difference on CPU/GPU

Figure 3: CPU/GPU execution ratio for each DNN layer.

Allocate DNN models to CPU and GPU in a layer-level granularity may cause low resource utilization and significant layer switching overhead.

Motivation

- **Layer-level DNN Inference**

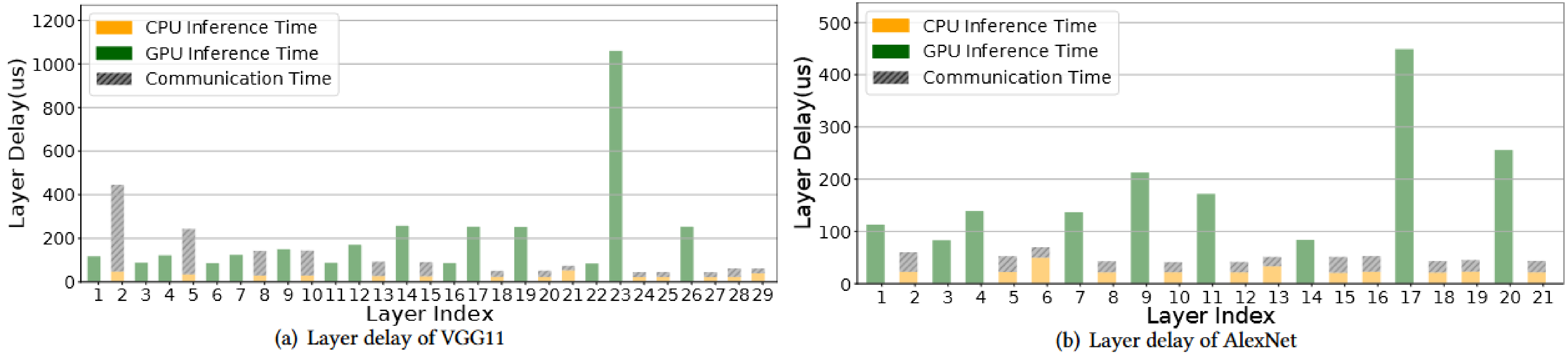
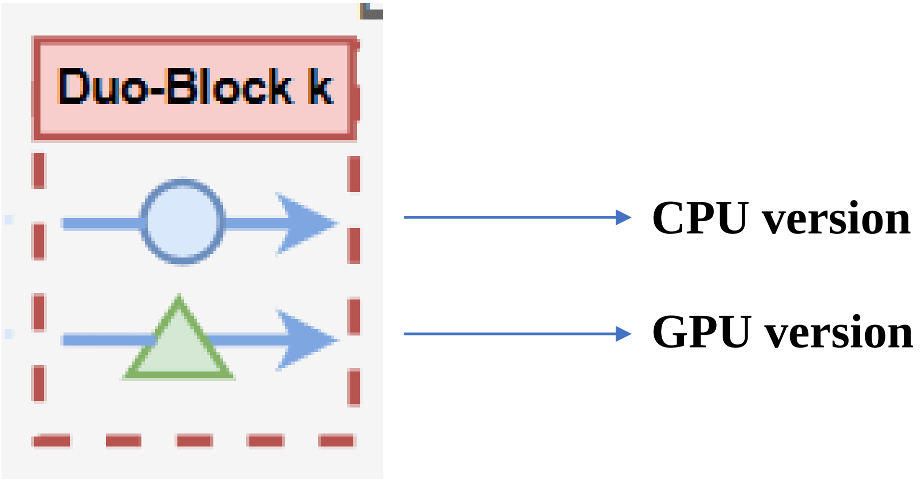


Figure 4: Layer delay by executing each layer on the processor with the shortest inference time (CPU Utilization: 9.05% for VGG11, 13.37% for AlexNet, GPU Utilization: 65.59% for VGG11, 73.39% for AlexNet, Communication Overhead: 25.35% for VGG11, 13.37% for AlexNet).

Allocate DNN models to CPU and GPU in a layer-level granularity may cause low resource utilization and significant layer switching overhead.

Design – A new abstraction of model partition

- The model-level allocation strategy often causes severe resource contention on the GPU while leaving the CPU idle.
- The layer-level allocation may lead to frequent layer switching and significant communication overhead.



Design – System Overview

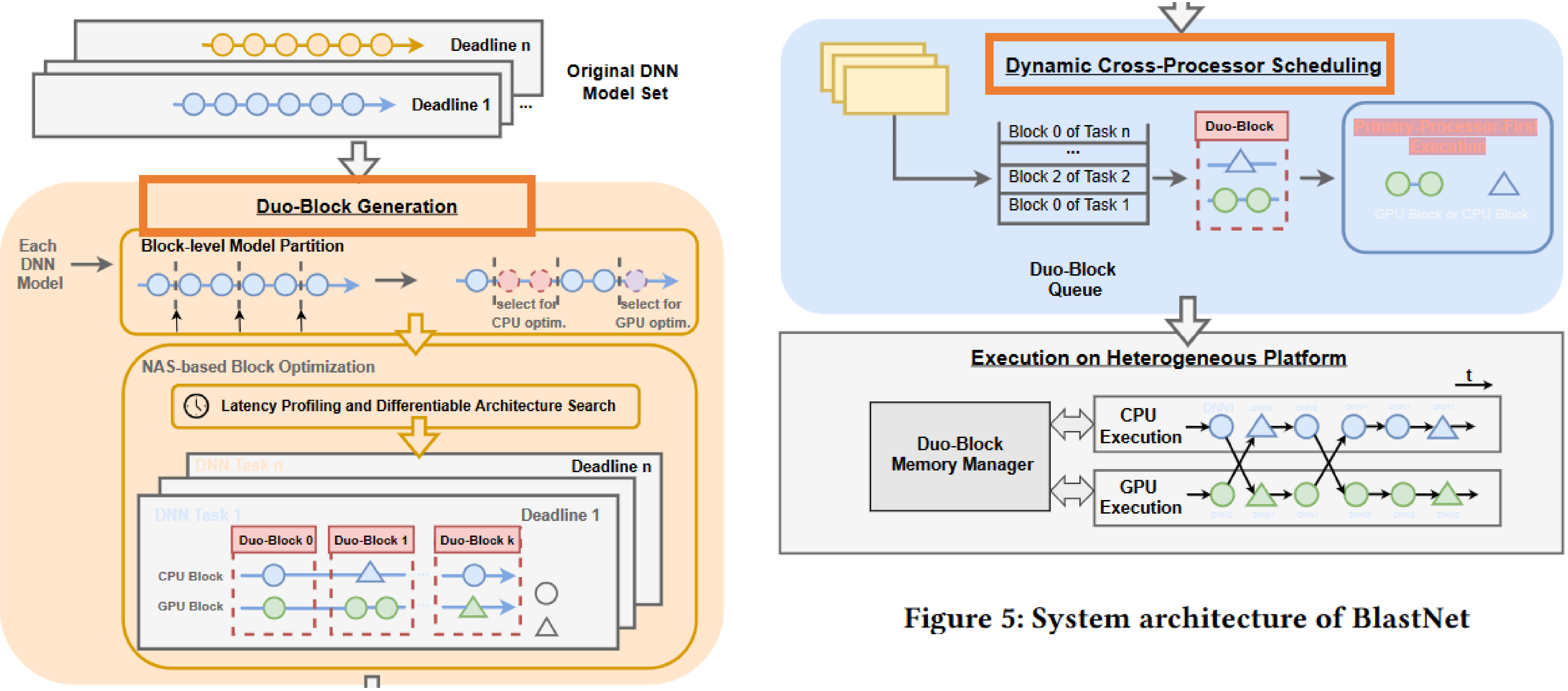
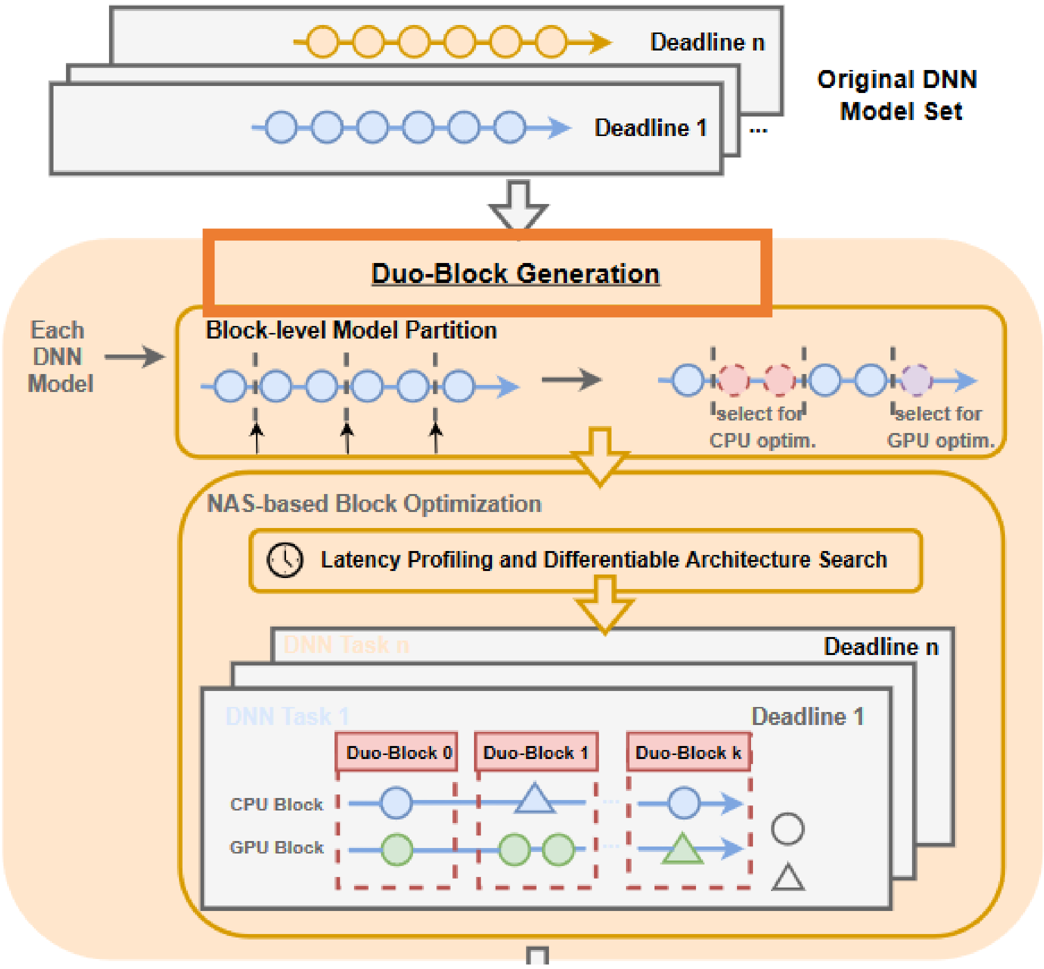


Figure 5: System architecture of BlastNet

Design – System Overview



- Block-level Model Partition
  - The layer-level computing
  - Communication characteristics
  - operator fusion rules

Design – duo-block generation

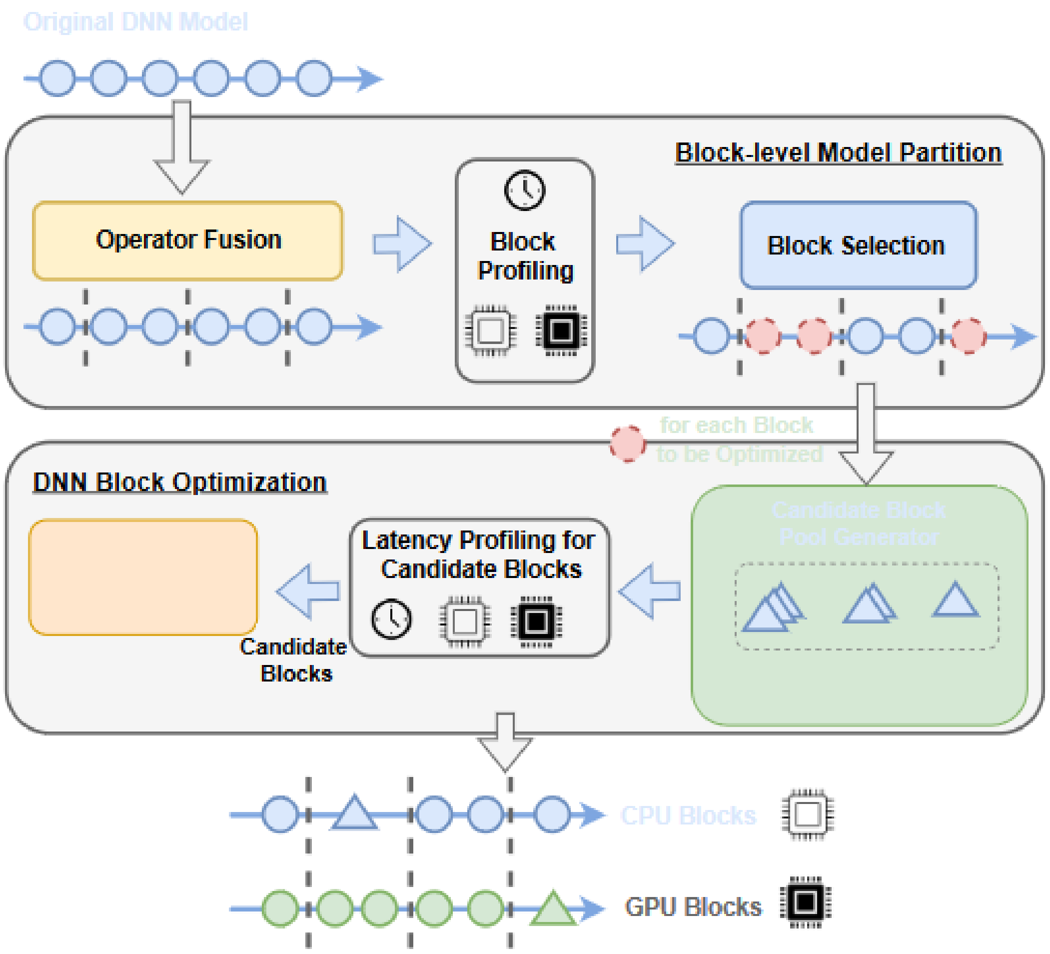
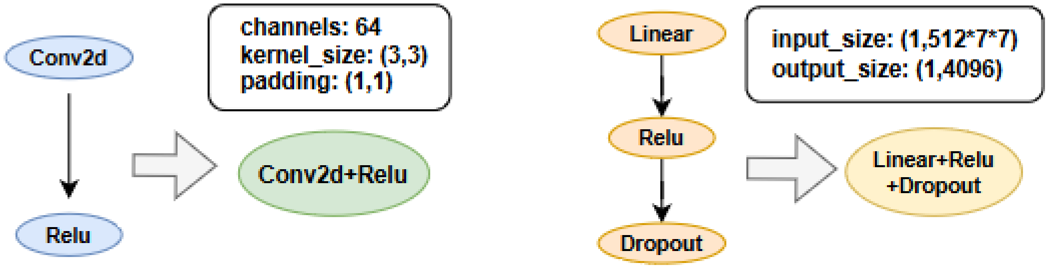


Figure 6: Generation procedure for cross-processor block

- Block-level Model Partition
  - Fuse DNN layers into blocks based on the general operator fusion rules and layer characteristics.
  - Determine the primary and the secondary processors for each block.
  - Optimize the block execution on its secondary processor.

Design – duo-block generation



Processor	T_{conv2d}	T_{relu}	T_{sum}	T_{fused_sum}	T_{linear}	$T_{relu+dropout}$	T_{sum}	T_{fused_sum}
GPU (ms)	2.701	1.147	3.848	3.697	12.395	0.078	12.475	12.415
CPU (ms)	0.178	0.092	0.27	0.242	1.016	0.067	1.083	1.022

Figure 7: Operator fusion and its benefit (evaluated under torchscript on the desktop platform with NVIDIA RTX 2060 GPU). $T_{operator\_name}$ denotes the execution time of the operator. T_{sum} denotes the sum of T_{conv2d}, T_{relu} or T_{linear}, $T_{dropout}$. T_{fused_sum} denotes the execution time of fused operators.

- **Block-level Model Partition**
  - **Fuse DNN layers into blocks based on the general operator fusion rules and layer characteristics.**
  - Determine the primary and the secondary processors for each block.
  - Optimize the block execution on its secondary processor.

Design – duo-block generation

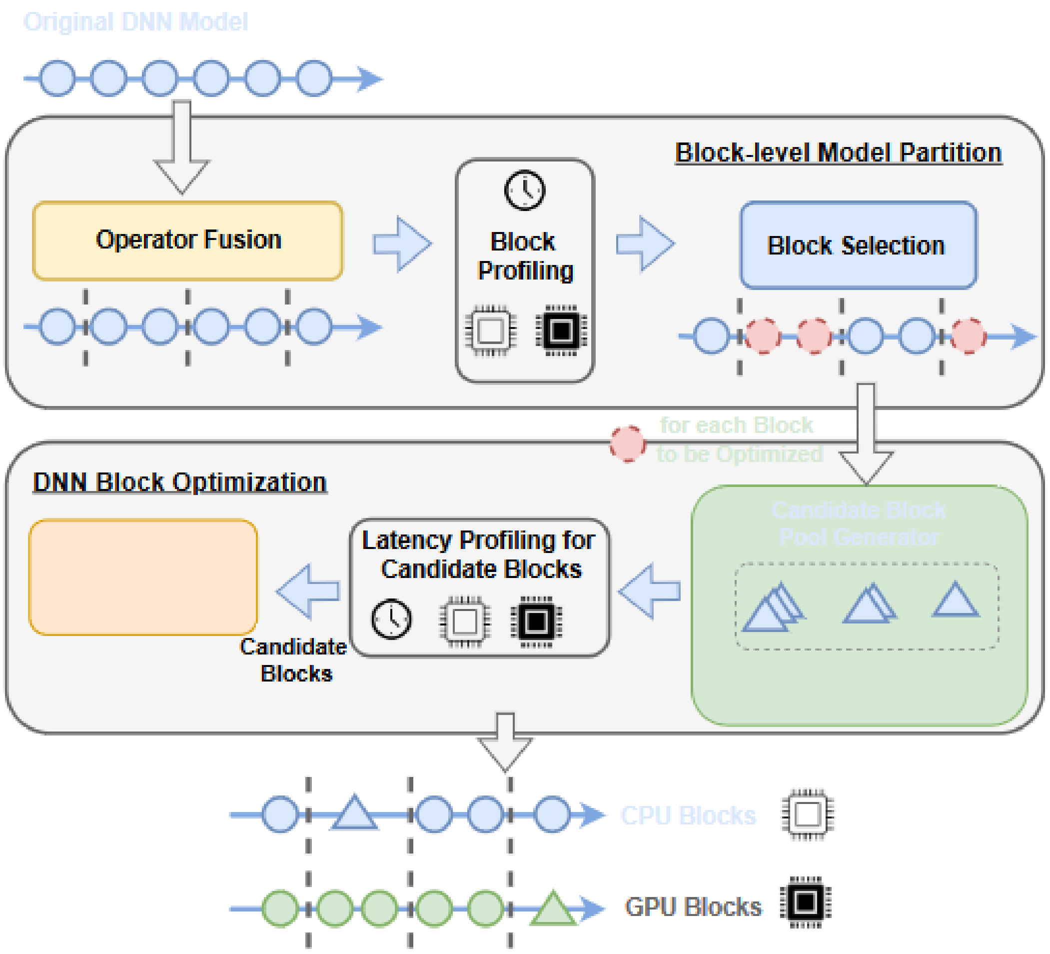
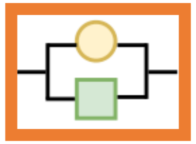


Figure 6: Generation procedure for cross-processor block

- **Block-level Model Partition**
  - Fuse DNN layers into blocks based on the general operator fusion rules and layer characteristics.



  - Determine the primary and the secondary processors for each block.
  - Optimize the block execution on its secondary processor.

Design – duo-block generation

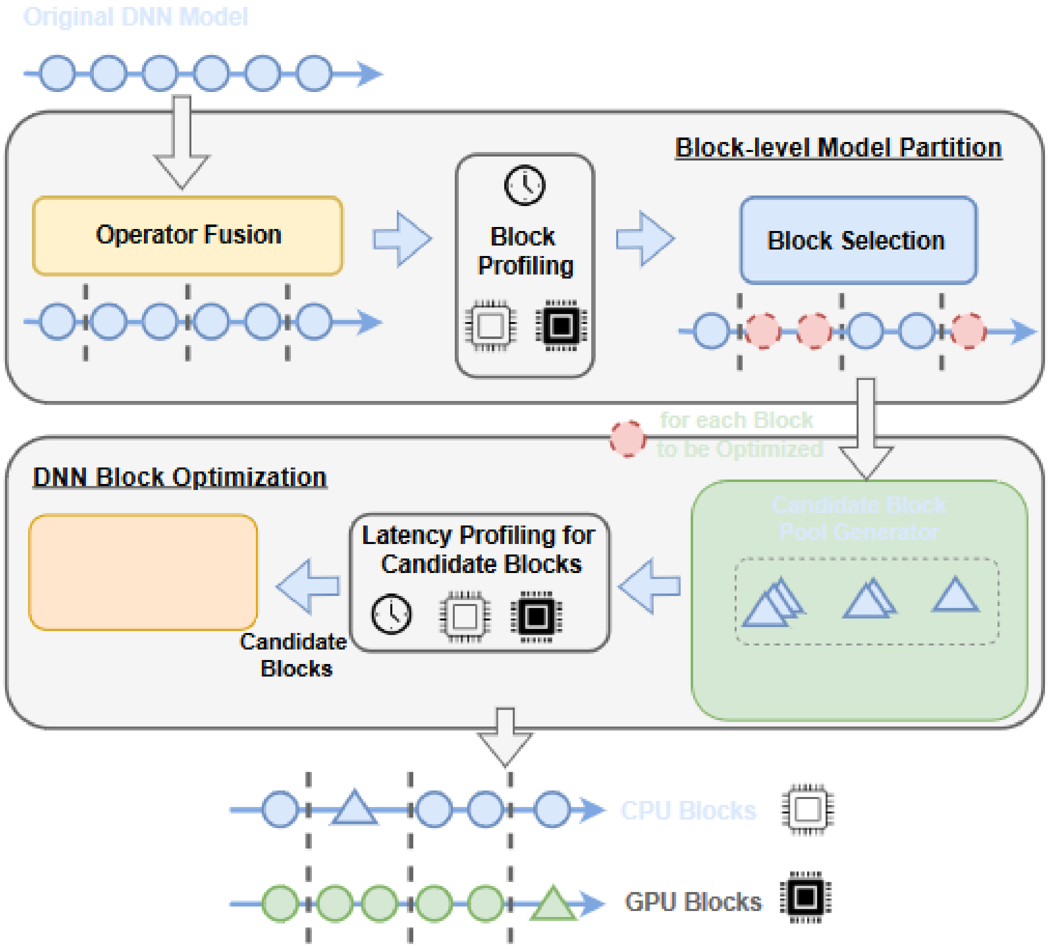


Figure 6: Generation procedure for cross-processor block

- **Block-level Model Partition**
  - Fuse DNN layers into blocks based on the general operator fusion rules and layer characteristics.
  - Determine the primary and the secondary processors for each block.
  - **Optimize the block execution on its secondary processor.**

$$CD = \frac{T^{sec}(B_k)}{T^{pri}(B_k)}, \quad WP = \frac{T^{pri}(B_k)}{\sum_{k=0}^{k_{max}} T^{pri}(B_k)}$$

$$CD > \varepsilon \qquad WP > 1/block\_num$$

Design – duo-block generation

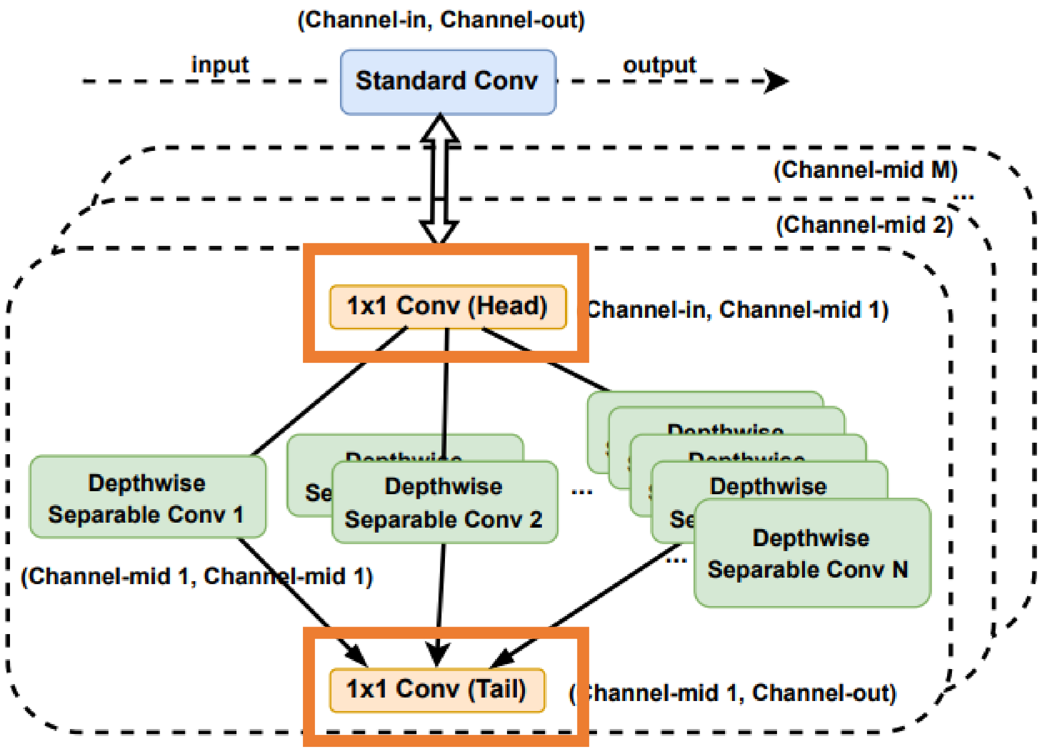


Figure 8: Example for candidate blocks of a convolutional layer on CPU

- NAS-based Block Optimization
  - Only consider convolutional and fully-connected layers since they account for the most execution time.
  - Convolutional layer optimized in CPU
    - choose the most CPU-friendly operators (i.e., depthwise separable convolutional layer)
  - Convolutional layer optimized in GPU
  - Fully-connected layer optimized

Design – duo-block generation

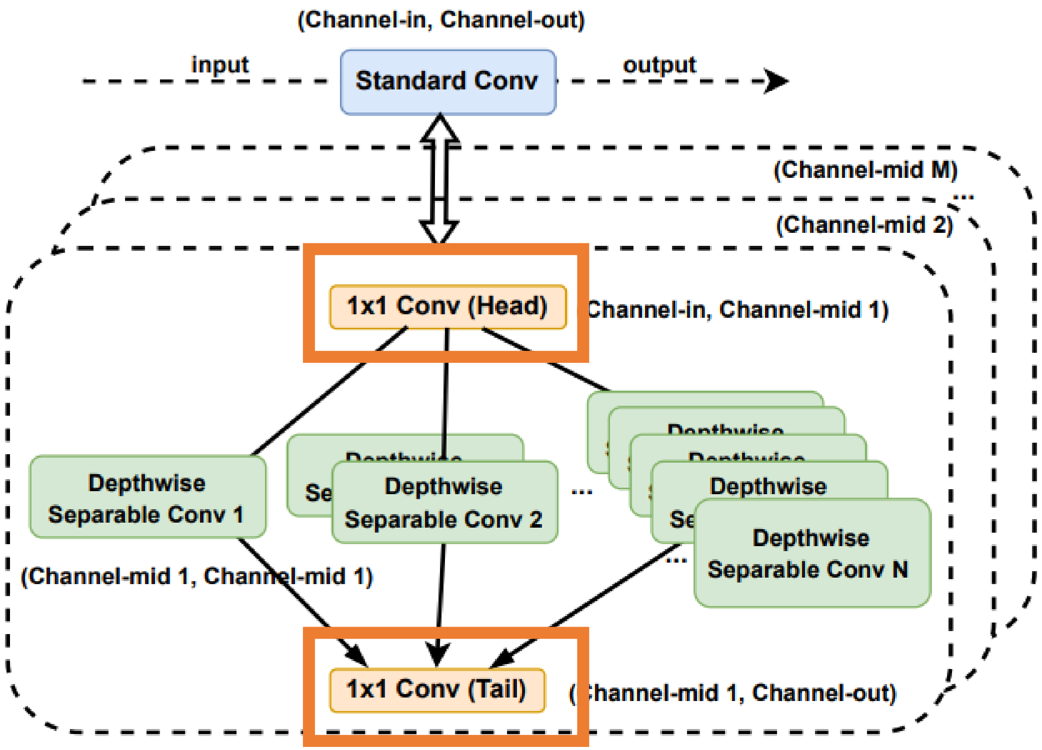


Figure 8: Example for candidate blocks of a convolutional layer on CPU

- NAS-based Block Optimization
  - Only consider convolutional and fully-connected layers since they account for the most execution time.
  - Convolutional layer optimized in CPU
  - Convolutional layer optimized in GPU
    - adopt denser convolutional layers
  - Fully-connected layer optimized

Design – duo-block generation

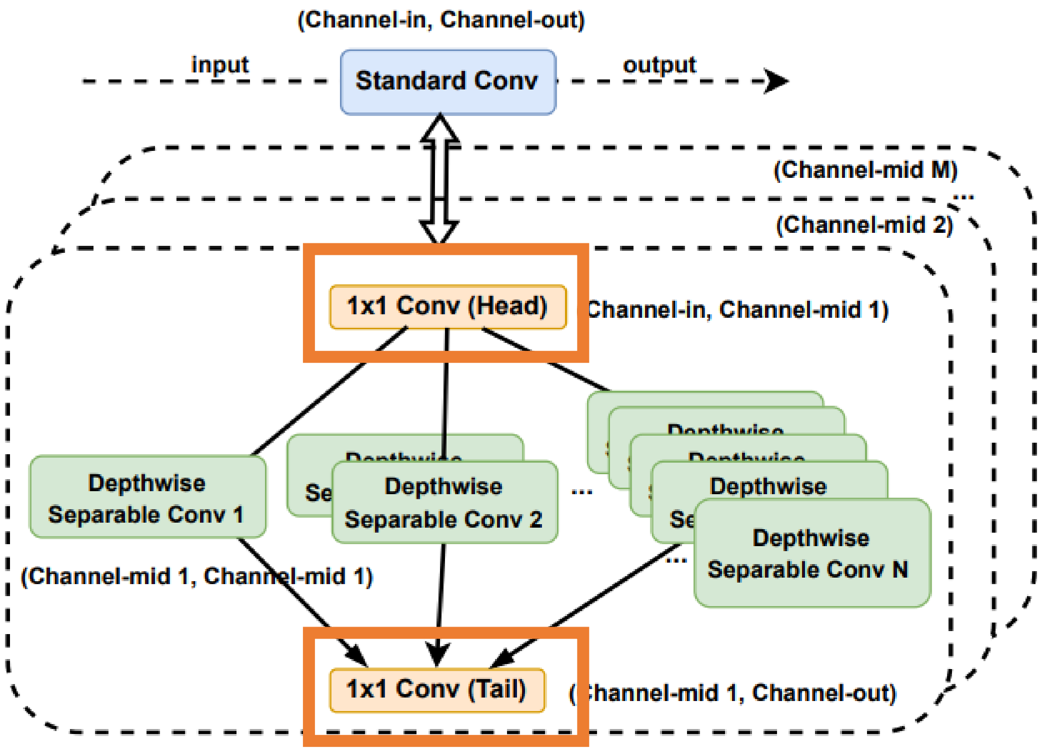


Figure 8: Example for candidate blocks of a convolutional layer on CPU

- NAS-based Block Optimization
  - Only consider convolutional and fully-connected layers since they account for the most execution time.
  - Convolutional layer optimized in CPU
  - Convolutional layer optimized in GPU
  - Fully-connected layer optimized
    - choose the fully-connected layer with a smaller/larger channel size

Design – duo-block generation

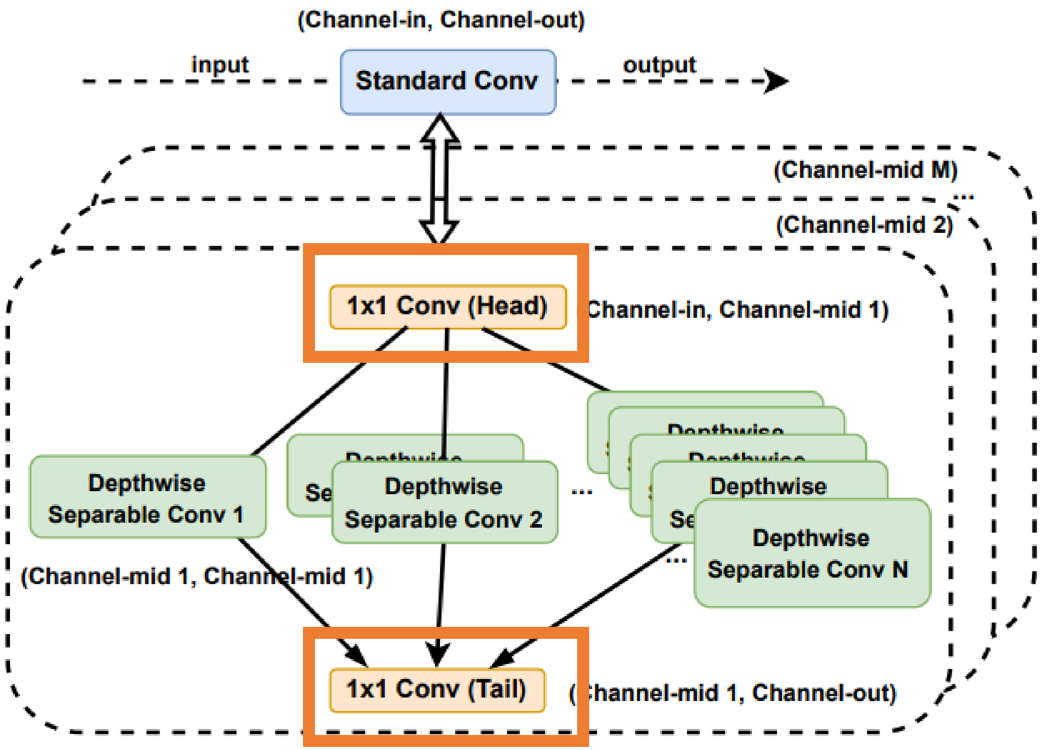


Figure 8: Example for candidate blocks of a convolutional layer on CPU

- **NAS-based Block Optimization**
  - **Each candidate block has a head module and a tail module.**
  - **Search for the optimal block that minimizes the accuracy loss.**
  - **The search algorithm is based on the Differentiable Architecture Search (DARTS) algorithm**

$$\forall\, k \quad \min \ \mathcal{L}(B_k^{new}, W^*)$$
$$s.t. \ \ T^{sec}(B_k^{old}) > T^{sec}(B_k^{new})$$

Design – dynamic cross processor scheduling

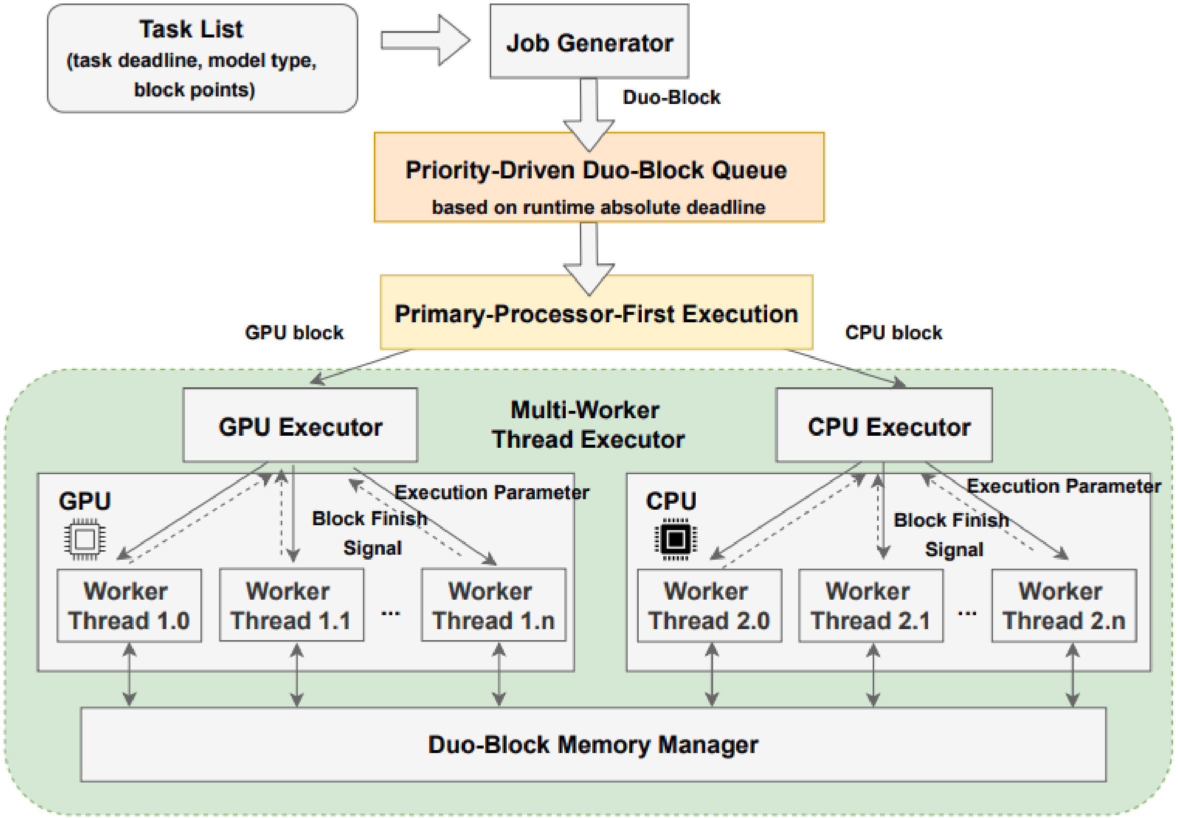


Figure 9: Procedure for cross-processor scheduling, worker thread 1.1 represents the worker thread for DNN model 1 on the GPU processor.

- The scheduler prioritizes each duo-block based on its task urgency.
- Primary processor-first execution mechanism decides the execution processor for each duo-block based on the status of the processors.

Design – dynamic cross processor scheduling

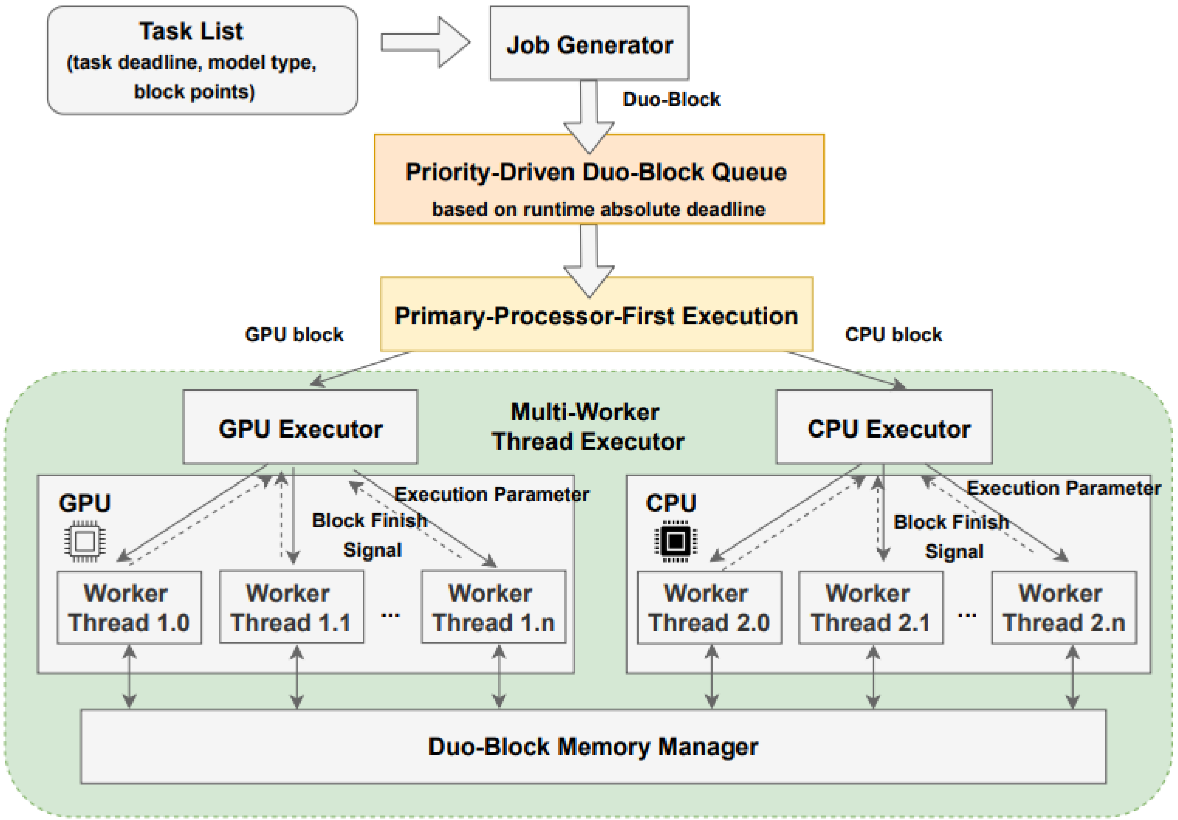


Figure 9: Procedure for cross-processor scheduling, worker thread 1.1 represents the worker thread for DNN model 1 on the GPU processor.

- The scheduler prioritizes each duo-block based on its task urgency.
- **Primary processor-first execution mechanism decides the execution processor for each duo-block based on the status of the processors.**

Design – dynamic cross processor scheduling

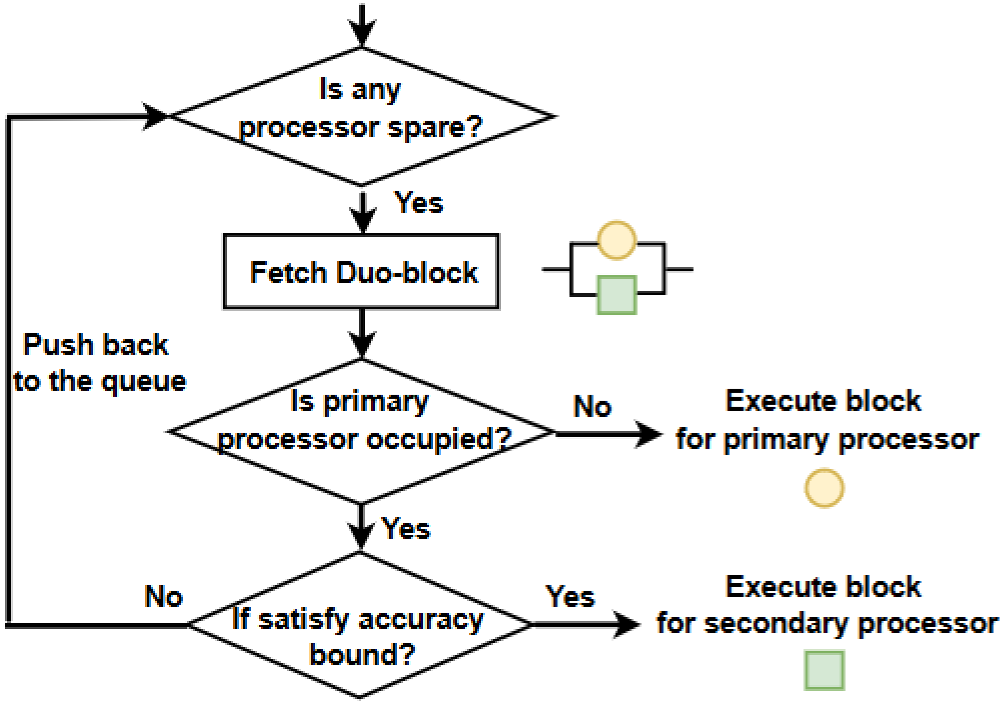


Figure 10: Primary-processor-first execution mechanism

- The scheduler prioritizes each duo-block based on its task urgency.
- **Primary processor-first execution mechanism decides the execution processor for each duo-block based on the status of the processors.**

Implementation

- **Platforms**

Table 1: Platforms used in evaluation experiments.

Platform	GPU	CPU	Memory	Storage
NVIDIA AGX Xavier	512-core Volta	8-core ARMv8.2	16GB	32GB
NVIDIA Jetson TX2	256-core Pascal	2-core ARM Denver + 4-core ARM A57	8GB	32GB
Desktop	RTX2080	8-core Intel i9-9900K	32GB	5TB

Implementation

- **Implementation**

DNN Inference Task Type	Dataset	DNN Model
Image Classification	CIFAR10 [24]	MobileNet[17], VGG11, AlexNet
Sign Recognition	GTSRB [48]	ResNet18
Object Detection	COCO [32]	YOLO [44]

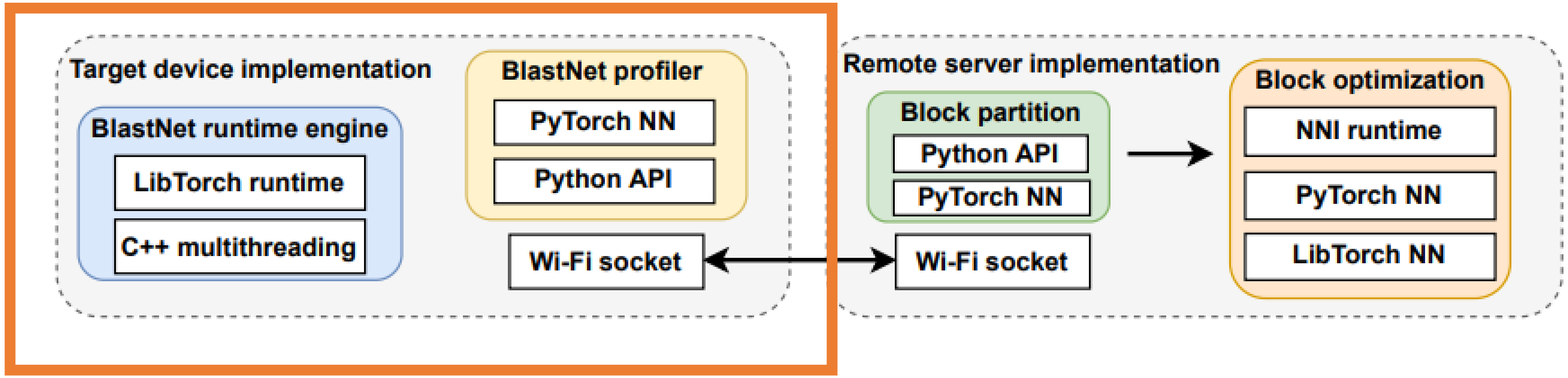


Figure 11: BlastNet Software Implementation

Implementation

- **Implementation**

DNN Inference Task Type	Dataset	DNN Model
Image Classification	CIFAR10 [24]	MobileNet[17], VGG11, AlexNet
Sign Recognition	GTSRB [48]	ResNet18
Object Detection	COCO [32]	YOLO [44]

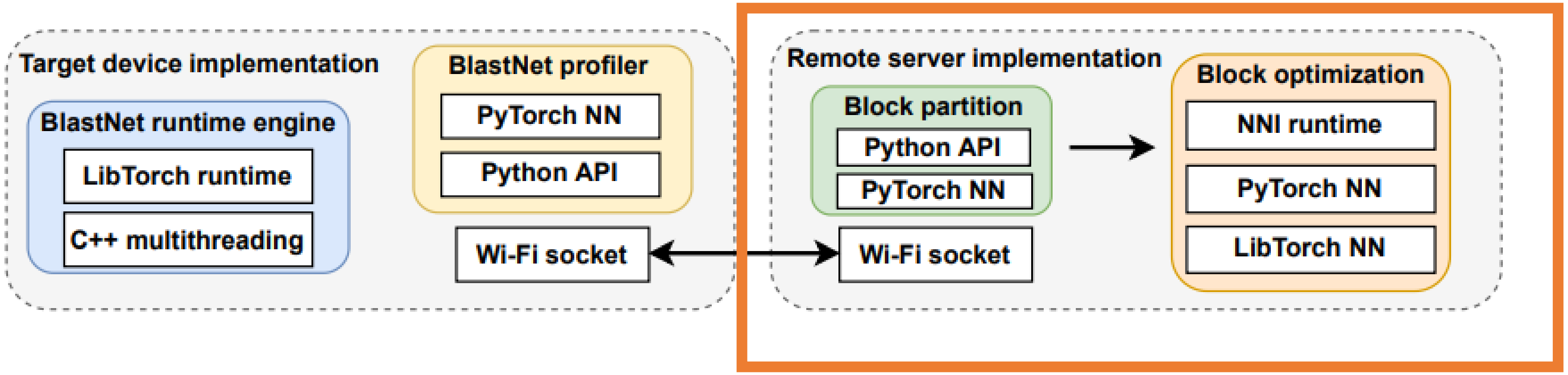


Figure 11: BlastNet Software Implementation

End-to-end System Evaluation

- **Autonomous driving testbed**

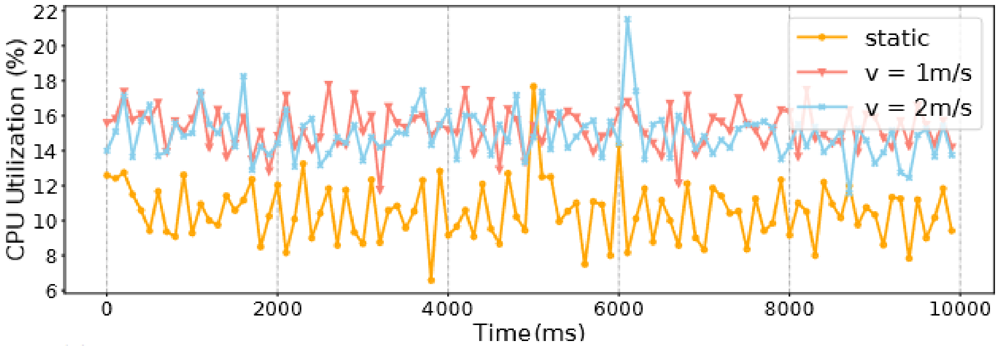
(a) F1/10 Vehicle (b) Testbed setup

Figure 12: F1/10 autonomous driving testbed.

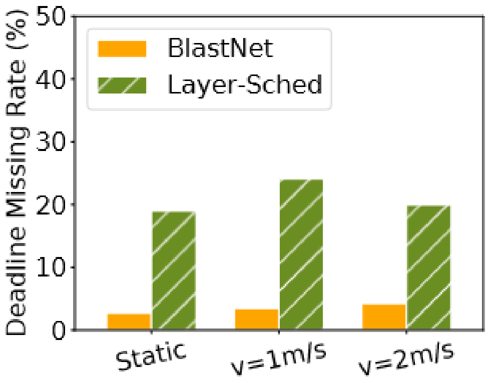
Four real-time DL tasks for traffic sign recognition and a lane detection task running on this platform.

End-to-end System Evaluation

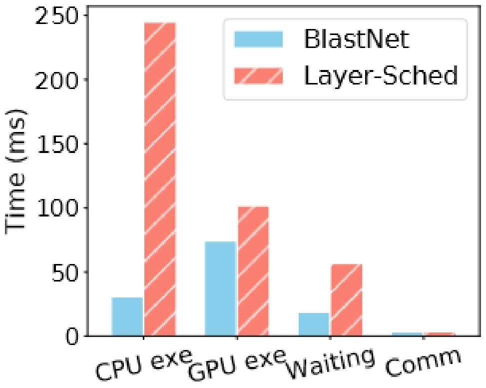
- **Autonomous driving testbed**



(a) CPU utilization under different car speed without DNN workload



(b) Deadline missing rate

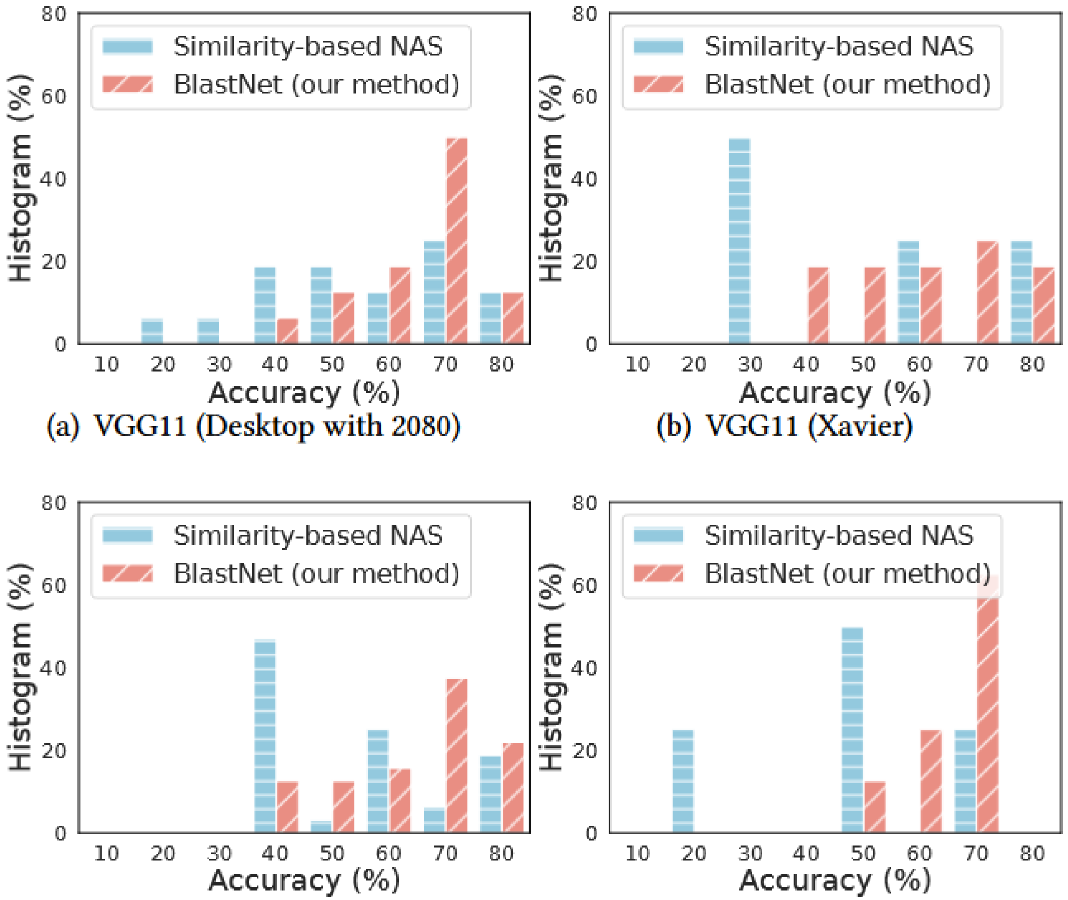


(c) Execution time for each category

Figure 13: Performance of BlastNet under various driving settings.

- **Baseline: *Layer _Sched* schedules the DNN model at the layer level**
- **Running the F1/10 autonomous vehicle at different speeds, resulting in different levels of resource utilization.**

Performance of Cross-processor Duo-block Generation



(a) VGG11 (Desktop with 2080) (b) VGG11 (Xavier)

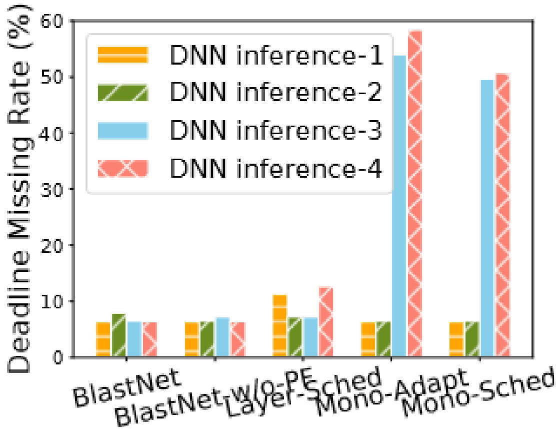
(c) VGG11 (TX2) (d) AlexNet (Xavier)

Figure 14: Model accuracy with all possible inference paths with the optimized blocks.

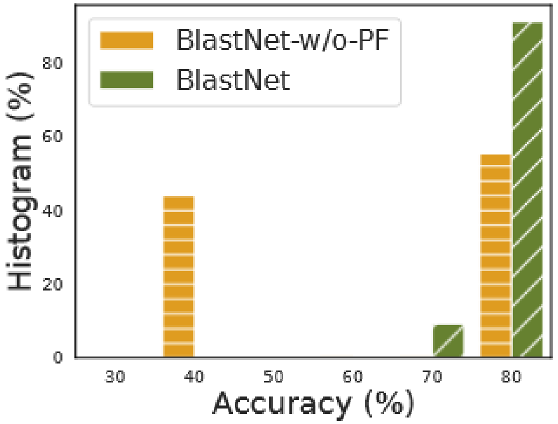
- **Baseline:** *Similarity-based NAS*

Overall Performance

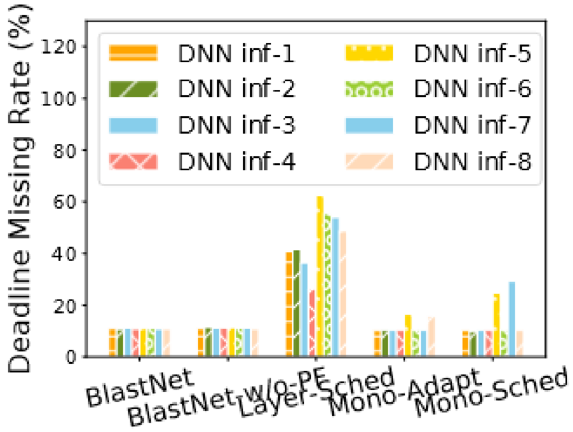
- **Impact of different DNN workloads**



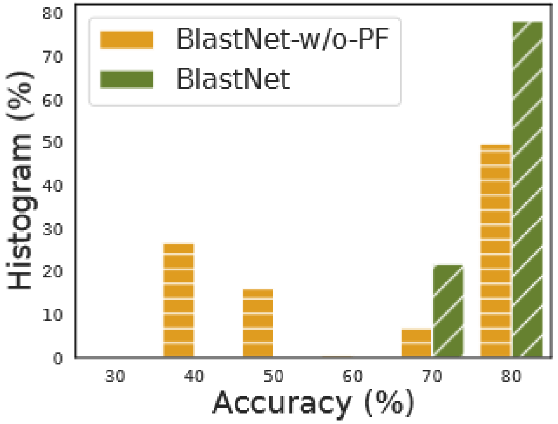
(a) 4DNN inference task set (Real-time performance)



(b) 4DNN inference task set (Accuracy performance)



(c) 8DNN inference task set (Real-time performance)



(d) 8DNN inference task set (Accuracy Performance)

- ***Mono_Sched*** **: Monolithically allocate the DNN models to heterogeneous resources.**
- ***Layer _Sched*** **: Schedule the DNN model at the layer level**
- ***Mono_Adapt*** **: Differs from BlastNet only in DNN scheduling by adopts monolithic scheduling to execute the models.**
- **BlastNet w/o-PF: Differ from BlastNet only in that it has no primary-first execution mechanism**

Overall Performance

- **Impact of different DNN workloads**

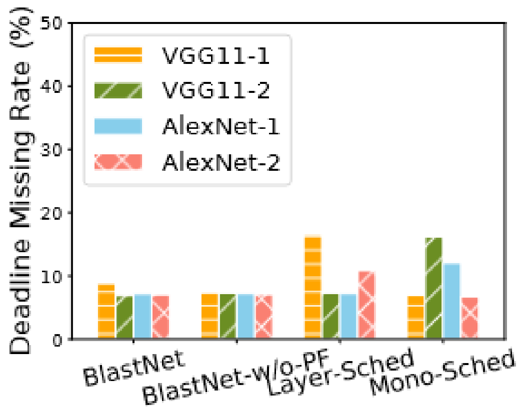
	Minimum	Average	1/4 Value	Maximum
BlastNet (4task)	73.05%	80.08%	80.79%	80.79%
BlastNet-w/o-PF (4task)	46.65%	65.69%	46.65%	80.79%
BlastNet (8task)	73.05%	79.16%	80.79%	80.79%
BlastNet-w/o-PF (8task)	46.65%	67.02%	46.65%	80.79%

(e) Statistics for accuracy

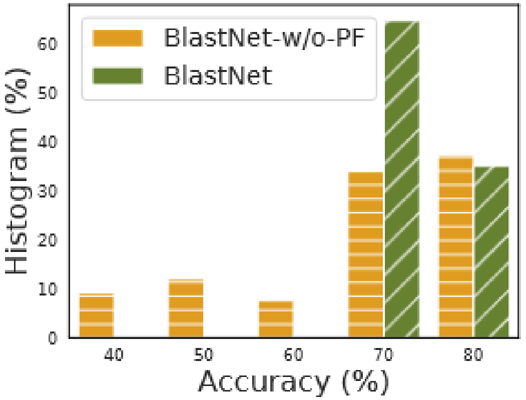
Figure 15: Real-time/Accuracy performance of BlastNet under different DNN workloads.

Overall Performance

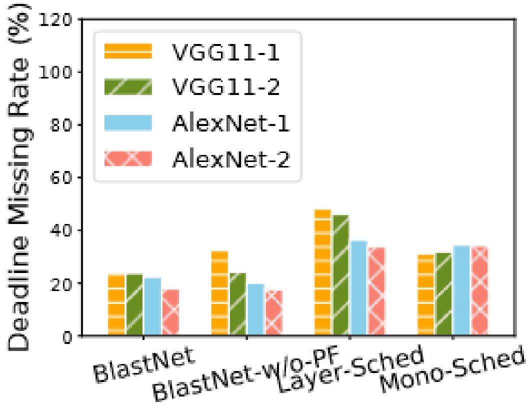
- **Impact of background load**



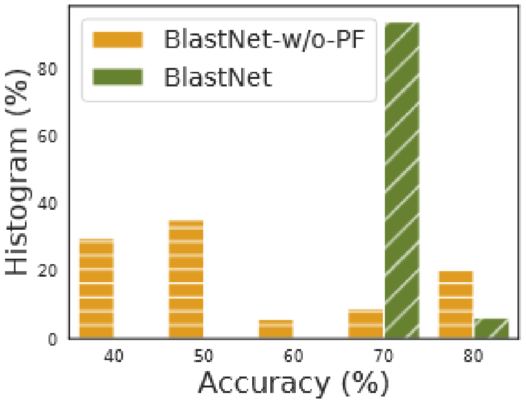
(a) Without background interference (Real-time Performance)



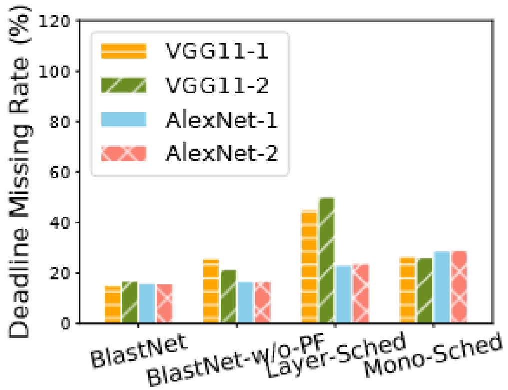
(b) Without background interference (Accuracy Performance)



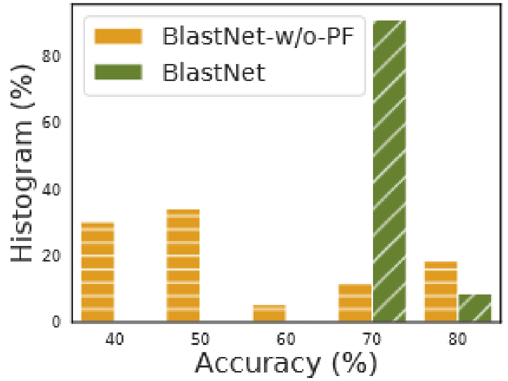
(c) GPU background interference (Real-time Performance)



(d) GPU background interference (Accuracy Performance)



(e) CPU background interference (Real-time Performance)



(f) CPU background interference (Accuracy Performance)

Figure 16: Real-time/Accuracy performance of BlastNet under interference.

Overall Performance

- **Different edge platforms**

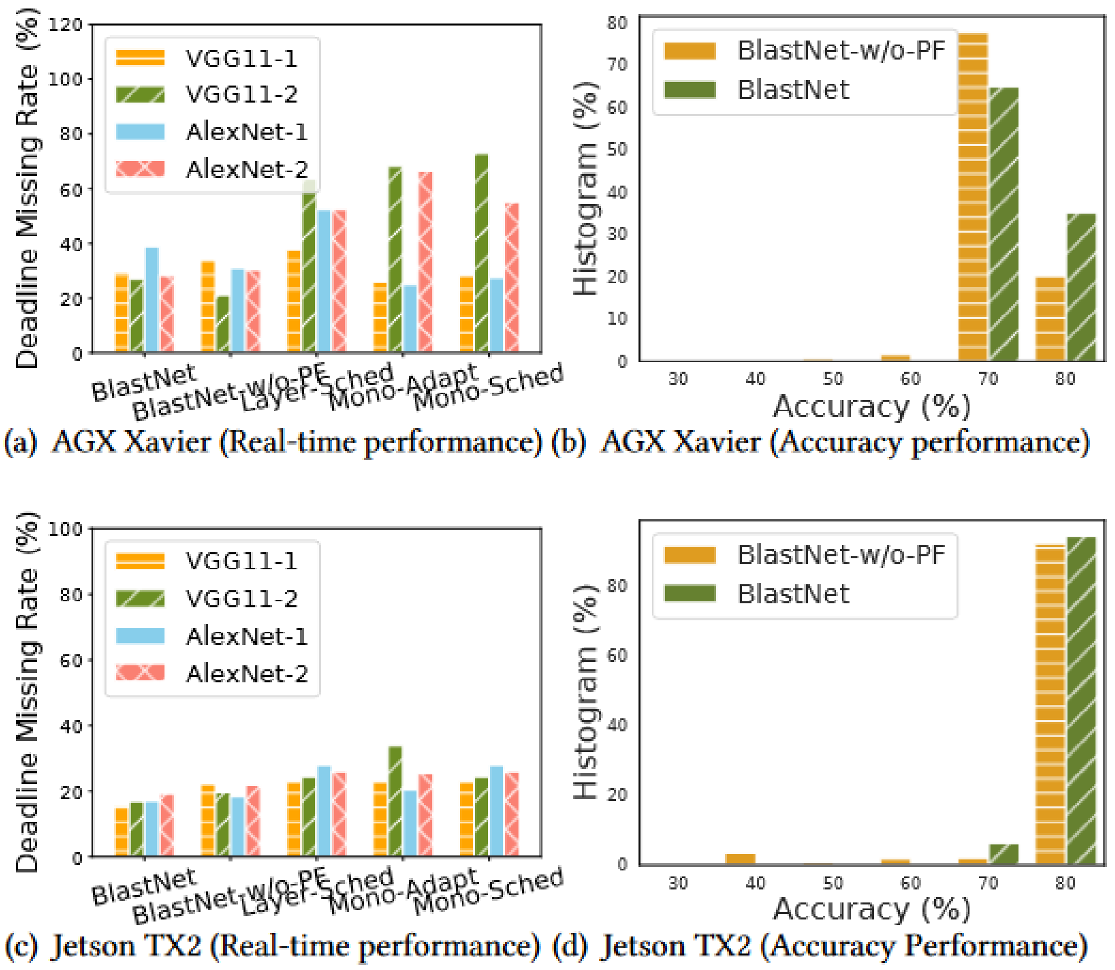
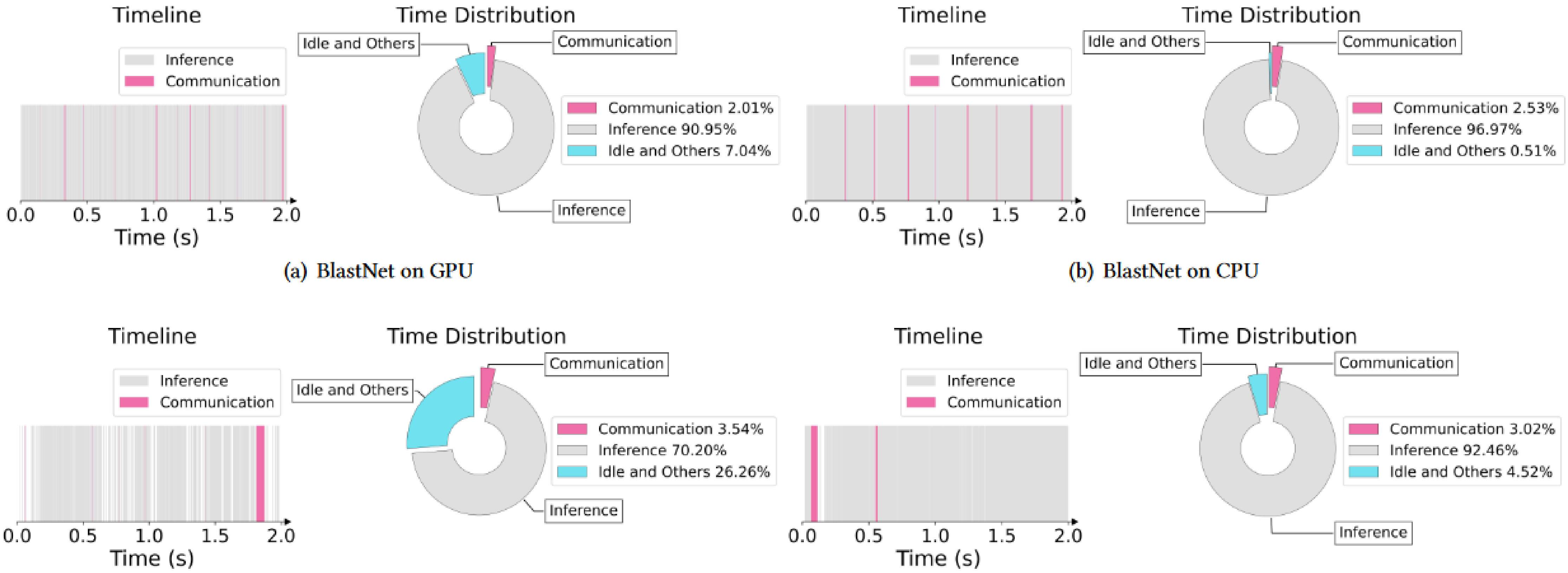


Figure 17: Performance comparison of BlastNet under different edge platforms.

Overall Performance

- **CPU/GPU utilization**



(a) BlastNet on GPU

(b) BlastNet on CPU

(c) Layer-Sched on GPU

(d) Layer-Sched on CPU

Figure 18: Execution timeline for inference, communication, and idle.

System Overhead

Table 3: CPU Overhead of Block-level DNN Scheduling

Task Set Size	Desktop	Xavier	TX2
2 DNN inference tasks	1.21%	1.61%	2.86%
4 DNN inference tasks	1.52%	1.38%	3.97%
6 DNN inference tasks	2.08 %	1.71%	3.20%

- **caused by the dynamic cross-processor DNN scheduler**

Advantage

- **Propose a new abstraction of model partition : duo-block.**
- **Efficient utilize the resource of GPU and CPU.**

Disadvantage

- **The online scheduling strategy is overly simplistic.**