CoActo: CoActive Neural Network Inference Offloading with Fine-grained and Concurrent Execution

Mobisys'24

Kyungmin Bin
Seoul National University
kmbin@snu.ac.kr

Jongseok Park
Seoul National University
cakeng@snu.ac.kr

Chanjeong Park
Seoul National University
cjpark99@snu.ac.kr

Seyeon Kim
University of Colorado Boulder
seyeon.kim@colorado.edu

Kyunghan Lee
Seoul National University
kyunghanlee@snu.ac.kr

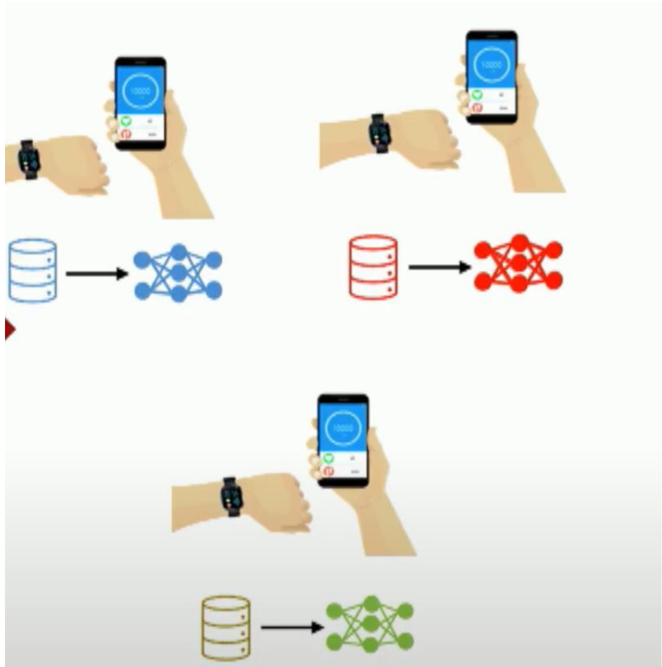
1 Introduction

Mobile can provide high-quality services that are comparable to those of human experts now

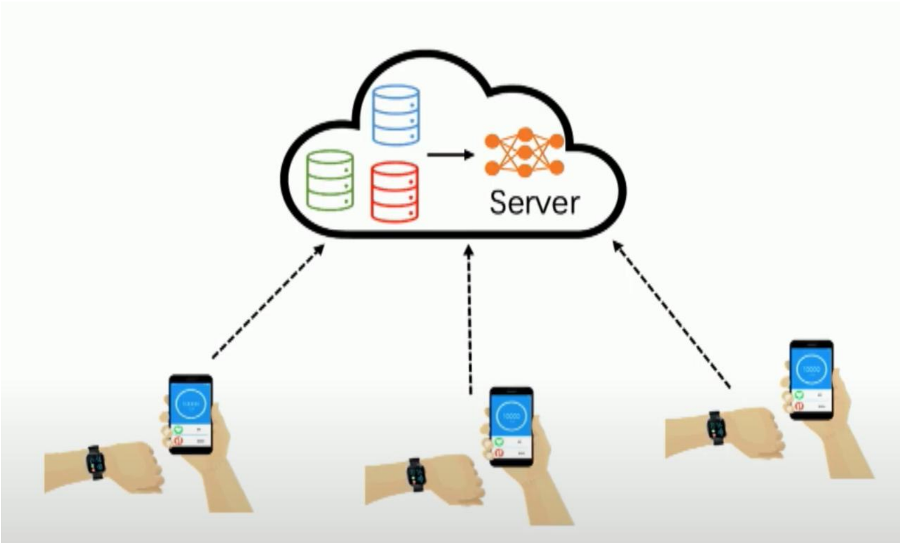
user interactions is important

Core target: Reduce the inference latency

1 Introduction



VS

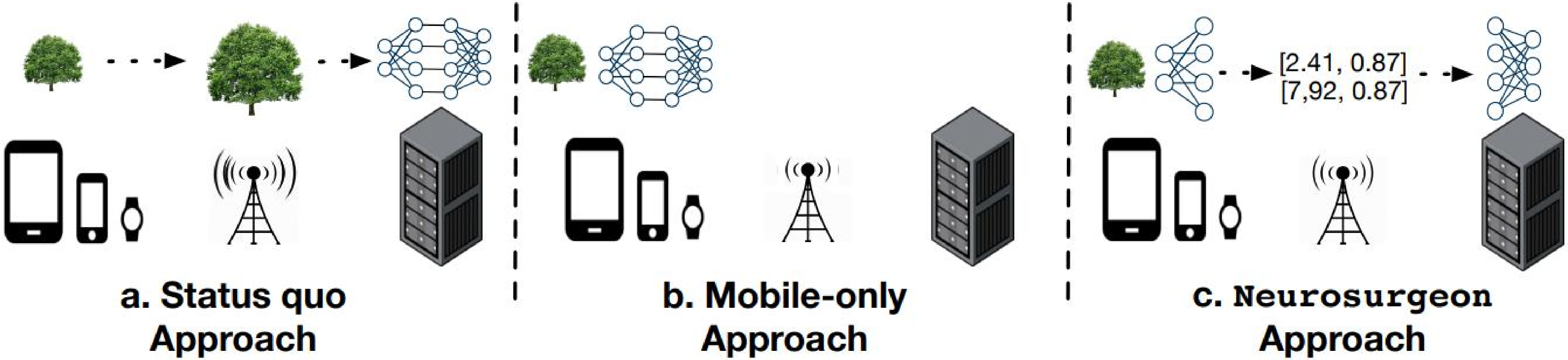


On-device Inference
(For more complex models, unable to achiece low latency)

Offload to server
(With more computing power to support more complex tasks)

1 Introduction

Offload all the model to the cloud is still not the best choice



server only

Mobile only

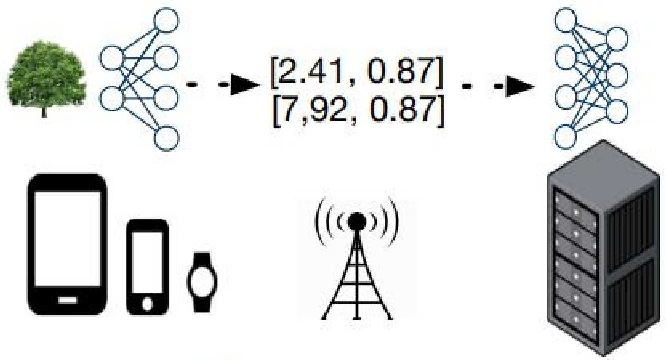
mobile-server collaborative inference

- mobile inference
- server inference
- transmittion cost
- optimal partitioning
- best scheduling scheme

[1]Kang Y, Hauswald J, Gao C, et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge[J]. ACM SIGARCH Computer Architecture News, 2017, 45(1): 615-629.

1 Introduction

For DNN offloading, this not only includes the partitioning and scheduling of the workload, but also the modeling of the workload, execution algorithm, dynamic load-balancing, possibility of multi-tenant execution, and many more.



c. Neurosurgeon Approach

mobile-server collaborative inference

- mobile inference
- server inference
- transmittion cost
- optimal partitioning
- best scheduling scheme

A more fundamental question:

How should a DNN execution system be designed for efficient mobile DNN offloading?

[1]Kang Y, Hauswald J, Gao C, et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge[J]. ACM SIGARCH Computer Architecture News, 2017, 45(1): 615-629.

1 Introduction

A more fundamental question:

How should a DNN execution system be designed for efficient mobile DNN offloading?

To realize efficient DNN inferences in a mobile-server offloading environment

Two key properties

- a fine-grained expression of DNNs
- flexibility of the system resource utilization

1 Introduction

A more fundamental question:
How should a DNN execution system be designed for efficient mobile DNN offloading?

To realize efficient DNN inferences in a mobile-server offloading environment

Two key properties

- a fine-grained expression of DNNs
  - DNNs are often expressed in units of layers.
    - too large
    - not supply enough parallelism to efficiently utilize all available resources.
- flexibility of the system resource utilization

1 Introduction

A more fundamental question:
How should a DNN execution system be designed for efficient mobile DNN offloading?

To realize efficient DNN inferences in a mobile-server offloading environment

Two key properties

- a fine-grained expression of DNNs
- flexibility of the system resource utilization
    - dynamic nature of DNN offloading
    - the system must be flexible enough to support dynamic changes
        - changes in available computation resources
        - network conditions
        - competing inference offloads

2 Background and Motivation

two representative approaches in collaborative inference:

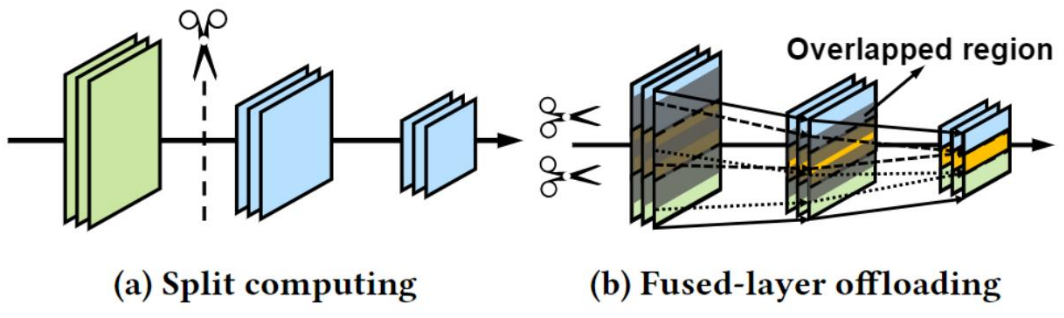


Figure 2: Illustrations of two collaborative inference approaches, (a) split computing and (b) fused-layer offloading.

- Split computing
  - splits the DNN model into two submodels at the layer level
  - the key is the split point
  - many studies try to find the the optimal split point
  - this sequential execution cannot make full use of the available computing resources

2 Background and Motivation

two representative approaches in collaborative inference:

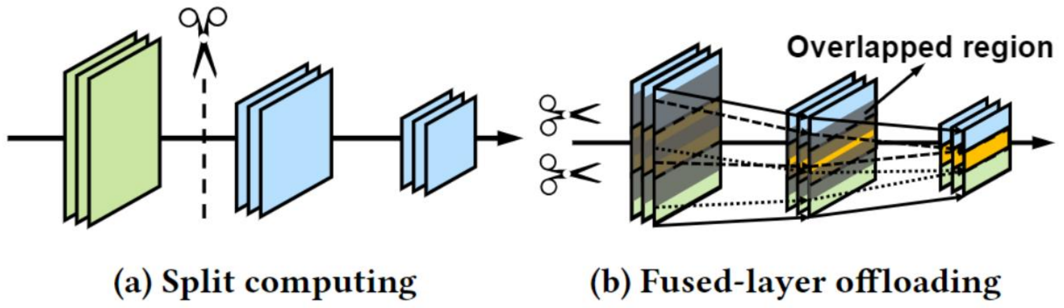


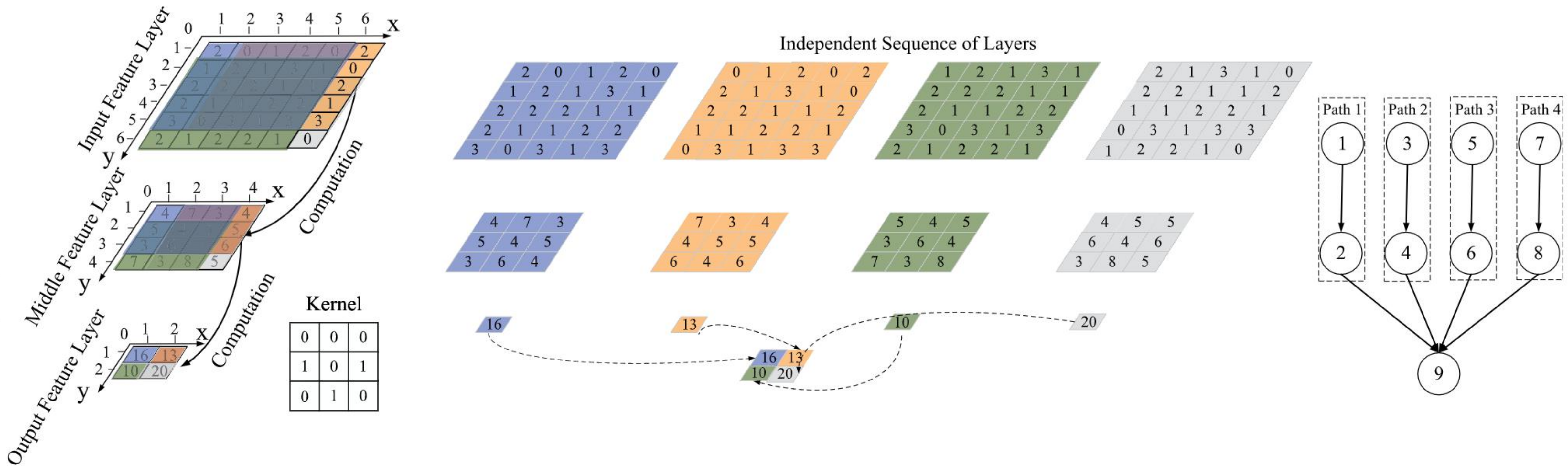
Figure 2: Illustrations of two collaborative inference approaches, (a) split computing and (b) fused-layer offloading.

- Fused-layer (FL) offloading
  - fuse multiple layers by exploiting the spatial locality of layers
  - several submodels with zero data dependencies
  - executed without any synchronization to the computation of other submodels,

2 Background and Motivation

two representative approaches in collaborative inference:

- Fused-layer (FL) offloading



(a) Correspondence of values in DNN

(b) Paths divided by the FL technique

Fig. 2. A simple DNN computation with the FL technique and homogeneously intercepted fused layer's size.

[1]Zhou H, Li M, Wang N, et al. Accelerating deep learning inference via model parallelism and partial computation offloading[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 34(2): 475-488.

2 Background and Motivation

two representative approaches in collaborative inference:

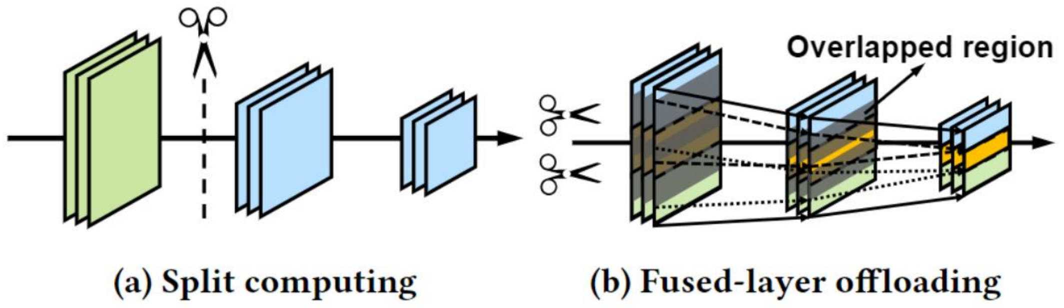


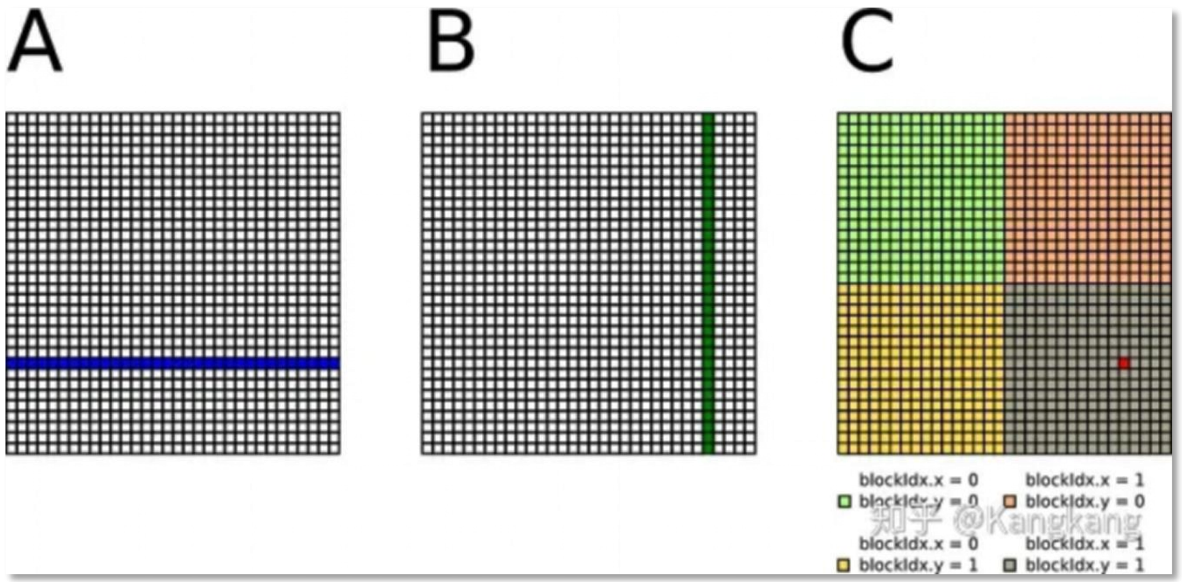
Figure 2: Illustrations of two collaborative inference approaches, (a) split computing and (b) fused-layer offloading.

- Fused-layer (FL) offloading
  - fuse multiple layers by exploiting the spatial locality of layers
  - several submodels with zero data dependencies
  - executed without any synchronization
  - suffers from limited scalability and high computation overhead

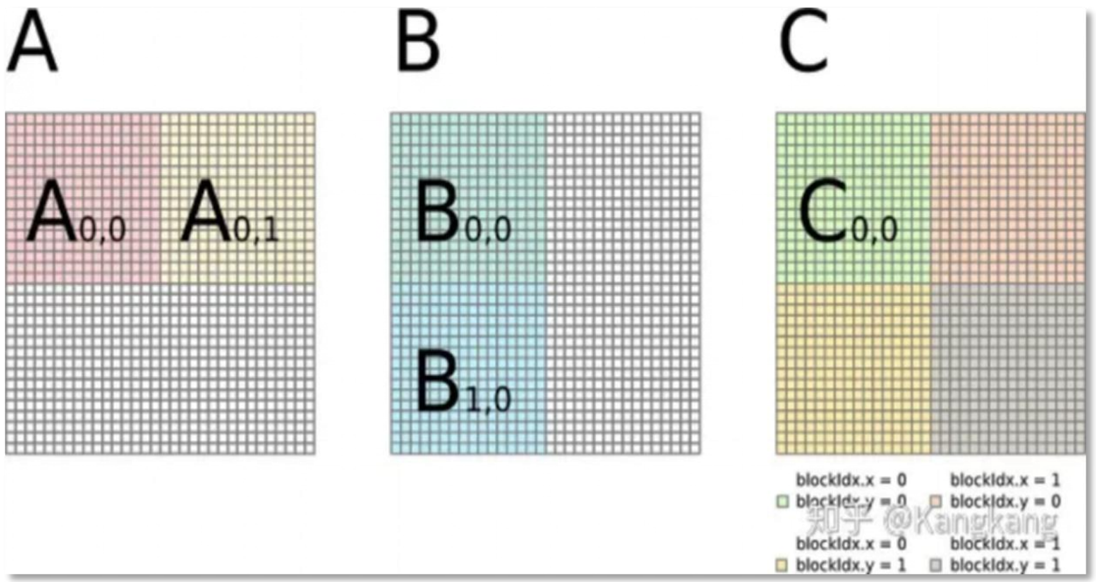
Layers are simply too large of a unit for a DNN offloading environment → Tiling for Collaborative Inference

2 Background and Motivation

Tiling for Collaborative Inference



native matrix multiplication



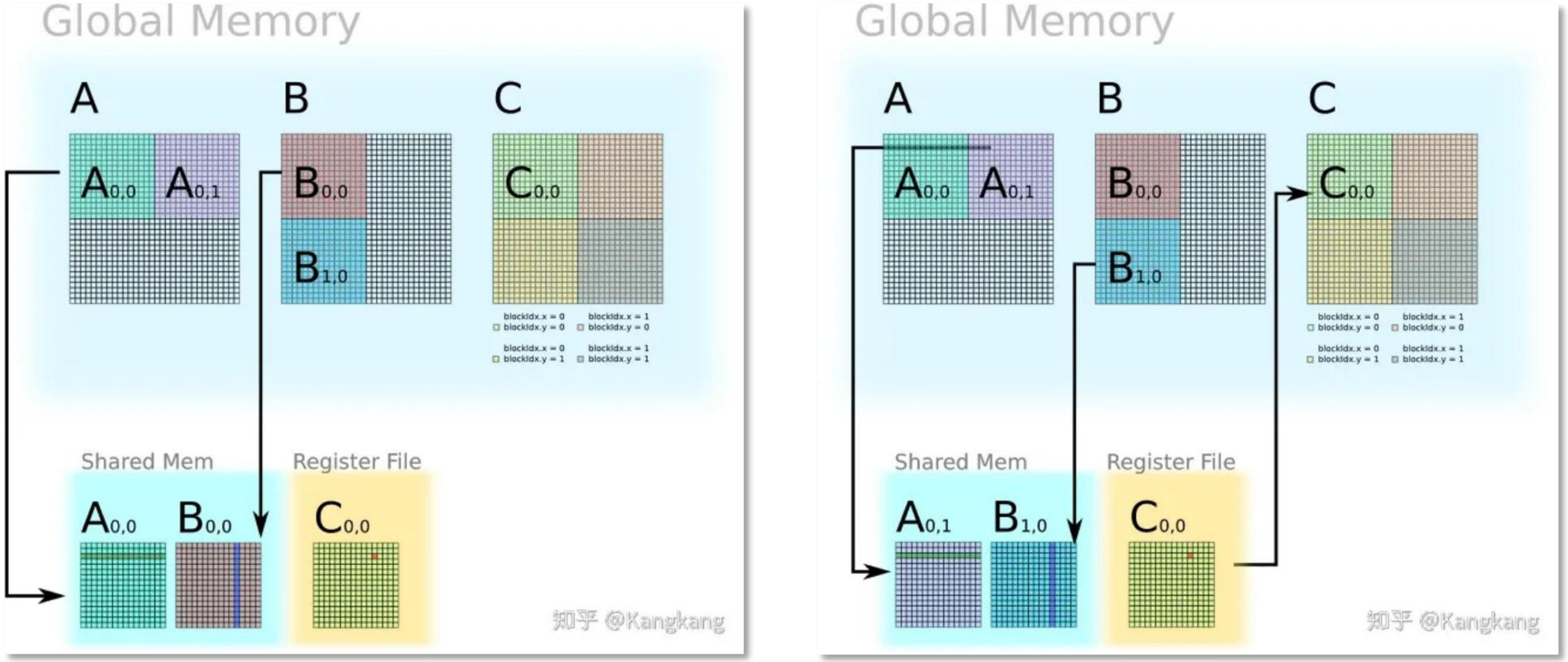
use tiling techinique

[1]reference:https://zhuanlan.zhihu.com/p/342103911

2 Background and Motivation

Tiling for Collaborative Inference

increase the parallelism
of the operation



tile sizes is flexible

tiles to be the ideal unit for fine-grained expression of DNN computation

[1]reference:https://zhuanlan.zhihu.com/p/342103911

2 Background and Motivation

Design Philosophy

Traditional approaches for collaborative inference primarily focus on model splitting

Not only to distribute the workload but also to ensure that runtime execution system components work in unison to make the best use of the available resources under the dynamic environments that exist during DNN offloading

2 Background and Motivation

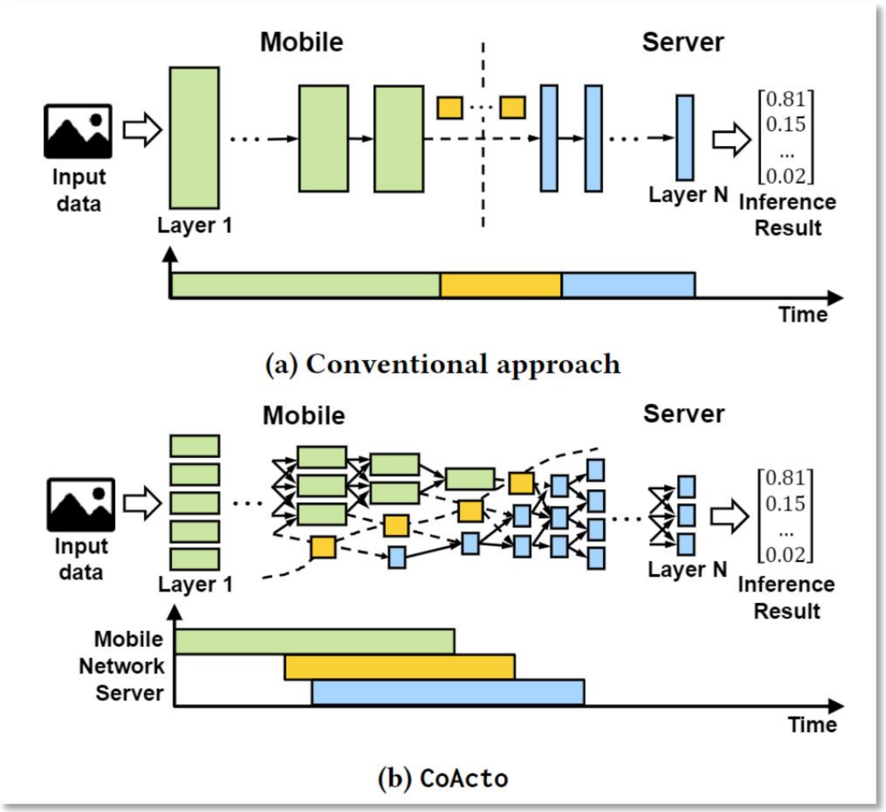
Two design philosophy

- Fine-grained DNN expression
  - Smaller workload size allows faster unit processing times
  - suitable for dynamic scheduling of parallel resources

2 Background and Motivation

Two design philosophy

- Fine-grained DNN expression
- Concurrency of runtime resources
  - Concurrent, rather than sequential, use of these resources is necessary to maximize parallel resource utilization.



(a) Conventional approach

(b) CoActo

2 Background and Motivation

Three challenges

1. Tile-based expression
    - expresses an arbitrary layer-wise computation graph as a tile-wise computation graph poses many challenges
        - determining the efficient tile dimensions and size for the given environment
        - automatically parsing
        - generating the independent data dependency flow graph between the tiles

2 Background and Motivation

Three challenges

1. Tile-based expression
2. Concurrent execution system
   - Tensorflow or PyTorch execute at layer level
   - tiling restricts the concurrency only to the intra-layer level
   - designing a concurrent execution system that enables overlapping the computation and communications of tiles is challenging

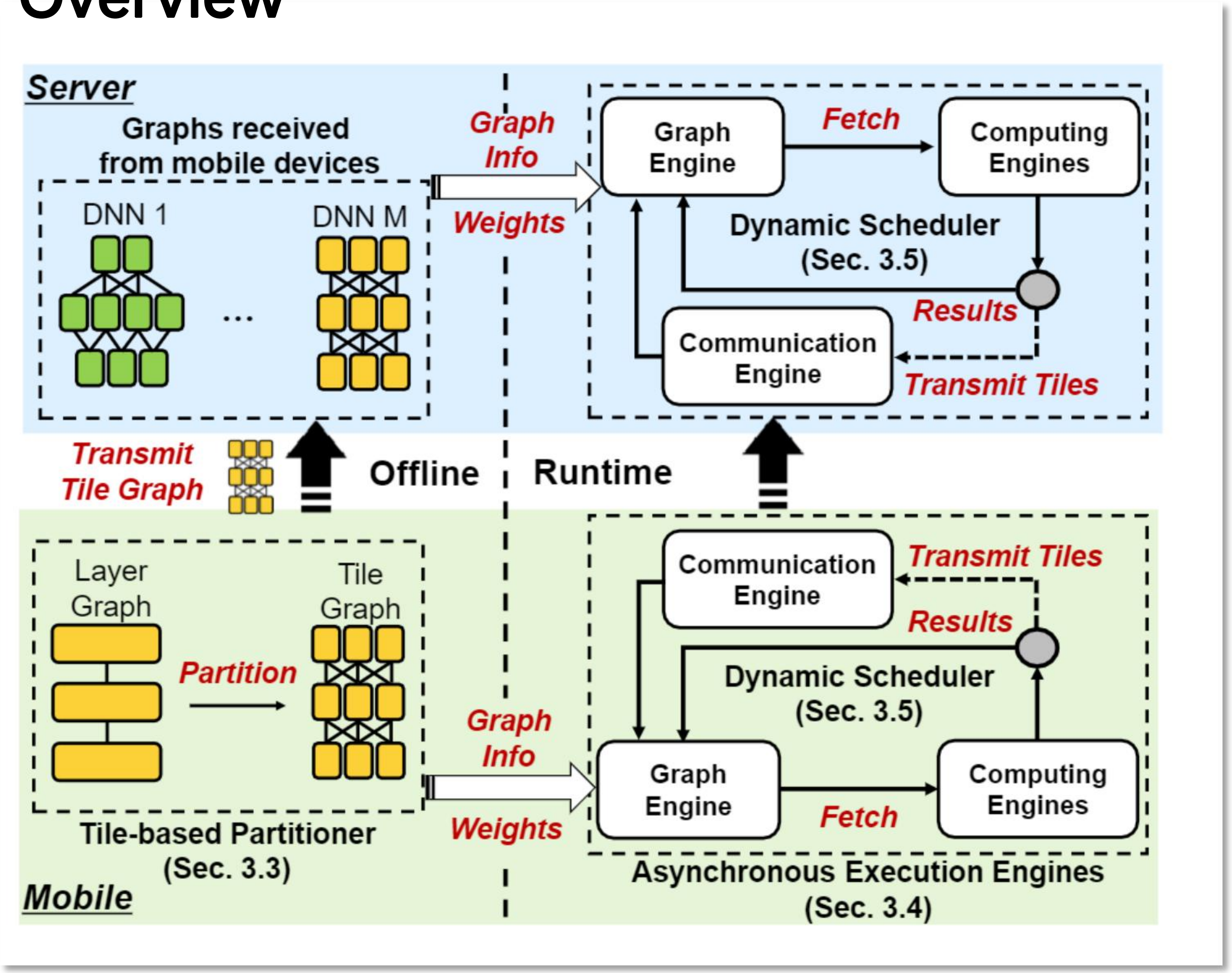
2 Background and Motivation

Three challenges

1. Tile-based expression
2. Concurrent execution system
3. Dynamic scheduling of tiles
   - balancing the model executions between the mobile, network, and server resources of the given environment
   - dynamic adaptation and balancing of complex fine-grained DNN between the concurrently operating resources is also challenging

3 *CoActo* Design

Overview



Three challenges

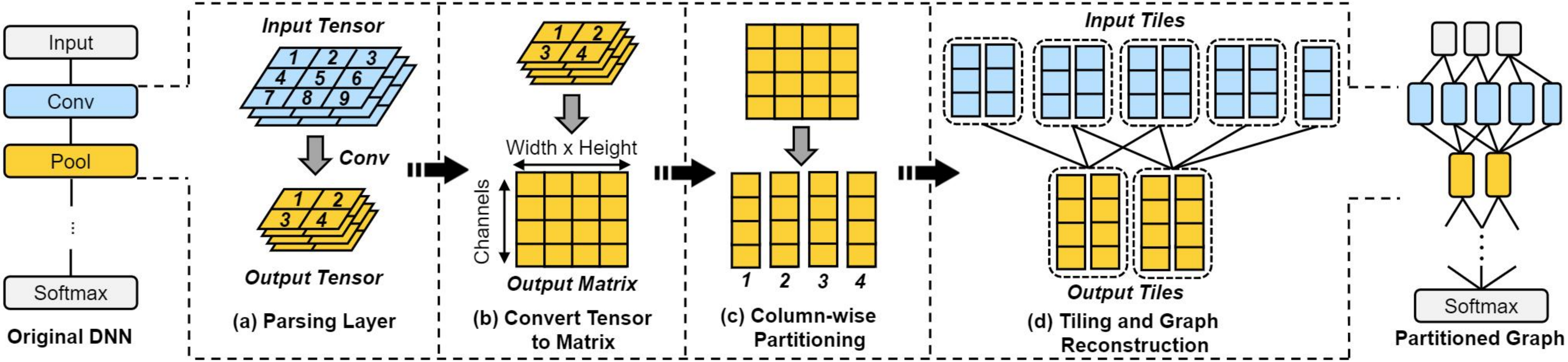
1. Tile-based expression
2. Concurrent execution system
3. Dynamic scheduling of tiles

Three components

- Tile-based Partitioner (TP)
- Asynchronous Execution Engines (AEEs)
- Dynamic Scheduler (DS)

3 *CoActo* Design

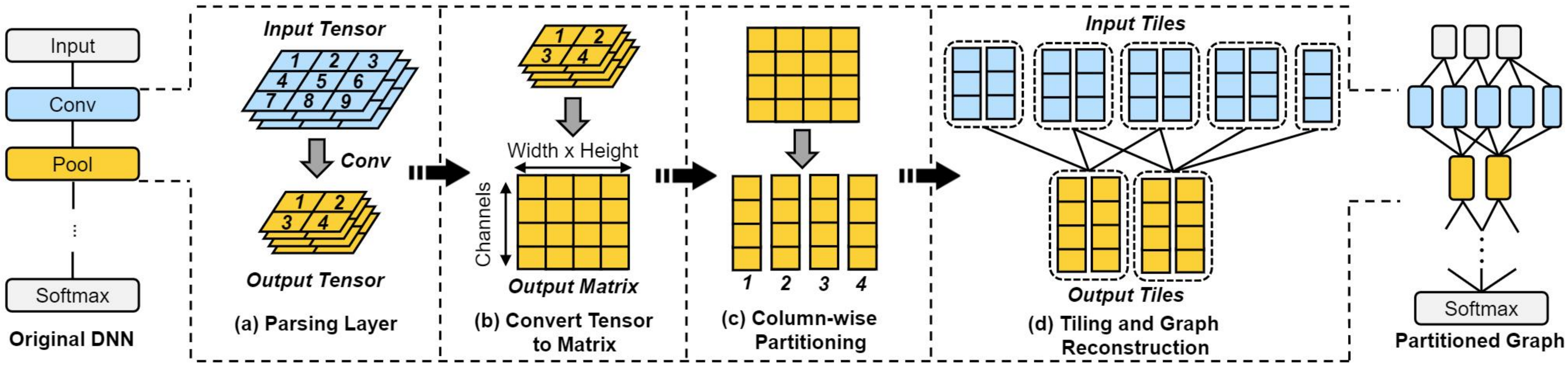
Tile-based Partitioner (TP)



1. analyze the size of each layer's output tensor
2. TP flattens the output tensors into a matrix
3. the matrices are decomposed into column-wise vectors
4. tiles are created by merging the partitioned columns（determined by the tile number which is predefined and changes by the environment)

3 *CoActo* Design

Tile-based Partitioner (TP)



The computation graph is then reconstructed by graphing the tiles into a directed acyclic graph (DAG)

3 *CoActo* Design

Tile-based Partitioner (TP)

Tiling allows a flexible and concurrent execution in DNN inference offloading

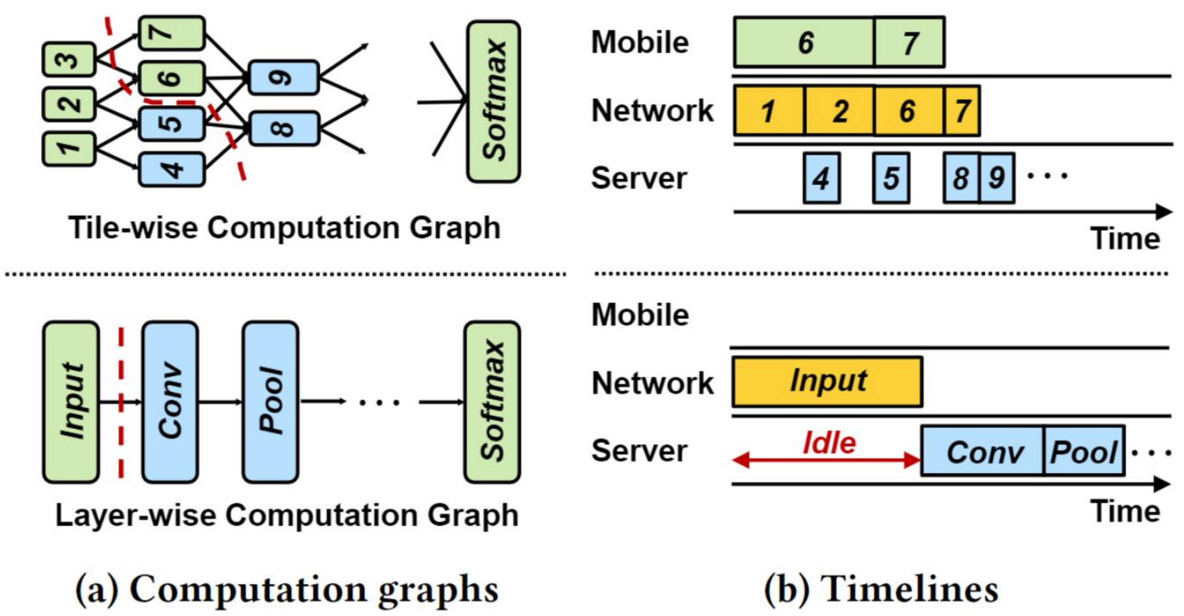


Figure 5: Examples of the collaborative inference with the tiles-wise computation graph (top) and the layer-wise computation graph (bottom).

3 *CoActo* Design

Asynchronous Execution Engines (AEEs)

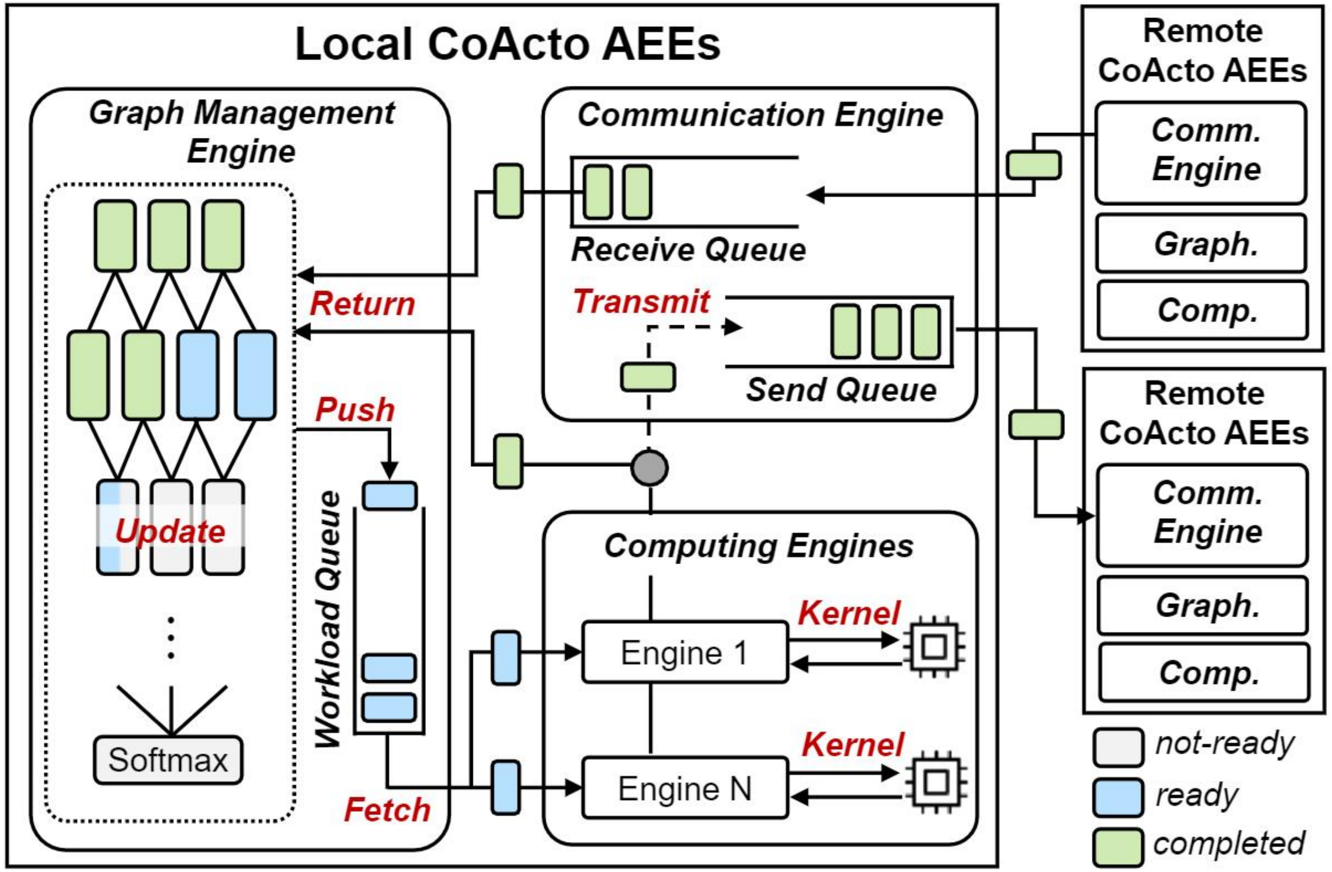


Figure 6: The overall execution workflows of Asynchronous Execution Engines.

Three types of execution engines

- Graph Management Engine
- Computing Engine
- Communication Engine

asynchronously and independently operate without waiting for each other

3 *CoActo* Design

Asynchronous Execution Engines (AEEs)

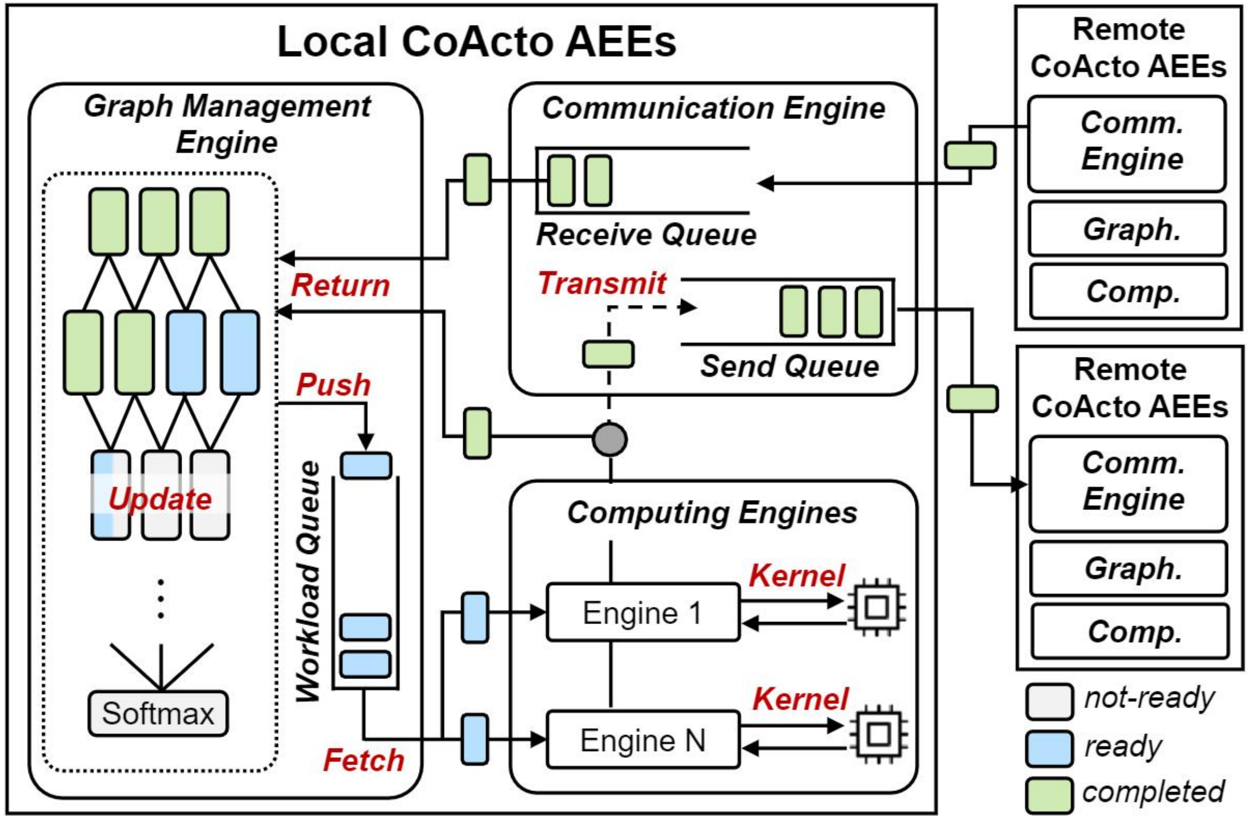


Figure 6: The overall execution workflows of Asynchronous Execution Engines.

Graph Management Engine

- managing the computation graph separately in each parallel computing engine requires frequent synchronization
- use Graph Management Engine
  - contains the entire computation graph information and its state

3 *CoActo* Design

Asynchronous Execution Engines (AEEs)

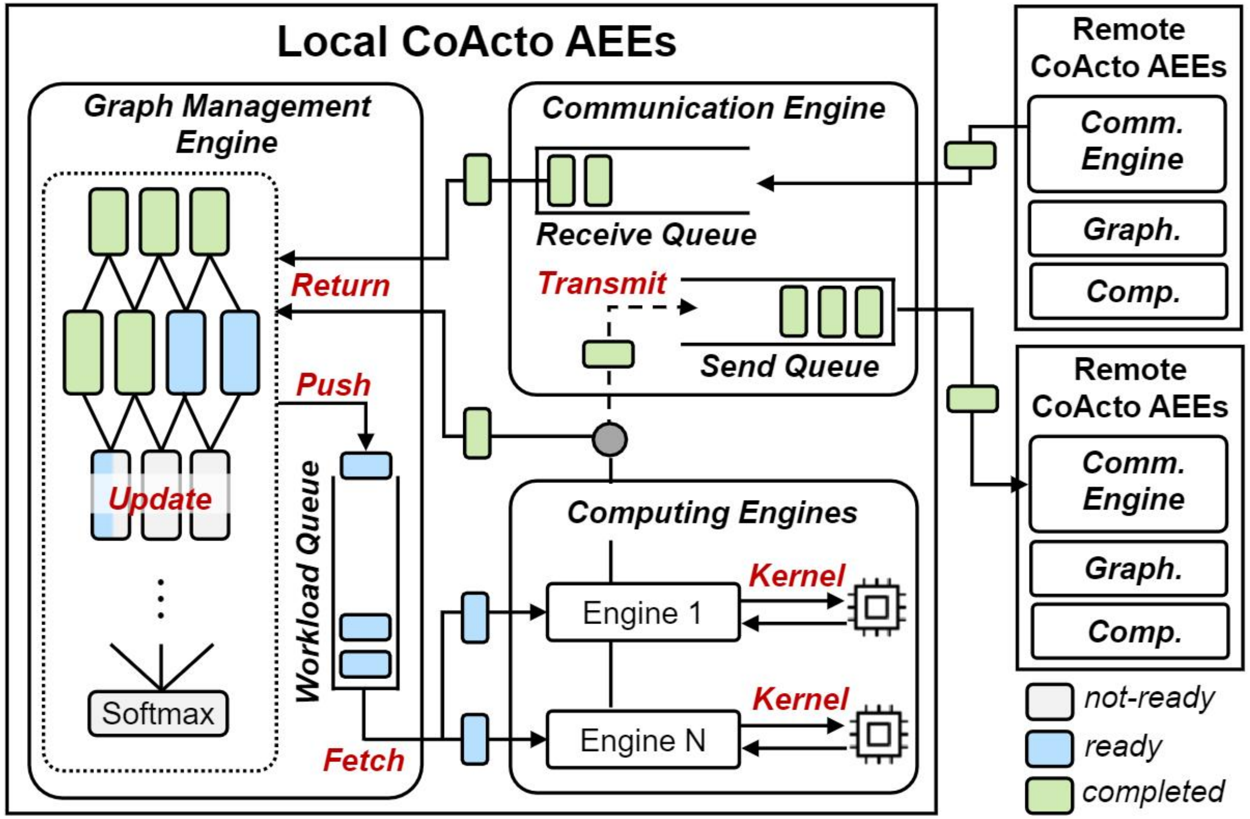


Figure 6: The overall execution workflows of Asynchronous Execution Engines.

Graph Management Engine

Three state of node(tile)

- completed
- ready
- not-ready

Workload queue

- pushes any child node that becomes ready to the workload queue

3 *CoActo* Design

Asynchronous Execution Engines (AEEs)

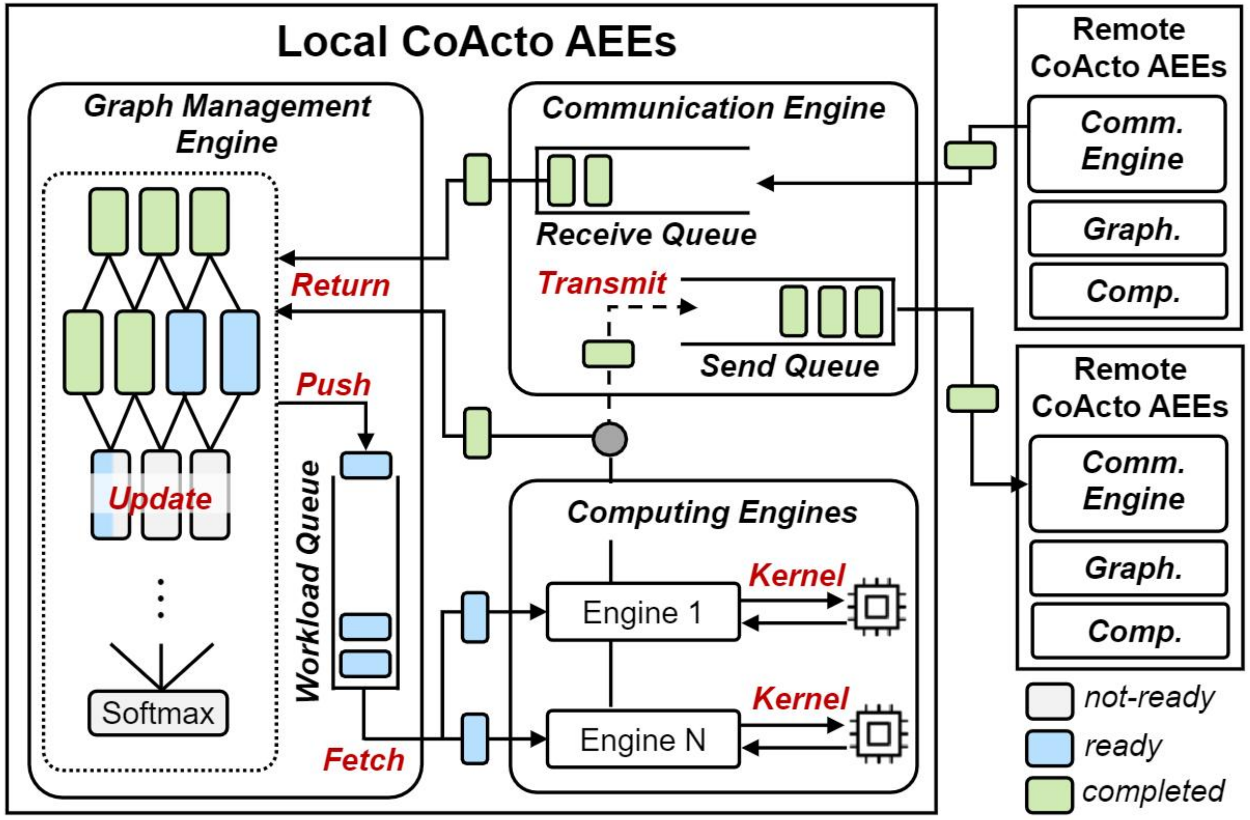


Figure 6: The overall execution workflows of Asynchronous Execution Engines.

Computing & Communication Engines

All computing and communication engines operate concurrently and asynchronously without any synchronization

3 *CoActo* Design

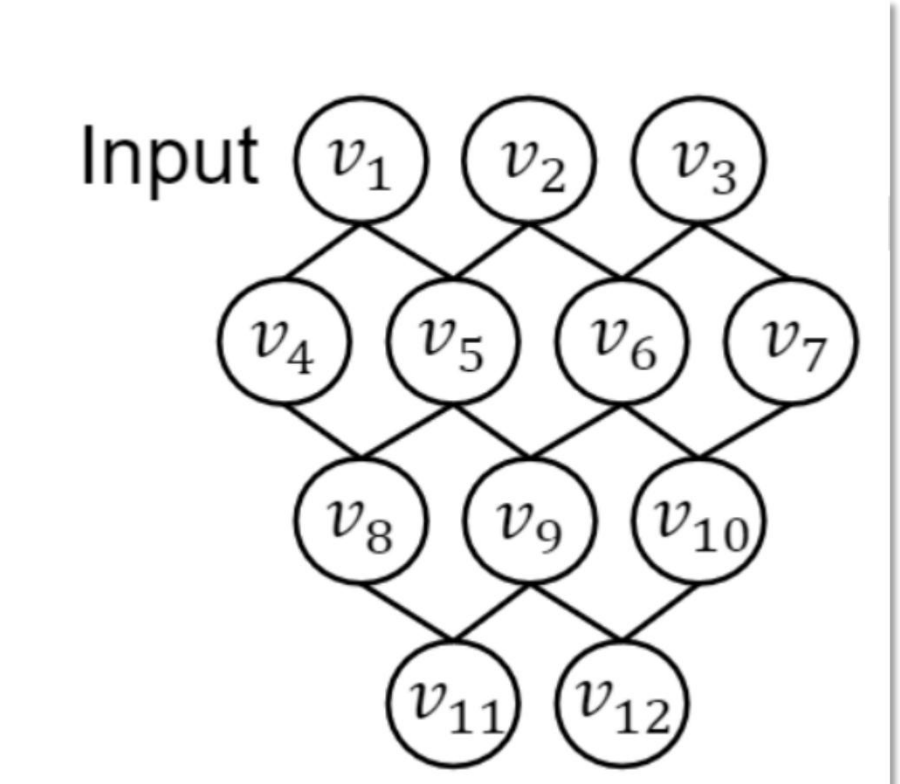
Dynamic Scheduler (DS)

With the tile-based partitioning technique, scheduling the tiles is interpreted as a complex DAG scheduling problem that is a well-known NP-complete problem

- Static DAG scheduling approaches is not a good choice for dynamic environments
  - unexpected network interference
  - an extremely burst request
- A dynamic offloading decision algorithm is better

3 *CoActo* Design

Dynamic Scheduler (DS)



(a) Sample DAG

Task model

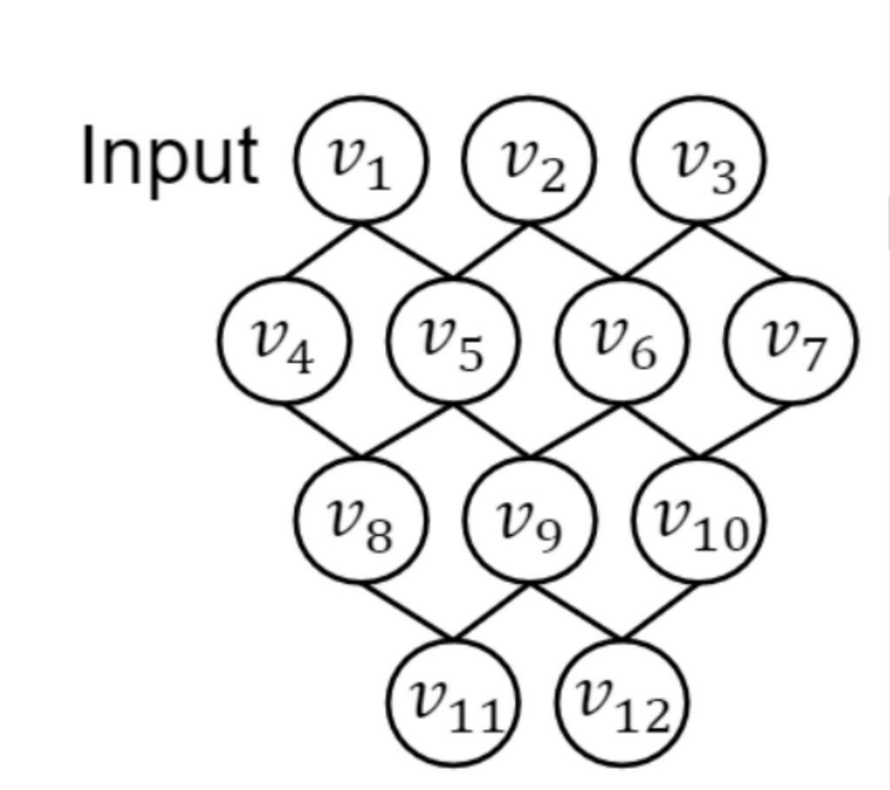
We define the partitioned computation graph as a DAG $G = <V, E>$

V: *nodes(tiles)*
E: *data dependency*
v point with no child is called v_{exit}

Each node v_i has a computation cost (FLOPs) of c_i, and output tile data size of d_i

3 *CoActo* Design

Dynamic Scheduler (DS)



(a) Sample DAG

Goal of the task

o_i denotes the offloading decision of a node v_i

$CT(v_i)$ represents the completion time of a node v_i

find the optimal offloading policy $O = \{o_1, \ldots, o_N\}$ that minimizes the maximum completion time of the exit nodes v_{exit}.

$$\min_{o_i \in O} \max CT(v_{exit}) \tag{1}$$

3 *CoActo* Design

Dynamic Scheduler (DS)

Basic idea

- first send all the input nodes to the server （sharing input data is cheap）
- dynamically decide to compute and send the outputs of the subsequent nodes on runtime

3 *CoActo* Design

Dynamic Scheduler (DS)

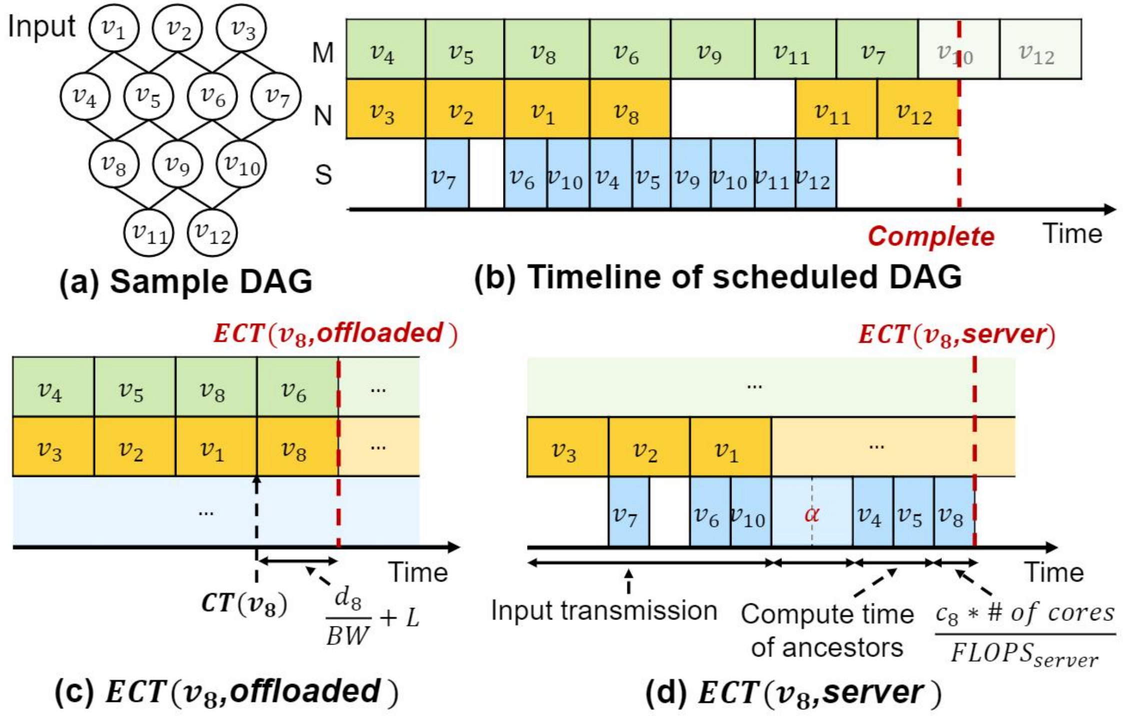


Figure 7: An example of dynamic offloading decision with (a) a sample DAG. It is performed by calculating $ECT(v_i, offloaded)$ and $ECT(v_i, server)$.

Dynamic offloading decision

- Mobile and server compute with the largest diameter in between to minimize duplicated computation
  - e.g. (b) mobile trans v3 while executing v4, server execute v7 after v3 arrives

3 *CoActo* Design

Dynamic Scheduler (DS)

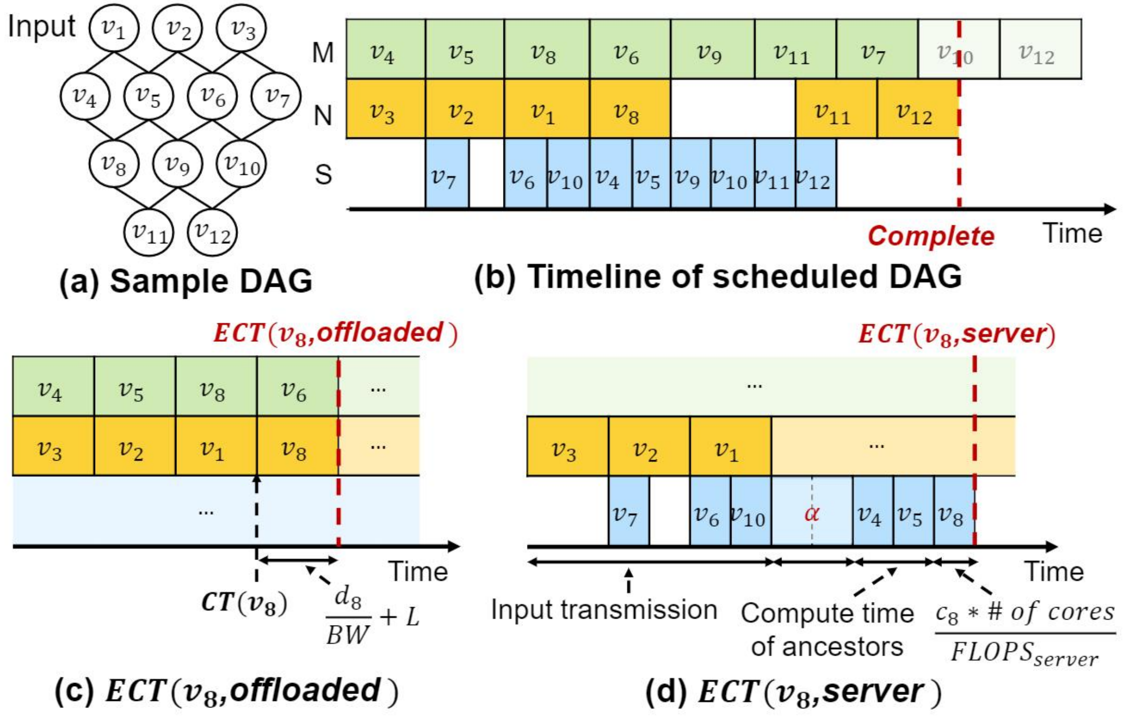


Figure 7: An example of dynamic offloading decision with (a) a sample DAG. It is performed by calculating $ECT(v_i$,*offloaded*$)$ and $ECT(v_i$,*server*$)$.

Dynamic offloading decision

- using a greedy approach in the mobile device
  - decision is made using the estimated completion times

$ECT(v_i$,offloaded$) < ECT(v_i$,server$)$
e.g v8

enables the maximal utilization of the powerful server resources

3 *CoActo* Design

Dynamic Scheduler (DS)

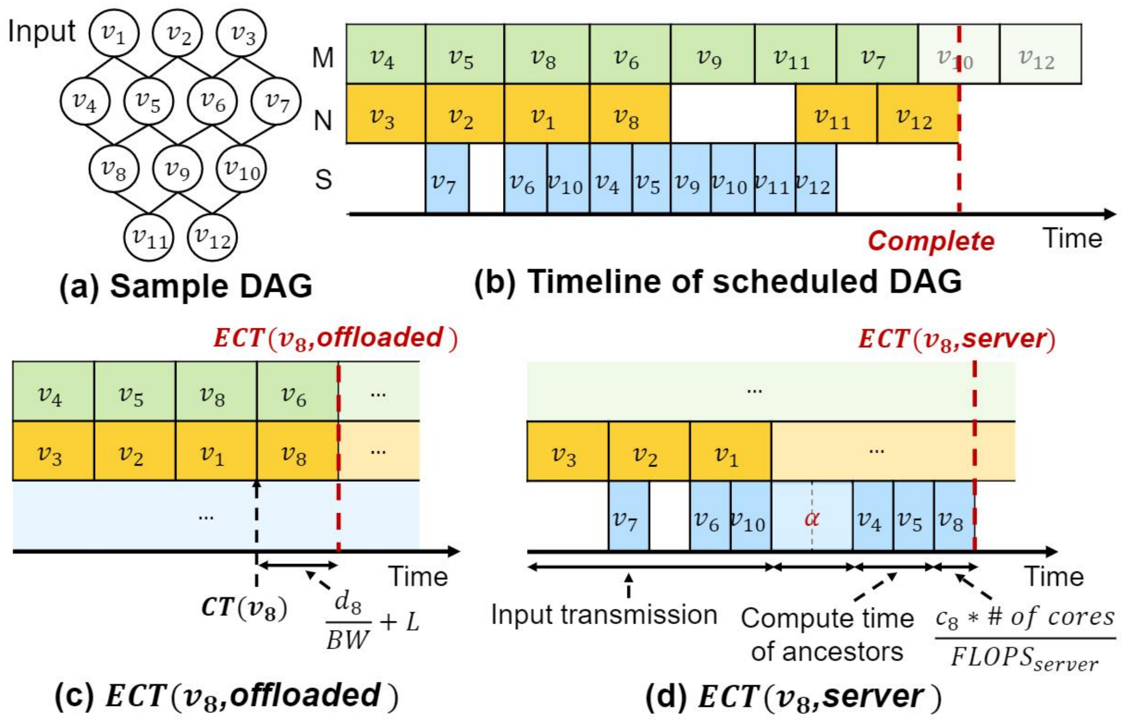


Figure 7: An example of dynamic offloading decision with (a) a sample DAG. It is performed by calculating $ECT(v_i,offloaded)$ and $ECT(v_i,server)$.

Estimating the completion times

- $ECT(v_8$, offloaded)
  - the completion time $CT(v_8)$
  - the queuing latency
    - obtaioned by total data size of the nodes in the send queue of and the profiled bandwidth.
  - the transmission time

3 *CoActo* Design

Dynamic Scheduler (DS)

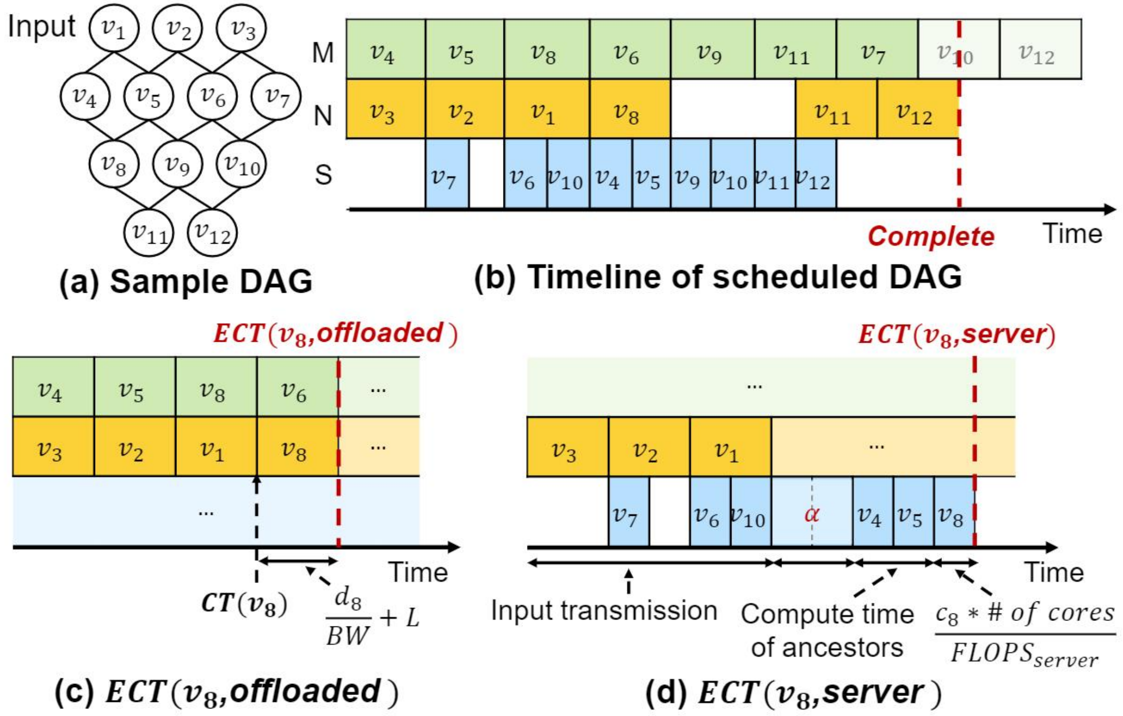


Figure 7: An example of dynamic offloading decision with (a) a sample DAG. It is performed by calculating $ECT(v_i, \textit{offloaded})$ and $ECT(v_i, \textit{server})$.

Estimating the completion times

- $ECT(v_8, \text{server})$
  - the transmission time of all input nodes
  - delayed time α_k of the mobile device k by the resource contention (dynamic)
  - the computation time of ancestors of v_8
  - the computation time of v_8

4 Evaluation

Experimental Setup

Platform	CPU	Memory
Server	64 Cores AMD Threadripper 3990X	128GB
Jetson AGX Xavier	8 Cores Carmel ARMv8.2	32GB
Raspberry Pi 4	4 Cores ARM Cortex-A72 1.8GHz	8GB
Pixel 5	1 Core ARM Cortex A-76 2.4GHz 1 Core ARM Cortex A-76 2.2GHz 6 Cores ARM Cortex A-55	8GB

Table 1: Specifications of the tested platforms.

4 Evaluation

Baselines

- Cloud-only: A status-quo approach that offloads whole DNN inference workloads to the cloud server by transmitting the input data.
- On-device: An approach that executes local inference on mobile platforms without offloading
- *SPINN:* The state-of-the-art split computing in collaborative inference
- *FL-offloading:* Fused-Layer (FL)-based collaborative inference approach

4 Evaluation

End-to-End Latency

Effectiveness in computation bottleneck

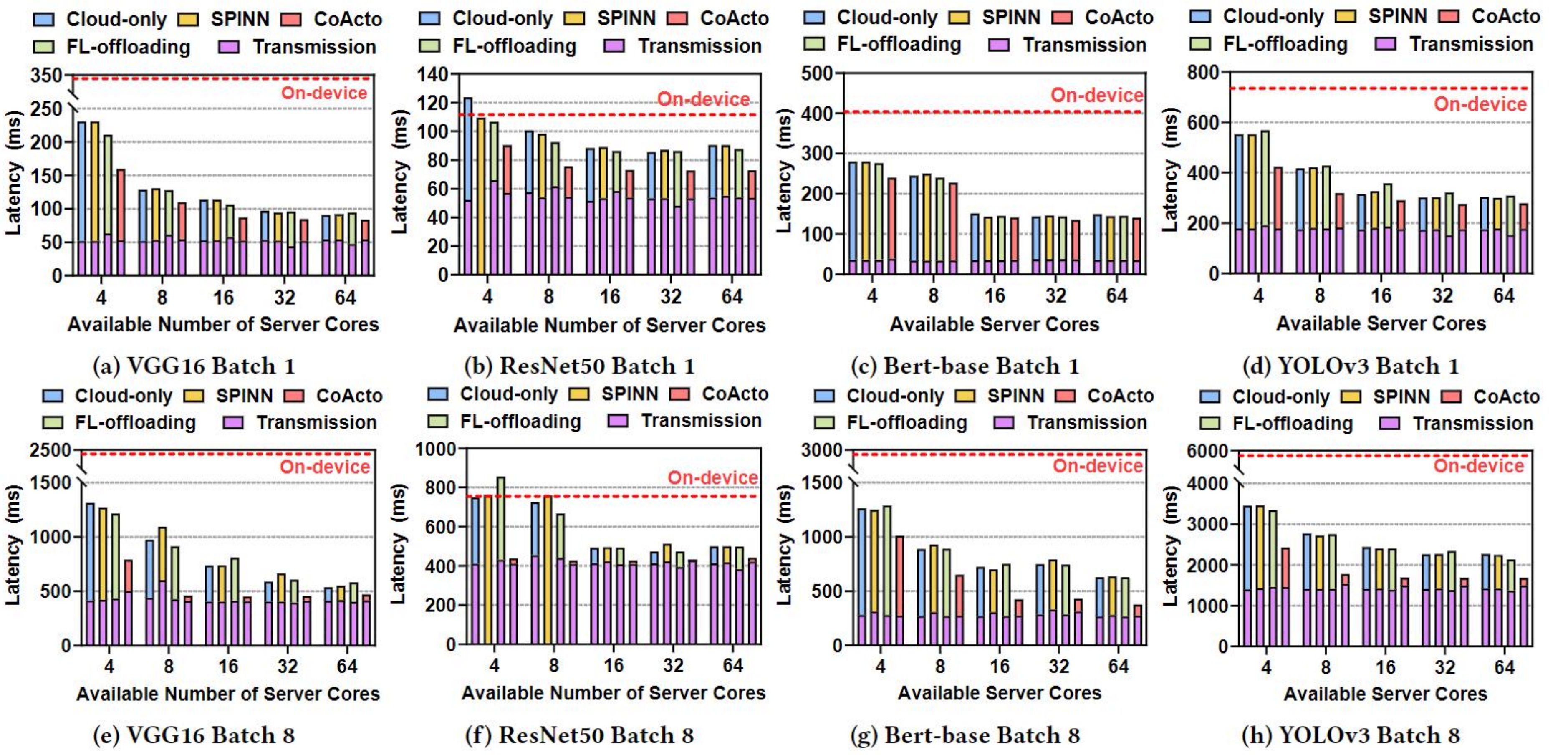
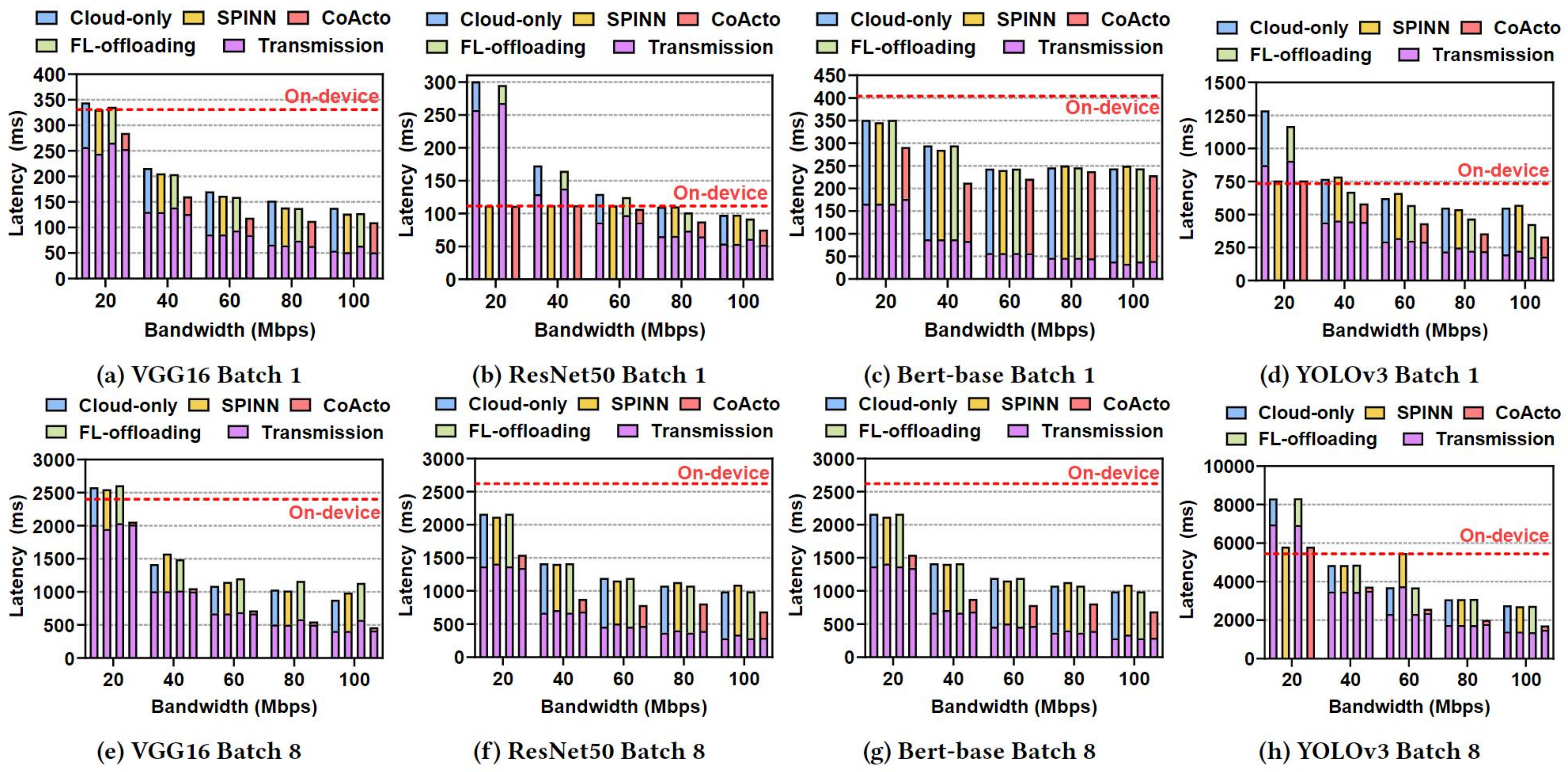


Figure 8: End-to-end latency using Jetson AGX Xavier with different number of available server cores, under a 100Mbps WiFi network.

4 Evaluation

End-to-End Latency

Effectiveness in network bottleneck



(a) VGG16 Batch 1 (b) ResNet50 Batch 1 (c) Bert-base Batch 1 (d) YOLOv3 Batch 1

(e) VGG16 Batch 8 (f) ResNet50 Batch 8 (g) Bert-base Batch 8 (h) YOLOv3 Batch 8

Figure 9: End-to-end latency using Jetson AGX Xavier under different network bandwidths and 8 cores available in the server.

4 Evaluation

Concurrency in Multi-tenant Scenario

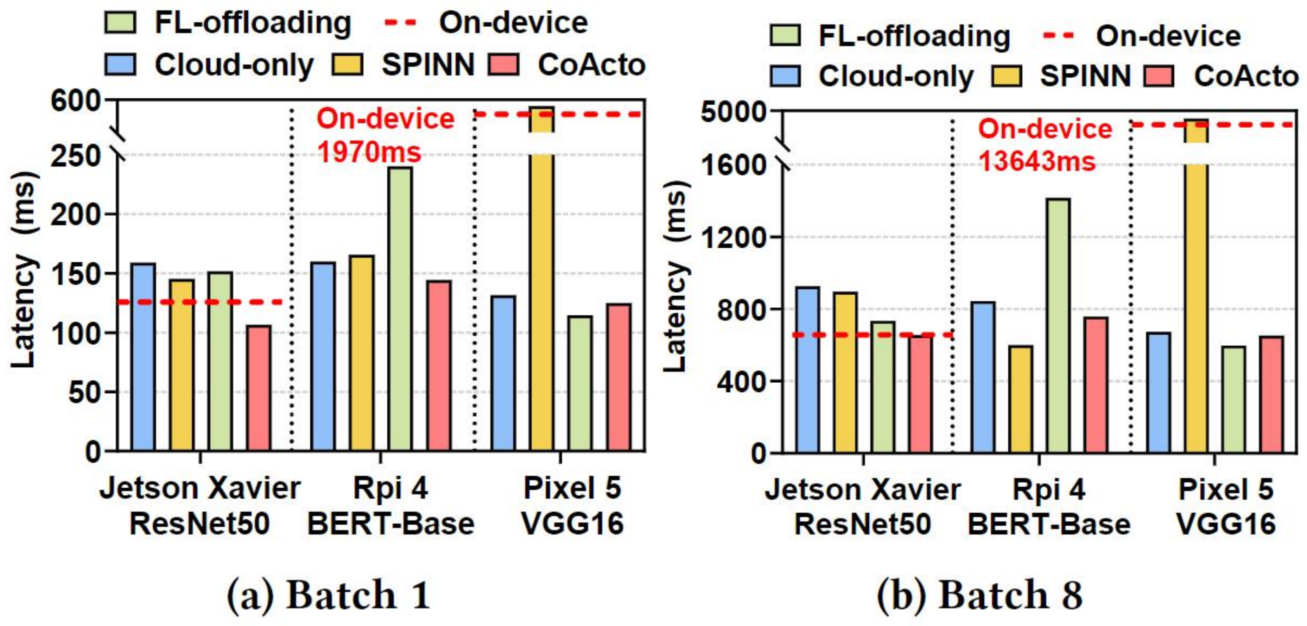


Figure 11: End-to-end latency in multi-DNN inference scenario that each device requests a distinct DNN inference query to the shared server with 100Mbps and 64 available cores settings.

4 Evaluation

Effectiveness of the Granularity

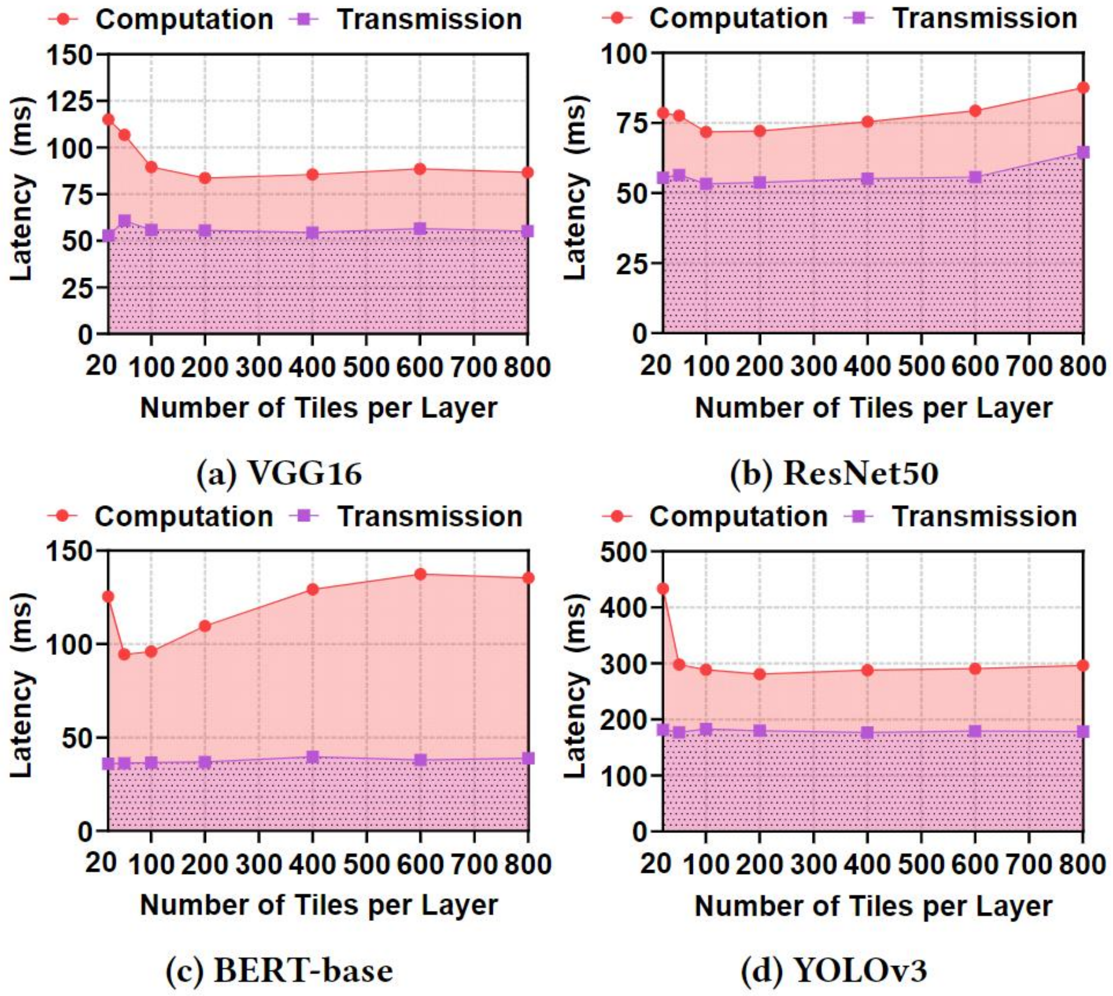


Figure 12: End-to-End latency of CoActo with different number of tiles per layer in 100MBps and 64 cores in the server. Note that the batch size is 1 for all the tested DNNs.

5 Thoughs

strengths

- optimize the mobile-server collaborative inference from the system view
- propose a fined-gained DNN expression tiling during execution
- support dynamic scheduling and concurrency of all the runtime resources

Weakness

- only support CPU inference now
    - can extend to GPU inference by implementating the tile-based computation kernels
- the system is less effective when the network is bad or when there is a large difference between the network transmission speed and the inference speed

Thank You.

2024.5.17