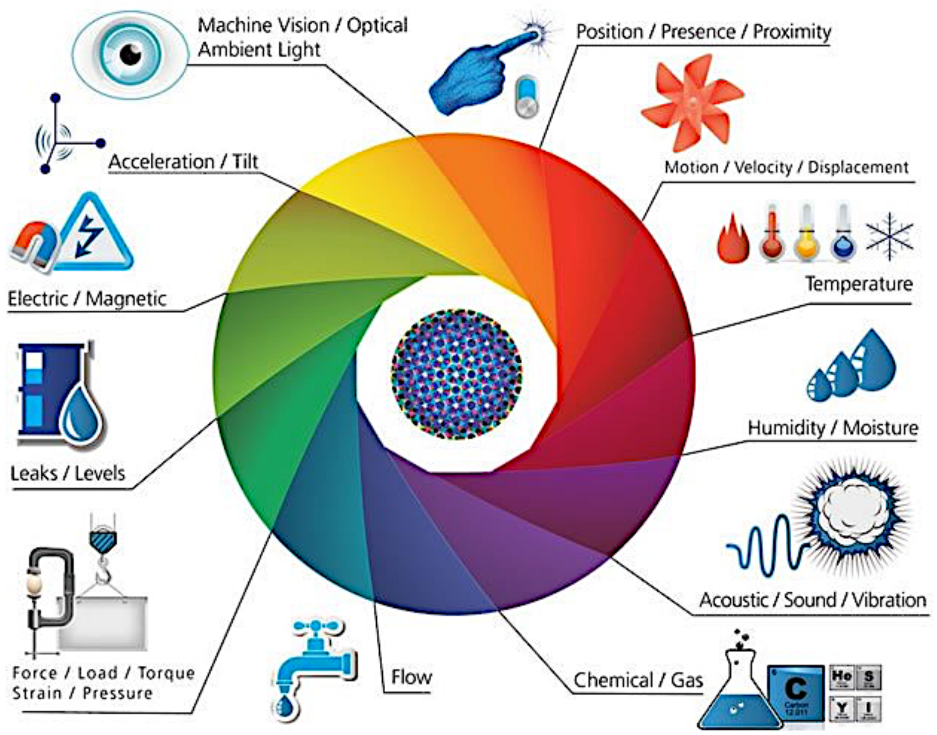
Internet of Things & Edge Devices

- Have access to an abundance of raw data
- In home, work, or vehicle

IoT & Edge: Raw Data & Processing

- Are gaining ground with the widespread of
  - Embedded processors
  - Ubiquitous wireless networks
- Access to raw data
  - Need to understand it
  - Have real-time constraints
  - Have limited resources
    - Power
    - Computation



IoT & Edge: DNN-based Processing

- With deep neural networks (**DNNs**):
  - IoT & Edge devices can
    - Process several new data types and
    - Understand behaviors
  - Examples: Speech, vision, video, and text

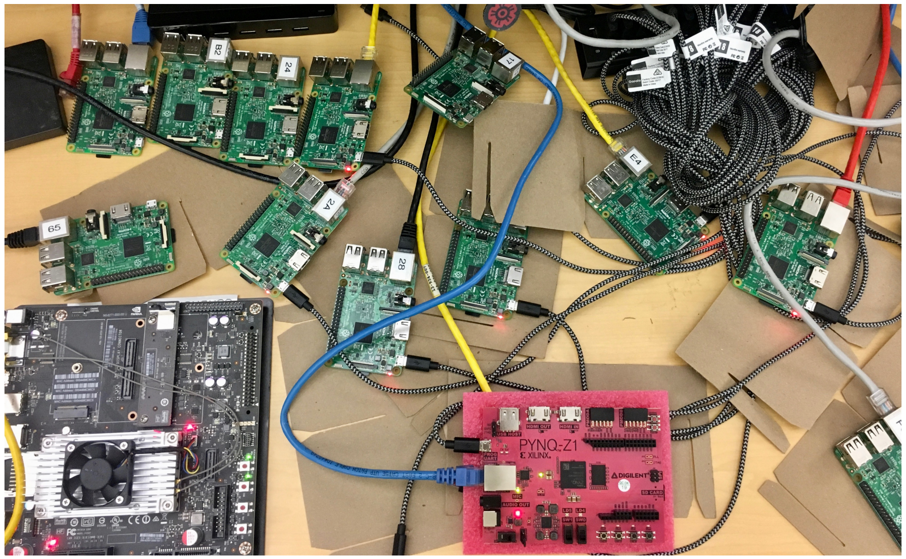
- But, DNNs are resource hungry
  - Cannot meet real-time constraints on IoT devices
  - Several DNNs cannot be executed on IoTs

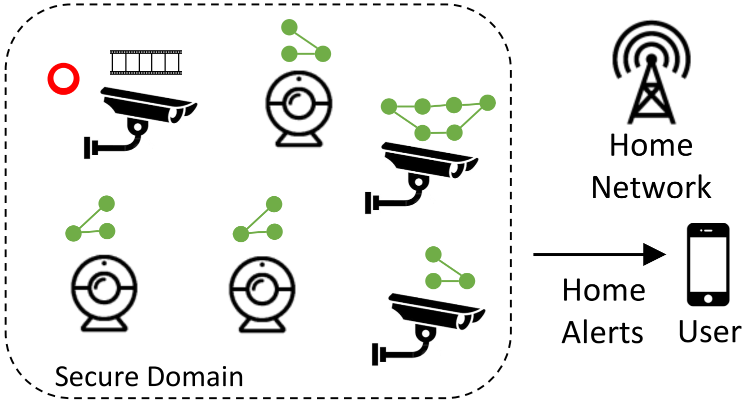
Approach 1: Offloading to Cloud

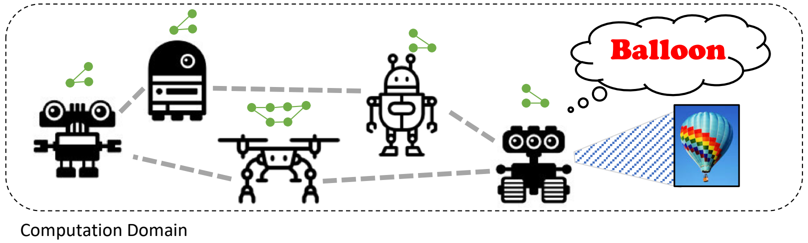
- **Why Cloud is Not Always a Good Solution?**
  - Connections to cloud are unreliable
  - Bandwidth is low and latency is high
  - Devices are not always connected
  - **Privacy**
    - User's data leaves the local network
    - Insecure connections and protocols threaten data
    - User loses ownership of data

Approach 2: In-The-Edge Collaboration

- Distributing computations with collaboration
  - To meet demands of DNNs
  - On top of common DNN techniques for constrained devices (e.g., pruning)





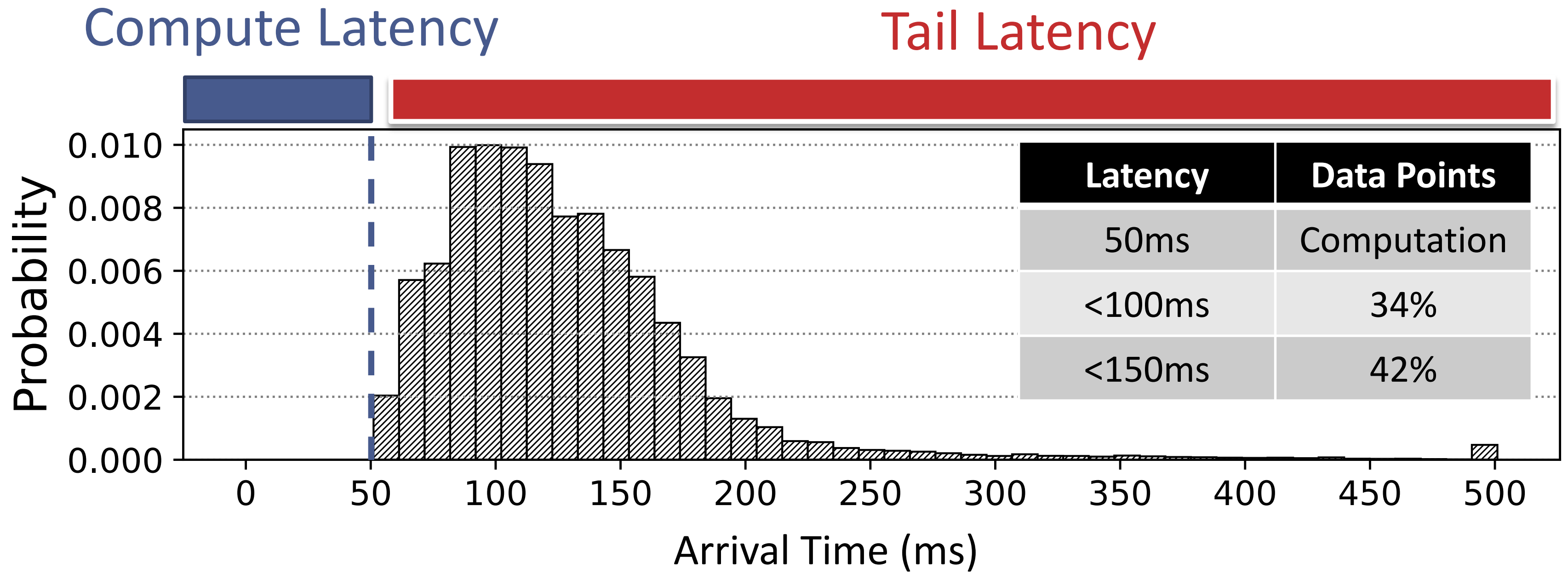


In-The-Edge Collaboration Pros & Cons

Pros	Cons
Does not Depend on Cloud	**Unreliable Latencies**
Preserves Privacy	**High Communication Overhead due to Model Interconnectivity**
Enables Personalized Insight	**Accuracy Drop due to Data Loss & Device Failures**

Unreliable Latency & Interconnected Models

- The histogram of arrival times in a 4-node system of RPis performing AlexNet (model parallelism).

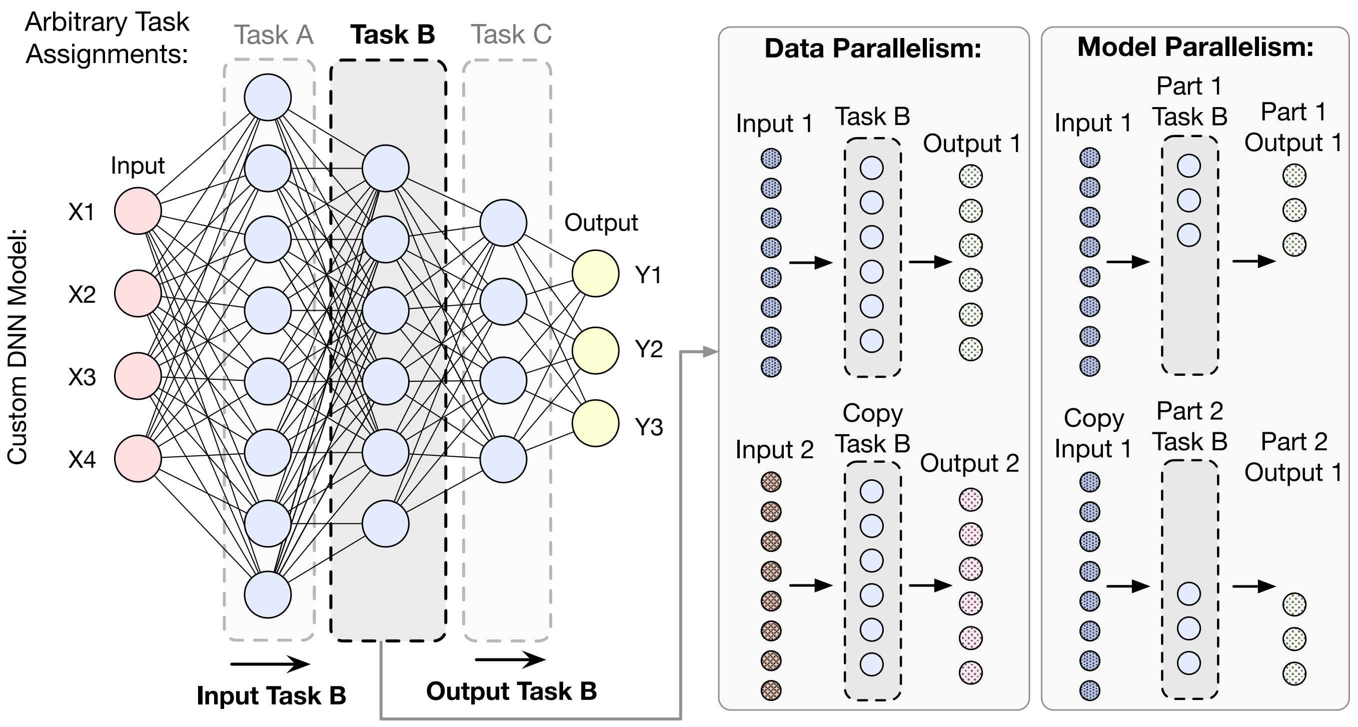


Latency	Data Points
50ms	Computation
<100ms	34%
<150ms	42%

- **Long Tail and Max Latency -> Straggler Problem**

Reason: Highly Interconnected Models

- Highly interconnected DNN models are not easy to distribute with common distribution methods:

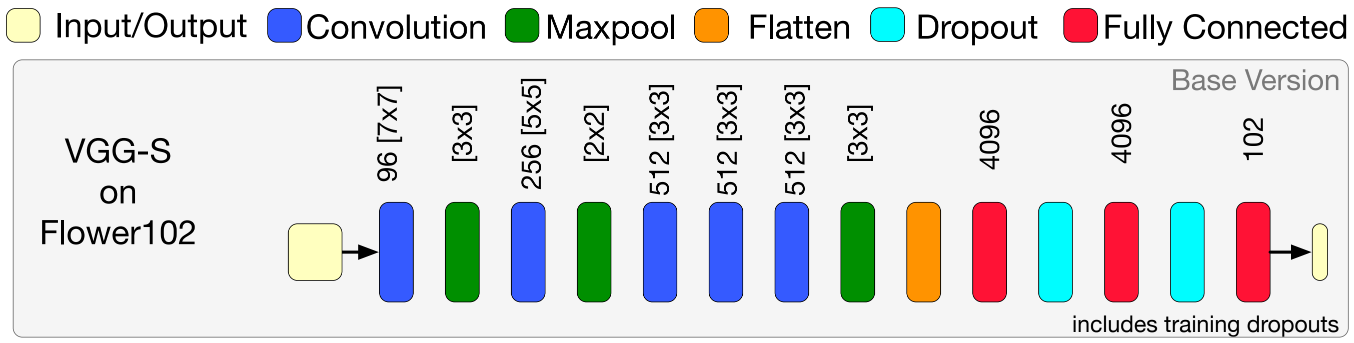


Model Parallelism: Needs several inputs

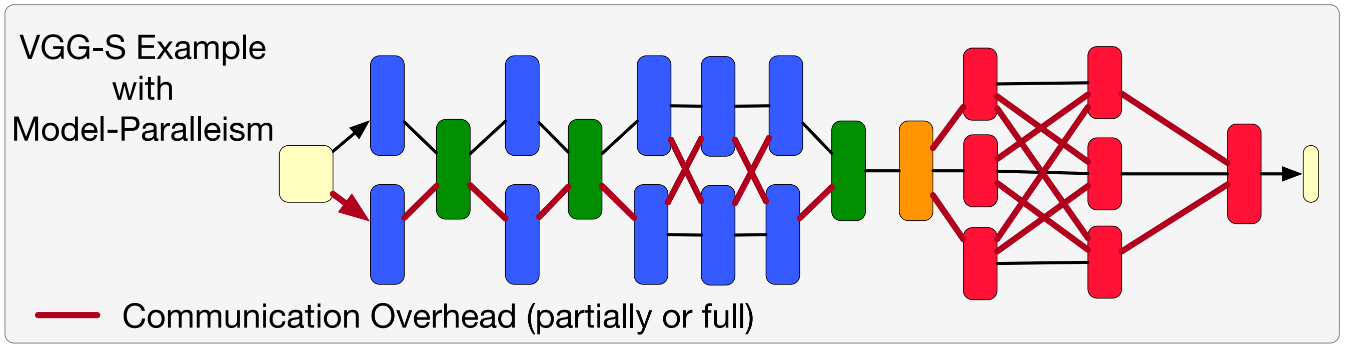
Data Parallelism: Creates several connections to aggregate/collect results

Reason: Highly Interconnected Models

- Highly interconnected DNN models are not designed for efficient distribution and low communication



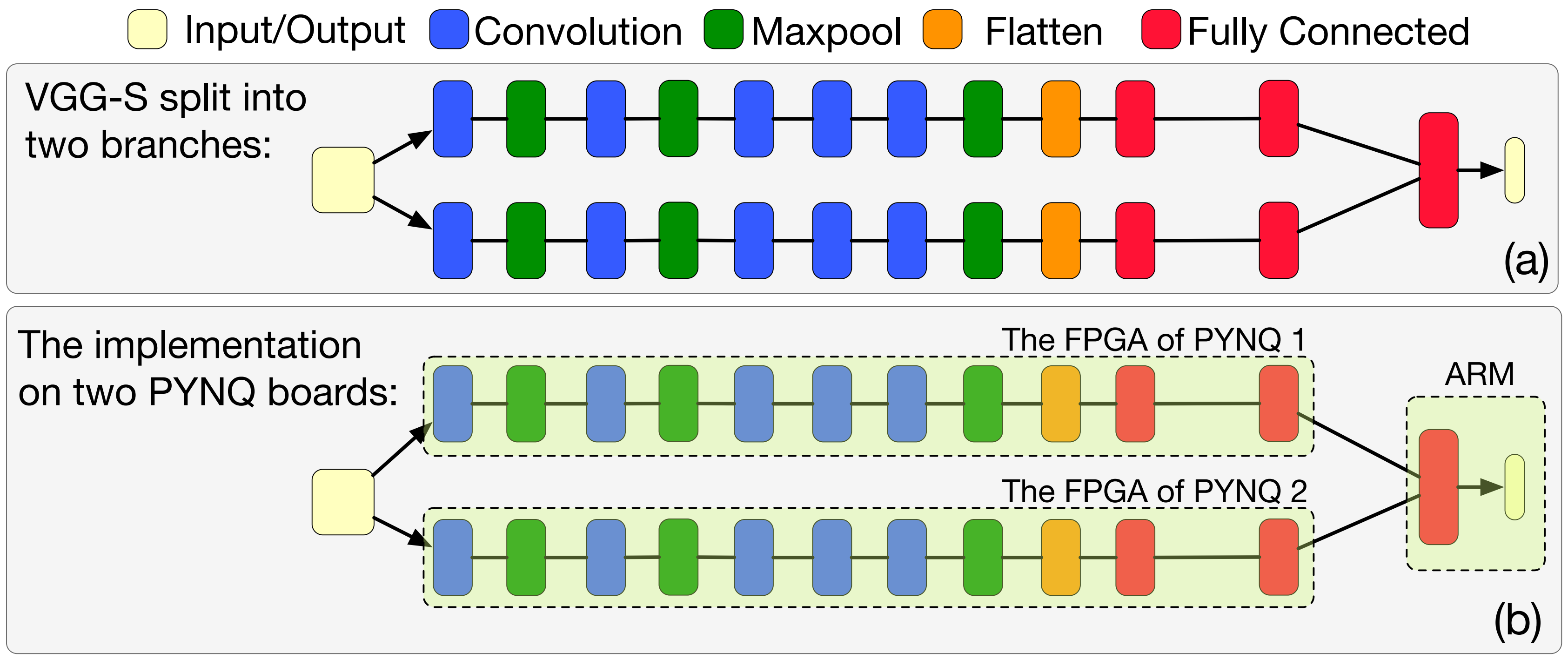
(a)


VGG-S Example with Model-Paralleism
Communication Overhead (partially or full)

(b)

Solution: Design New DNN Models

- Design new DNN models that are efficient with distribution and has low communication



Our FPGA Implementation

- Using PYNQ boards
- ResNet18 model: Divided in two branches
  with less than 5% accuracy loss
  2.4x higher throughput
- Using TVM/VTA* stack for FPGA implementation
- Models build with MXNet Gluon
- RPC server-client model for communication
- Code available on GitHub

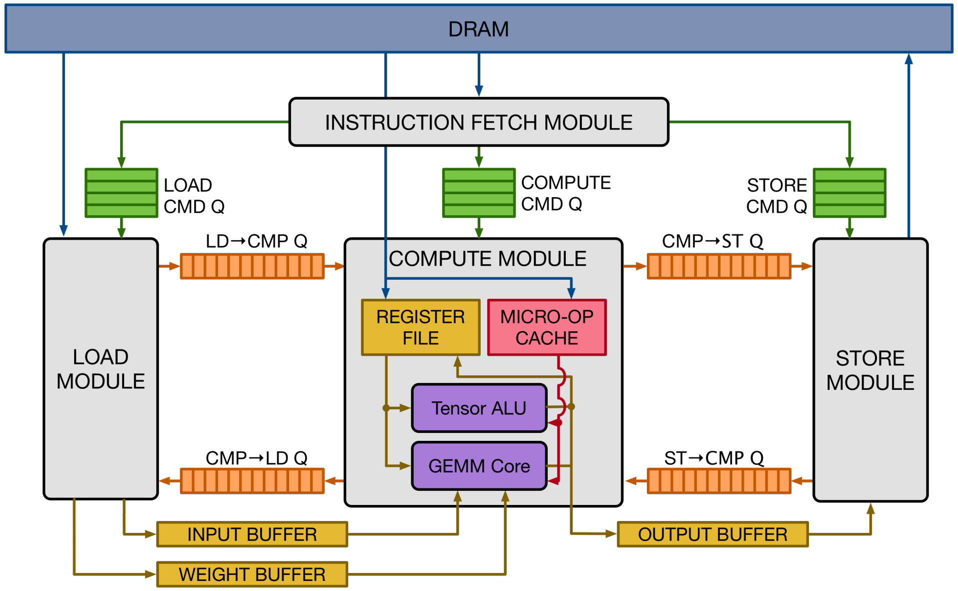
FPL Demo Website:
http://comparch.gatech.edu/hparch/fpl19

SCAN ME

*Chen, Tianqi, et al. "TVM: end-to-end optimization stack for deep learning." arXiv preprint arXiv:1802.04799 (2018): 1-15.

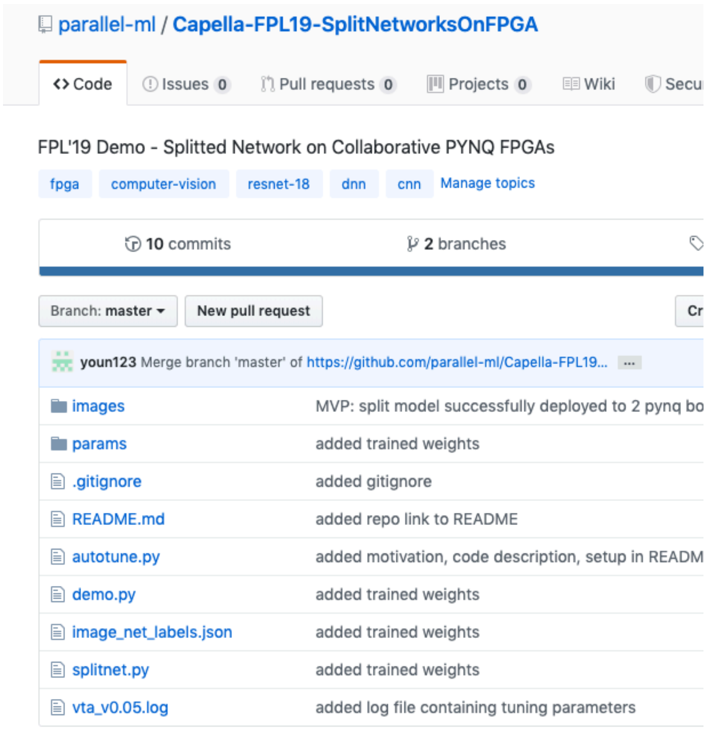
TVM/VTA Stack

- Vanilla Tensor Accelerator (VTA) is a generic deep learning accelerator built around a GEMM core
- VTA provides design and JIT runtime compatible with TVM stack
- VTA integrates a RISC-like processor for dense algebra that works on tensor registers
- design adopts decoupled access-execute to hide memory access latency.
- design adopts decoupled access-execute to hide memory access latency.
- More at: docs.tvm.ai/vta



Source Code Available on Github

- We released documented source code on Github
- We include our new DNN model description, training procedure, importing to VTA, and live camera demo files



parallel-ml / Capella-FPL19-SplitNetworksOnFPGA

<> Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Secu

FPL'19 Demo - Splitted Network on Collaborative PYNQ FPGAs

fpga | computer-vision | resnet-18 | dnn | cnn | Manage topics

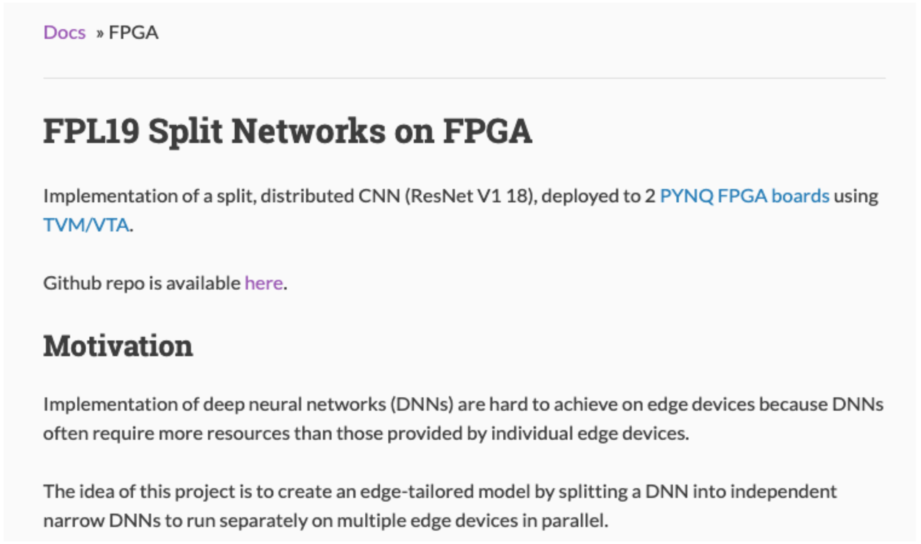
10 commits | 2 branches

Branch: master | New pull request | Cr

youn123 Merge branch 'master' of https://github.com/parallel-ml/Capella-FPL19...

images	MVP: split model successfully deployed to 2 pynq bo
params	added trained weights
.gitignore	added gitignore
README.md	added repo link to README
autotune.py	added motivation, code description, setup in READM
demo.py	added trained weights
image_net_labels.json	added trained weights
splitnet.py	added trained weights
vta_v0.05.log	added log file containing tuning parameters

- Accessible through demo website



Docs » FPGA

FPL19 Split Networks on FPGA

Implementation of a split, distributed CNN (ResNet V1 18), deployed to 2 PYNQ FPGA boards using TVM/VTA.

Github repo is available here.

Motivation

Implementation of deep neural networks (DNNs) are hard to achieve on edge devices because DNNs often require more resources than those provided by individual edge devices.

The idea of this project is to create an edge-tailored model by splitting a DNN into independent narrow DNNs to run separately on multiple edge devices in parallel.

Our Experience

- PYNQ boards are unique in bridging between regular ARM cores with Linux tools & FPGAs
- TVM/VTA Stack:
  - Not every model currently complies to VTA compatible code. We had to change our new model parameters to match the hardware specification (MXNet Gloun)
  - VTA profiles lots of hardware parameters to tune. However, we were not able to use automated profiling tools
    - Example: A non-optimized hardware implementation is even slower than CPU-based implementations
  - Finding a suitable model and frontend that actually can run on the TVM VTA due to the shape constraints
  - Tight coupling of RPC server and compilation machine

*Increasing Reliability in Edge Systems

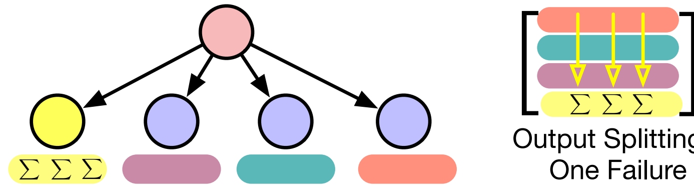
- DAC'19

  Robustly Executing DNNs in IoT Systems Using Coded Distributed Computing

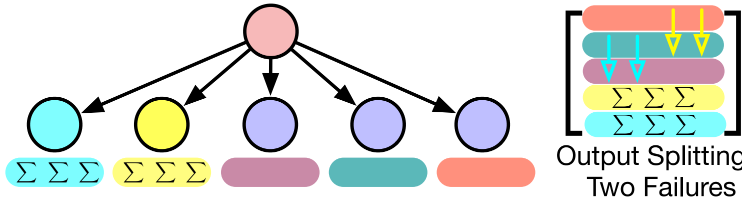
  - Using Coded Distributed Computing (CDC) to increase reliability

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{11}+w_{21} & w_{12}+w_{22} \end{bmatrix} \times \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_1+a_2 \end{bmatrix}$$

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w^{cdc}_{:1} & w^{cdc}_{:2} \end{bmatrix} \times \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a^{cdc} \end{bmatrix}$$



Output Splitting for 4 nodes
One Failure Tolerance



Output Splitting for 5 nodes
Two Failures Tolerance

DAC'19 Paper:
https://dl.acm.org/citation.cfm?id=3322474