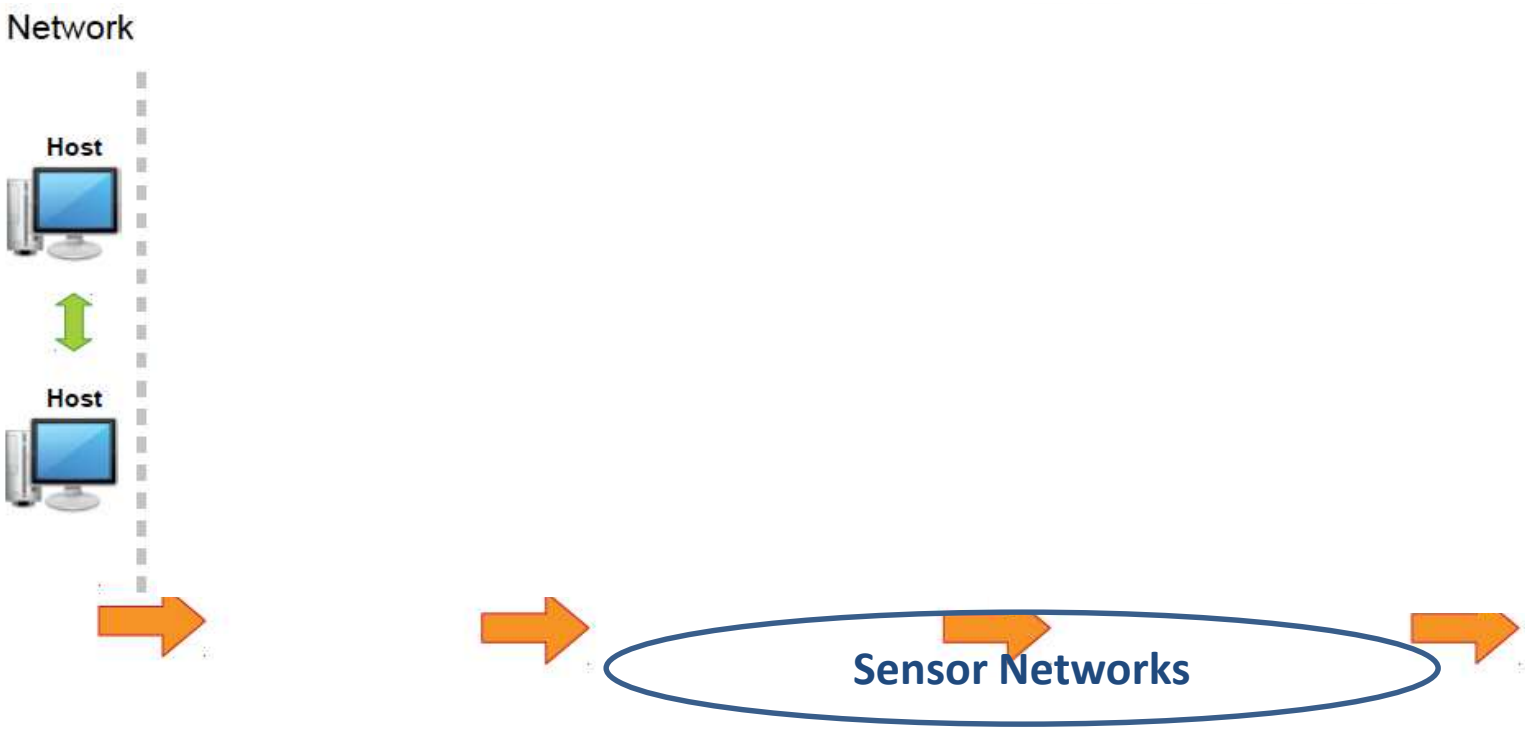
Accelerating IoT Systems & Resolving Reliability and Maintainability Challenges

Prof. Tajana Simunic Rosing

& SEE Lab Team @ UCSD

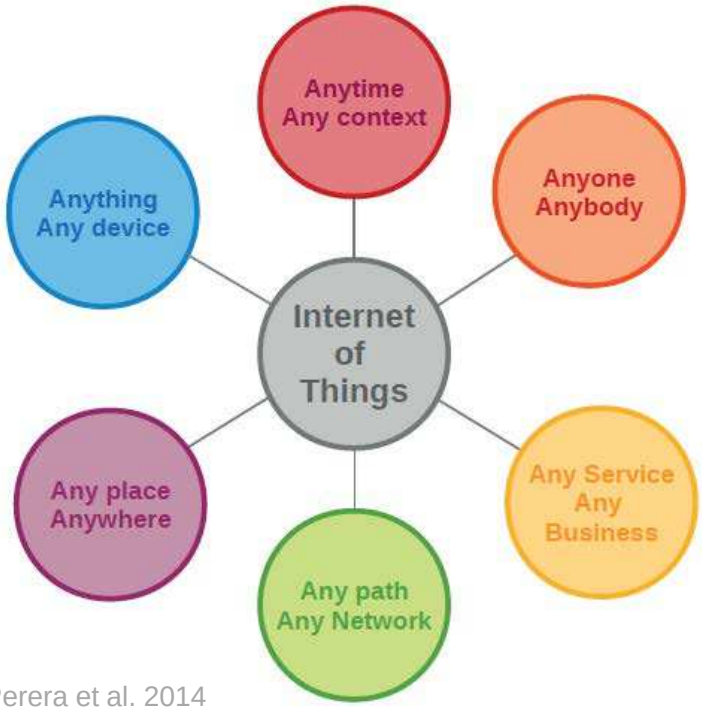
Internet Evolution



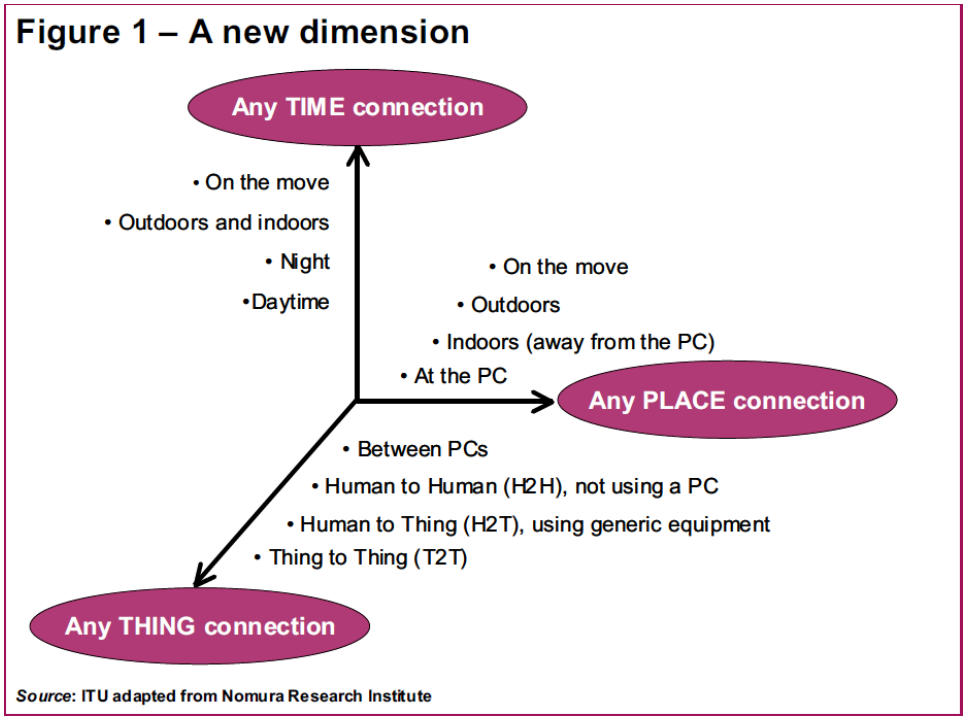
Starts with a network and evolves into *everything* that can be *connected with a network*

Source: Perera et al. 2014

What is IoT?



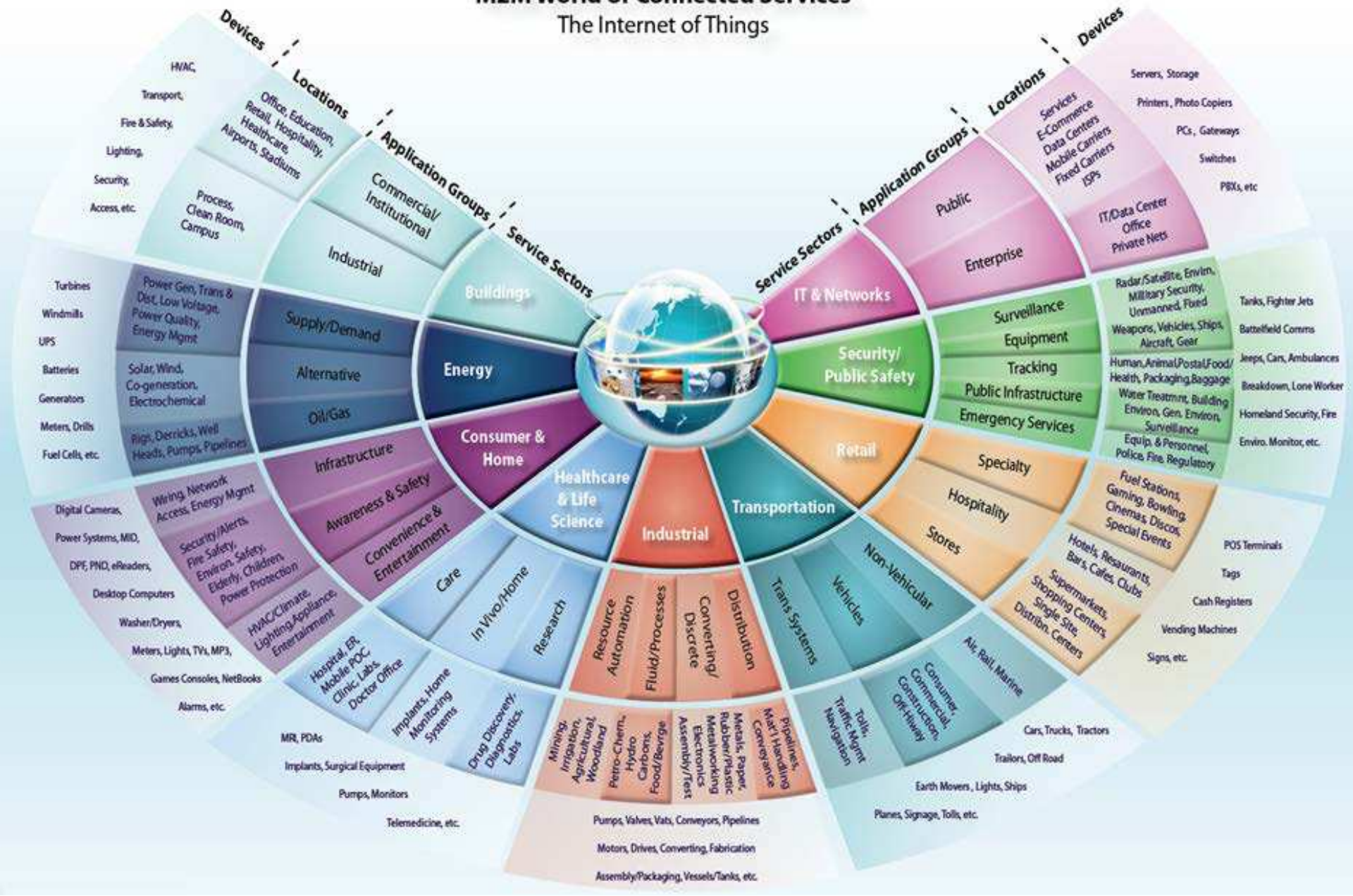
Source: Perera et al. 2014



The Internet of Things allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any service.

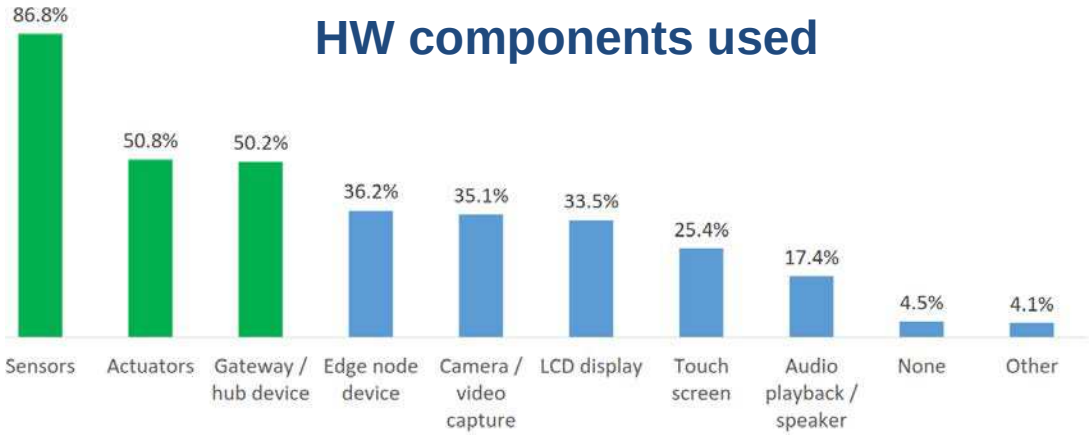
M2M World of Connected Services

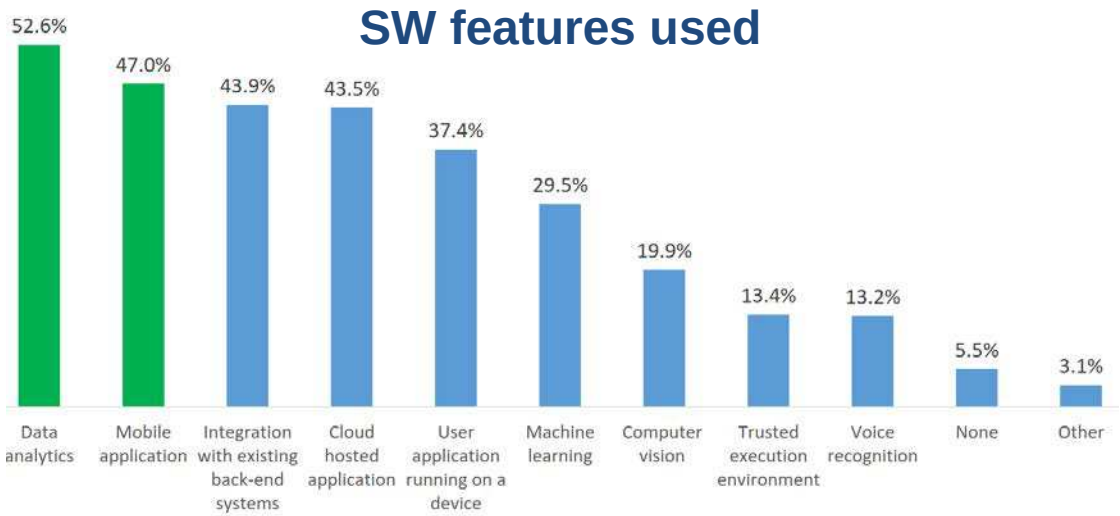
The Internet of Things



Source: Beecham Research

IoT HW & SW

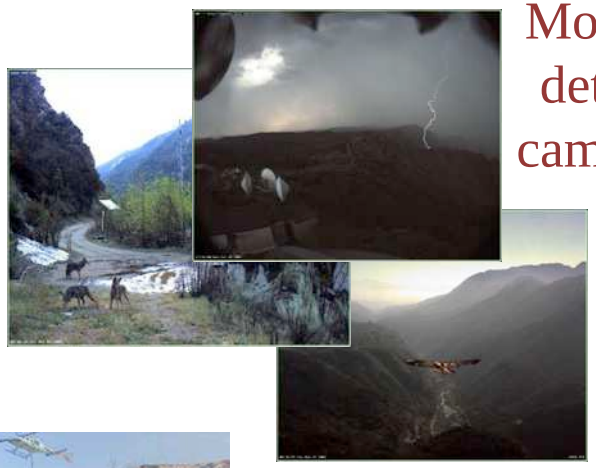




IoT Developer Survey 2017 - Copyright Eclipse Foundation, Inc.

Large Scale High Performance Wireless Research and Educational Network - HPWREN

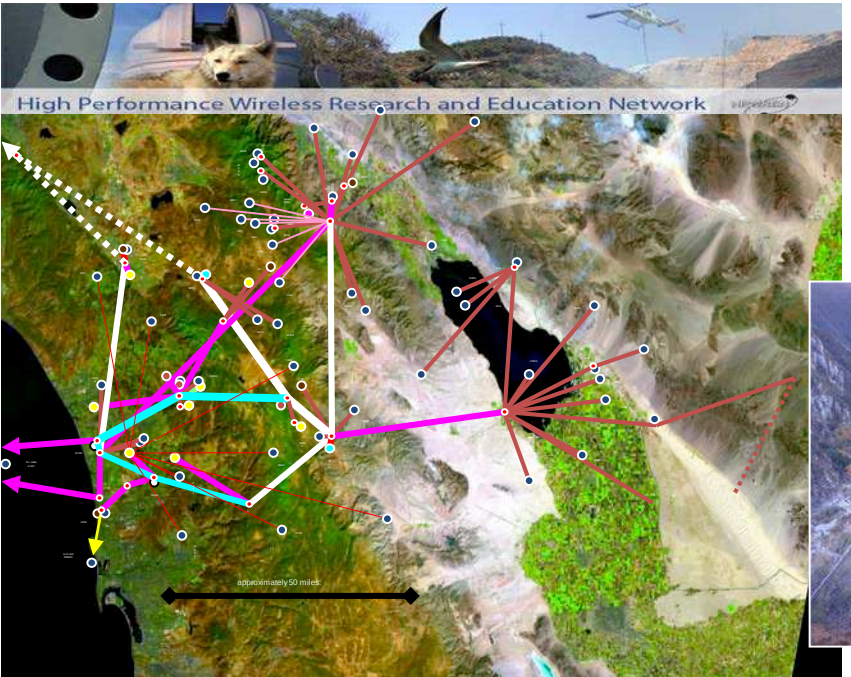
HPWREN wireless connectivity covers 20k sq. mile area with numerous sensors



Motion detect cameras

Wildfire tracking cams

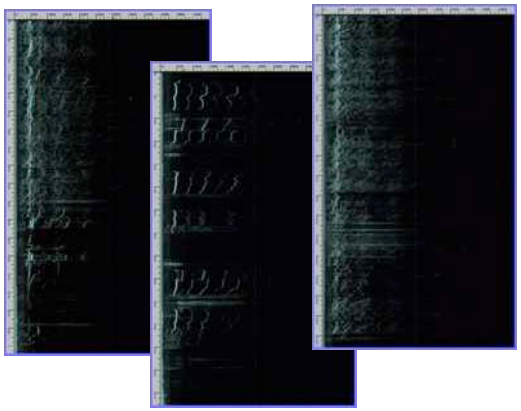
Cal Fire Dept.



Environmental sensors & cams

Acoustic sensors

Wolf howls at the CA Wolf Center

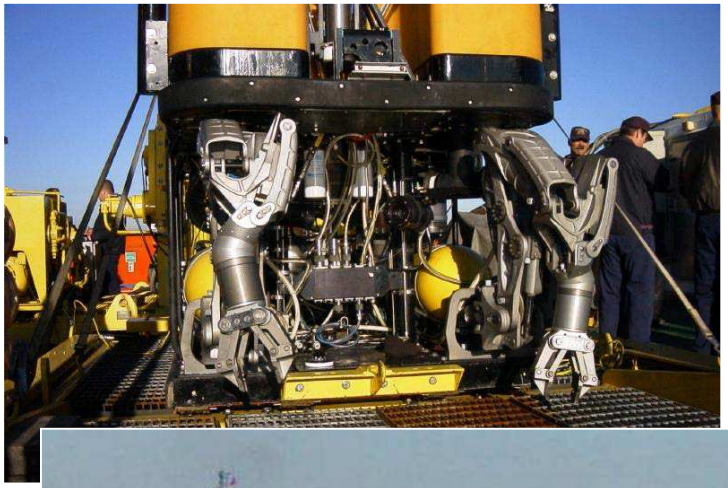


HW Braun, F. Vernon, T. S. Rosing, UCSD

Santa Margarita Ecological Reserve

75 Cluster Heads connected via WLAN

U.S. Navy Deep Submergence Unit

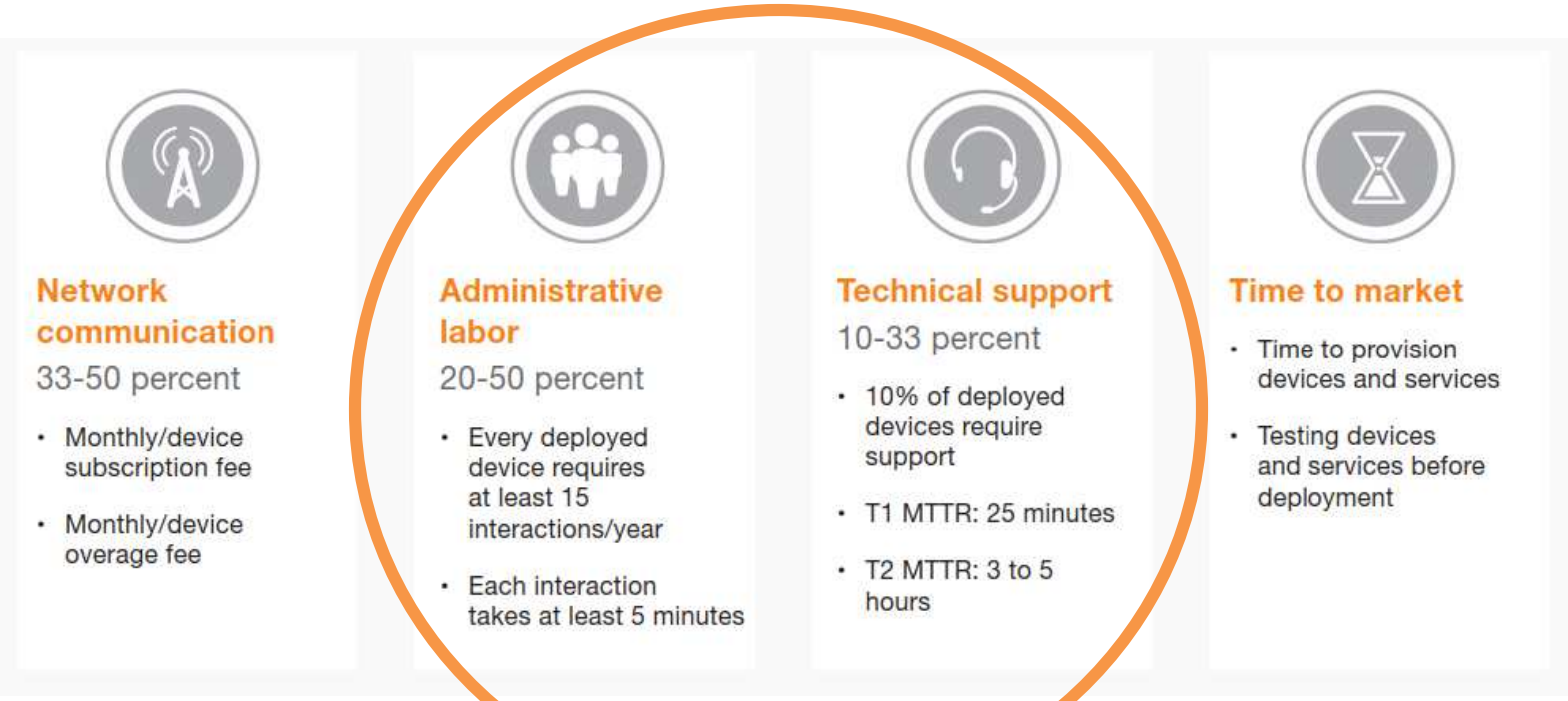


http://www.csp.navy.mil/csda5/dsu/dsu.htm

Volcan Fire HPWREN connection, September 2005

Hidden Costs of IoT

- Initial costs:
  - Design, HW sourcing and manufacturing, implementation
- Operational Expenses (OpEx) are recurring costs:



Source: Jasper, Cisco

Average Annual *Administrative* Costs for 100,000 Devices

- Data analysis of more 3500 companies worldwide
- Assumptions:
  - Average salary for professional operations administrator in the USA is $48/hr
  - Each interaction requires 5 minutes of labor
  - Number of interactions varies by industry:

Average Annual *Technical Support* Costs for 100,000 Devices

- Data analysis of more 3500 companies worldwide
- Assumptions:
  - Average tier 1 engineer salary is $20/hr, and call time is 25min
  - Average tier 2 engineer salary is $40/hr, and the mean time to resolution is 4hrs
  - 20% of support has to be escalated from Tier 1 to Tier 2
  - 10-30% of devices require technical support

Security & home automation

30% require technical support

Without platform:

$1.45M

- $250K for T1
- $1.2M for T2

Point-of-sales company

20% require technical support

Without platform:

$967K

- $167K for T1
- $800K for T2

Industrial equipment

25% require technical support

Without platform:

$1.2M

- $208K for T1
- $1M for T2

Transportation & logistics

10% require technical support

Without platform:

$483K

- $83K for T1
- $400K for T2

Sources of failure

- User errors
- Installation errors
- Communication problems
- Power issues (battery lifetime)
- Hardware failures
  - Soft errors: transient
  - Hard errors: permanent

Reliability, MTTF & Maintainability

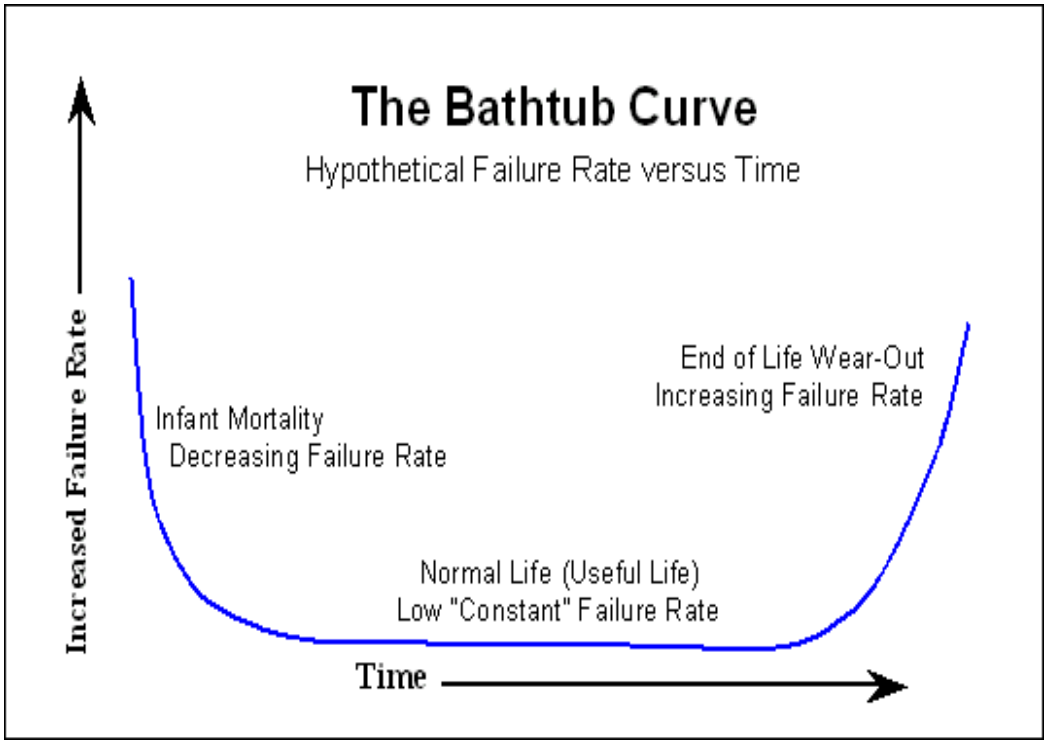
- Reliability R(t):
  - Probability function that a system will operate correctly in [0,t)

- Mean time to failure (MTTF): $\int_0^\infty R(t)dt$

- Maintainability: a measure of ability to restore a device to specified condition when maintenance is performed
  - If we maintain systems at an interval T, with the total number of maintenance events N, the reliability of the maintained system during the time $NT \leq t < (N+1)T$ is:

$$R_M(t) = R(T)^N R(t - NT)$$

Permanent Failures

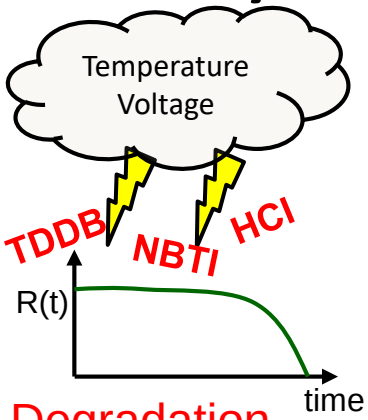


- Reliability during useful lifetime:
  - Modeled with exponential distribution
  - Constant failure rate (λ_f) :

$$R(t) = e^{-\lambda_f t}$$

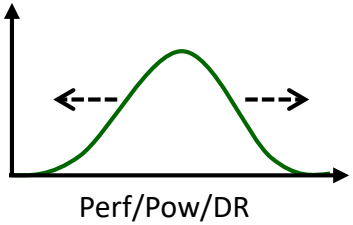
Power, Thermals, Reliability & Variability

Interrelated problems!



Degradation mechanisms

- Technology is rapidly scaling
- Accuracy of fabrication process is limited



Power

- Multiple power hungry subsystem
- Intense application requirements
- Limited battery capacity

Temperature

- Power dissipation increases temperature
- Big issue with no active cooling (fans)

Reliability

- Utilization and temperature stress reduces device lifetime

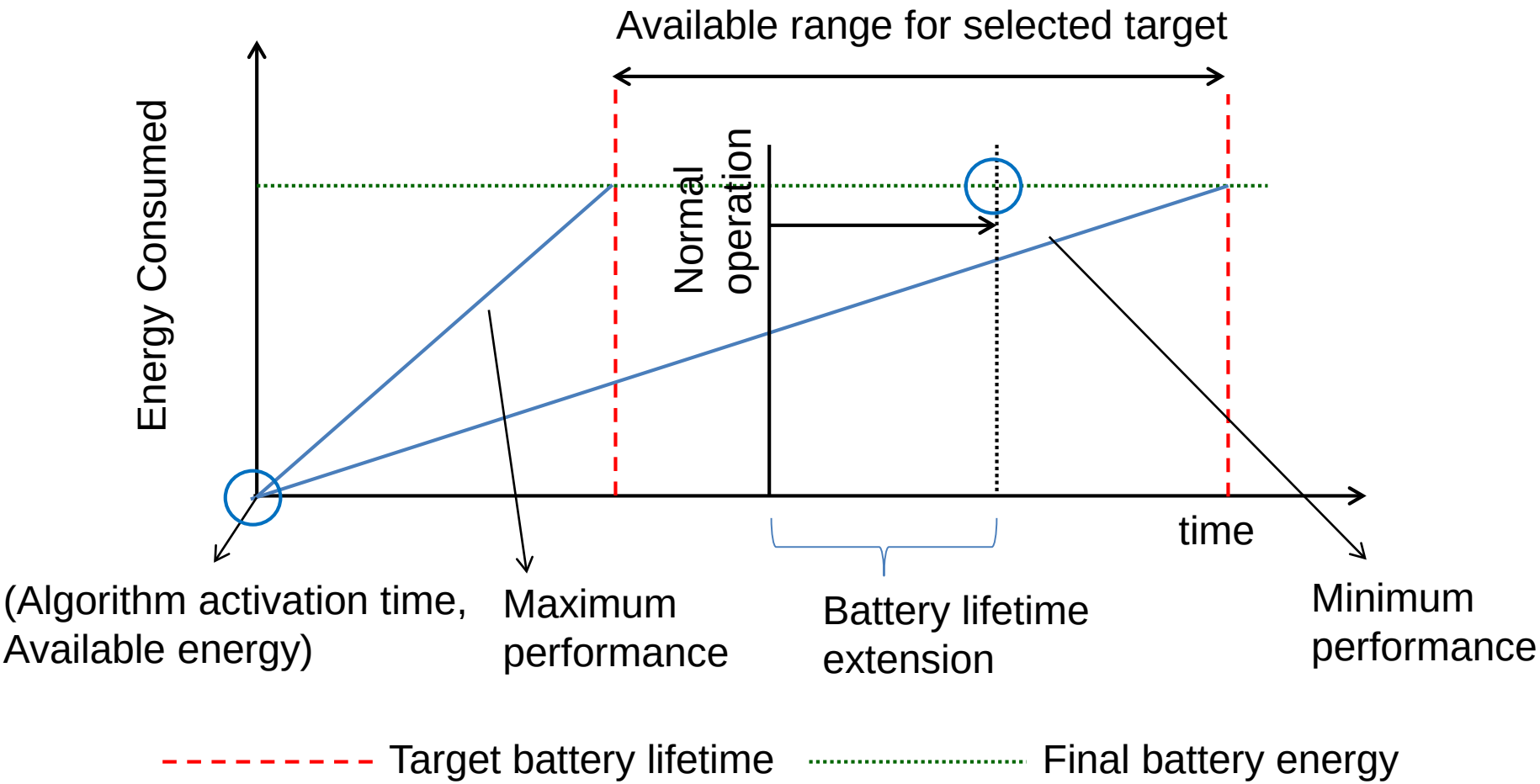
Variability

- Performance
- Power
- Degradation Rate

Performance-related metrics

Energy Optimization

Max(Perf) s.t. Target Battery Lifetime



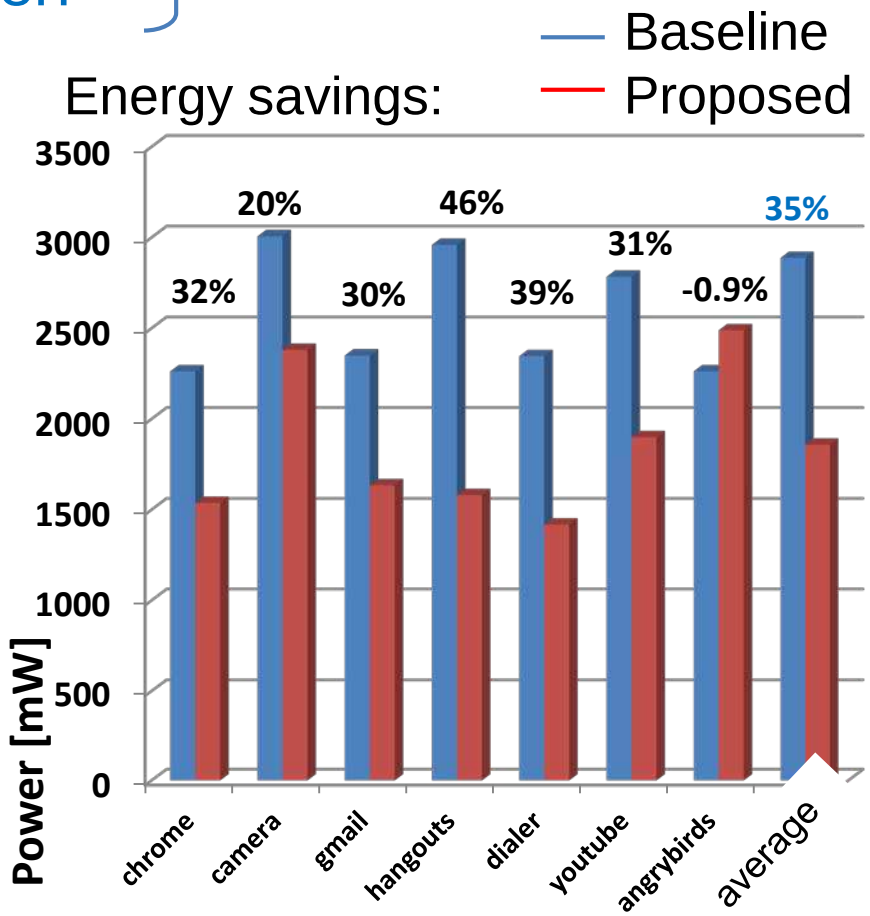
Integrating Temperature: Proactive Approach

Max(Perf) s.t. Target Battery Lifetime
or
Max(Battery Lifetime) s.t. App Perf

s.t. Temp < Threshold

Application	CPU Freq (MHz)	GPU Freq (MHz)	Display Brightness
Chrome	1000	200	150
Camera	1500	350	150
Gmail	800	200	70
Hangouts	900	200	100
Dialer	1300	200	55
Youtube	1500	450	150
Angrybirds	2260	450	255
Default	500	200	100

- Nexus 5 smartphone
- Profile of popular application
- Comparison between off-the-shelf baseline and proposed framework



Reliability Degradation due to Hot Spots

- Thermal hot spots [Failure Mechanisms for Semiconductor Devices, JEDEC]
  - **Electromigration:**
    - Transport of material caused by gradual movement of ions in the interconnect lattice
    - Eventually leads to opens and shorts
  - **Stress migration:**
    - Mechanical stress causes metal atoms in the interconnect to migrate
    - Similar failures as electromigration
  - **Time dependent dielectric breakdown:**
    - Result of the gate dielectric's gradual wear-out
    - Leads to transistor failure

$$\lambda \propto e^{\frac{-E_a}{kT}}$$

λ: Failure rate; T: temperature

E_a: Activation energy, k: Boltzman's constant

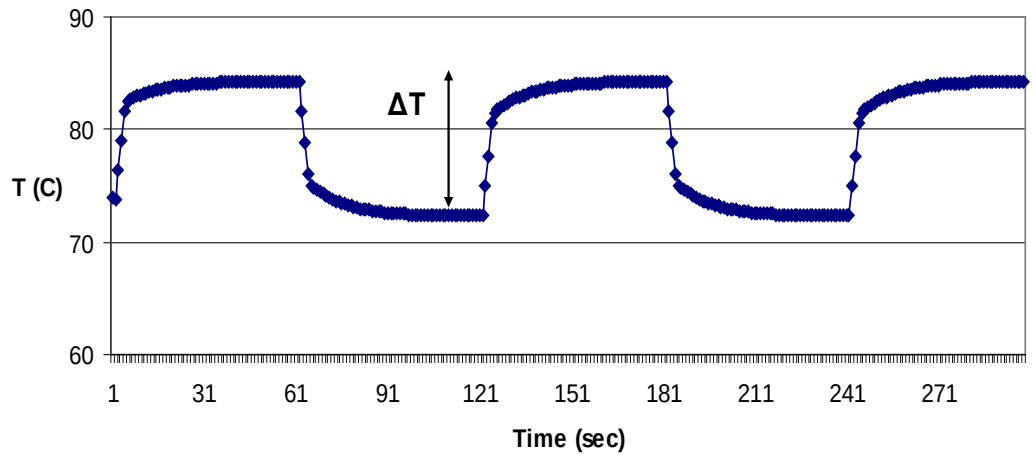
- Increasing temperature increases λ
- **10 – 15 C increase in temperature causes ~2X increase in failure rate**

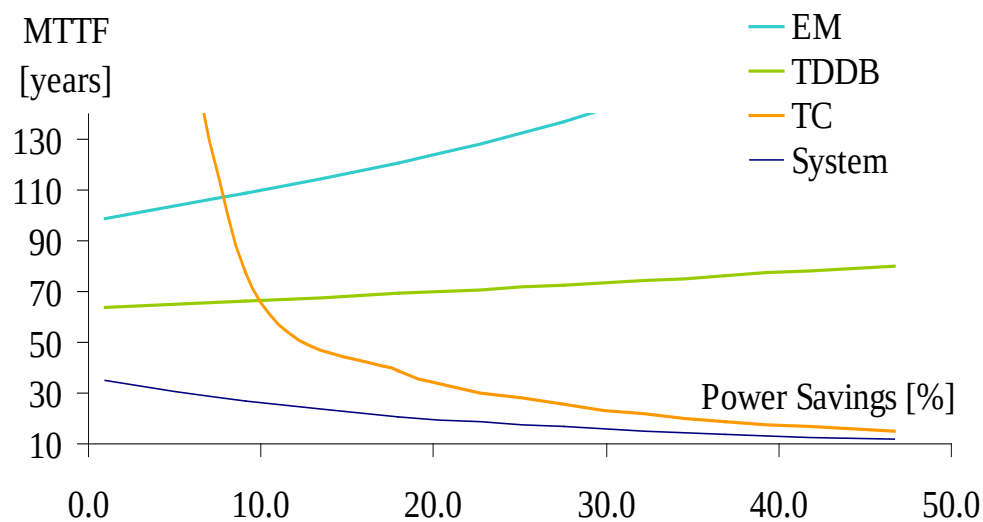
Reliability Degradation due to Thermal Cycles

- **Thermal cycling** [JEDEC]
  - Different expansion coefficients
  - Fatigue failures

$$\lambda \propto |\Delta T|^q \; f$$

  - Increases with:
    - ΔT : Magnitude of variation
    - f : Frequency of cycles
  - ΔT increases by 10°C (q=4 for metallic structures)
    - Failures happen 16 times sooner





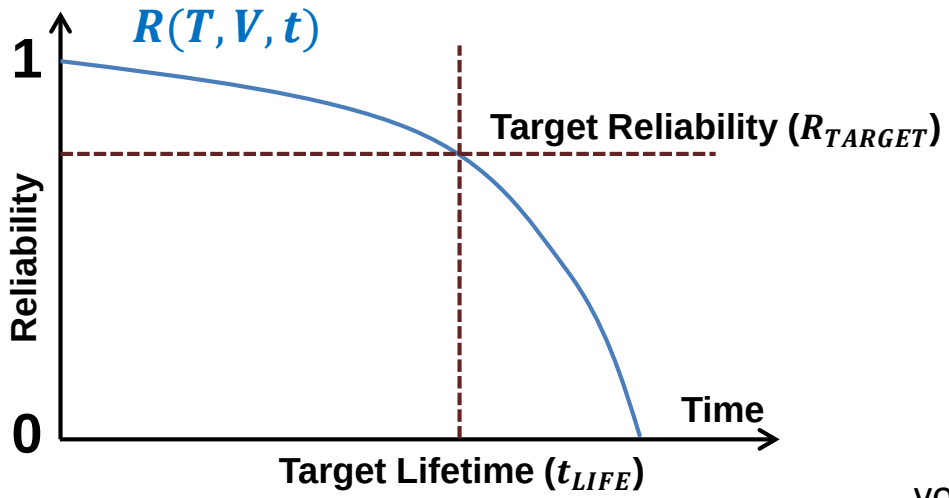
[Rosing et al., TVLSI'07]

Reliability Management

Borrowing strategy

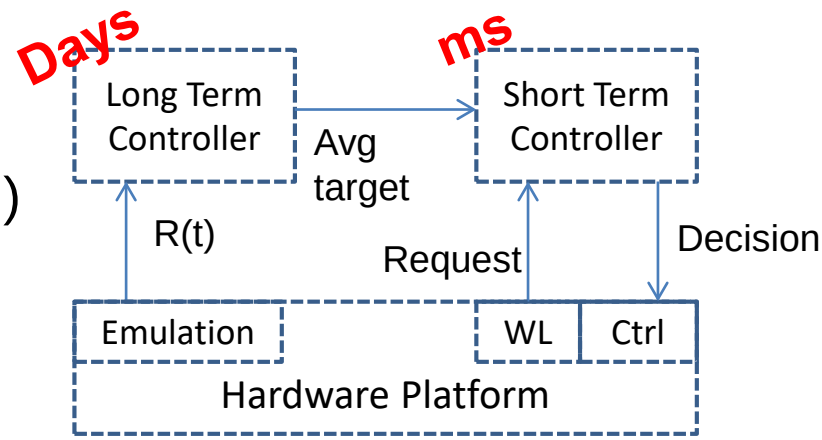
$$V_{average} \leq V_{LTC}$$
$$T_{average} \leq T_{LTC}$$

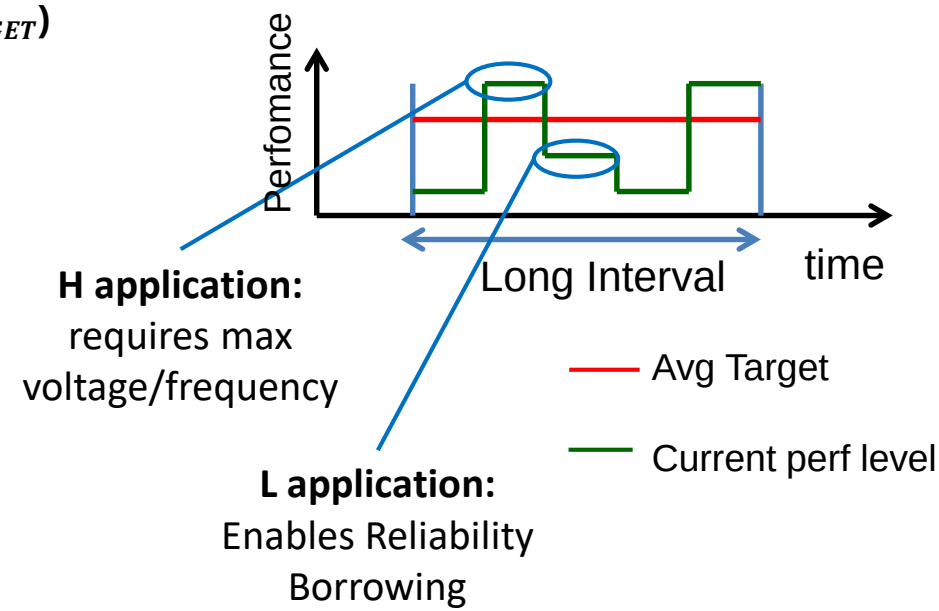
- **Reliability:** "measure" of device integrity
  - Sensors (available only on prototypes)
  - Mathematical models (Online Emulation)
- Depend on temperature and voltage stress:



Dynamic Reliability Management:

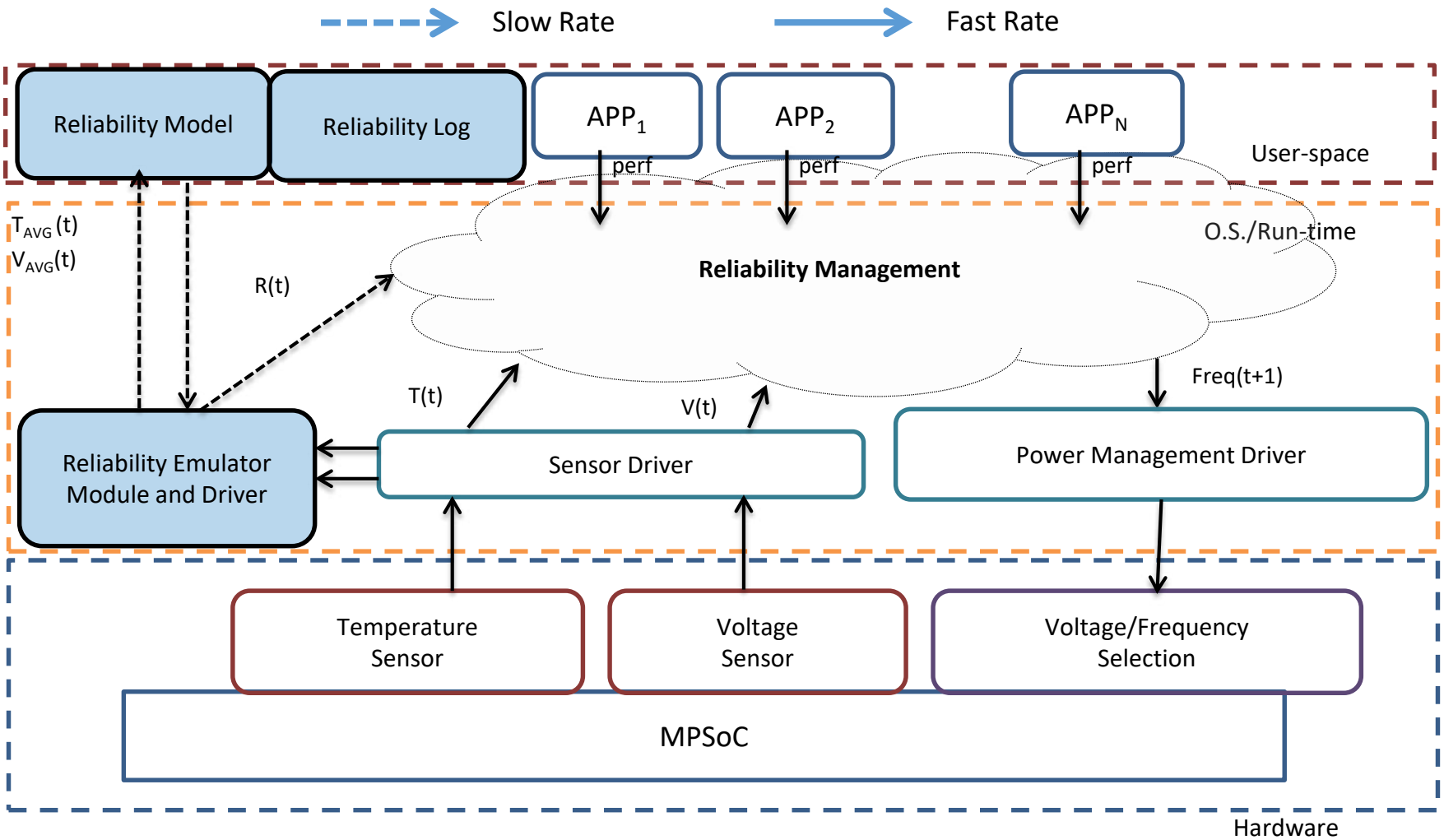
$$R(t_{LIFE}) \geq R_{TARGET}$$



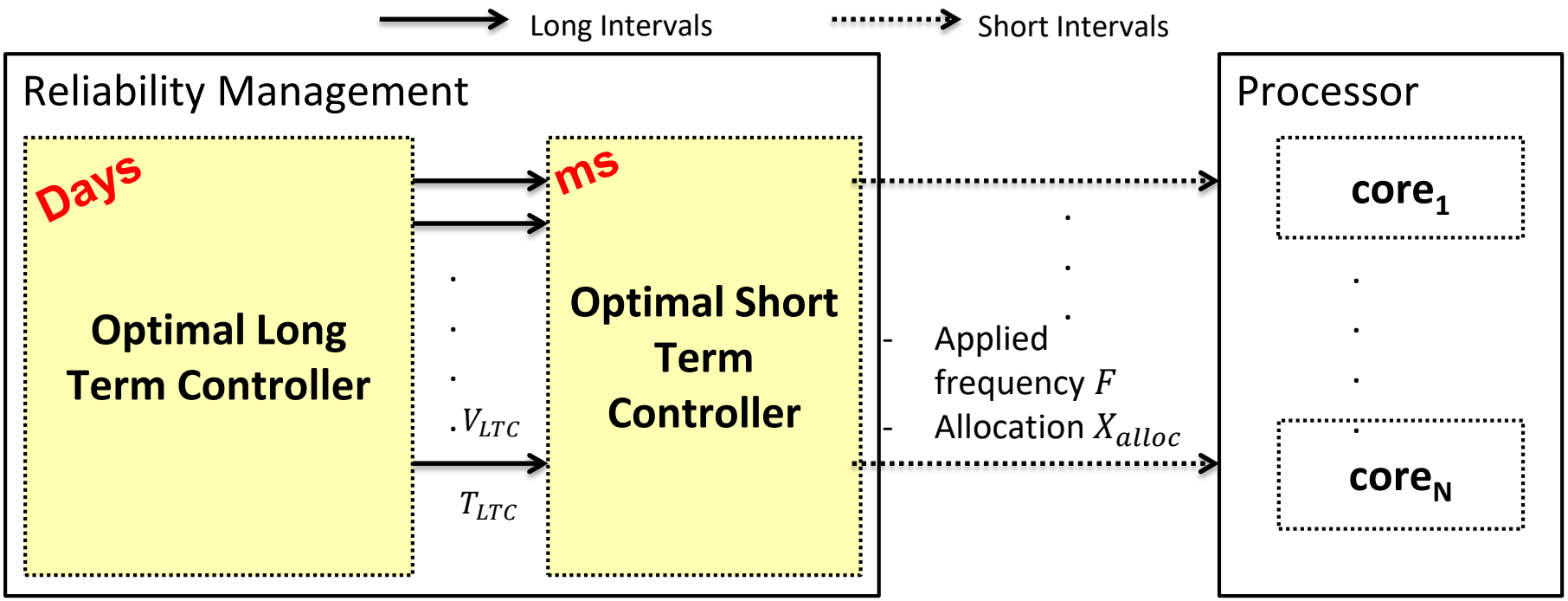


Reliability Emulation ("RelDroid")

RelDroid is used for prototyping on real devices



Optimal Reliability Controller



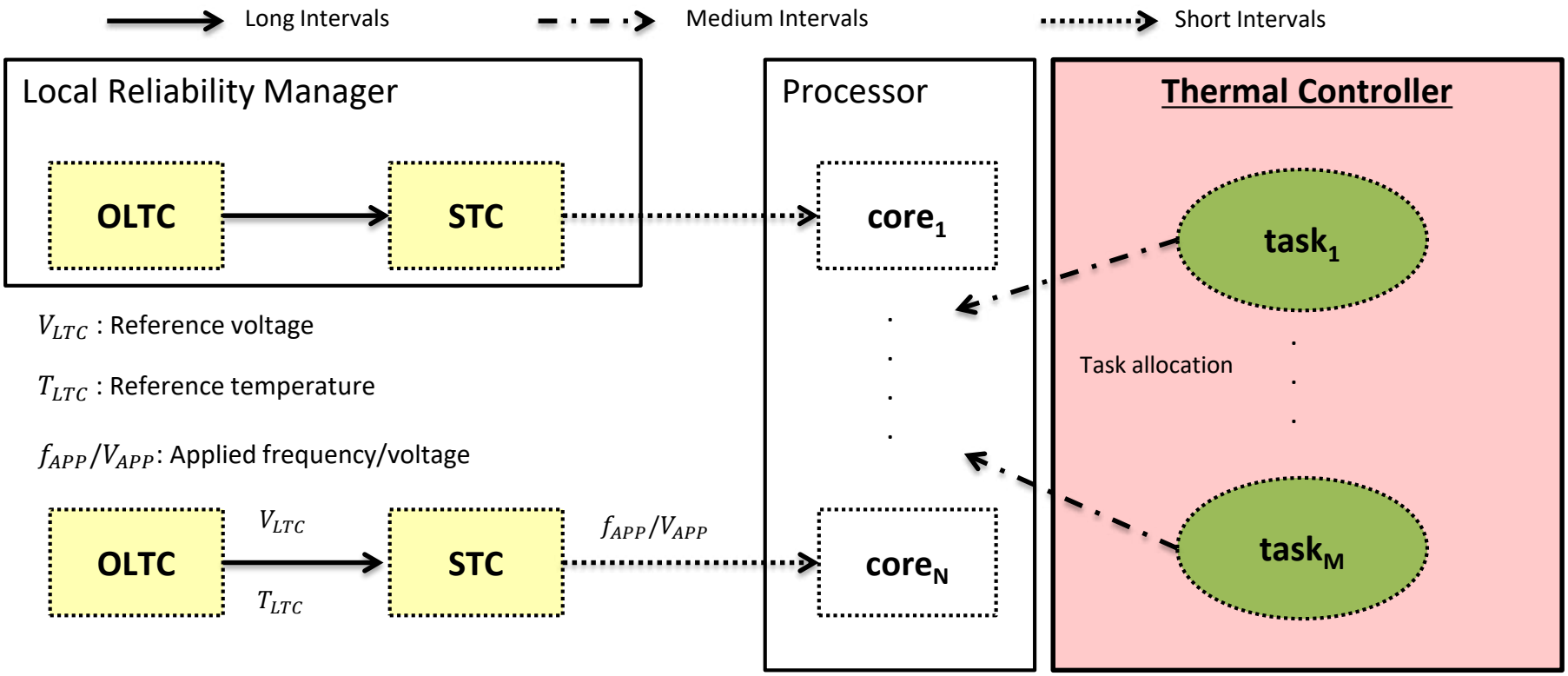
Optimal Long Term Controller: computes the average target values for voltage and temperature V_{LTC} and T_{LTC}

Optimal Short Term Controller: determines frequency to apply F and task allocation X_{alloc}

Solve optimal problem using **convex optimization**

→ Large computation overhead for OSTC

Workload Aware Reliability Manager



Short Term Controller:

- Short Interval (SI, ms)

Thermal Controller:

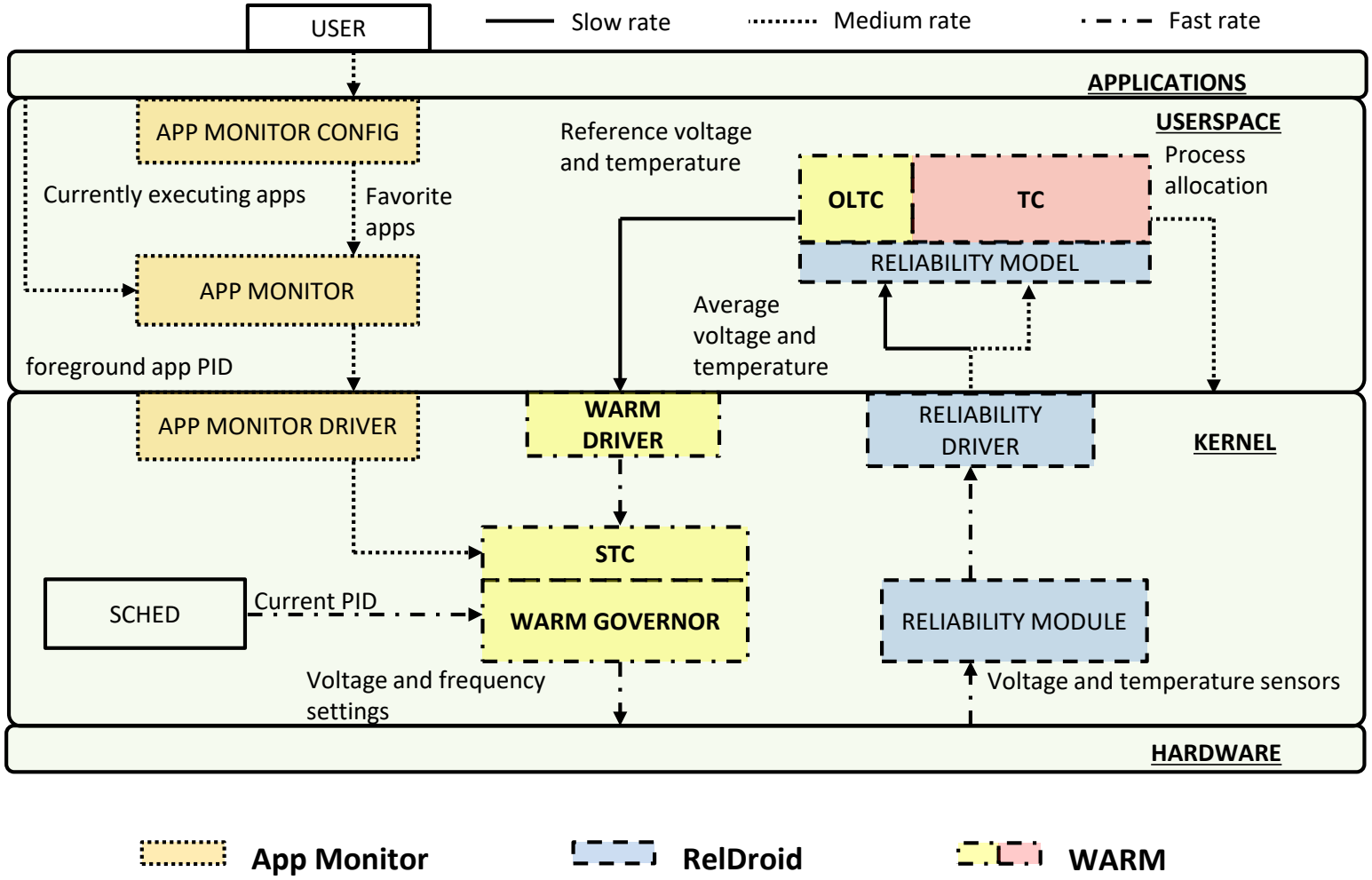
- Medium Intervals (MI, s)
- Assigns H tasks to cooler cores first (X_{alloc})

Borrowing strategy:

$$V_{average} \leq V_{LTC}$$

$$T_{average} \leq T_{LTC}$$

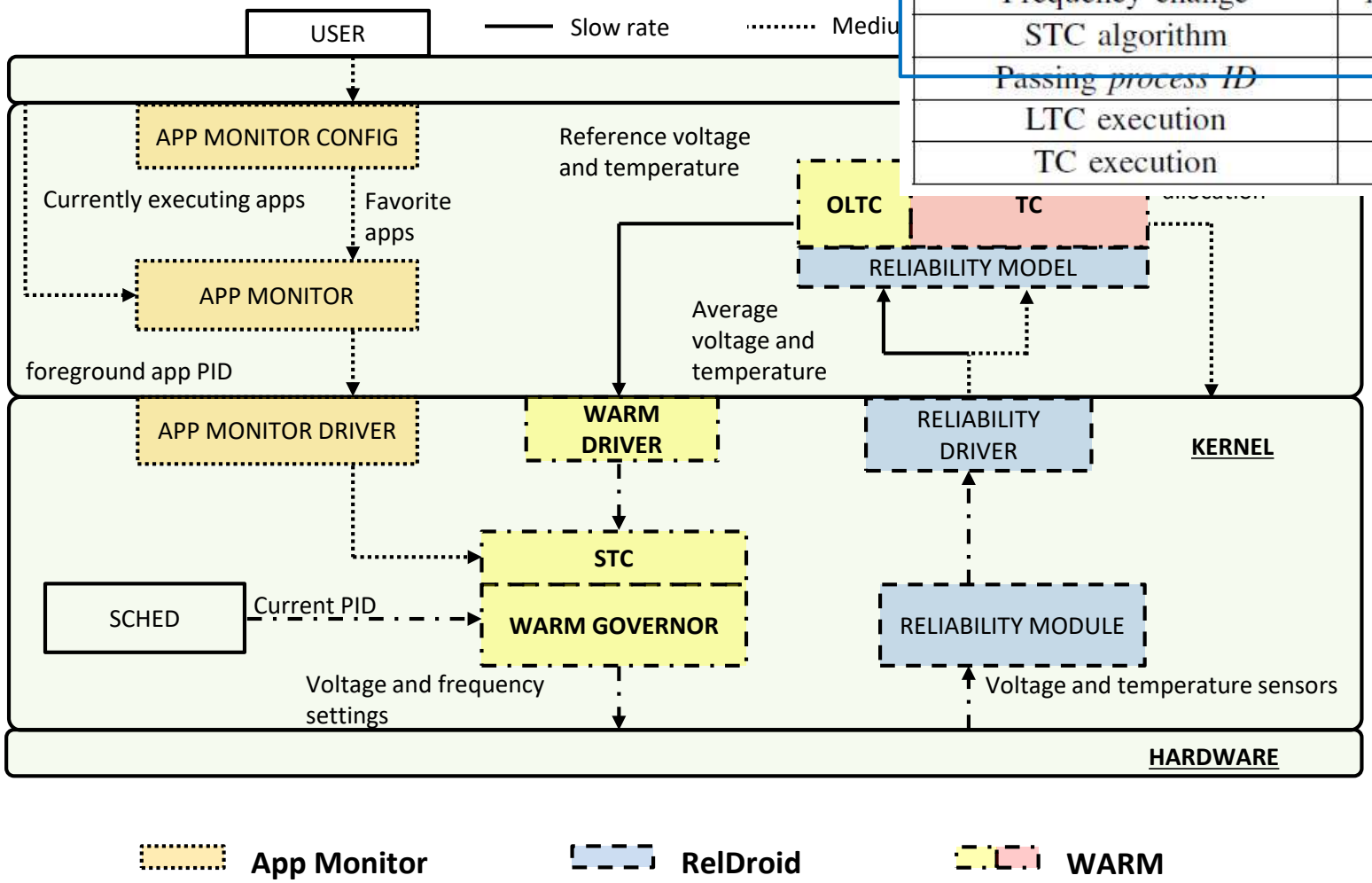
Implementation



Implementation

Implementation overhead

Overhead	Metric
Temperature sensor reading	160 cycles
Voltage sensor reading	170 cycles
Frequency change	16824 cycles
STC algorithm	74 cycles
Passing *process ID*	5 us
LTC execution	342 ms
TC execution	1.5 ms



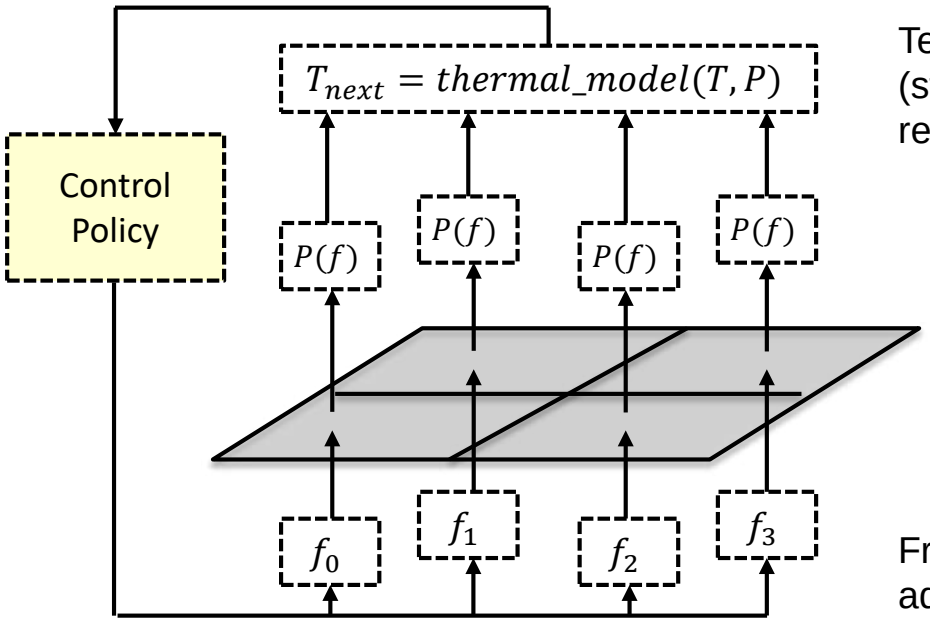
Testing and Validation

Odroid XU3 development board:

- ARM big.LITTLE architecture
- OS is Android 4.4.4 with Linux kernel 3.10.9.
- The device has 4 integrated voltage/current /power monitoring sensors (LITTLE cluster, big cluster, GPU and memory)

Virtual platform:

- 4 cores
- configured based on measurements from the Cortex A15 cores of the ARM big.LITTLE
- CVX for solving optimization



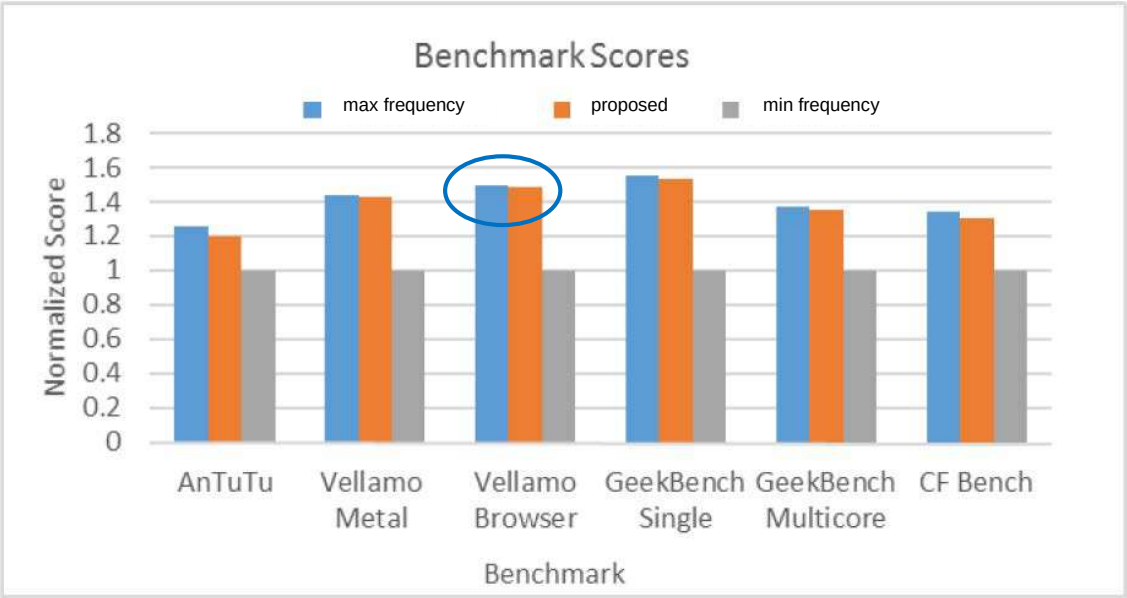
Temperature model (state space representation)

Power model: $P(f) = af^3 + bf$

Virtual cores

Frequency adaptation

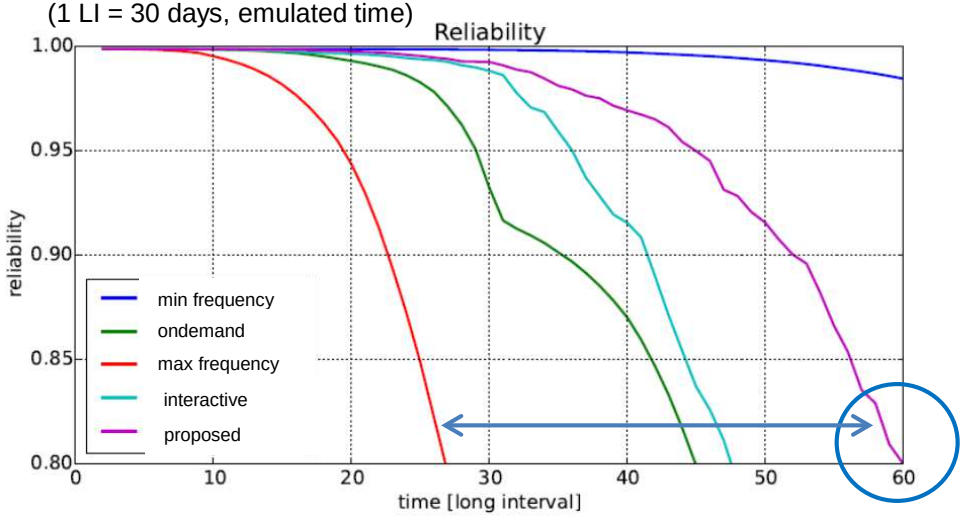
Measurement Comparison with Standard Governors



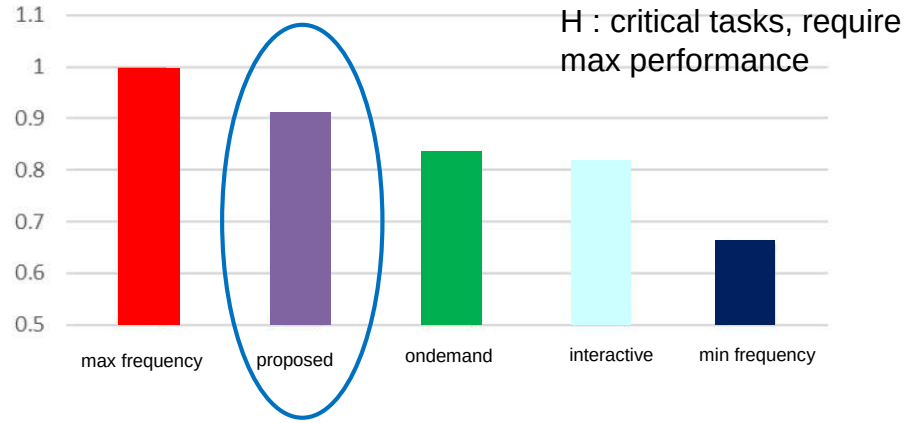
Without background activity:
Performance comparable to max frequency for high quality (H) applications (in the 4%)

With background activity:
it improves performance while meeting the target lifetime.
100% lifetime improvement

(1 LI = 30 days, emulated time)

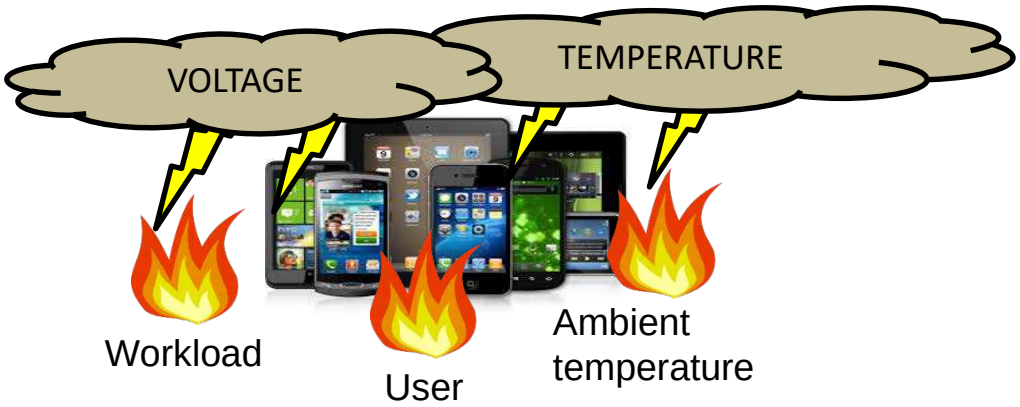


AnTuTu Benchmark (H) – Normalized Score



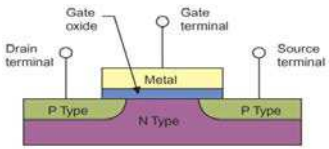
Impact of Variability

DYNAMIC VARIATIONS:



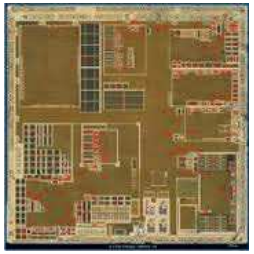
STATIC VARIATIONS:

- Channel length
- Threshold voltage



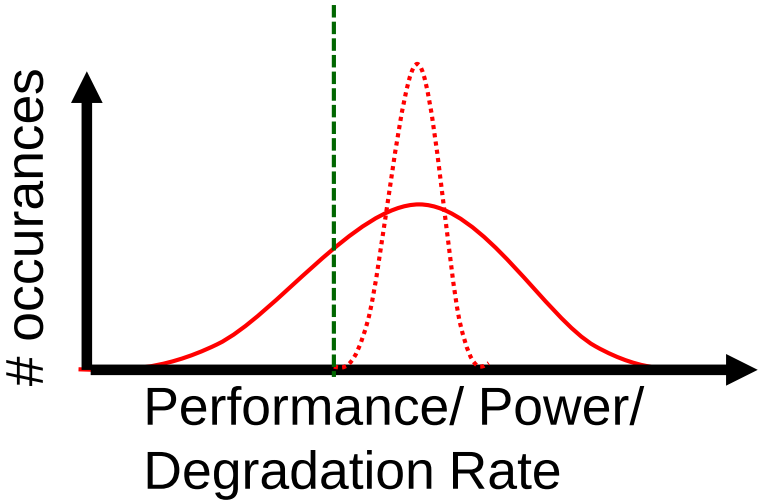
TRANSISTOR

PROCESSOR



- Operating frequency (performance)
- Power consumption
- Degradation rate (DR)

DYNAMIC VARIABILITY MANAGEMENT (DVM)

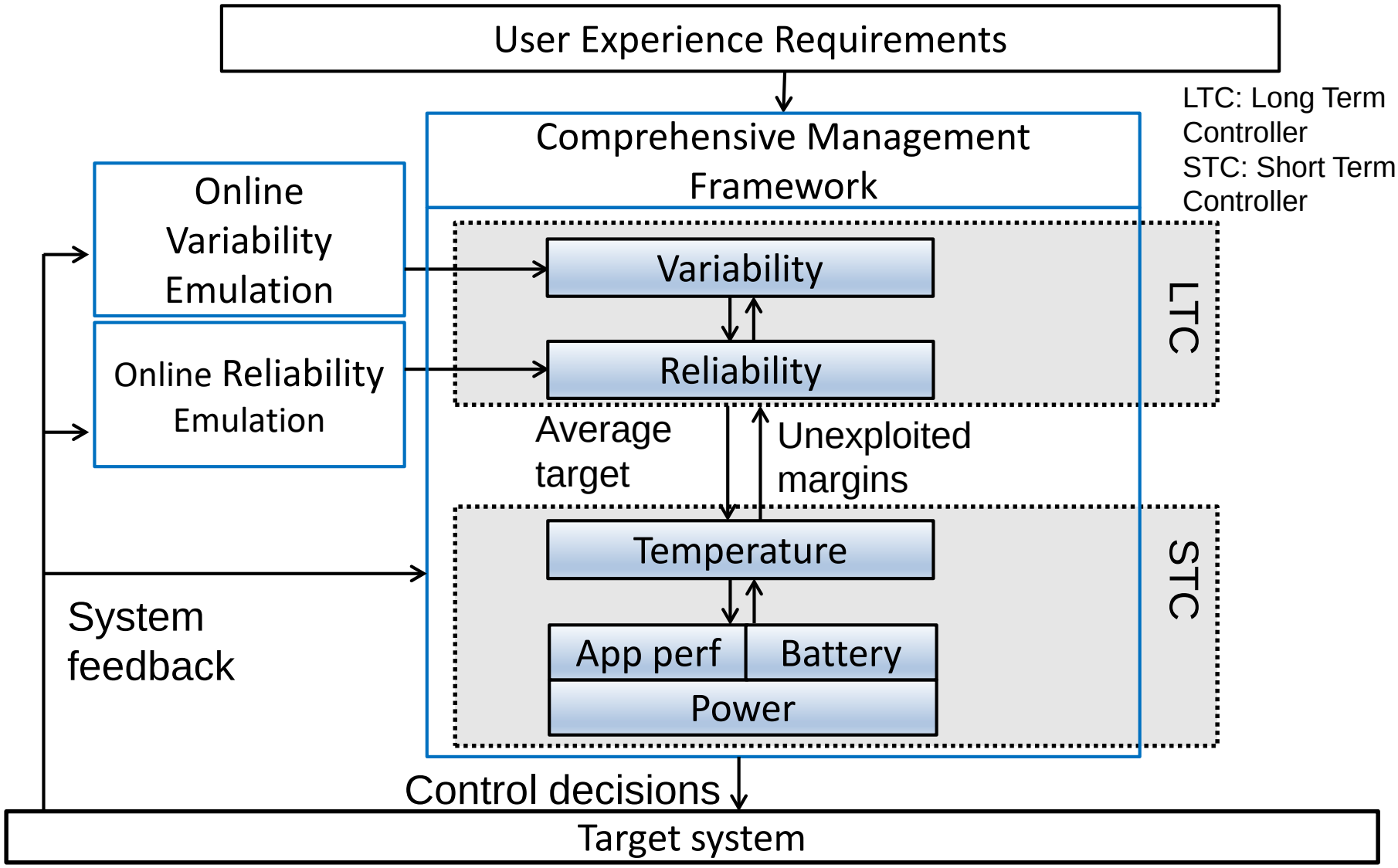


Without DVM

With DVM

Product requirement

A comprehensive management framework for power, temperature, reliability, & variability management in individual devices



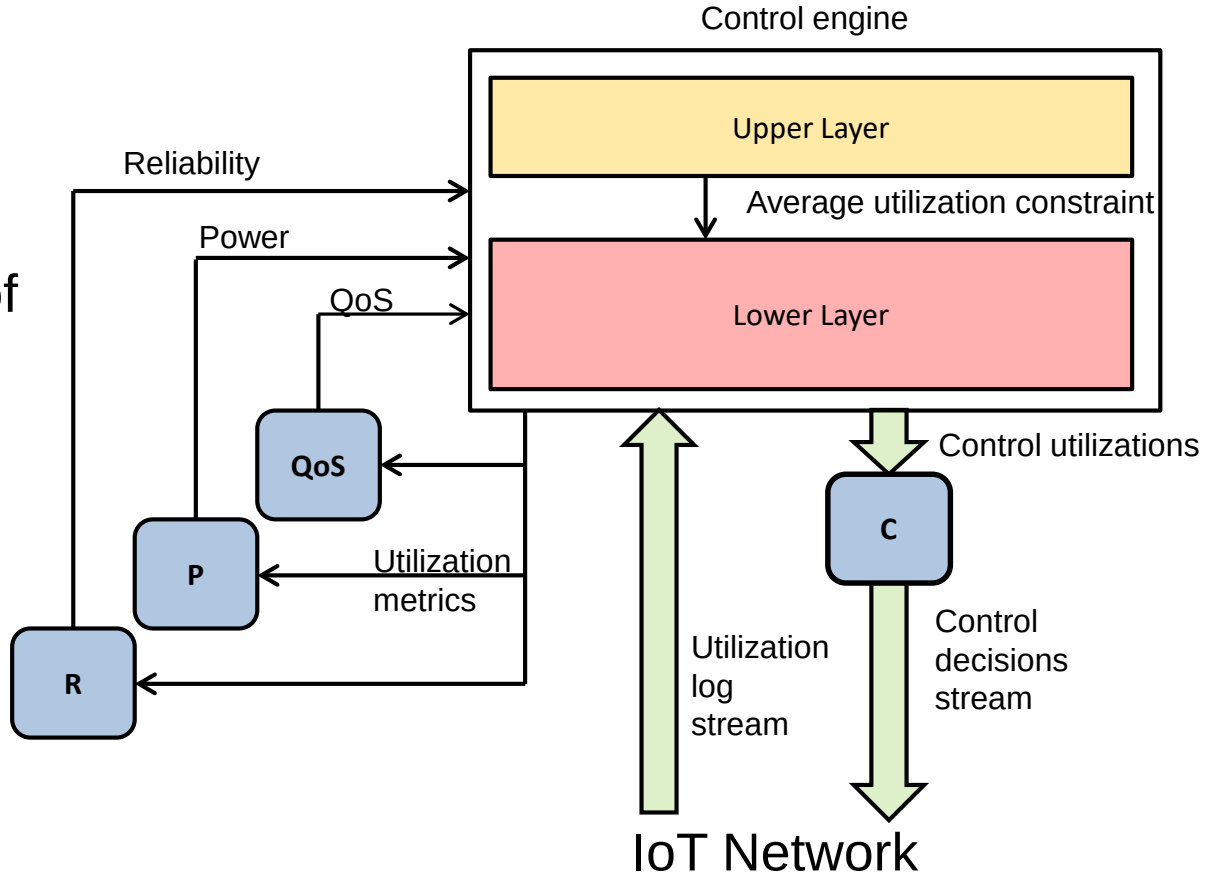
Networks of IoT Devices

Internet of Things:

- Energy and reliability are important issues -> maintenance costs
- Challenges: heterogeneity, distributed nature, large scale

Research directions:

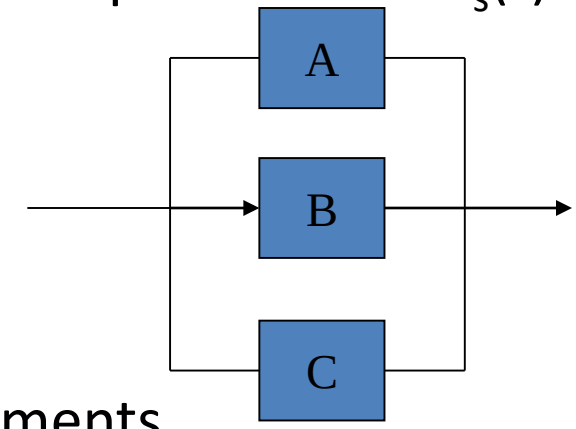
- Framework for energy / reliability management of the IoT infrastructure
- Control of multiple heterogeneous control variables
- Automatic recognition of user experience requirements



Reliability of Networks of Devices

- Series combination: all components must operate for the system to function correctly - $R = R_A \times R_B \times R_C$

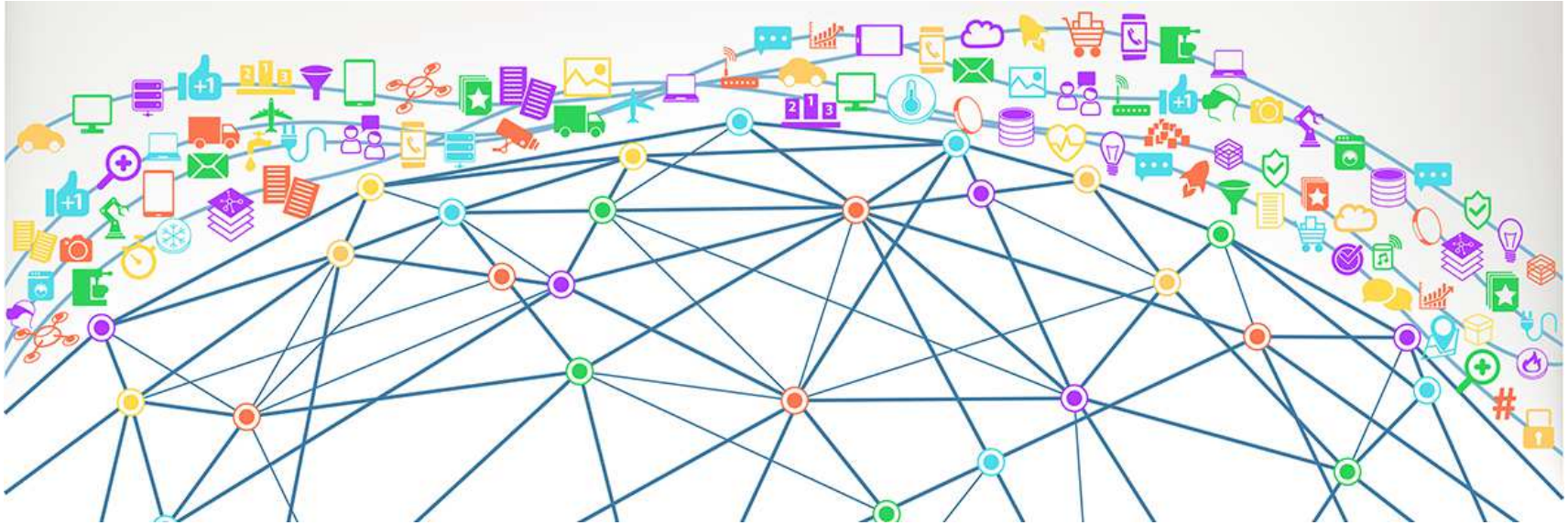
- Parallel combination: system fails if all components fail: $F_s(t) = F_a(t)\, F_b(t)\, F_c(t)$ where $F(t) = 1 - R(t)$



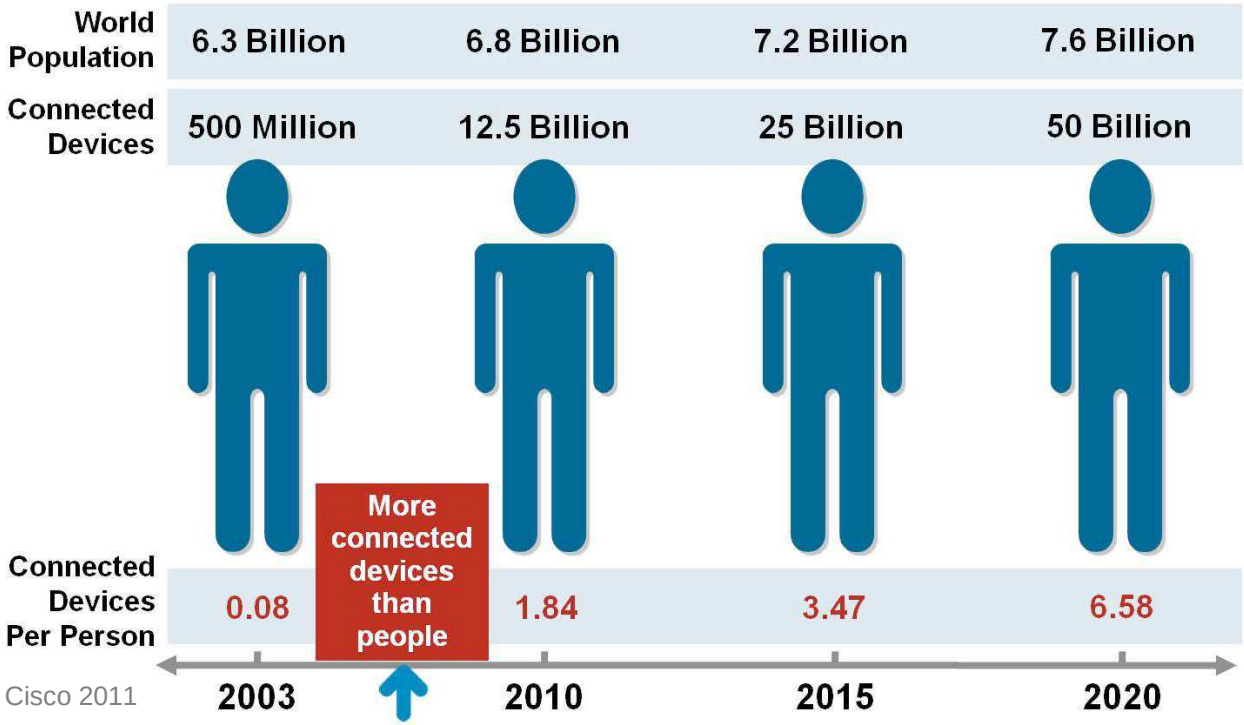
- Most IoT systems are mixed
  - Soft redundancy: imperfect replacements

Research Challenge

Designing an IoT infrastructure that meets performance requirements while maximizing reliability and minimizing maintenance costs



The changing nature of computing



Cisco 2011

50 Billion connected devices

- Impossible to manage using centralized control ->
  distributed resource management and control are key

CitiSense: Environment & Health

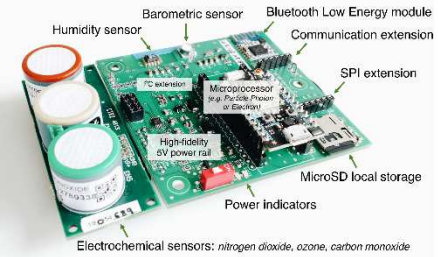
THE WALL STREET JOURNAL.

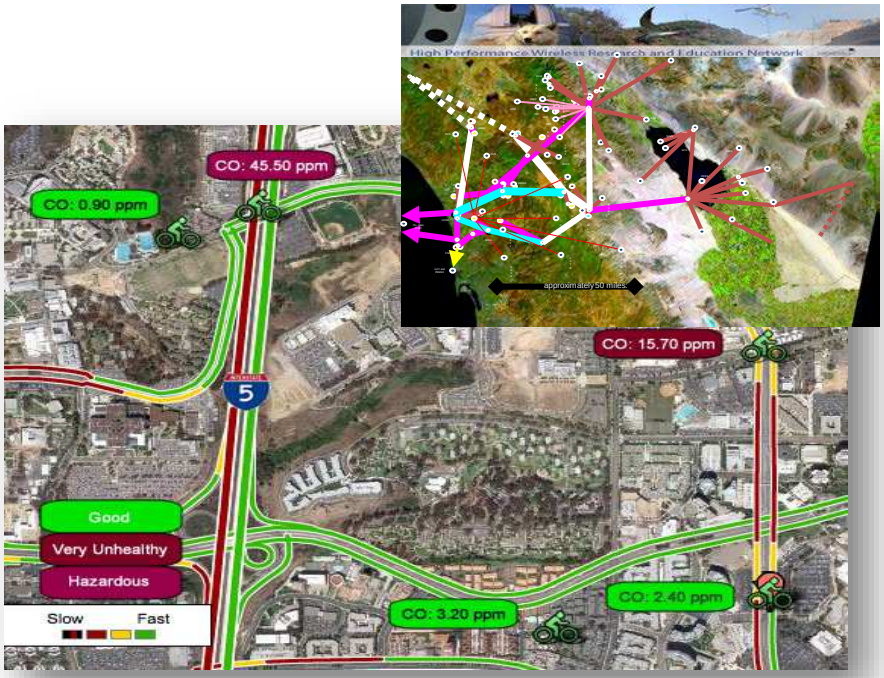
Bad Air at State and Main

Need advance warning of ozone at the n
air-quality sensors that feed data to sm
and others to avoid the heaviest concent

Jacobs School of Engineering/UC San Diego

Portable sensors send air-pollution data to smartphones.





- Sensors and phones given to commuters using various transportation means throughout the greater San Diego area
- Results found “urban valleys” where buildings trapped pollution
- Major routes reported contrastingly high/low AQI for the same location

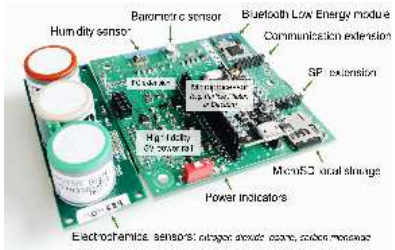
Sponsored by NSF

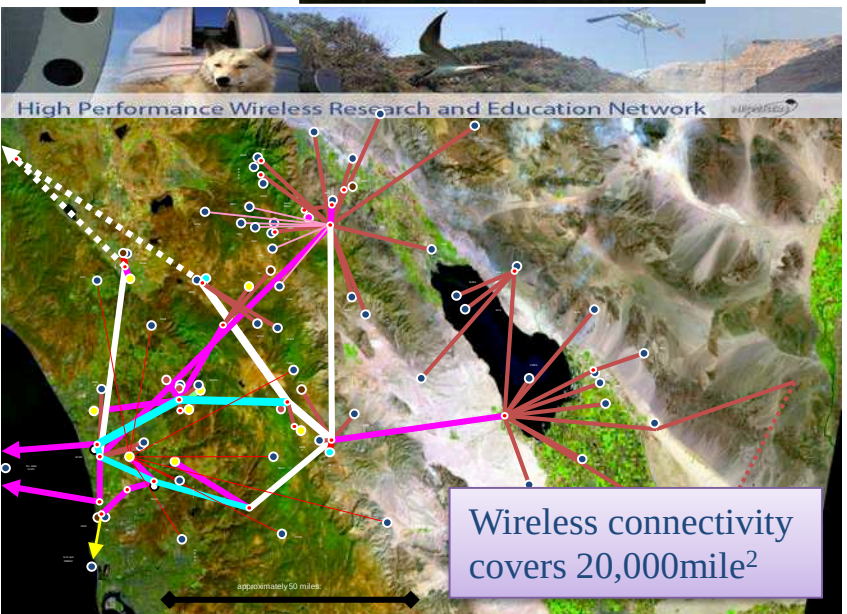
Proactive Fire Tracking

Fire tracking cameras and weather sensors
Cal Fire Dept.

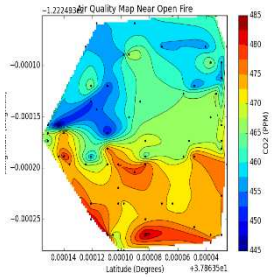
State of the art:
Stationary sensors & cameras placed in high risk areas

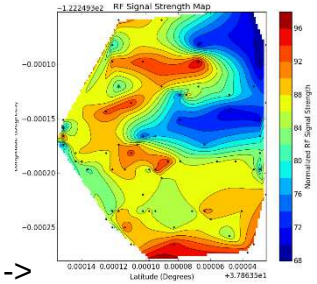
Goal: A proactive infrastructure that leverages stationary & drone based sensors to **sniff out fires & provide adaptive connectivity**





Wireless connectivity covers 20,000mile^2





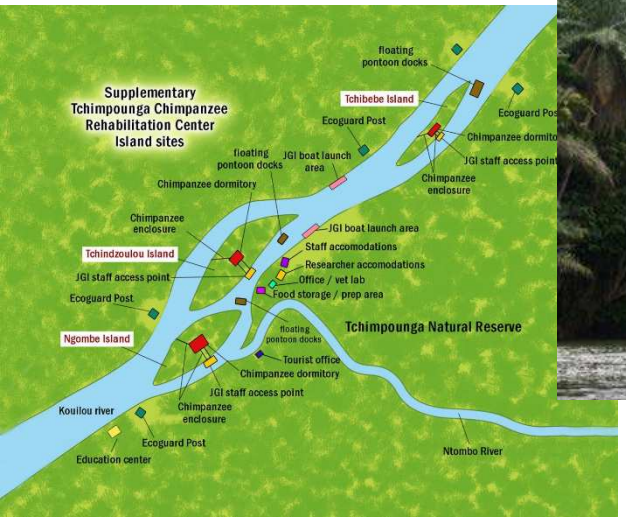
<- Air quality & RF spectrum ->

Sponsored by NSF

Tracking the great apes

The largest chimpanzee sanctuary in the world (Tchimpounga) and the only bonobo sanctuary in the world (Lola Ya) need ways to track groups of animals in enclosures/islands while in the jungle to learn information about them for future release in the wild

Tchimpounga



Lola Ya

3 islands: 100, 60 and 30 hectares
Release site: 20.000 hectares
Individuals per island: 50, 30 & 20

3 enclosures:
15, 10 and 5 hectares
with 20 individuals each

AI for Healthy Aging

- Community Research
- Ethnographic Studies
- Reading Humans
- Robots
- Cognitive Infrastructure

- Cognitive Wellness Dataset
  - Five years of neuropsychological, ethnographic, & sensor data tied to psychosocial predictors of healthy aging
- Cognitive Wellness Framework
  - Model of daily living
  - Behavioral categorization metrics
  - Personalized prediction of cognitive changes
  - Personalized interventions via cognitive objects & robots

For the first time in history, we will longitudinally study the impact that daily habits, the environment, genetics, and the microbiome have on older adult cognition.

We will model the subtle changes of aging, and deploy personalized health interventions via robots & cognitive infrastructure to support independent living.

AI for Healthy Aging

- Community Research
- Ethnographic Studies
- Reading Humans
- Robots
- Cognitive Infrastructure

- Cognitive Wellness Dataset
  - Five years of neuropsychological, ethnographic, & sensor data tied to psychosocial predictors of healthy aging
- Cognitive Wellness Framework
  - Model of daily living
  - Behavioral categorization metrics
  - Personalized prediction of cognitive changes
  - Personalized interventions via cognitive objects & robots

For the first time in history, we will longitudinally study the impact that daily habits, the environment, genetics, and the microbiome have on older adult cognition.

We will model the subtle changes of aging, and deploy personalized health interventions via robots & cognitive infrastructure to support independent living.

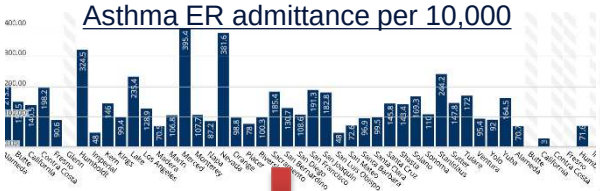
Proactive Health at Scale

Goal: Enable citizens to take *personal* sensor readings and view them in the larger context of *population-level knowledge* for personalized health insights

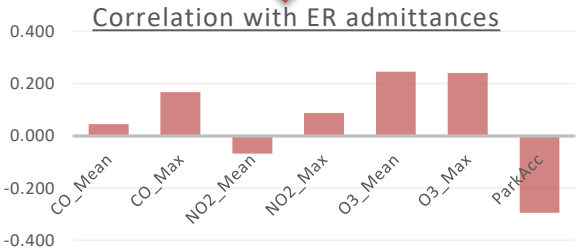
Population Level Analysis

- Per-county demographics, habits and environmental data -> context engine models

"Let's Get Healthy California" Task Force: Asthma ER admittance per 10,000



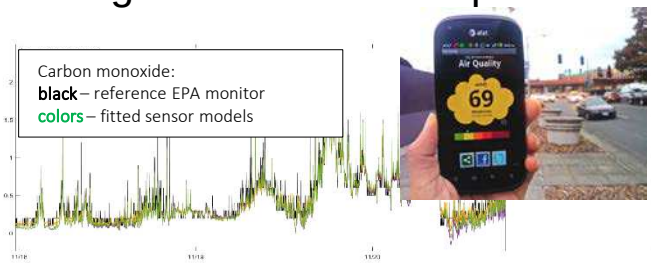
Correlation with ER admittances



Personal Sensing

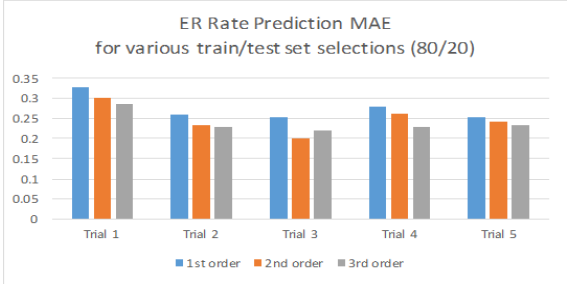
Collect individual context

- Current environmental exposure *(e.g. exercise, daily pollutant exposure & movement etc.).*
- Ensures seamless and adaptive integration of new components



Proactive Feedback

Provide actionable feedback based on personal readings, aggregated local data for and population level models, all combined via context engine infrastructure

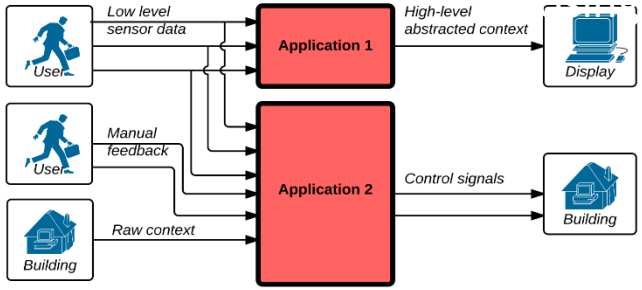


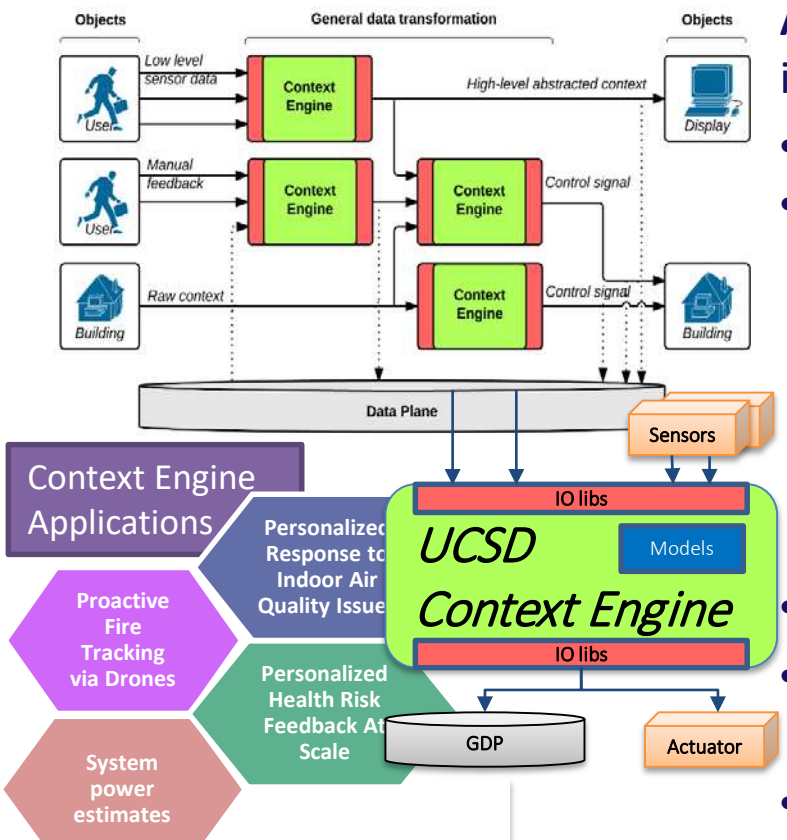
In collaboration with UCSD School of Medicine

Context Engine & Applications

Goal: seamless analysis and reuse of context

- Applications process raw input data → output actions
  - Avoids processing redundancy & monolithic applications
- Provide ontological information
  - Ontology provides range, discretization, type of input data



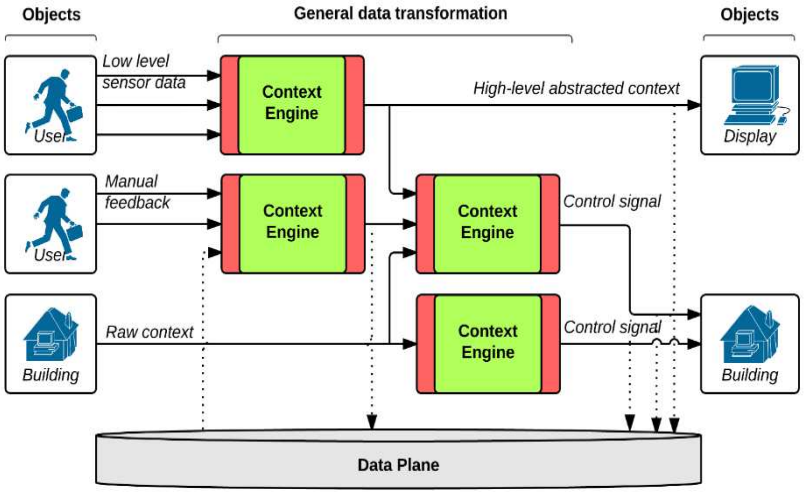


Approach: Replace black-box applications with multi-input single-output functional units – ***context engines***

- Composable units allow data reuse & parallelization
- General purpose statistical learning to generate output based on ontological metadata:
  - Classification: Decision Trees, SVM
  - Regression: KNN, SVR, Linear Regression, TESLA
  - Clustering: K-means, D-Stream II
  - Anomaly Detection
- Simple API facilitates development; includes full RSA
- Runs on small devices (Arduino, RPi3), and on larger systems (laptops, servers)
- Easily scalable and capable of adapting to change

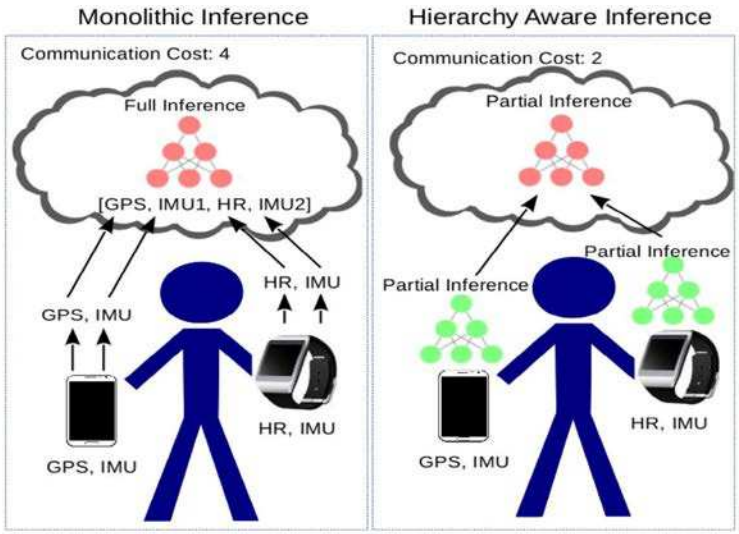
Hierarchical Machine Learning

- **Motivation:**
  - Leverage computation capabilities throughout the IoT hierarcy
    - Reduces communication costs -> increases battery lifetime
    - Cloud only has access to derived features -> accuracy



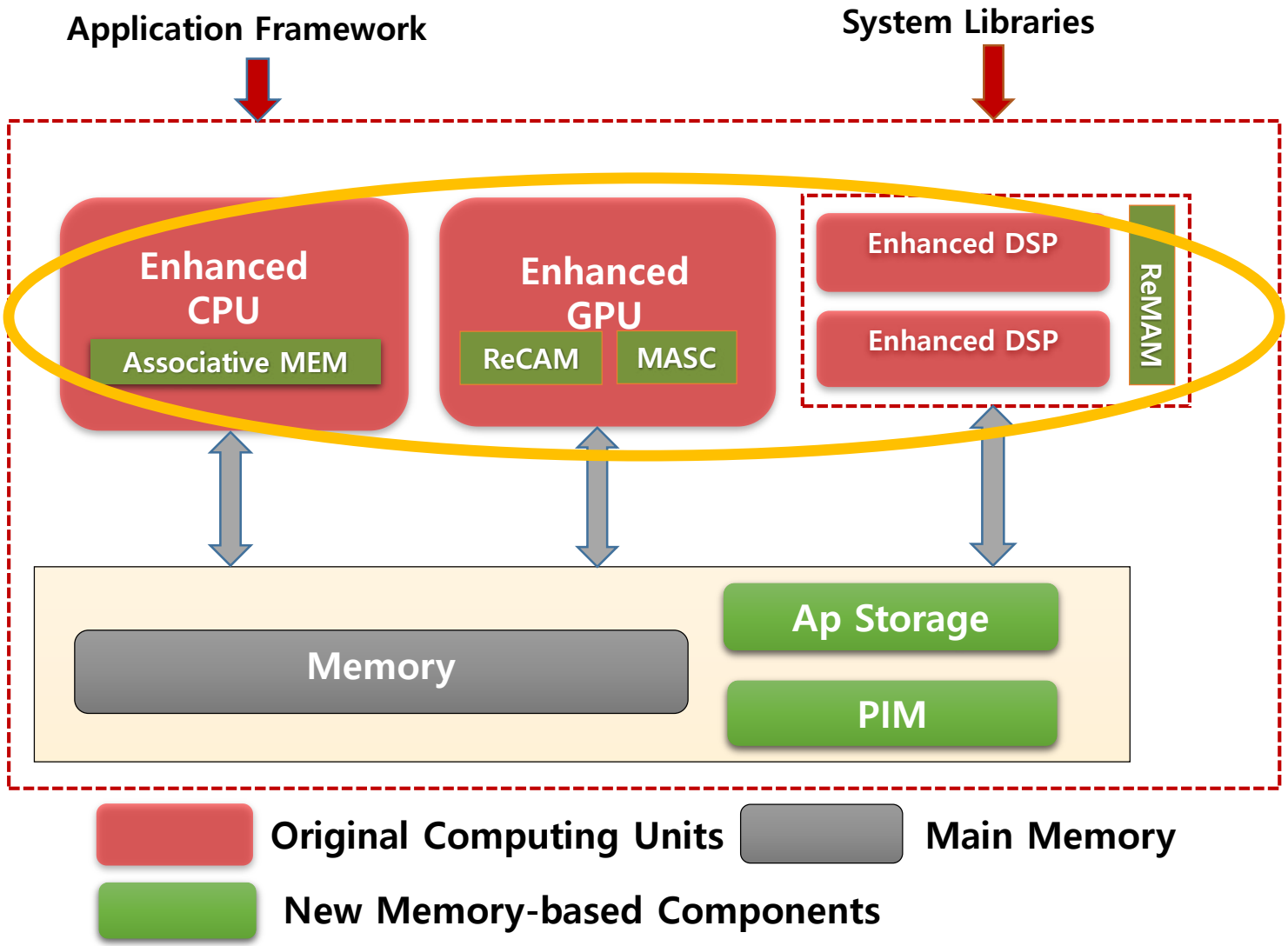
- **Neural Networks fit perfectly!**
  - Optimal compression through local NNs
    - Formulate bounds on accuracy-communication trade off
  - On-line adaptation

- **Results for activity classification:**
  - 640k observations, 36 variables
  - Hierarchical ML has better accuracy & 3x lower communication cost vs. state of the art techniques

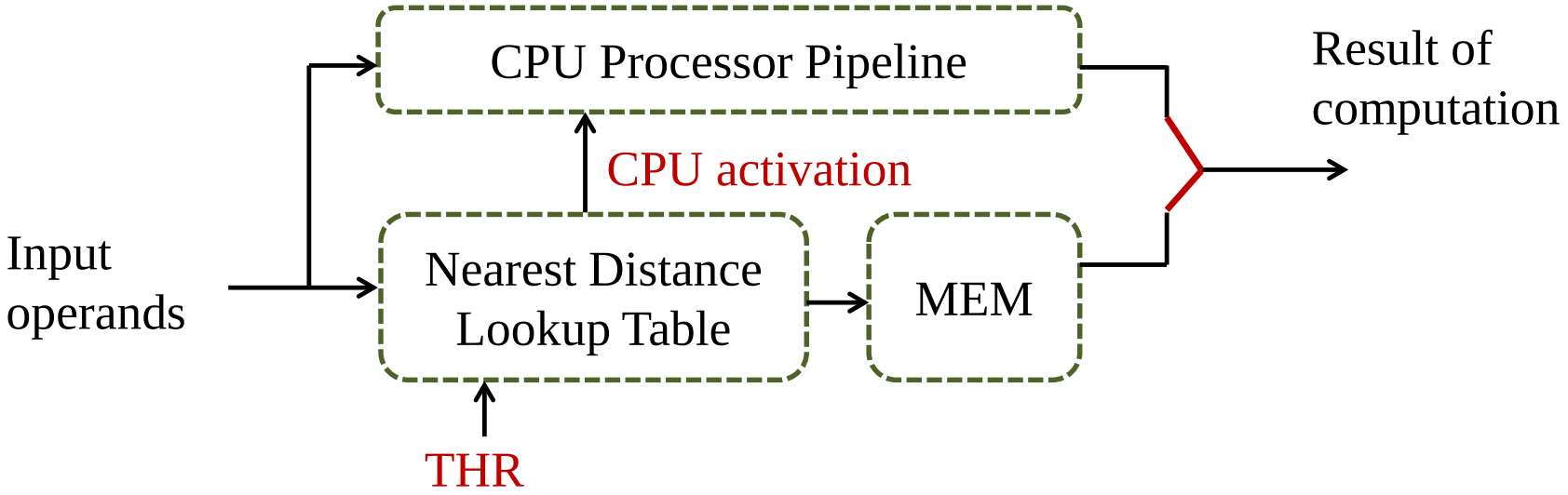


Sponsored by TerraSwarm, published in [SPIE'17]

Acceleration of Large Scale Workloads: CPUs, GPUs, DSPs, PIM



Tunable CPU Approximation [TETC'17]



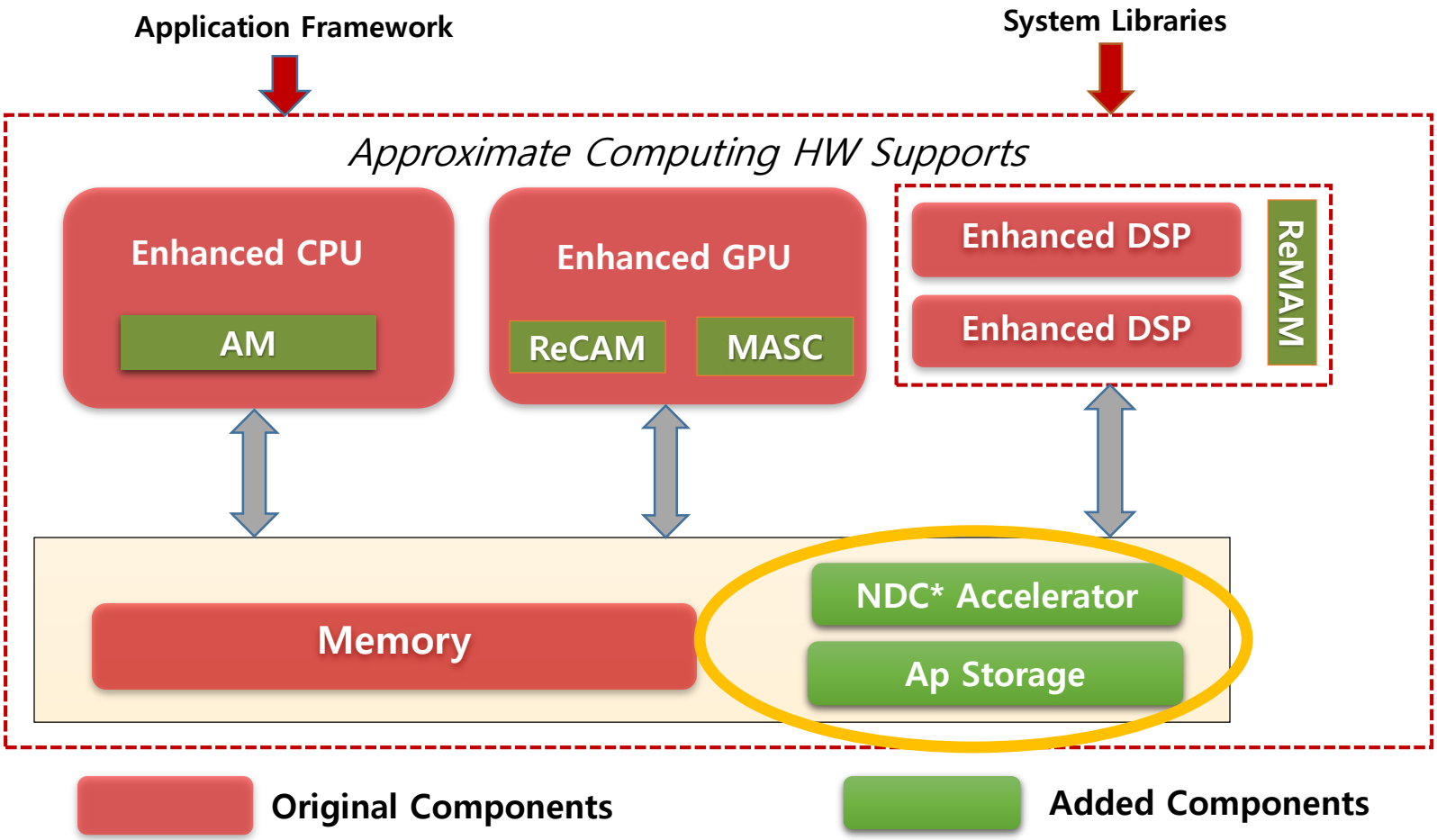
- Profile code and identify common input/output pairs
- Approximately process code on lookup table
- Threshold value (THR) to control output error
  - Common computations handled by RCA
  - Less frequent, but critical computations handled by CPU
  - Runtime control of approximation error

Examples of Recent Associative Memory Designs

- CPU Approximation [TETC'17]
  - Store highly frequent patterns of application in associative memory
  - Approximately model a function by doing similarity check
  - Over six CPU application: 6.5x energy improvement and 4x speedup as compared to CPU with less than 10% quality loss

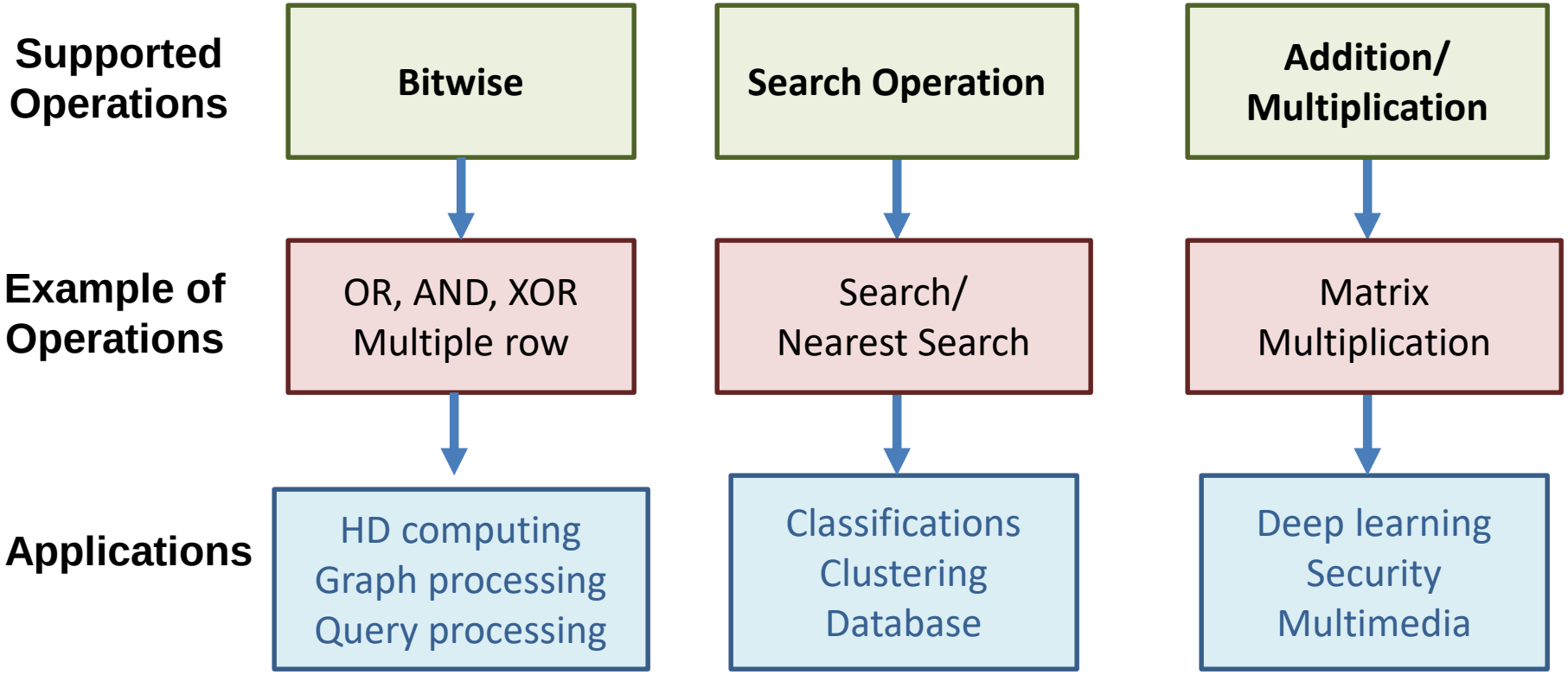
- GPU Approximation [TETC'16, DATE'16, ISQED'15]
  - Uses small associative memory to store frequent FPU patterns
  - Retrieves the pattern exactly or approximately from memory
  - Tunable approximation provides 48% energy savings vs. general GPU

In-memory Acceleration

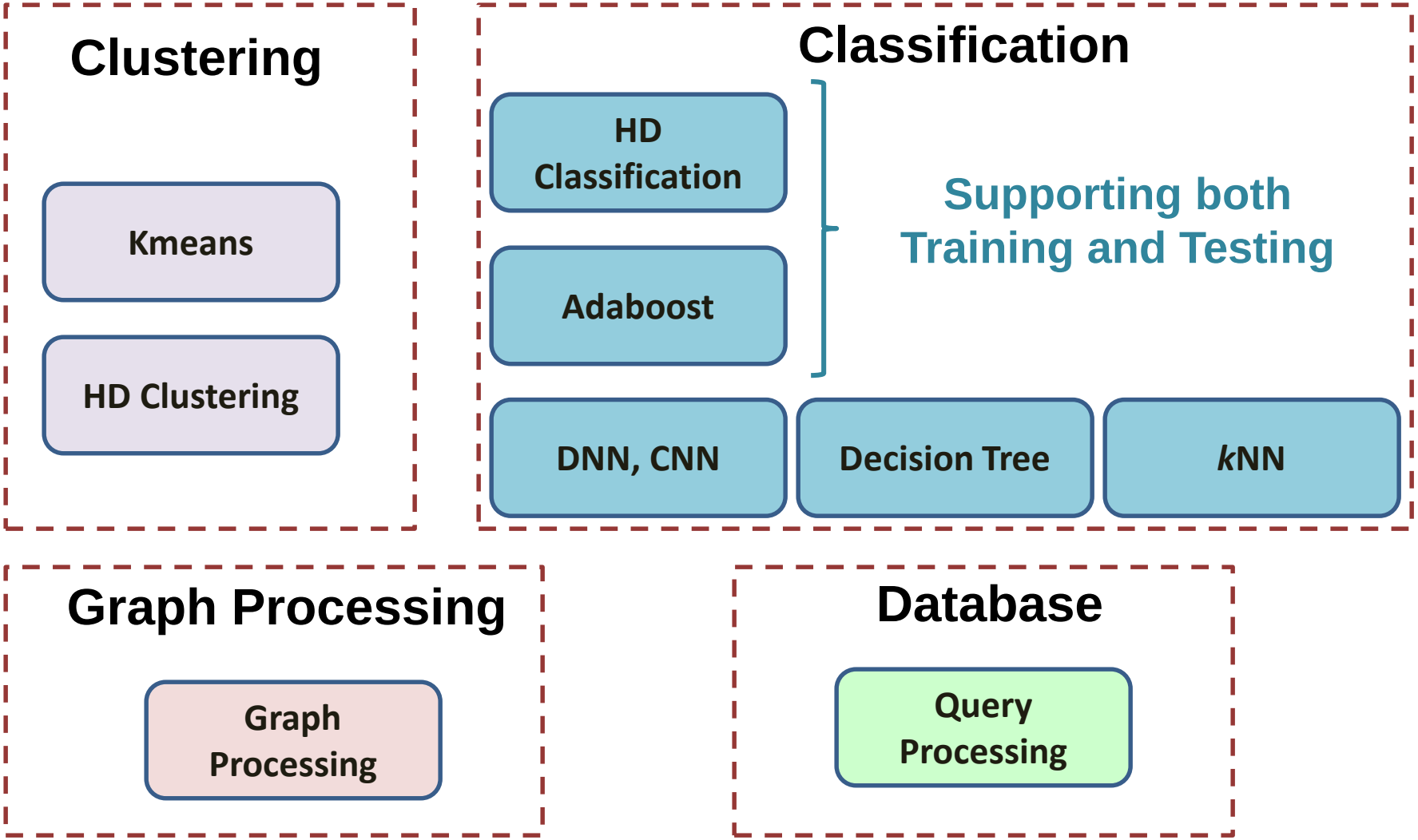


*NDC: Near-Data Computing

Supporting In-Memory Operations



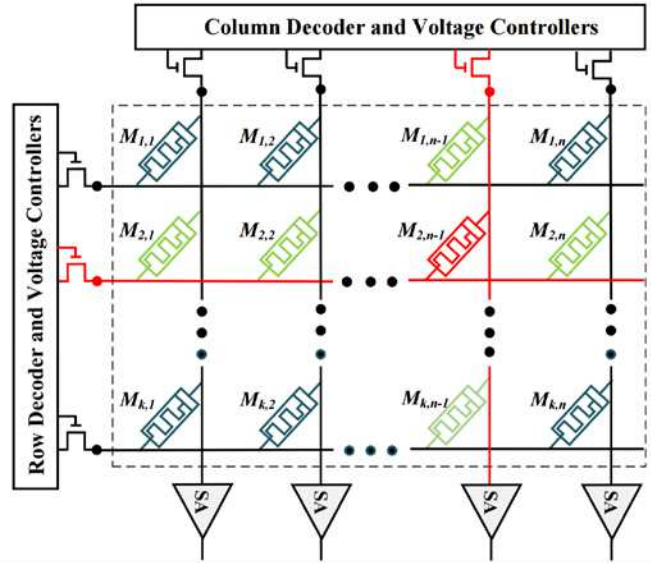
In-Memory Computing Accelerator



In-Memory Addition/ Multiplication

- Crossbar memristive memory which supports:
  - **Addition** for multiple in-memory inputs in exact & approximate modes
  - **Parallel multiplications** in a pipeline stage
- Application: at the compiler level map instructions to addition and multiplication (e.g. via Taylor expansion) then run application in-memory

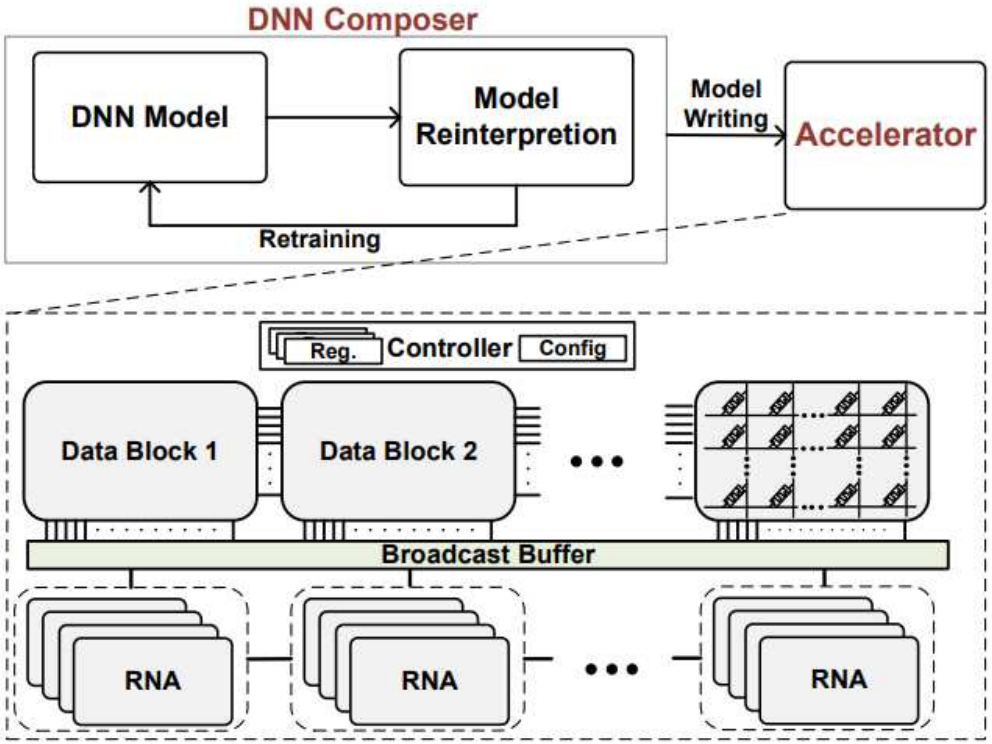
- **Applications:** data-intensive workloads such as machine learning, security, video/image processing, etc.
- **Results**: **20×** performance improvement and **480×** improvement in energy-delay product over GPU running six



[DAC'17] M. Imani, S. Gupta, T. Rosing "Ultra-Efficient Processing In-Memory for Data Intensive Applications", IEEE/ACM Design Automation Conference (DAC), 2017.
[ISQED'17] M. Imani, T. Rosing, "CAP: Configurable Resistive Associative Processor for Near-Data Computing," IEEE International Symposium on Quality Electronic Design (ISQED), 2017.

Example: DNN Accelerator

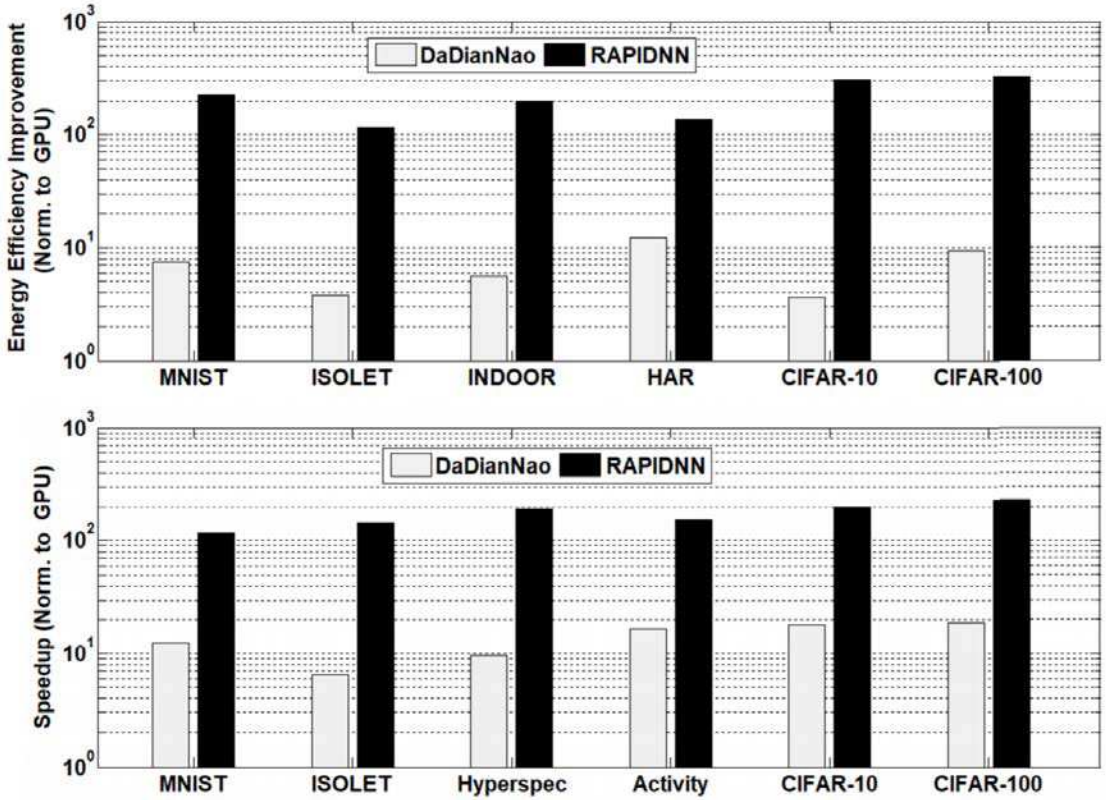
- Reduce the DNN domain:
  - **Sharing** the input/weights using clustering
  - **Retraining** the network to compensate the impact of sharing
  - **Encoding** the shared weights and inputs



- Approximately model neuron computations with basic memory elements
  - Multiplication: single memory access
  - Accumulation: parallel in-memory addition
  - Activation function & pooling implemented with lookup-based search

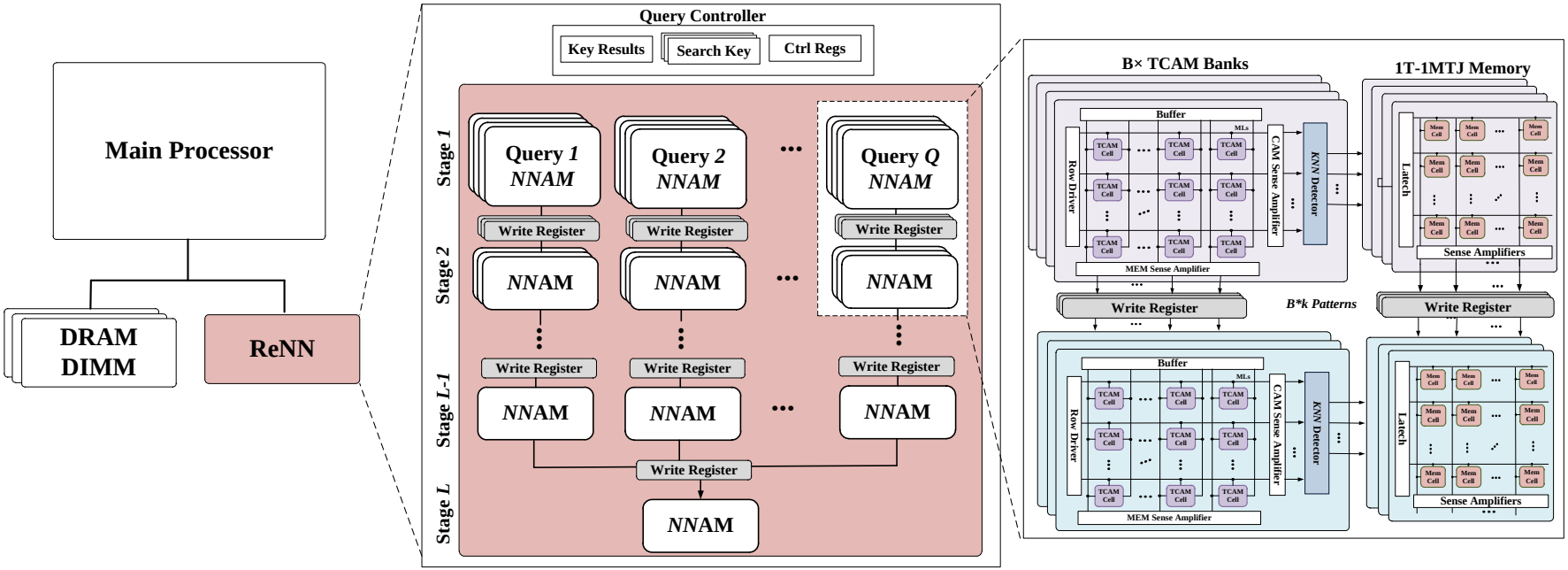
Example: DNN Accelerator

- Results:
  - 210x energy efficiency improvement and 396x, speedup vs GPU
  - 31x energy efficiency improvement and 12.6x speedup vs DadianNao at less than 0.1% of quality loss



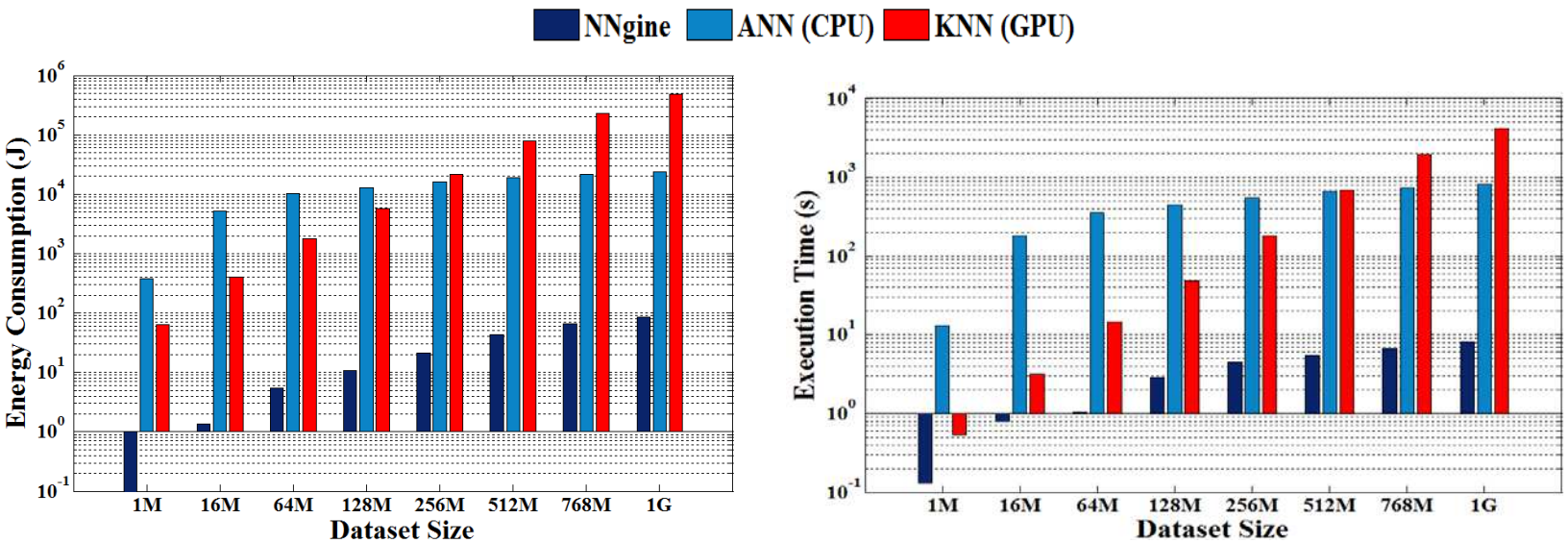
Example:
NN*gine:* KNN Accelerator [ICRC17]

- **NN*gine*:** Nearest neighbor search accelerator
- **Applications:** pattern recognition, statistical classification, biology, computer vision, databases, coding theory, computational geometry
- Performing the search operation inside the magnetic memory next to DRAM



*NNAM: Associative memory with capability of finding k-nearest neighbor

NN*gine* Results

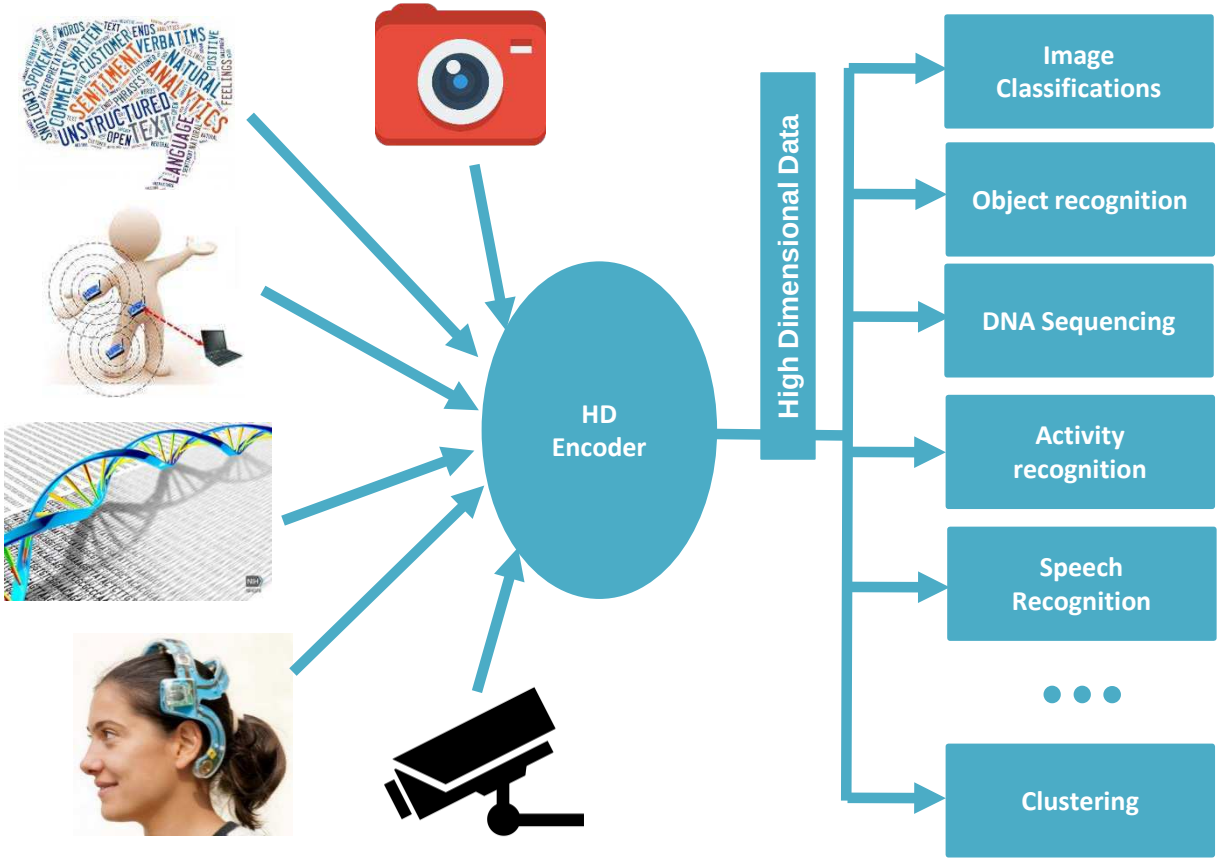


NN*gine* improvement compare to CPU/GPU up to:

NN*gine* Energy Improvement	5590x
NN*gine* Speed up	510x

- Implementation is not sensitive to dataset and query sizes
- Results are very accurate – less than 0.5% QoS loss

Brain-inspired Hyperdimensional Computing



Hyperdimensional computing:

- Represent data using bits encoded at large dimensionality (> 10,000); such randomly encoded vectors are orthogonal
- Purely statistical, thrives on **randomness**
- Supports **full algebra**
- **Reversible**: data can be recovered after encoding

Superb properties:

- **General** and scalable model of computing
- **One-shot** learning
- Robust against **most failure mechanisms and noise**

Hyperdimensional Computing

Brain Checks for Similarity!

Training

Reversible Encoding

Cat hypervector

-1 +1 -1 -1 +1 . . . +1

Encoding

Dog hypervector

-1 -1 +1 -1 -1 . . . +1

Similarity Check

Testing

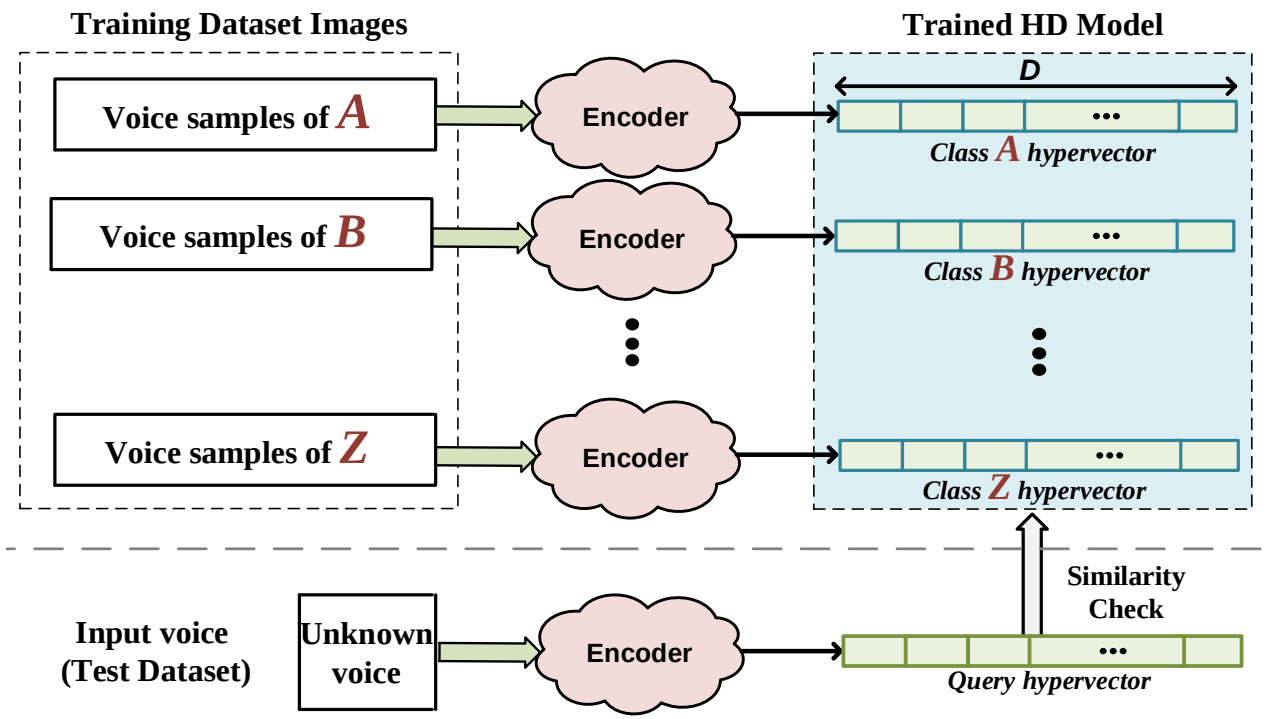
Encoding

+1 +1 +1 -1 -1 . . . +1

Query hypervector

Example: Speech recognition [ICRC'17]

- Speech recognition dataset (ISOLET)
  - ISOLET includes voice of letters of alphabet. The objective is to classify an input voice to one of 26 letters, A to Z.
  - 150 subjects spoke the name of each letter of the alphabet twice. Hence, there are 52 training examples from each speaker.



Speech Recognition: VoiceHD [ICRC'17]

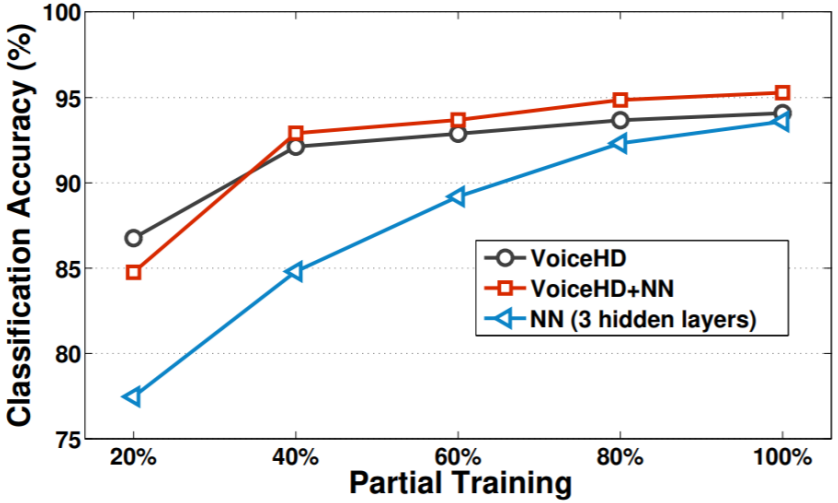
Accuracy and efficiency:

- Comparing VoiceHD vs NN
  - *Training:* 3x faster
  - *Testing:* 1.3x faster and 8.5x more energy efficient
  - *Accuracy*: VoiceHD (95.3%), NN (93.6%)

		NN	VoiceHD
Training	Execution Time/Dataset	17min	3.7min
Testing	Energy Consumption/Query Execution Time/Query	454mJ 4.61ms	38mJ 0.87ms

Partial Training:

- Training on 40% of dataset:
  - *VoiceHD can maintain high accuracy of 91.7%*
  - *NN accuracy sharply drops < 85%*



Example: Image Classification with Hyperdimensional Computing

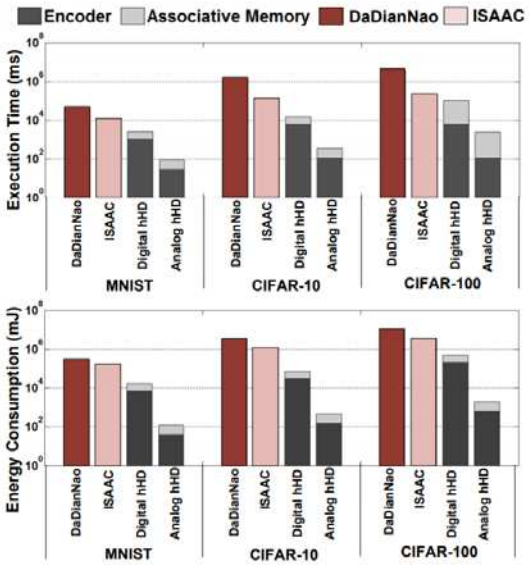
In memory acceleration vs, ISSAC [ISCA'16]:

Digital HD:

- 18× higher energy efficiency & 8.5× speedup

Analog HD:

- **1417× higher energy efficiency & 135× faster**



CIFAR-100			
iHD	**hHD**	**DNN-3**	**CNN-4**
30.1	38.4	120.2	151.6
54.5	69.3	2,560.0	3,932.6
57.6	64.3	105.3	116.8
12.7	16.1	21.8	27.4
58.3%	67.9%	63.6%	69.4%

Dataset		**MNIST**					**CIFAR-10**			
		iHD	**hHD**	**DNN-1**	**CNN-1**	**CNN-2**	**iHD**	**hHD**	**DNN-2**	**CNN-3**
Training	*Power (W)*	21.2	27.8	75.0	115.2	128.6	28.9	35.2	109.7	143.1
	Execution Time(min)	27.6	44.1	2,591.6	785.3	1,516.6	53.1	67.6	1,727.5	2,793.2
Testing	*Power (W)*	37.6	45.7	78.6	66.2	74.1	46.0	50.7	96.1	105.4
	Execution Time(min)	5.2	7.1	14.6	10.3	12.9	8.7	13.1	16.8	20.4
	Max Accuracy	97.4%	98.9%	96.5%	98.9%	99.1%	83.3%	89.8%	87.9%	90.3%

Example: DNA Sequencing

Classifying the DNA sequences:

- Tested on *Empirical* Dataset consisting of long sequences
- HD can achieve 2x more efficient and 4.7x faster while providing more accurate classification results

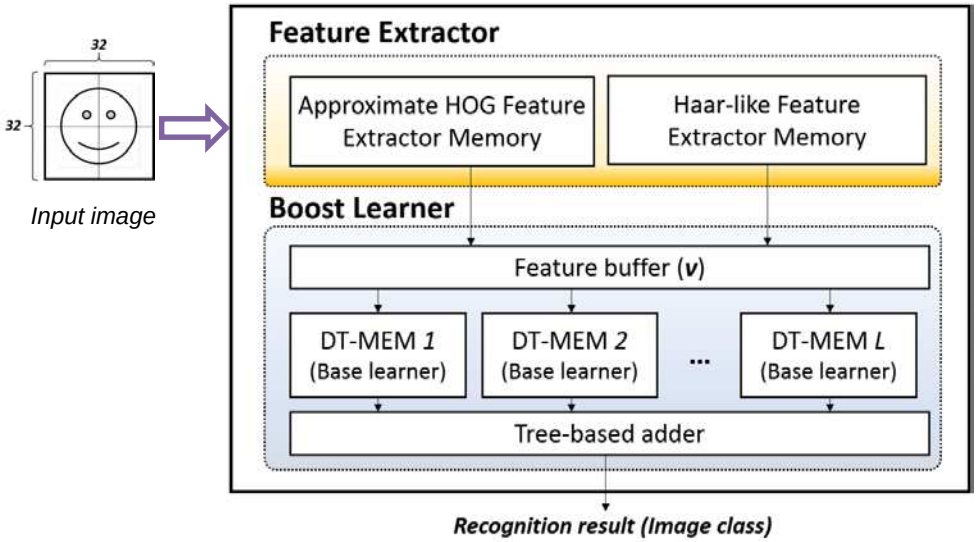
	Classes	SVM	Bayes	Encoder I	Encoder II
Accuracy	*Cypraeidae*	94.3%	93.2%	100%	100%
	Drosophila	98.3%	96.5%	100%	100%
	Inga	89.8%	91.5%	100%	100%
	Bats	100.0%	100.0	98.2%	100%
	Fishes	95.5%	97.3%	100%	95.2%
	Birds	98.4%	94.3%	99.7%	100%
	Fungi	80.0%	70.0%	100%	100%
	Algae	100.0%	100.0%	100%	100%
	Average	**94.53%**	**92.85%**	**99.74%**	**99.40%**
Energy Consumption (mJ)		62.03	47.51	14.53	23.16
Execution Time (ms)		2.77	1.73	0.58	0.44

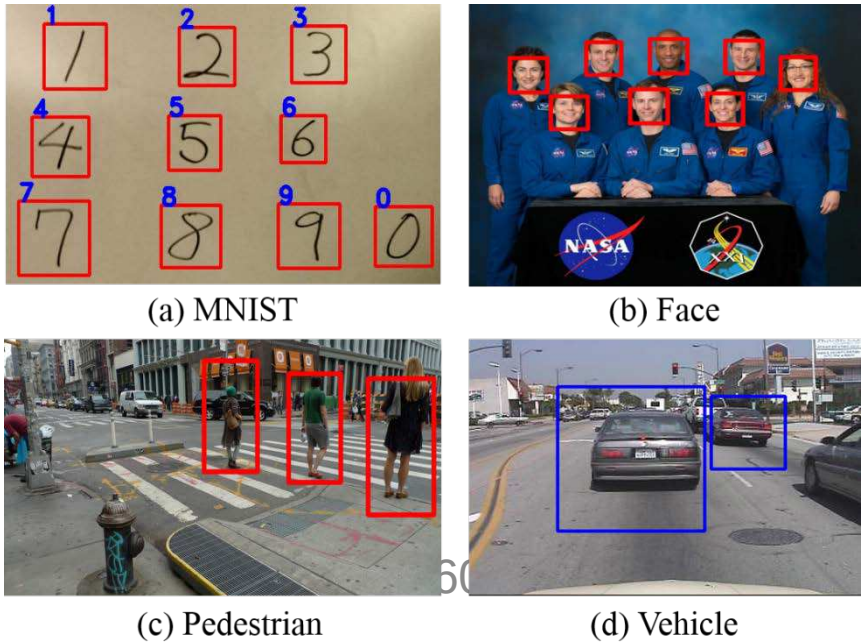
DETAILS OF USED EMPIRICAL DATASET

Class	*Seq.Len.*	*#.Seq*	*#species*	*Class*	*Seq.Len.*	*#.Seq*	*#species*
Cypraeidae	614	2,008	211	***Fishes***	419	626	82
Drosophila	663	615	19	***Birds***	255	1,700	150
Inga	1838	913	56	***Fungi***	510	50	8
Bats	659	826	82	***Algae***	1,128	26	5

Example: ORCHARD
Object Recognition Acceleration [ICCAD'17]

- AdaBoost: One of the best off-the-shelf learning algorithms
  - Exploits ensembles of weak learning models (e.g., decision trees) => robust & general purpose
  - Memory blocks are designed for in-memory feature extraction
  - Classification decision happens with ensemble decision tree block



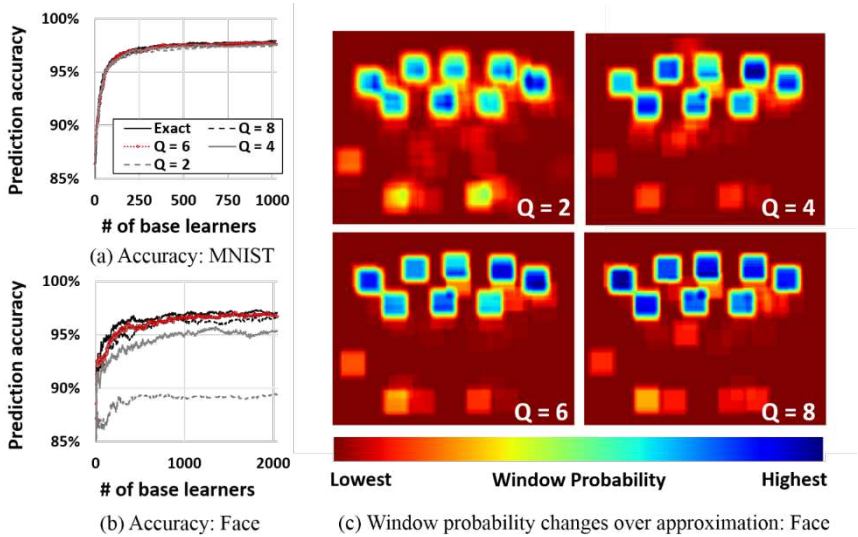


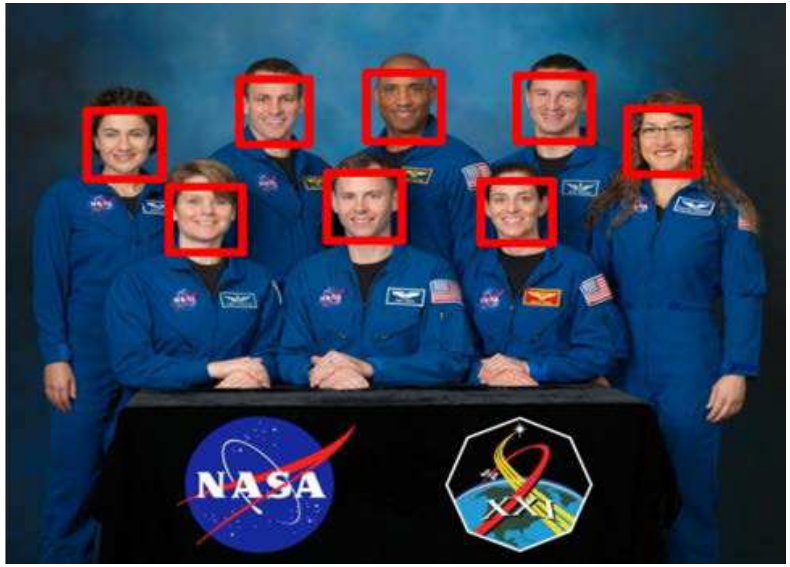
(a) MNIST (b) Face

(c) Pedestrian (d) Vehicle

Object Recognition Acceleration [ICCAD'17]

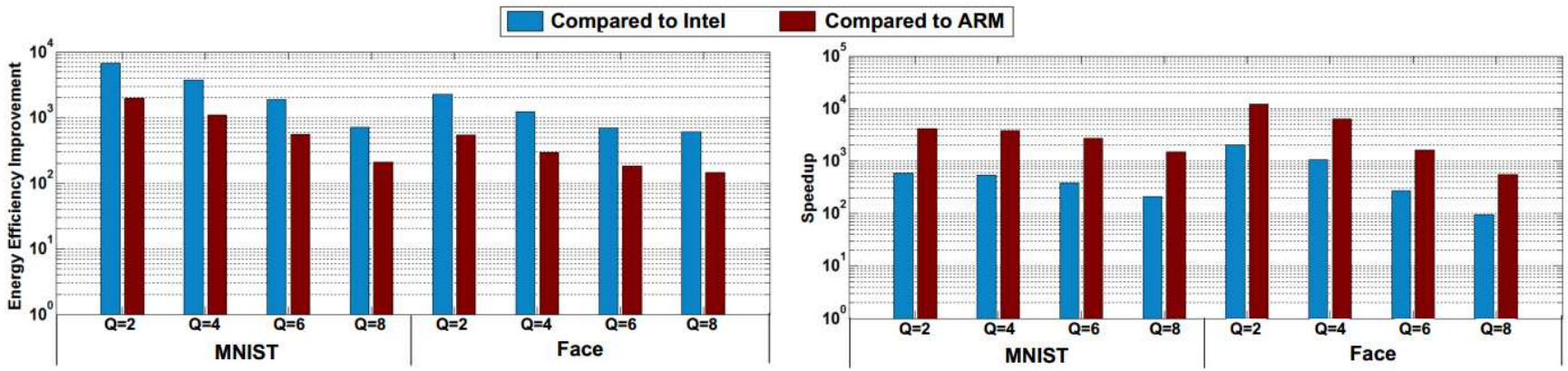
- Our design quantizes and encodes each pixel into Q levels
  - E.g. Q = 4, pixel values are quantized to 4 values, 00, 01, 10, and 11.
- Approximate in-memory feature extraction has minimal accuracy loss:
  - MNIST: only 0.4% (97.5%) for Q = 2 and L = 1024
  - WebFace: only 0.3 % error (96.7%) for Q = 6 and L = 2048



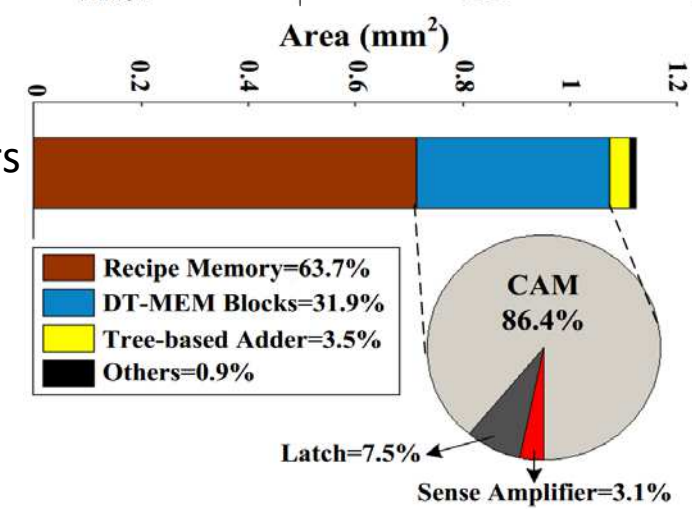


ORCHARD:
Energy & Performance Comparison [ICCAD'17]

- ORCHARD executing all the tasks inside memory:
  - **1896X** energy efficiency and **376X** speedup vs. Intel Xeon E5440
  - **552X** energy efficiency and **2654X** speedup vs. ARM Cortex A53



- Area:
  - The crossbar memory of two feature extractors takes 63.7% of the total area
  - Decision tree blocks are 31.9% of tot. area
  - Tree-based adder is 3.5% of tot. area



Summary

- IoT presents a whole new set of challenges, including needs for:
  - Scalable, reliable and maintainable systems
  - Strategies to learn in place, over a hierarchy of devices
  - Novel HW/SW designs that are orders of magnitude faster & energy efficient
- IoT reliability and maintainability
  - Reliability and maintainability can account for as much as 80% cost
  - System design has to be considered as a whole, across multiple time scales, ranging from ms to years
- Machine learning over hierarchy of devices
  - State of the art assumes all data is available in the cloud for processing – cannot scale for future IoT
  - Novel infrastructure has to be developed to trade off energy efficiency, privacy, with accuracy
- Learning acceleration:
  - Approximate computing reduces energy at the cost of computation inaccuracy
  - Processing in-memory enables both simple operations such as addition/multiplication and search; along with accelerating more complex applications such as DNN, graph processing
  - Hyperdimensional computing supports both efficient training and inference; it can be easily accelerated in HW,

Hardware Architectures for Efficient Machine Learning

Hun-Seok Kim

Assistant Professor

EECS, University of Michigan

hunseok@umich.edu

ELECTRICAL & COMPUTER ENGINEERING
UNIVERSITY OF MICHIGAN

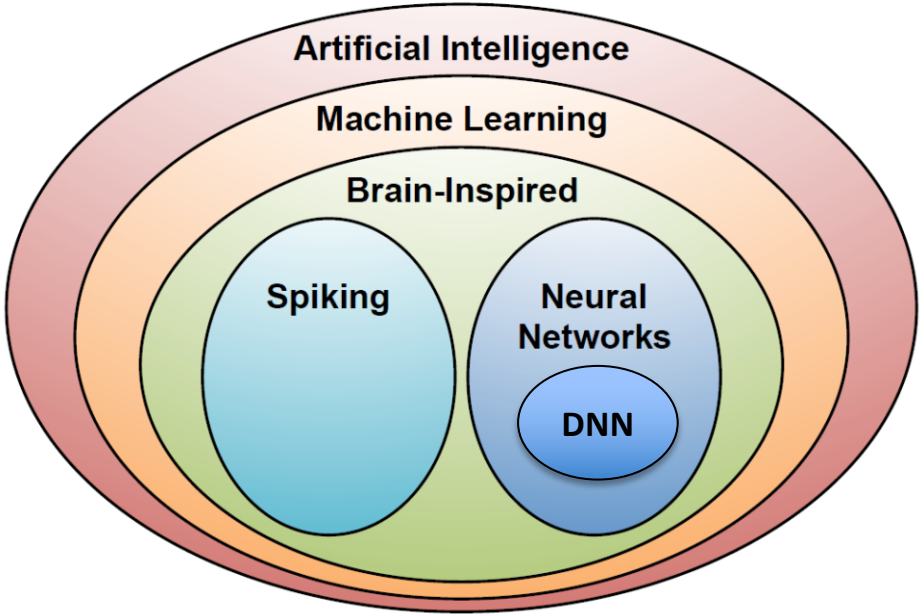
Takeaways

- Overview of machine learning in the context of DNNs
- Key HW architecture design considerations for DNNs
- Tradeoffs between various HW architectures
- Various optimization approaches for energy efficient DNN HW
- Recent HW architecture trends and opportunities for DNN

Outline

- Motivation: Energy efficient DNN
- Overview of DNNs
- Complexity breakdown of DNN
  - Computation and memory accesses patterns of DNN
- Techniques for energy efficient DNN architecture
- Conclusion

Deep Neural Networks (DNN)

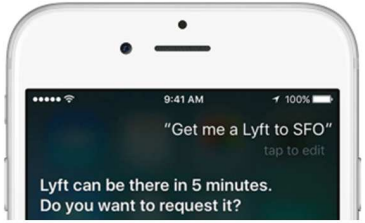


[V. Sze, Hardware Architectures for Deep Neural Networks, http://eyeriss.mit.edu/tutorial.html]

Hierarchical Deep Learning

- Mobile devices send data
- Servers compute Deep Neural Networks (DNNs) for AI

→ *Latency & energy overhead due to communication*



Security and privacy concerns

Hierarchical Deep Learning

Always-on IoT devices

Low-power Wireless Comm. (BLE 20mW)

Portable devices

Internet Connection (WiFi 300mW)

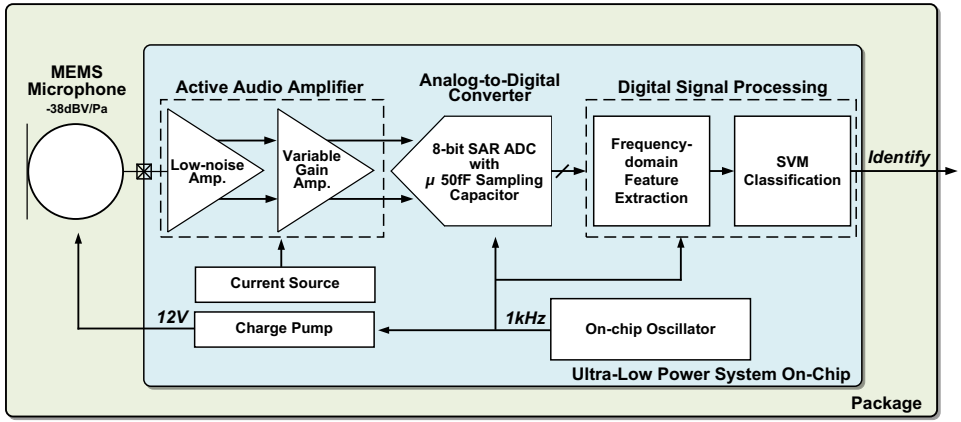
Small DNN	*Medium DNN*	*Large DNN*
Power budget *< BLE power (20mW)*	*Power budget* *< WiFi power (300mW)*	*Power budget* *not limited*
• Time-critical, simple tasks • Send data to high performance machines to process more complicated tasks		• Complicated tasks

Objective: ***Save Energy & Improve Latency***

Enhanced security and privacy

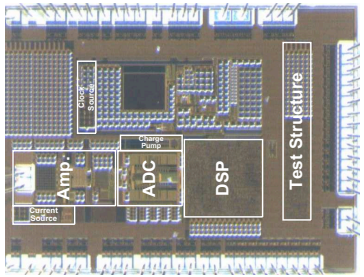
Ultra-Low Power (ULP) DNNs

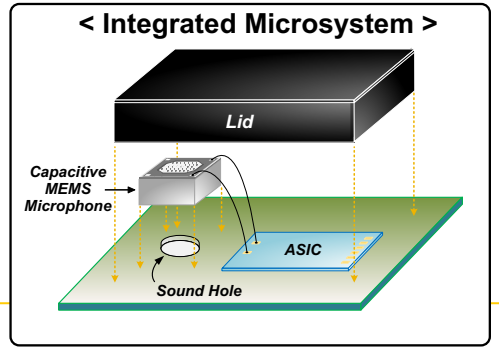
- Intelligence on always-on, energy autonomous, and unattended sensors
- DARPA Near-Zero power acoustic sensing platform program

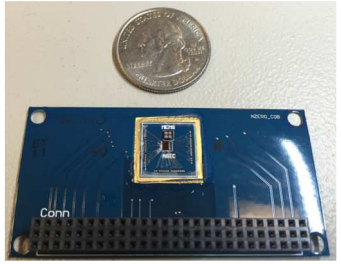


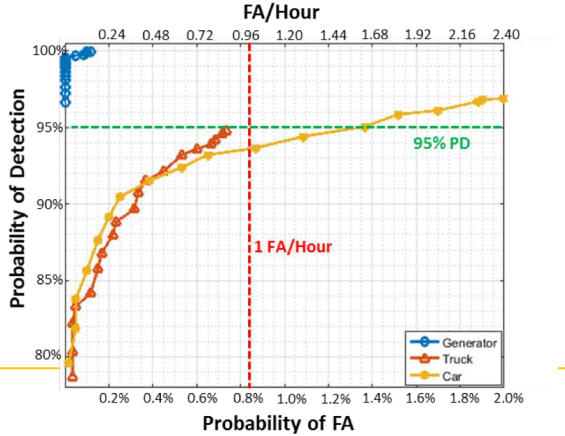
[S. Jeong et al., ISSCC2017]

- **20nW active power** for machine type detection (electric generator, car, truck, etc.) via acoustic sensing DNN
- Single chip ASIC solution: analog frontend and custom fully connected neural network processor backend





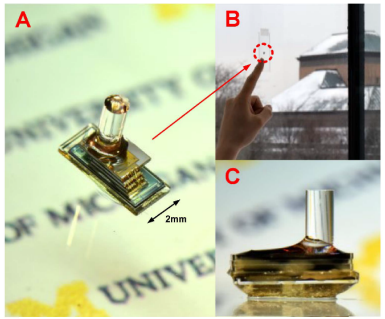




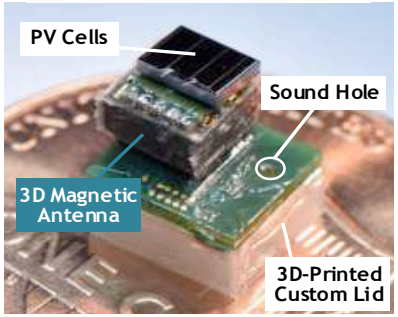
Ultra-Low Power (ULP) DNNs

- **Intelligence on millimeter-scale IoT platforms**

Die-stacked IoT system
[VLSIC 2014, Kim]



[VLSIC 2017, Cho]



Scope of "Smart IoT node"

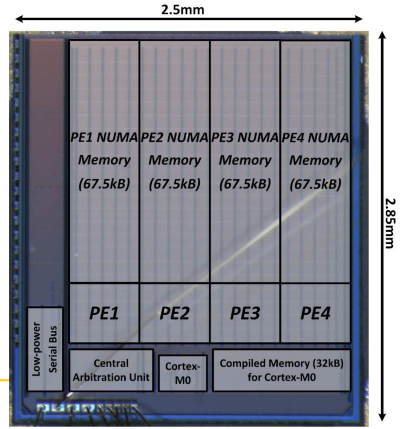
- **Mobile intelligence**
- **Simple classification tasks (DNN)**
  - **Speech/Image detection**
  - **Offline training**
- **DSP tasks (e.g. FFT)**

Proposed Techniques

- **Parallel processing elements (PEs)**
- **Non-uniform memory architecture (NUMA)**
- **Custom low-leakage SRAM**
- **Configurable data precision**

[S. Bang et al., ISSCC2017]

- **300µW active power** deep neural network processor
- Face detection, keyword recognition, FFT, etc. on millimeter-scale Michigan Micro Mote M^3 platforms



Outline

- Motivation: Energy efficient DNN
- Overview of DNN
- Complexity breakdown of DNN
  - Computation and memory accesses patterns of DNN
- Techniques for energy efficient DNN architecture
- Conclusion

Deep Neural Networks (DNN) Overview

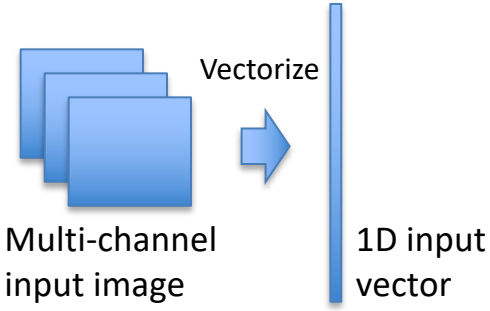
- DNN: $\boldsymbol{y} = f(\boldsymbol{x})$
  - $f(\boldsymbol{x}) = \sigma_L\left(W_L\sigma_{L-1}\left(W_{L-1} \dots \sigma_2\left(W_2\sigma_1(W_1\boldsymbol{x})\right) \dots\right)\right)$
  - Each layer of DNN consists of
    - **Linear function: $\boldsymbol{W_i x}$**
    - **Non-linear activation function: $\boldsymbol{\sigma_i(\quad)}$**
  - Deep: L >> 1

Deep Neural Networks (DNN) Overview

- DNN: $\boldsymbol{y} = f(\boldsymbol{x})$
  - $f(\boldsymbol{x}) = \sigma_L\left(W_L\sigma_{L-1}\left(W_{L-1} \dots \sigma_2\left(W_2\sigma_1(W_1\boldsymbol{x})\right) \dots\right)\right)$
  - Each layer of DNN consists of
    - **Linear function: $\boldsymbol{W_i x}$**
    - **Non-linear activation function: $\boldsymbol{\sigma_i}(\quad)$**
  - Deep: L >> 1
- Training a DNN
  - Supervised learning: numerically optimize and obtain 'weights' W_L, W_{L-1}, … W_1 that minimize a cost function $\sum_n \mathrm{C}\left(f\left(\boldsymbol{x}_{train}^{(n)}\right), \mathbf{y}_{train}^{(n)}\right)$ using training dataset $(\boldsymbol{x}_{train}^{(n)}, \boldsymbol{y}_{train}^{(n)})$, n =1, …, N

Linear function in DNN

- $W_i \boldsymbol{x}$
  - Matrix-vector multiplication: Fully Connected Layer (FCL)



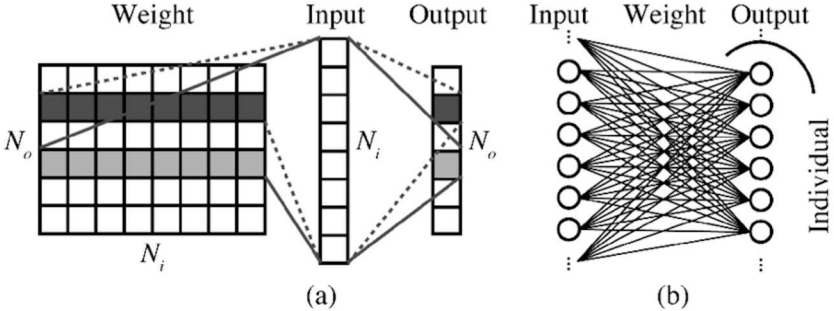
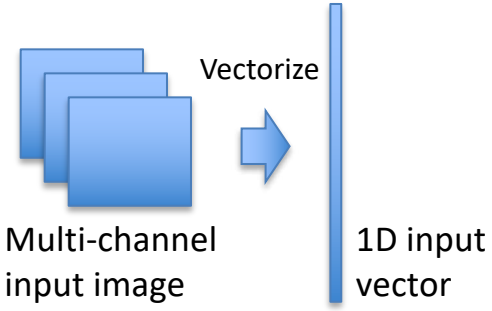


Diagram from [K. Ando, Circuits and Systems 2018]

Linear function in DNN

- $W_i \boldsymbol{x}$
  - Matrix-vector multiplication: Fully Connected Layer (FCL)



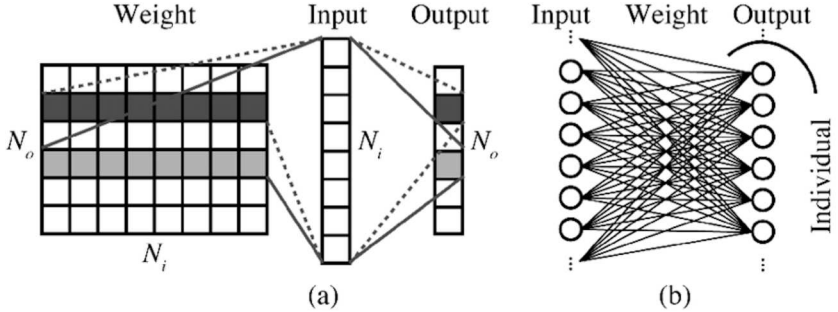
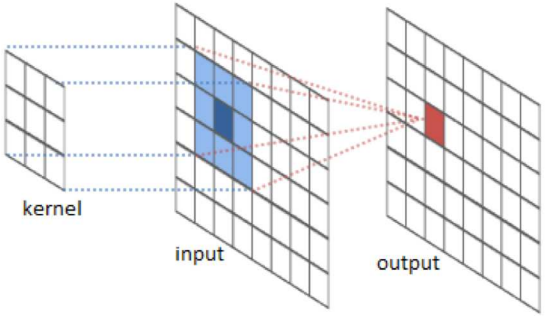


Diagram from [K. Ando, Circuits and Systems 2018]

  - Convolution Layer



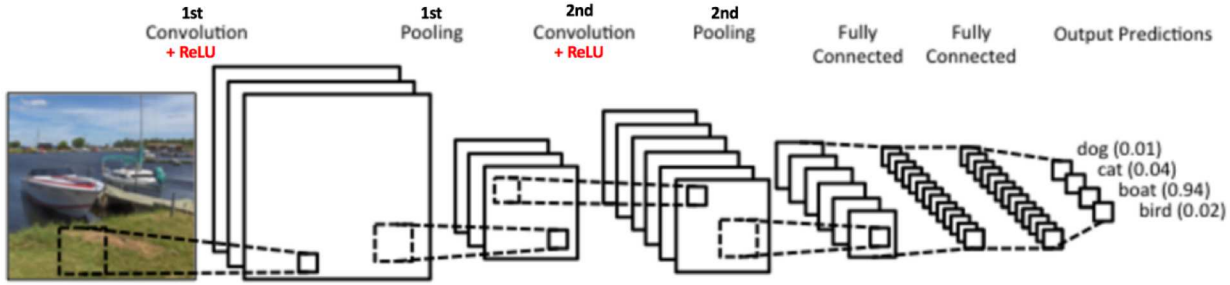
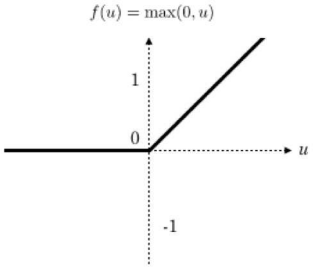


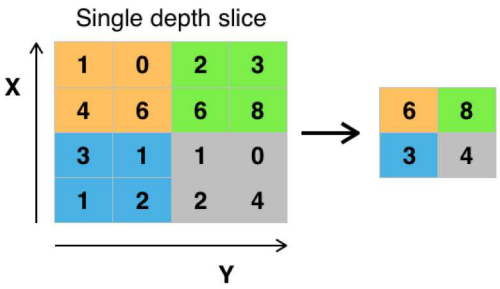
Diagram from [https://www.clarifai.com/technology]

Non-linear activation function in DNN

- $\boldsymbol{\sigma_i}(W_i\boldsymbol{x}\ )$
  - ReLu, performed element-wise



  - Max-pooling



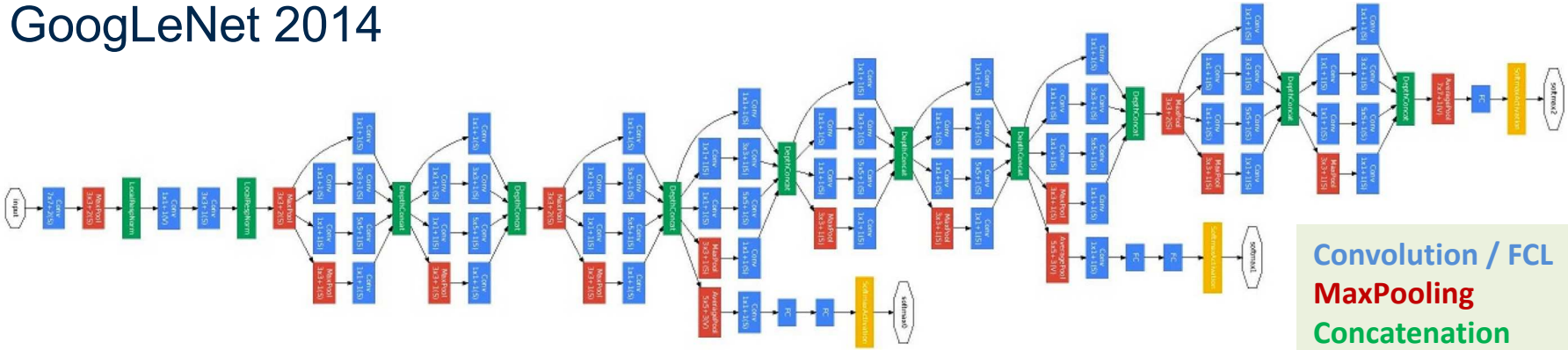
  - Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

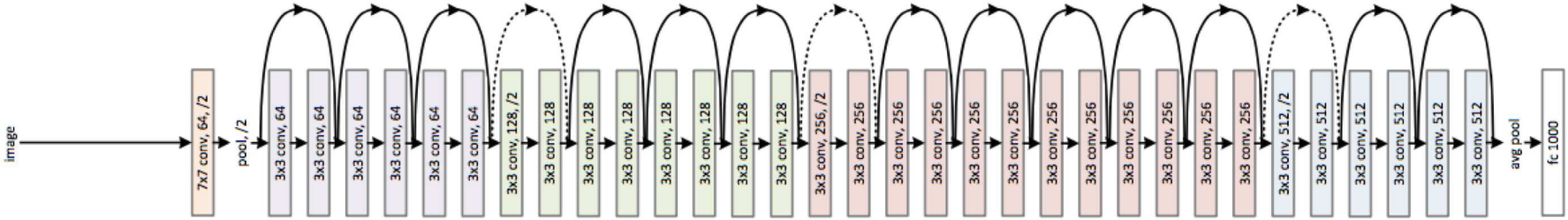
- Much lower complexity compared linear function $W_i\boldsymbol{x}$

Deep Convolutional Neural Network (CNN)

- GoogLeNet 2014

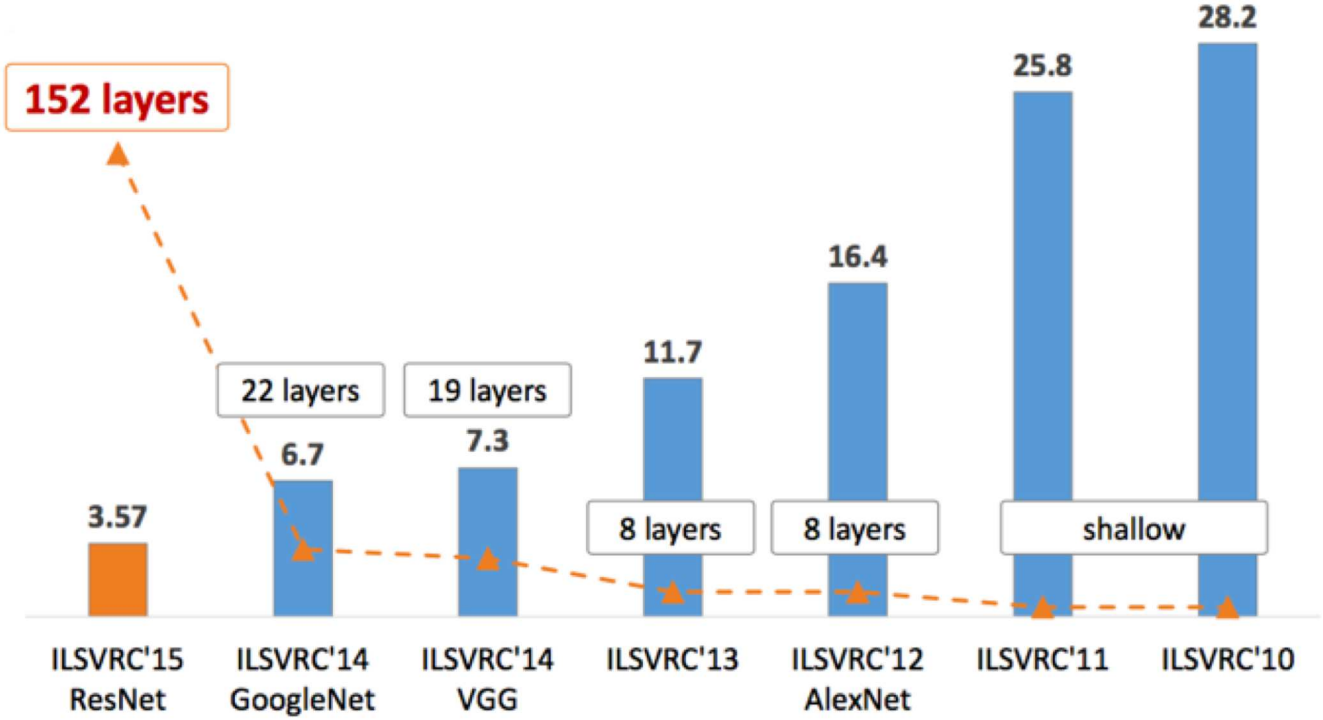


- ResNet 2015



Deep Convolutional Neural Network (CNN)

- ImageNet Top 5 Classification Errors



[Russakovsky et al., IJCV 2015]

Outline

- Motivation: Energy efficient DNN
- Overview of DNN
- Complexity breakdown of DNN
  - Computation and memory accesses patterns of DNN
- Techniques for energy efficient DNN architecture
- Conclusion

Complexity of a Fully Connected Layer

- N_ix1 input vector, N_oxN_i weights
- Multiplication and accumulation (MAC)
  - N_oxN_i
- Memory access
  - Weight: N_oxN_i reads, **1 read per weight**
  - Input: N_oxN_i reads, **N_o reads per input**
- # of MAC and memory accesses: same order
- Weight sizes of various DNNs' FCL
  - GoogLeNet: 1million → 4MB w/ single precision
  - AlexNet: 58.6 million → 58.6MB
  - VGG16: 144 million → 144MB

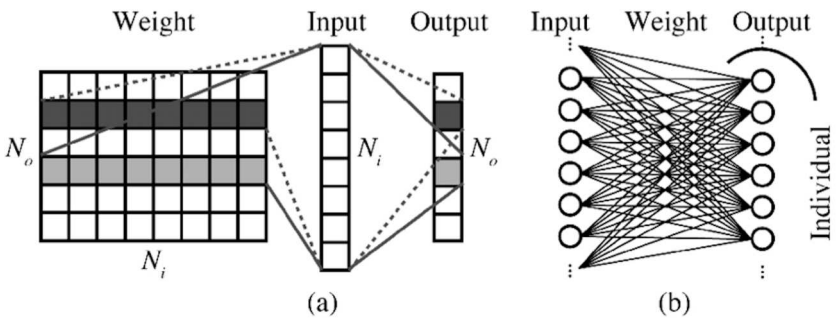
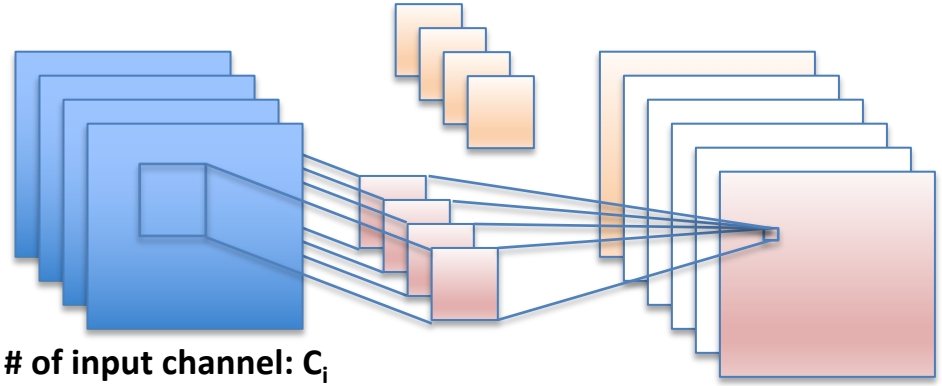


Diagram from [K. Ando, Circuits and Systems 2018]

Intel i7-920 on-chip SRAM cache size: 8MB

Complexity of a Convolutional Layer

- Dimension of input/output channel
  - M x N
- Dimension of each conv kernel
  - K x K
- # of input channels: C_i
- # of output channels: C_o
- # of conv kernels: C_i x C_o
- Weight storage memory (per layer)
  - K^2 x C_i x C_o
- MAC operations (per layer)
  - K^2 x C_i x C_o x M x N
- **# of MAC >> # of memory accesses**
  - **Architecture dependent!**

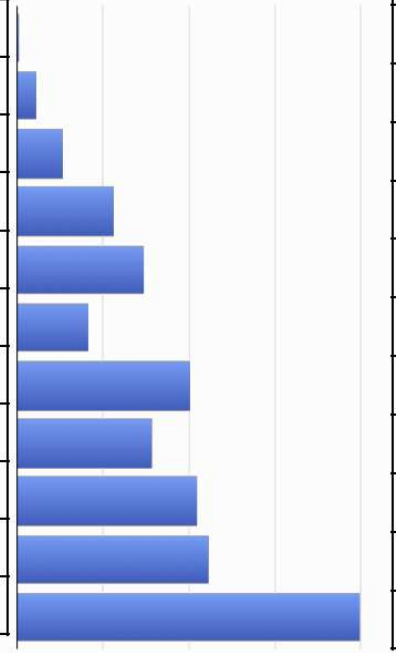


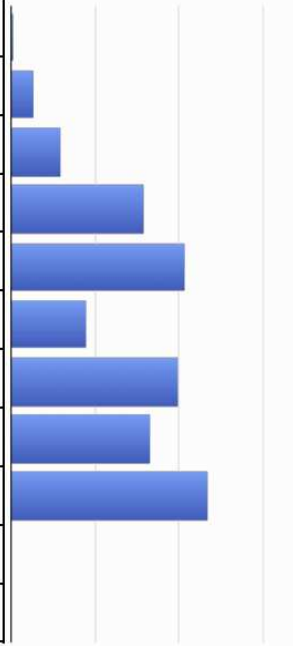
Dimension of each input/output channel: M x N
Dimension of each conv. kernel: K x K

State-of-the-art DNNs (e.g., ResNet)
- **Kernel size K: 1, 3, 5, 7**
- **# of channels per layer: thousands**
- **# of unique kernels per layer: thousands**
- **Total # of weights: 10s of millions**
- **Total # of MACs: 10s of billions**

Complexity to Energy Mapping

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A

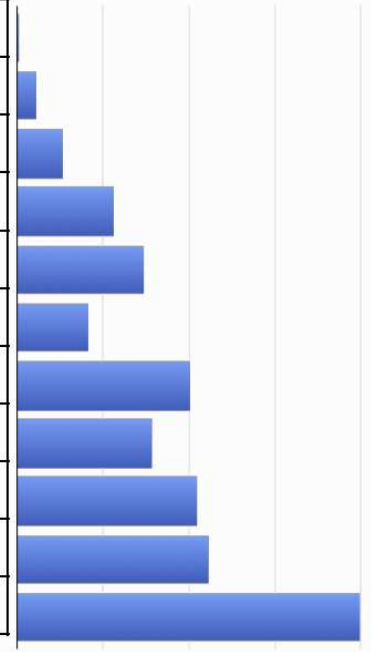


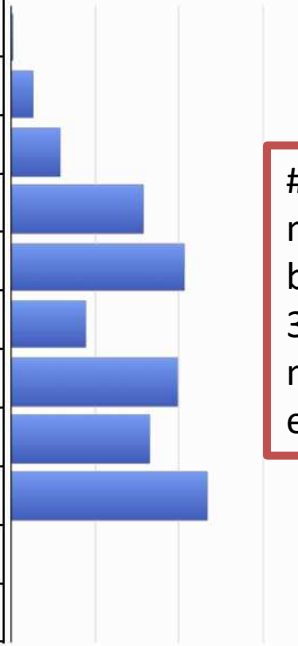


[Horowitz, “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014]

Complexity to Energy Mapping

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





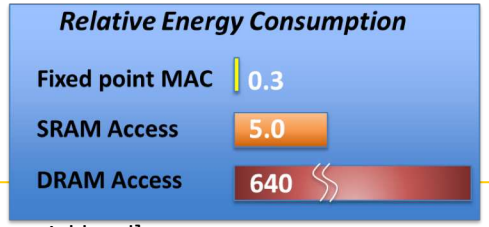
# of MAC >> # of memory accesses but DRAM access is 3 – 4 orders of magnitude more expensive!

[Horowitz, “Computing’s Energy Problem (and what we can do about it)”, ISSCC 2014]

Complexity of Popular DNNs

Deeper →

		LeNet5	AlexNet	VGG16	GoogLeNet	ResNet-50
ImageNet Top5 Error %		N/A	16.4	7.4	6.7	5.3
# of layers	Conv	2	5	16	21	49
	FCL	2	3	3	1	1
# of weights	Conv	2.6k	2.3M	14.7M	6.0M	23.5M
	FCL	58k	58.6M	124M	1M	2M
# of MACs	Conv	283k	666M	15.3G	1.43G	3.86G
	FCL	2.6k	2.3M	14.7M	6.0M	23.5M
Total Weights		60k	61M	138M	7M	25.5M
Total MACs		341k	724M	15.5G	1.43G	3.9G



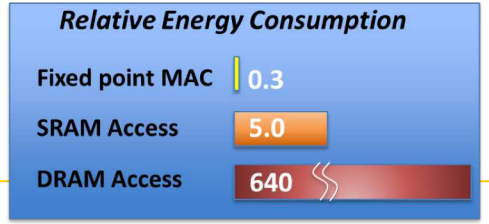
[V. Sze, Hardware Architectures for Deep Neural Networks, http://eyeriss.mit.edu/tutorial.html]

Complexity of Popular DNNs

Deeper →

		LeNet5	AlexNet	VGG16	GoogLeNet	ResNet-50
ImageNet Top5 Error %		N/A	16.4	7.4	6.7	5.3
# of layers	Conv	2	5	16	21	49
	FCL	2	3	3	1	1
# of weights	Conv	2.6k	2.3M	14.7M	6.0M	23.5M
	FCL	58k	58.6M	124M	1M	2M
# of MACs	Conv	283k	666M	15.3G	1.43G	3.86G
	FCL	2.6k	2.3M	14.7M	6.0M	23.5M
Total Weights		60k	61M	138M	7M	25.5M
Total MACs		341k	724M	15.5G	1.43G	3.9G

- **Shallower networks' complexity dominated by FCL**

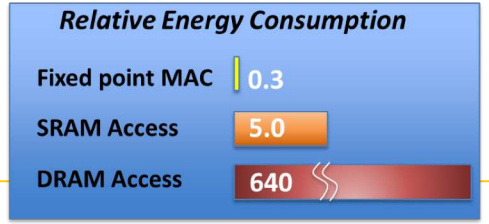


Complexity of Popular DNNs

Deeper →

		LeNet5	AlexNet	VGG16	GoogLeNet	ResNet-50
ImageNet Top5 Error %		N/A	16.4	7.4	6.7	5.3
# of layers	Conv	2	5	16	21	49
	FCL	2	3	3	1	1
# of weights	Conv	2.6k	2.3M	14.7M	6.0M	23.5M
	FCL	58k	58.6M	124M	1M	2M
# of MACs	Conv	283k	666M	15.3G	1.43G	3.86G
	FCL	2.6k	2.3M	14.7M	6.0M	23.5M
Total Weights		60k	61M	138M	7M	25.5M
Total MACs		341k	724M	15.5G	1.43G	3.9G

- Shallower networks' complexity dominated by FCL
- Deeper networks' complexity dominated by convolution layer



NN Applications Workload on Google TPU

- [N P. Jouppi et al., ISCA 2017]
  - Google TPU in July 2016 workload is mostly FCL

Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% of Deployed TPUs in July 2016*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	100	5				5	ReLU	20M	200	200	61%
MLP1	1000	4				4	ReLU	5M	168	168	
LSTM0	1000	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1500	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1000		16			16	ReLU	8M	2888	8	5%
CNN1	1000	4	72		13	89	ReLU	100M	1750	32	

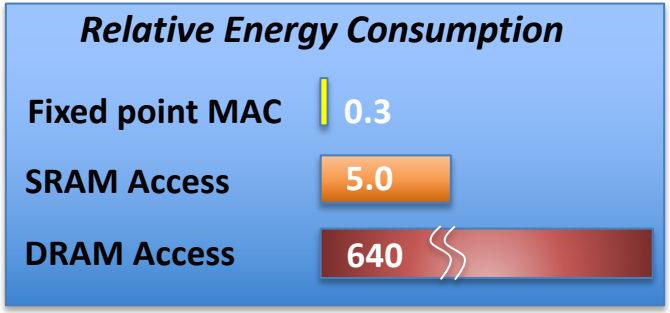
Table 1. Six NN applications (two per NN type) that represent 95% of the TPU's workload. The columns are the NN name; the number of lines of code; the types and number of layers in the NN (FC is fully connected, Conv is convolution, Vector is self-explanatory, Pool is pooling, which does nonlinear downsizing on the TPU; and TPU application popularity in July 2016. One DNN is RankBrain [Cla15]; one LSTM is a subset of GNM Translate [Wu16]; one CNN is Inception; and the other CNN is DeepMind AlphaGo [Sil16][Jou15].

Outline

- Motivation: Energy efficient DNN
- Overview of DNN
- Complexity breakdown of DNN
  - Computation and memory accesses patterns of DNN
- Techniques for energy efficient DNN architecture
- Conclusion

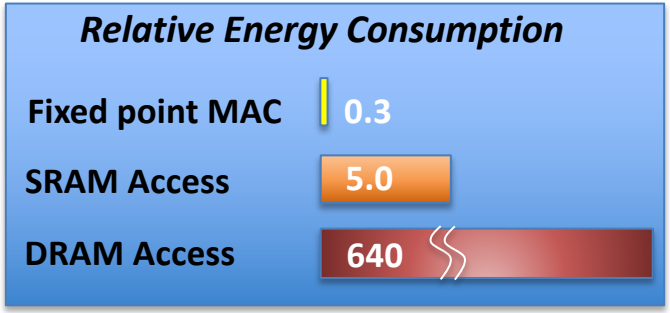
Techniques for Energy Efficient DNN

- Energy-optimized HW architecture for DNN
  - Exploit data locality
    - Data reuse for convolution and FCL
    - More (less) frequent local (remote) accesses



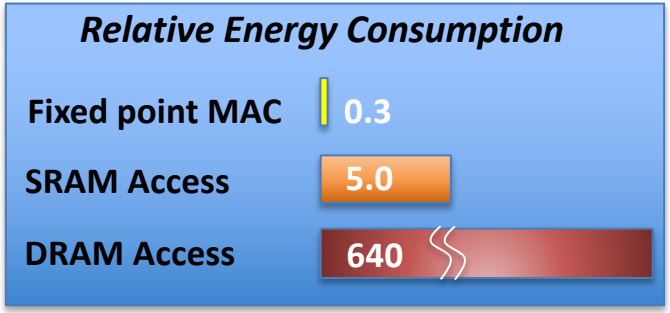
Techniques for Energy Efficient DNN

- Energy-optimized HW architecture for DNN
  - Exploit data locality
    - Data reuse for convolution and FCL
    - More (less) frequent local (remote) accesses
  - Parallelism
    - Voltage and frequency scaling



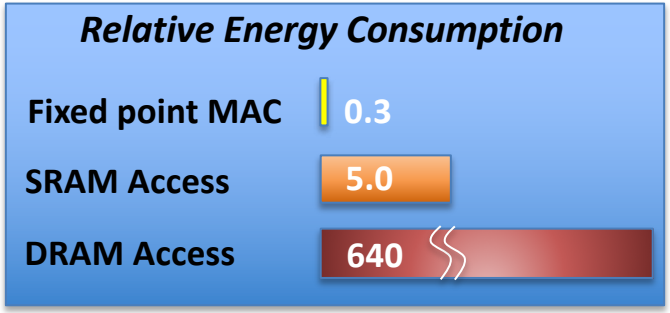
Techniques for Energy Efficient DNN

- Energy-optimized HW architecture for DNN
  - Exploit data locality
    - Data reuse for convolution and FCL
    - More (less) frequent local (remote) accesses
  - Parallelism
    - Voltage and frequency scaling
  - Weight precision reduction
    - Less on-chip memory access and lower power MAC
  - Weight compression
    - Minimal or no off-chip DRAM access



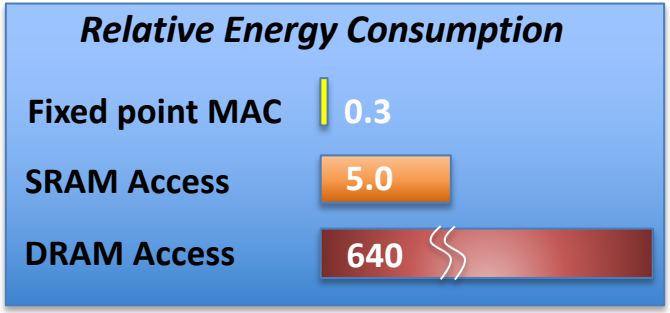
Techniques for Energy Efficient DNN

- Energy-optimized HW architecture for DNN
  - Exploit data locality
    - Data reuse for convolution and FCL
    - More (less) frequent local (remote) accesses
  - Parallelism
    - Voltage and frequency scaling
  - Weight precision reduction
    - Less on-chip memory access and lower power MAC
  - Weight compression
    - Minimal or no off-chip DRAM access
  - Exploit sparsity
    - Significant portion of activations and weights are zero



Techniques for Energy Efficient DNN

- Energy-optimized HW architecture for DNN
  - Exploit data locality
    - Data reuse for convolution and FCL
    - More (less) frequent local (remote) accesses
  - Parallelism
    - Voltage and frequency scaling
  - Weight precision reduction
    - Less on-chip memory access and lower power MAC
  - Weight compression
    - Minimal or no off-chip DRAM access
  - Exploit sparsity
    - Significant portion of activations and weights are zero
  - In-memory compute using specialized memories



Data Reuse in DNN

- Data reuse leads to more local (RF, smaller SRAM) accesses
- Input (image) reuse, weight reuse

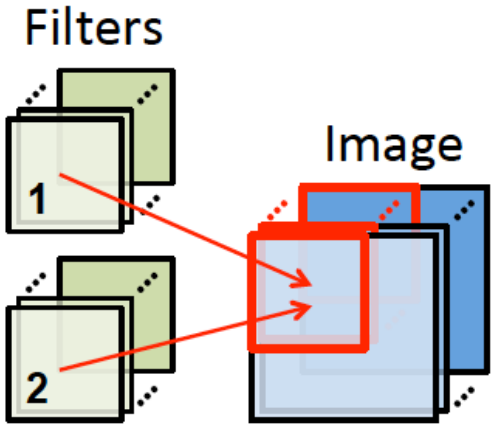
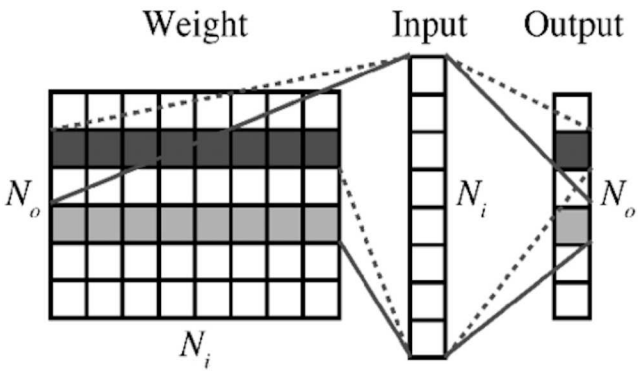
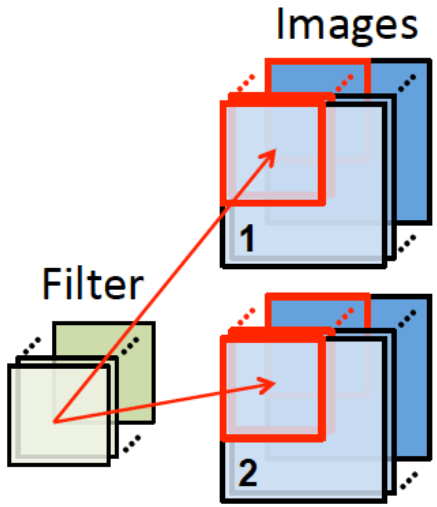


Image Reuse
(pixels)



Input Reuse in FCL



Filter Reuse
(weights)

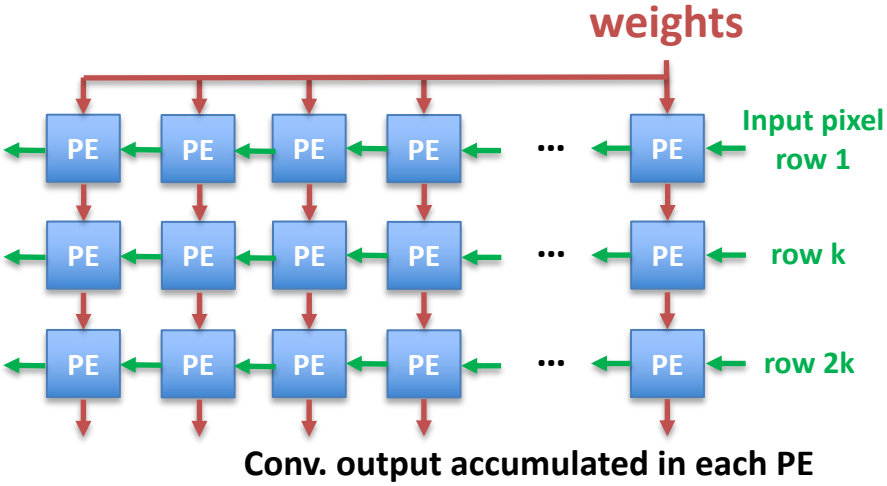
Diagrams from [V. Sze, Hardware Architectures for Deep Neural Networks, http://eyeriss.mit.edu/tutorial.html] & [K. Ando, Circuits and Systems 2018]

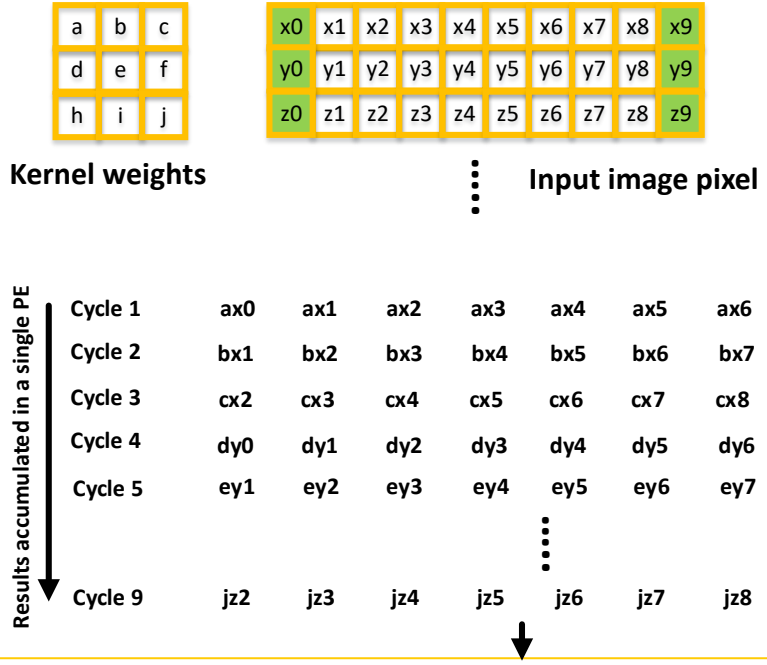
Inefficiency of General Purpose Processors for DNN

- Inefficiency of GPPs and GPUs for DNN
  - Not enough parallelism (GPP)
  - Datapath not specifically designed for DNN data reuse patterns
  - Unbalanced computation vs. memory BW
  - Excessive bit precision for MAC and weight storage
  - Excessive overhead for compression/decompression of weights
  - Unable to exploit sparsity well

Multi-PE Architecture for DNN

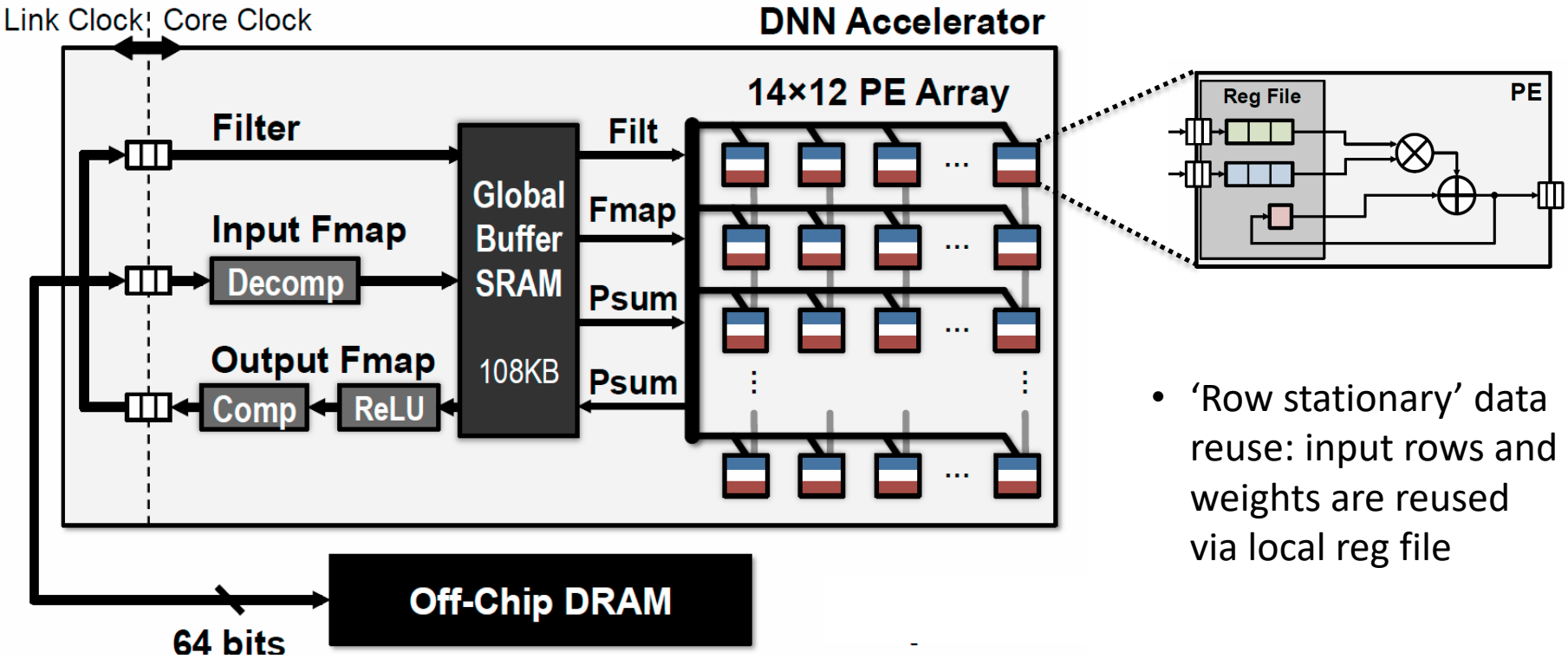
- Parallel PE (Processing Elements) based systolic architecture
- Local memory and/or local communication among PEs
- Weight and input reuse architecture
- **Enables parallelism + data reuse**





Multi-PE Architecture Prior Works

- Eyeriss [Chen et al., ISSCC2016], [Chen et al., ISCA 2016]

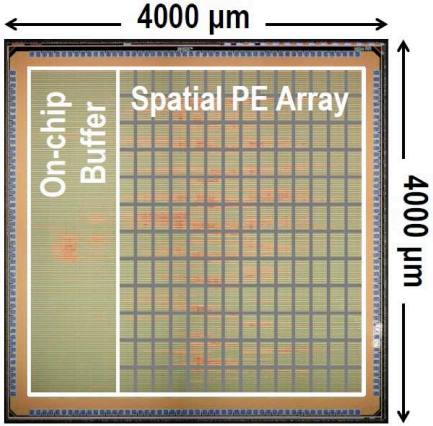


- 'Row stationary' data reuse: input rows and weights are reused via local reg file

Multi-PE Architecture Prior Works

- Eyeriss [Chen et al., ISSCC2016], [Chen et al., ISCA 2016]

Technology	TSMC 65nm LP 1P9M
Core Area	3.5mm×3.5mm
Gate Count	1852 kGates (NAND2)
On-Chip Buffer	108 KB
# of PEs	168
Scratch Pad / PE	0.5 KB
Supply Voltage	0.82 – 1.17 V
Core Frequency	100 – 250 MHz
Peak Performance	33.6 – 84.0 GOPS (2 OP = 1 MAC)
Word Bit-width	16-bit Fixed-Point
Filter Size*	1 – 32 [width] 1 – 12 [height]
# of Filters*	1 – 1024
# of Channels*	1 – 1024
Stride Range	1–12 [horizontal] 1, 2, 4 [vertical]



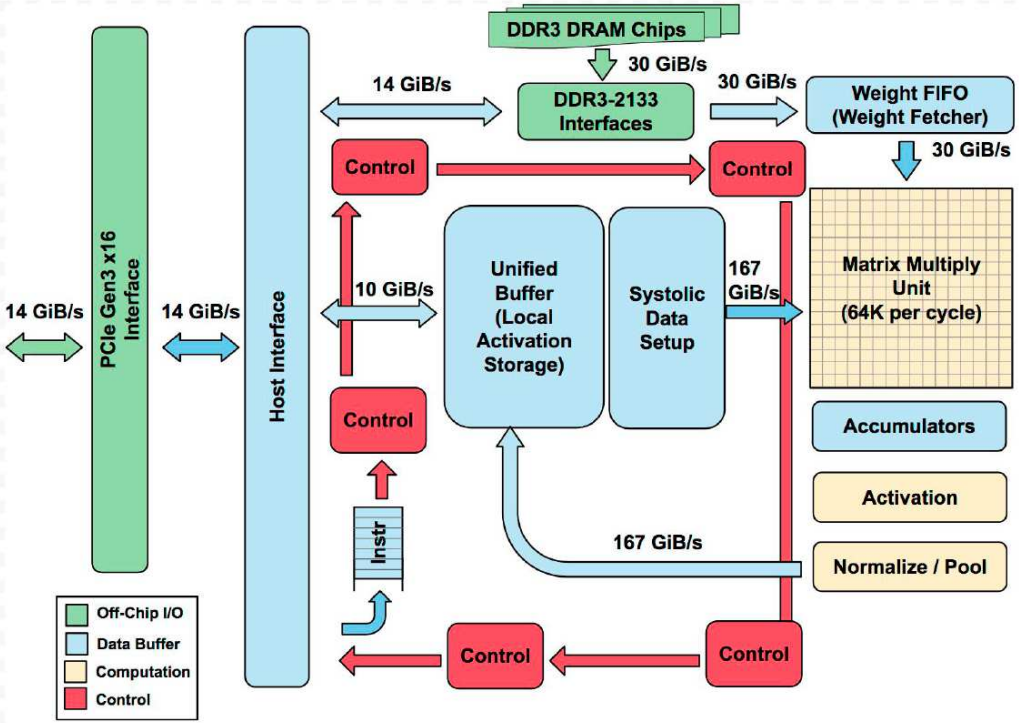
* Natively Supported

	This Work	NVIDIA TK1 (Jetson Kit)
Technology	65nm	28nm
Clock Rate	200MHz	852MHz
# Multipliers	168	192
On-Chip Storage	Buffer: 108KB Spad: 75.3KB	Shared Mem: 64KB Reg File: 256KB
Word Bit-Width	16b Fixed	32b Float
Throughput[1]	34.7 fps	68 fps
Measured Power	278 mW	Idle/Active[2]: 3.7W/10.2W
DRAM Bandwidth	127 MB/s	1120 MB/s [3]

1. AlexNet Convolutional Layers Only
2. Board Power
3. Modeled from [Tan, SC11]

Multi-PE Architecture Prior Works

- Google TPU [N P. Jouppi et al., ISCA 2017]



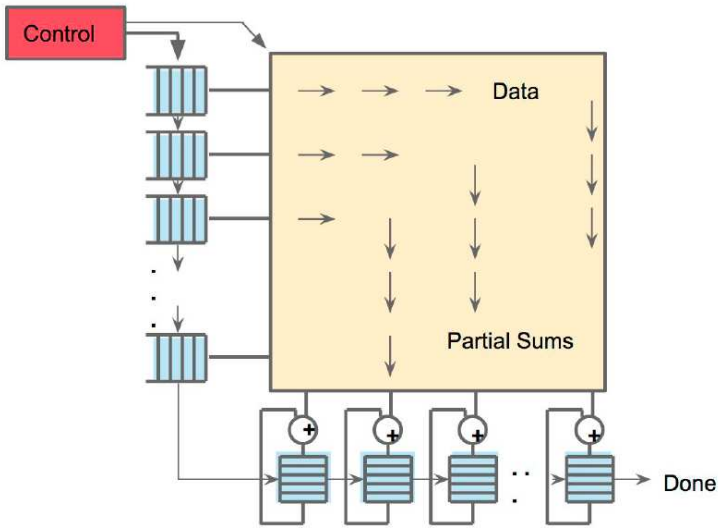


Figure 4. Systolic data flow of the Matrix Multiply Unit. Software has the illusion that each 256B input is read at once, and they instantly update one location of each of 256 accumulator RAMs.

Multi-PE Architecture Prior Works

- Google TPU [N P. Jouppi et al., ISCA 2017]

Model	Die										Benchmarked Servers				
	mm^2	nm	MHz	TDP	Measured		TOPS/s		GB/s	On-Chip Memory	Dies	DRAM Size	TDP	Measured	
					Idle	Busy	8b	FP						Idle	Busy
Haswell E5-2699 v3	662	22	2300	145W	41W	145W	2.6	1.3	51	51 MiB	2	256 GiB	504W	159W	455W
NVIDIA K80 (2 dies/card)	561	28	560	150W	25W	98W	--	2.8	160	8 MiB	8	256 GiB (host) + 12 GiB x 8	1838W	357W	991W
TPU	NA*	28	700	75W	28W	40W	92	--	34	28 MiB	4	256 GiB (host) + 8 GiB x 4	861W	290W	384W

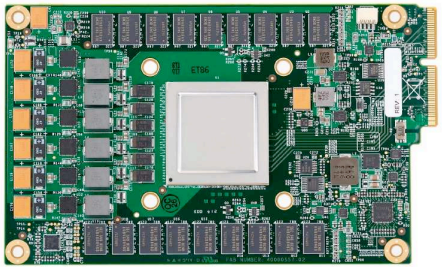
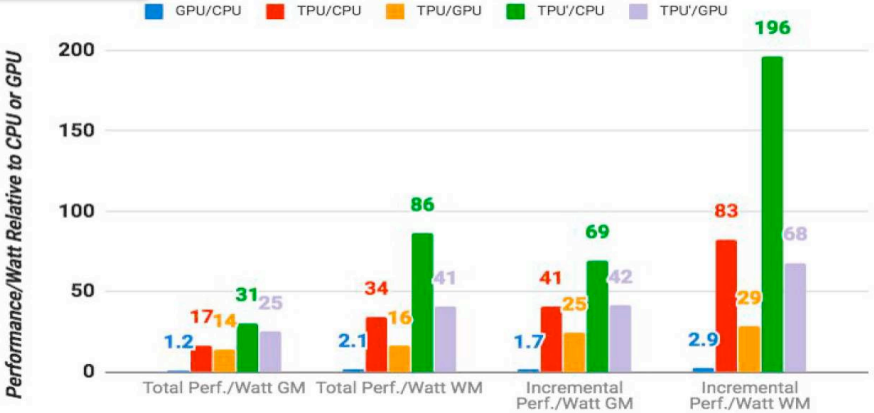
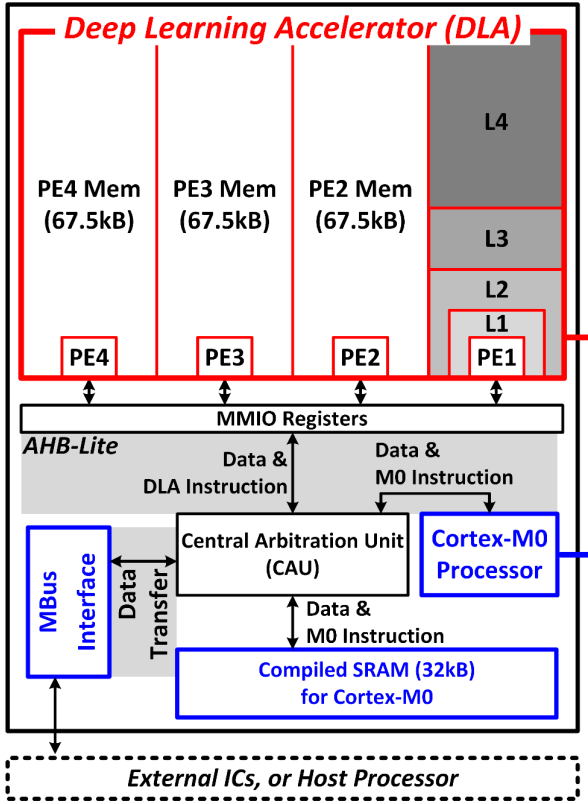


Figure 3. TPU Printed Circuit Board. It can be inserted in the slot for an SATA disk in a server, but the card uses PCIe Gen3 x16.



ULP NN Processor Exploiting Data Locality [S. Bang, ISSCC2017]

- 300uW Deep Learning Accelerator for mm-scale IoT
- [S. Bang et al., ISSCC 2017]



NUMA Memory for a Single PE

	L1	L2	L3	L4
Unit bank (kB)	0.375	1.5	3.0	12.0
# Banks	4	4	4	4
Total size (kB)	1.5	6.0	12.0	48.0

DLA PE instructions

- Fully-connected layer (FCL) → main focus
- FFT
- Data block move
- Non-linear activation function (LUT-based)

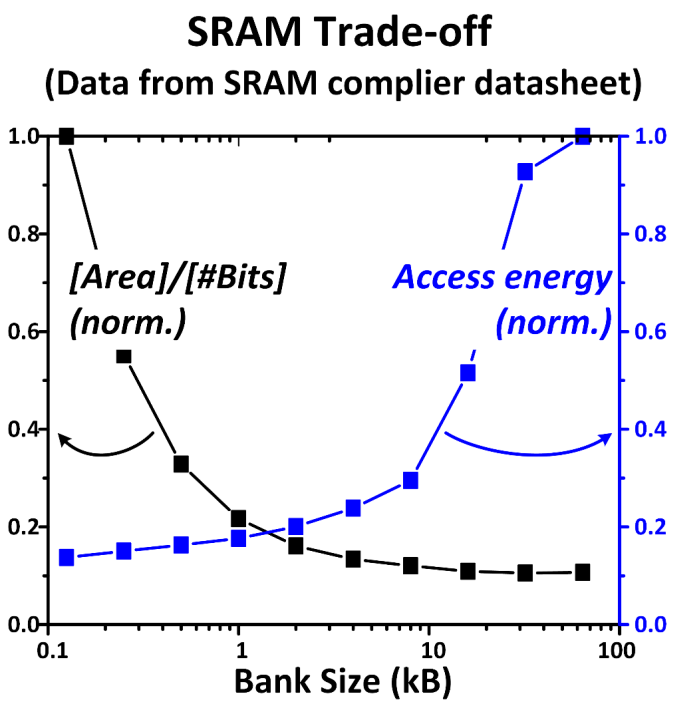
ARM Cortex-M0

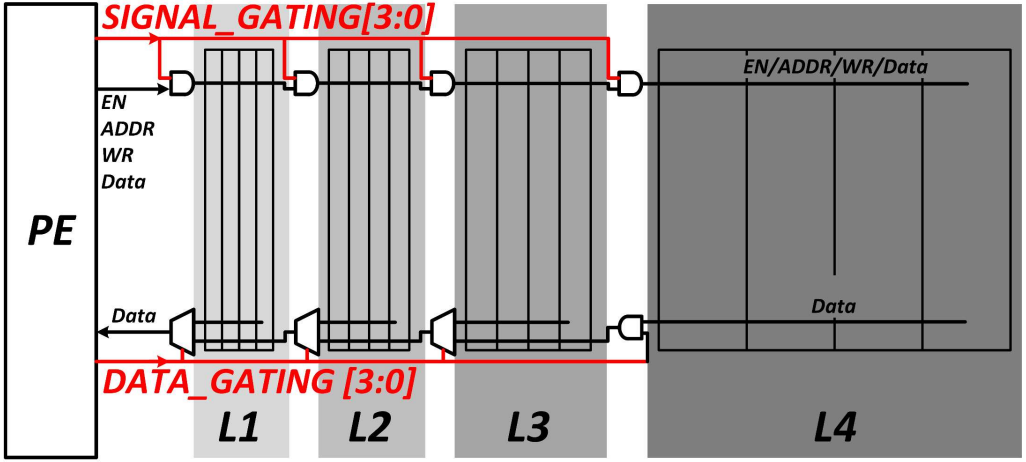
- Load instruction of DLA PEs
- Load data of DLA PEs
- Control DLA operation sequence

MBus Interface

- Low-power protocol for sensor system

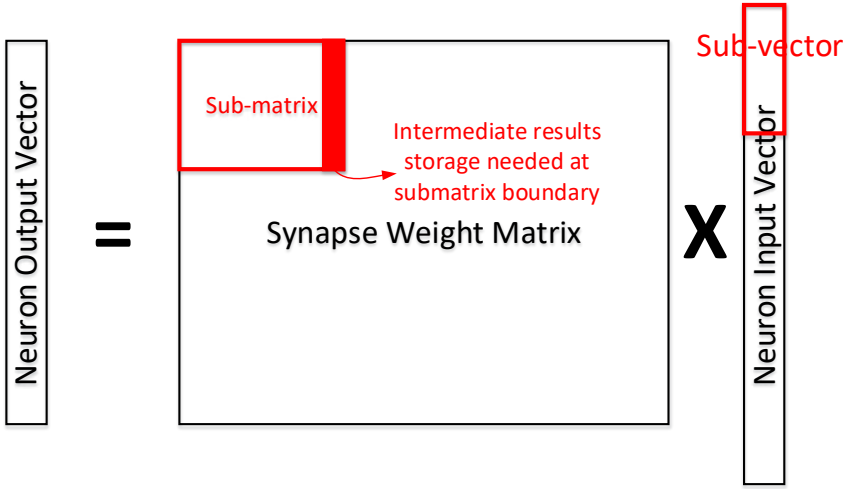
ULP NN Processor Exploiting Data Locality [S. Bang, ISSCC2017]





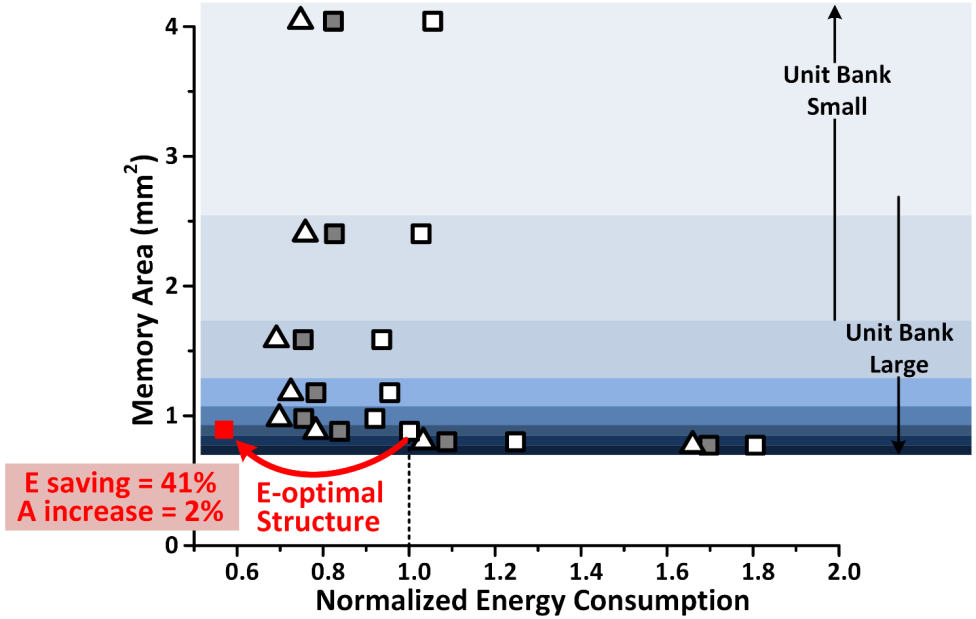
ULP NN Processor Exploiting Data Locality [S. Bang, ISSCC2017]

- Fully Connected Layer on Non-Uniform Memory Architecture (NUMA)
- FCL partitioned to smaller sub-matrices to maximize data locality



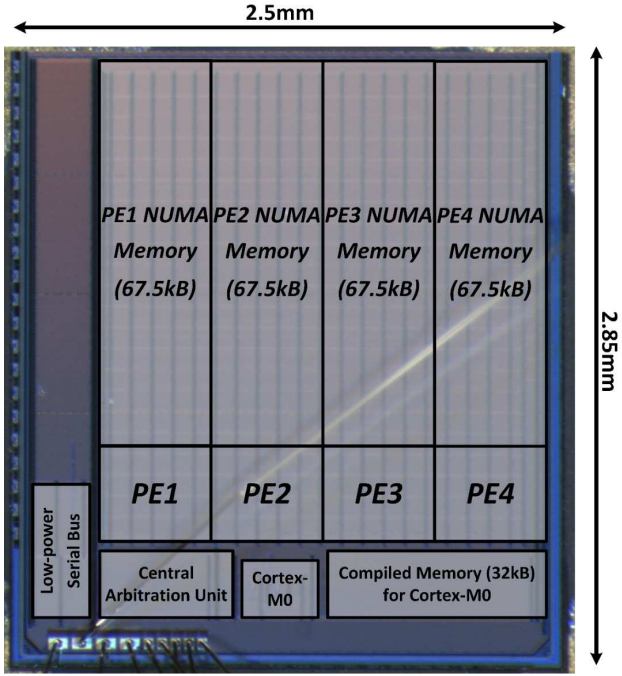
Symbol	Memory	Locality	Tiling	No. PE
□	Uniform	No	No	1
■	Uniform	Yes	No	1
△	Uniform	Yes	Yes	4
■	NUMA	Yes	Yes	4

*Total memory = 256kB
*Uniform memory structures
*Non-uniform memory structure (optimal)



ULP NN Processor Exploiting Data Locality [S. Bang, ISSCC2017]

- ASIC fabricated in TSMC 40nm LP process



App.	Layer*	#PE in use	Weight matrix	Precision (bit)				Power (μW)	Latency (msec)	GOPS/W
				W	I	O	TO			
General	1st	4	384 × 408	6	12	12	32	288	2.97	374

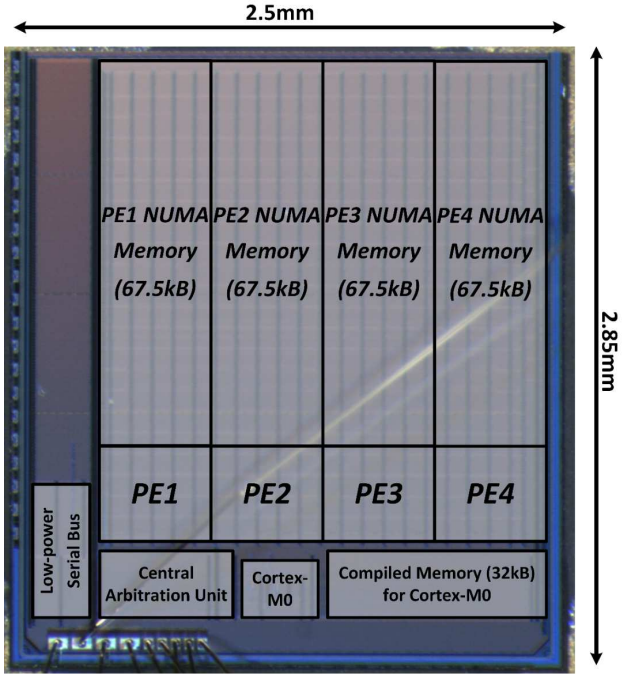
App.	Layer*	#PE in use	Weight matrix	Precision (bit)				Power (μW)	Latency (msec)	GOPS/W
				W	I	O	TO			
Keyword spotting	1st	4	384 × 408	6	12	12	32	360	2.98	316
	2nd	4	384 × 384	6	12	12	32	312	2.80	365
	3rd	1	24 × 384	6	12	12	32	189	0.72	140
Total								321	6.50	318
Face detection	1st	1	16 × 1032	8	8	16	32	208	1.70	113
	2nd	1	2 × 16	8	16	16	32	195	0.02	112
Total								208	1.72	113

*Fully-connected layer

Measured @ 0.65V and 3.9MHz

ULP NN Processor Exploiting Data Locality [S. Bang, ISSCC2017]

- ASIC fabricated in TSMC 40nm LP process



	This work	ISSCC 2016, Chen	VLSI 2016, Moons
Process	40nm	65nm	40nm
Chip area	7.1mm^2	16mm^2	2.4mm^2
Design target	Fully-connected layer & FFT	Convolutional layer	Convolutional layer
V_{DD}	0.63 – 0.9 V	0.82 – 1.17 V	0.55 – 1.1 V
Frequency	1.9 – 19.3 MHz	100 – 250 MHz	12 – 204 MHz
Efficiency	374 GOPS/W*,†	9.6 GOPS/W‡,#	300 – 2600 GOPS/W‡
Power	0.288 mW†	278 mW‡	76 mW‡
On-chip core SRAM	270 kB	181.5 kB	144 kB
Off-chip memory	No	Yes	Yes
Variable precision	6/8/12/16/24/32-bit	No	1 – 16-bit

*OPs = addition/multiplication/load/store. †Operating at 0.65V and 3.9MHz.
‡DRAM access power not included. #Calculated based on reported GOPS and power.

Weight Precision Reduction

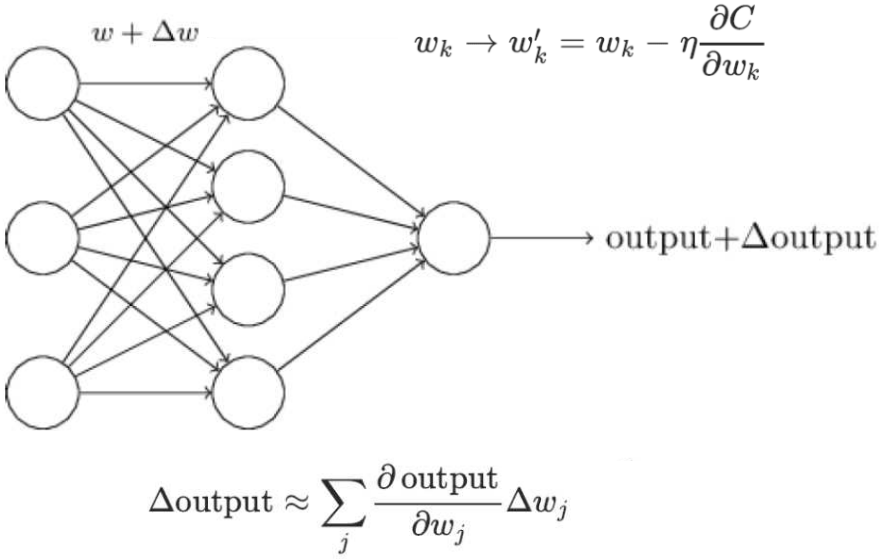
- DNNs inherently tolerant to imprecise computing
- But…

DNN is a function

$$\boldsymbol{y} = f(\boldsymbol{x}) = \sigma_L\left(W_L\sigma_{L-1}\left(W_{L-1}\ldots\sigma_2\left(W_2\sigma_1(W_1\boldsymbol{x})\right)\ldots\right)\right)$$

DNN training goal: minimize 'cost function'

$$\sum_n C\left(f\left(\boldsymbol{x}_{train}^{(n)}\right), \mathbf{y}_{train}^{(n)}\right)$$



Weight Precision Reduction

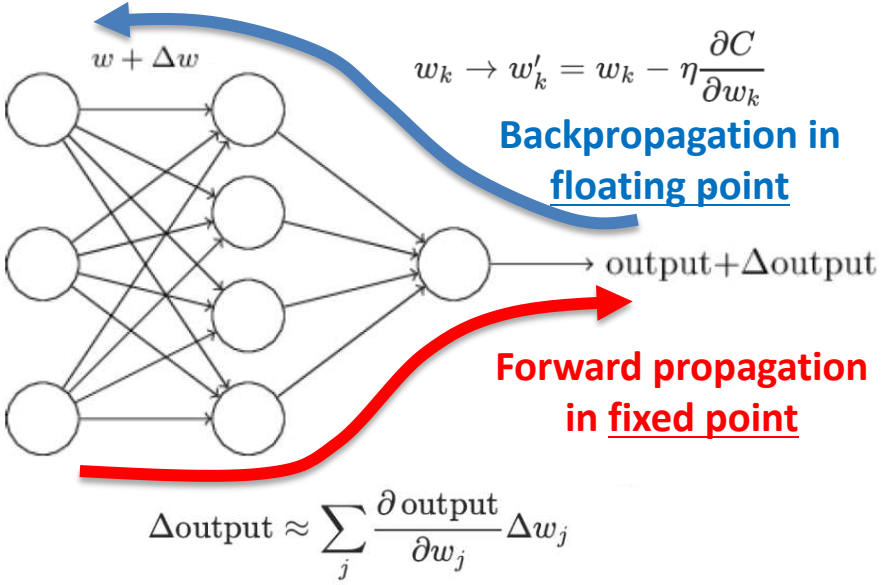
- DNNs inherently tolerant to imprecise computing
- But… **directly quantizing trained weights leads to significant accuracy loss**
- **Retrain with fixed point quantized weights** to avoid accuracy loss

DNN is a function

$$\boldsymbol{y} = f(\boldsymbol{x}) = \sigma_L\left(W_L\sigma_{L-1}\left(W_{L-1}\ldots\sigma_2\left(W_2\sigma_1(W_1\boldsymbol{x})\right)\ldots\right)\right)$$

DNN training goal: minimize 'cost function'

$$\sum_n C\left(f\left(\boldsymbol{x}_{train}^{(n)}\right), \mathbf{y}_{train}^{(n)}\right)$$



Precision vs. Accuracy

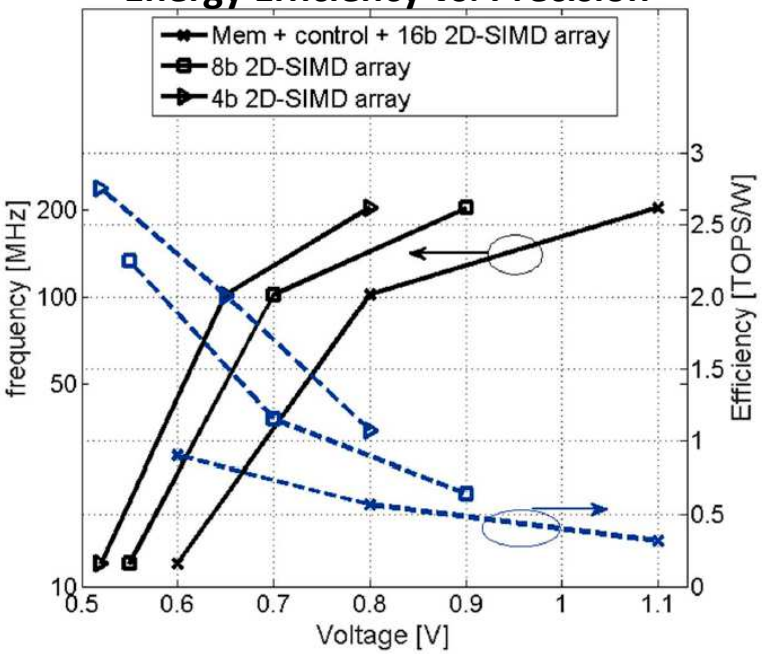
- Precision reduction leads to lower energy
  - Less frequent memory access
  - Lower complexity MAC

SVHN classification error rate

	Type of network	L	C	V
	Floating result	*6.45*	*5.65*	*4.50*
2-bit (3 point)	Direct	45.68	23.17	73.55
	Conventional	8.37	7.10	5.24
	Adaptive	**8.01**	**6.65**	**5.02**
3-bit (7 point)	Direct	10.14	7.88	6.73
	Conventional	7.04	5.97	4.57
	Adaptive	**6.92**	**5.91**	**4.53**
4-bit (15 point)	Direct	7.85	6.03	4.79
	Conventional	6.60	**5.76**	4.74
	Adaptive	**6.46**	5.86	**4.60**

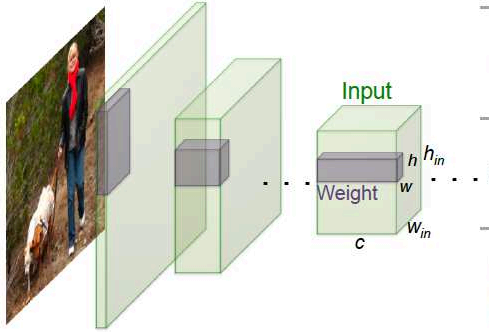
[S. Shin et al., Fixed-point optimization of deep neural networks with adaptive step size retraining, 2017]

Energy Efficiency vs. Precision



[B. Moons, VLSI 2016]

XNOR-Net [M. Rastegari et al., 2016]



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34, -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41, -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34, -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1, -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (XNOR-Net)	Binary Inputs: 1 -1 ... -1, -1 1 ... 1; Binary Weights: 1 -1 ... 1, -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Classification Accuracy(%)									
Binary-Weight				Binary-Input-Binary-Weight				Full-Precision	
BWN		BC[11]		XNOR-Net		BNN[11]		AlexNet[1]	
Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
56.8	**79.4**	35.4	61.0	**44.2**	**69.2**	27.9	50.42	56.6	80.2

ImageNet classification using AlexNet

	ResNet-18		GoogLenet	
Network Variations	top-1	top-5	top-1	top-5
Binary-Weight-Network	60.8	83.0	65.5	86.1
XNOR-Network	51.2	73.2	N/A	N/A
Full-Precision-Network	69.3	89.2	71.3	90.0

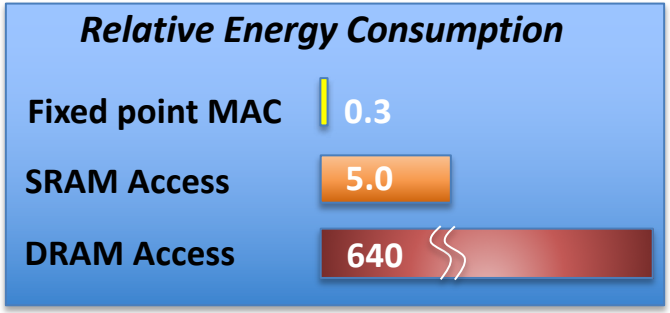
ImageNet classification using deeper networks (ResNet, GoogLenet)

Observations: Precision vs. Accuracy

- **Small and medium NNs**
  - Small accuracy loss from 1 or 2-bit weights
  - Synergy between binary NNs and in-memory computing
  - Recent binary NNs report >10TOPs/W
  - But ... accuracy is significantly worse than the state-of-the-art large NNs
- **Deeper NNs**
  - Not designed for aggressive quantization, difficult to train with low precision
  - Noticeable accuracy loss from aggressive quantization
  - New NN structures optimized for binary weights?
  - Lacks comprehensive energy efficiency analysis on binary NN vs. non-binary NN with an iso-accuracy constraint

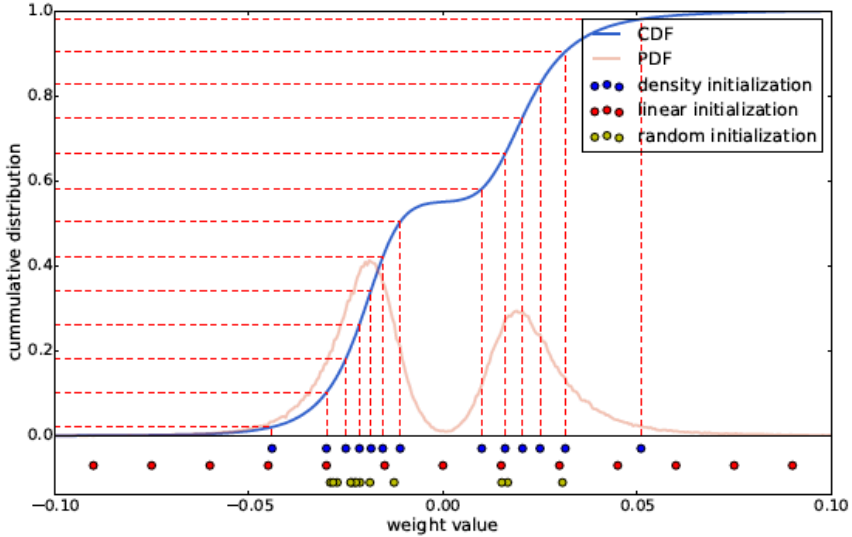
Techniques for Energy Efficient DNN

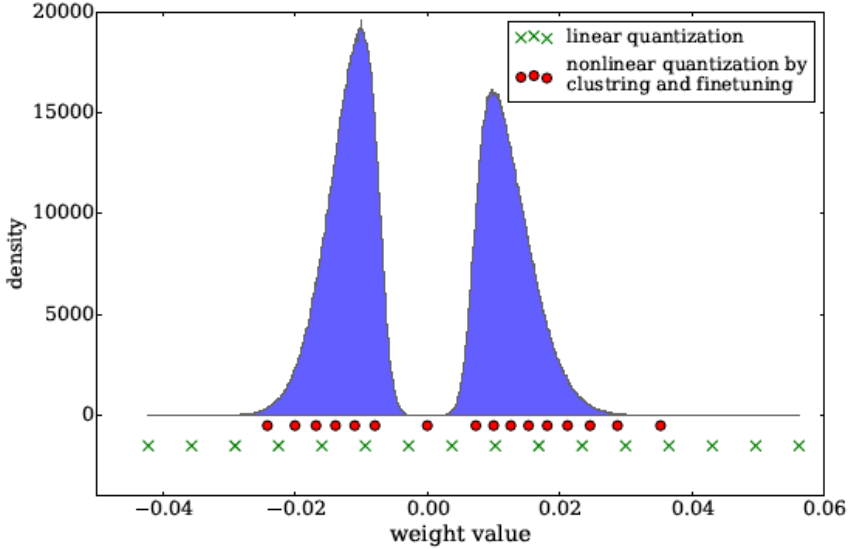
- Energy-optimized HW architecture for DNN
  - Exploit data locality
    - Data reuse for convolution and FCL
    - More (less) frequent local (remote) accesses
  - Parallelism
    - Voltage and frequency scaling
  - Weight precision reduction
    - Less on-chip memory access and lower power MAC
  - Weight compression
    - Minimal or no off-chip DRAM access
  - Exploit sparsity
    - Significant portion of activations and weights are zero
  - In-memory compute using specialized memories



Deep Compression [S. Han, ICLR 2016]

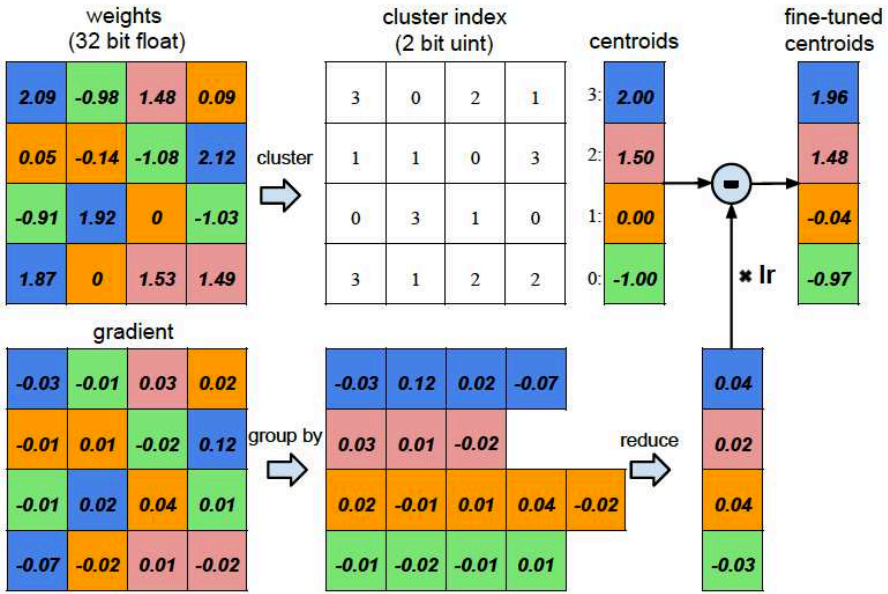
- Non-uniform quantization
  - Same accuracy with fewer quantization levels compared uniform quantization

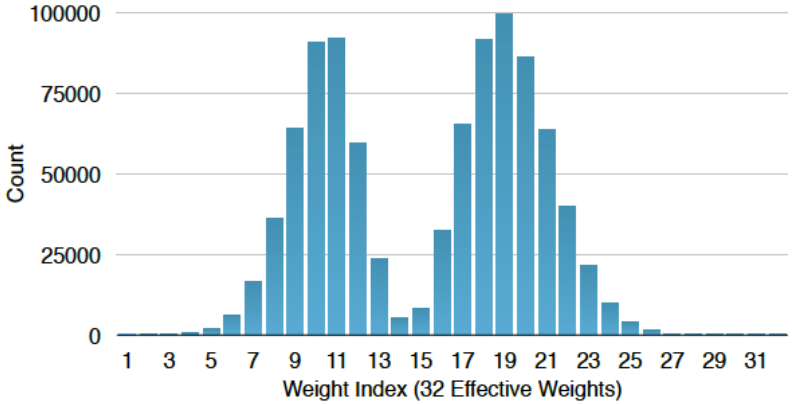




Deep Compression [S. Han, ICLR 2016]

- Non-uniform weight clustering and retraining
- Each cluster of weights share the same centroid value



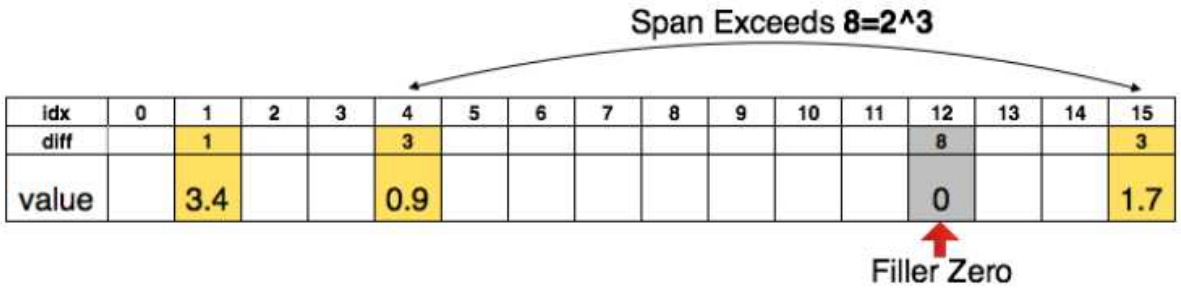


Non-uniform distribution of weights

Entropy (Huffman) coding of weight indices for compression

Deep Compression [S. Han, ICLR 2016]

- Prune weights with small magnitude
  - Weights become ‘sparse’
- Run-length encoding of non-zero positions
  - Run-lengths do not have uniform distribution
  - Apply Huffman code to run-lengths



Deep Compression [S. Han, ICLR 2016]

Table 4: Compression statistics for AlexNet. P: pruning, Q: quantization, H:Huffman coding.

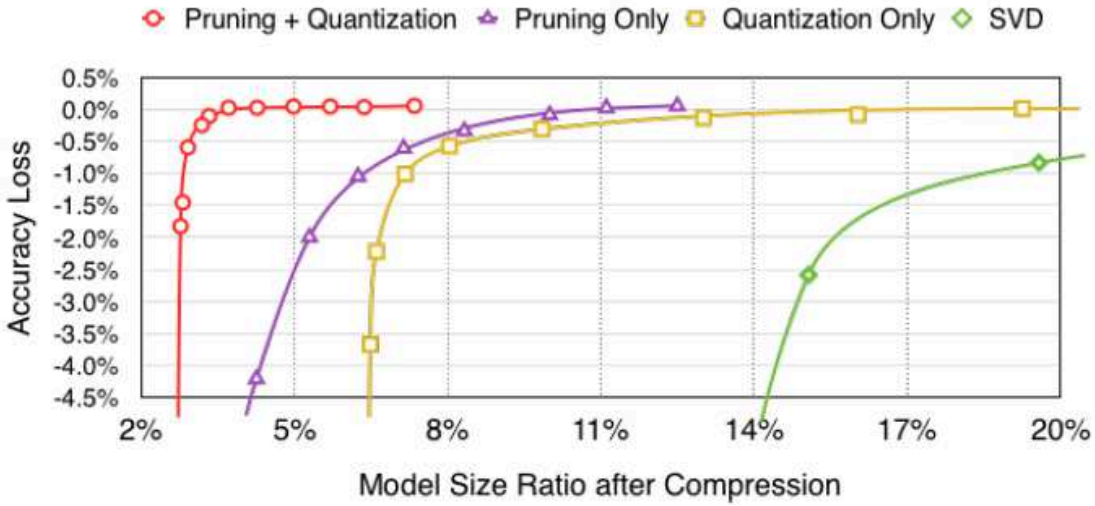
Layer	#Weights	Weights% (P)	Weight bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	35K	84%	8	6.3	4	1.2	32.6%	20.53%
conv2	307K	38%	8	5.5	4	2.3	14.5%	9.43%
conv3	885K	35%	8	5.1	4	2.6	13.1%	8.44%
conv4	663K	37%	8	5.2	4	2.5	14.1%	9.11%
conv5	442K	37%	8	5.6	4	2.5	14.0%	9.43%
fc6	38M	9%	5	3.9	4	3.2	3.0%	2.39%
fc7	17M	9%	5	3.6	4	3.7	3.0%	2.46%
fc8	4M	25%	5	4	4	3.2	7.3%	5.85%
Total	61M	11%(9×)	5.4	4	4	3.2	3.7% (**27**×)	2.88% (**35**×)

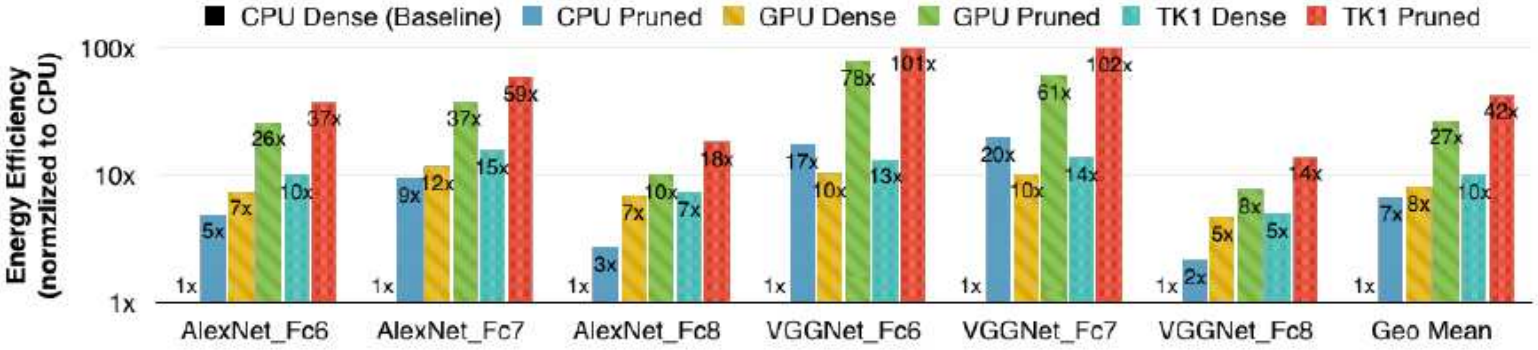
Deep Compression [S. Han, ICLR 2016]

Table 5: Compression statistics for VGG-16. P: pruning, Q:quantization, H:Huffman coding.

Layer	#Weights	Weights% (P)	Weigh bits (P+Q)	Weight bits (P+Q+H)	Index bits (P+Q)	Index bits (P+Q+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1_1	2K	58%	8	6.8	5	1.7	40.0%	29.97%
conv1_2	37K	22%	8	6.5	5	2.6	9.8%	6.99%
conv2_1	74K	34%	8	5.6	5	2.4	14.3%	8.91%
conv2_2	148K	36%	8	5.9	5	2.3	14.7%	9.31%
conv3_1	295K	53%	8	4.8	5	1.8	21.7%	11.15%
conv3_2	590K	24%	8	4.6	5	2.9	9.7%	5.67%
conv3_3	590K	42%	8	4.6	5	2.2	17.0%	8.96%
conv4_1	1M	32%	8	4.6	5	2.6	13.1%	7.29%
conv4_2	2M	27%	8	4.2	5	2.9	10.9%	5.93%
conv4_3	2M	34%	8	4.4	5	2.5	14.0%	7.47%
conv5_1	2M	35%	8	4.7	5	2.5	14.3%	8.00%
conv5_2	2M	29%	8	4.6	5	2.7	11.7%	6.52%
conv5_3	2M	36%	8	4.6	5	2.3	14.8%	7.79%
fc6	103M	4%	5	3.6	5	3.5	1.6%	1.10%
fc7	17M	4%	5	4	5	4.3	1.5%	1.25%
fc8	4M	23%	5	4	5	3.4	7.1%	5.24%
Total	138M	7.5%(13×)	6.4	4.1	5	3.1	3.2% (**31**×)	2.05% (**49**×)

Deep Compression [S. Han, ICLR 2016]



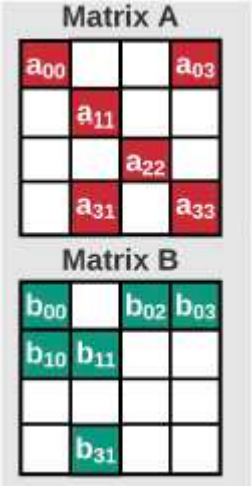


Exploiting Sparsity

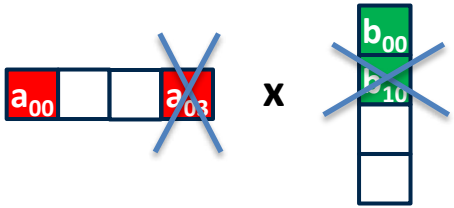
- Non-zero weight density after pruning [S. Han, ICLR 2016]
  - Conv. Layer: 25 – 60%
  - **FCL: 4% - 25%**
- ReLu activation: ~50% density
- Significant gain in FCL from sparse matrix-vector multiplication
  - Higher overhead in exploiting sparsity for conv. layers
- In FCL
  - Weights are used only once (no data reuse)
  - Sparse inputs can be reused (but how?)
- Architecture for sparse matrix-vector multiplication
  - [S. Pal, OuterSpace, HPCA 2018], [S. Han, EIE, ISCA 2016], etc.

Outer-Product based Sparse Matrix Acclerator

- Outer-product based multiplication maximizes data reuse and minimize input memory read [S. Pal, HPCA 2018]
- Each input element is read only once and maximally reused
- Merging overhead is low when non-zero element density is low

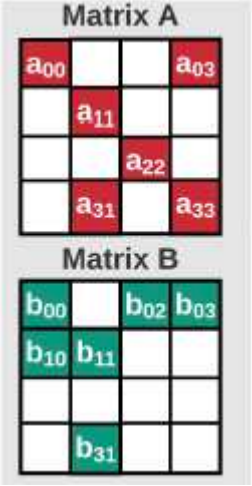


Inner-product:
row A x column B
→ unused non-zero read data

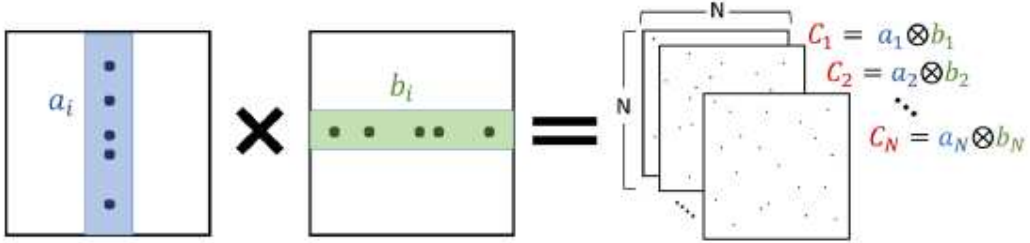


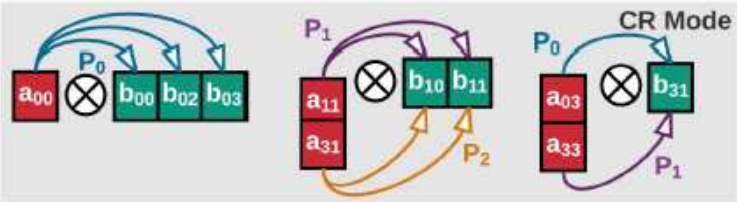
Outer-Product based Sparse Matrix Acclerator

- Outer-product based multiplication maximizes data reuse and minimize input memory read [S. Pal, HPCA 2018]
- Each input element is read only once and maximally reused
- Merging overhead is low when non-zero element density is low



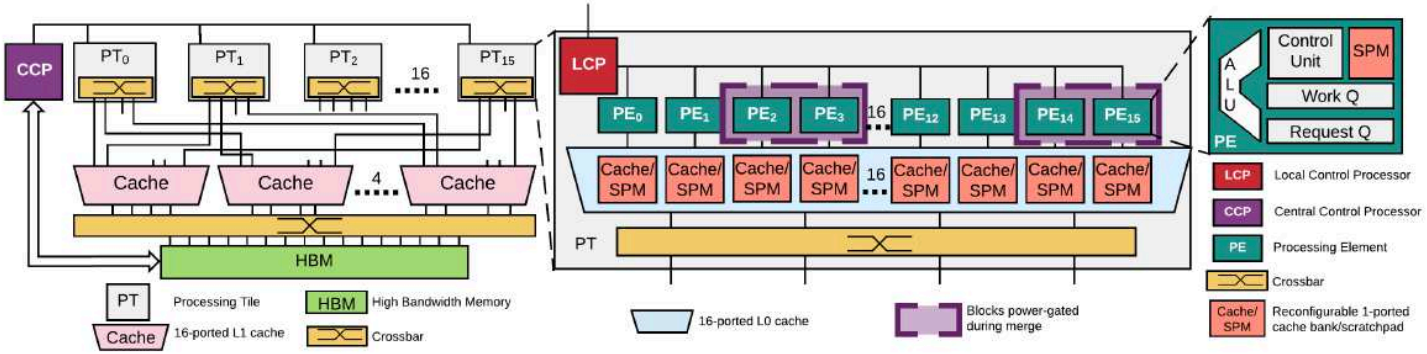
Outer-product: column A x row B maximizes data reuse





Outer-Product based Sparse Matrix Accelerator

- [S. Pal, HPCA 2018]



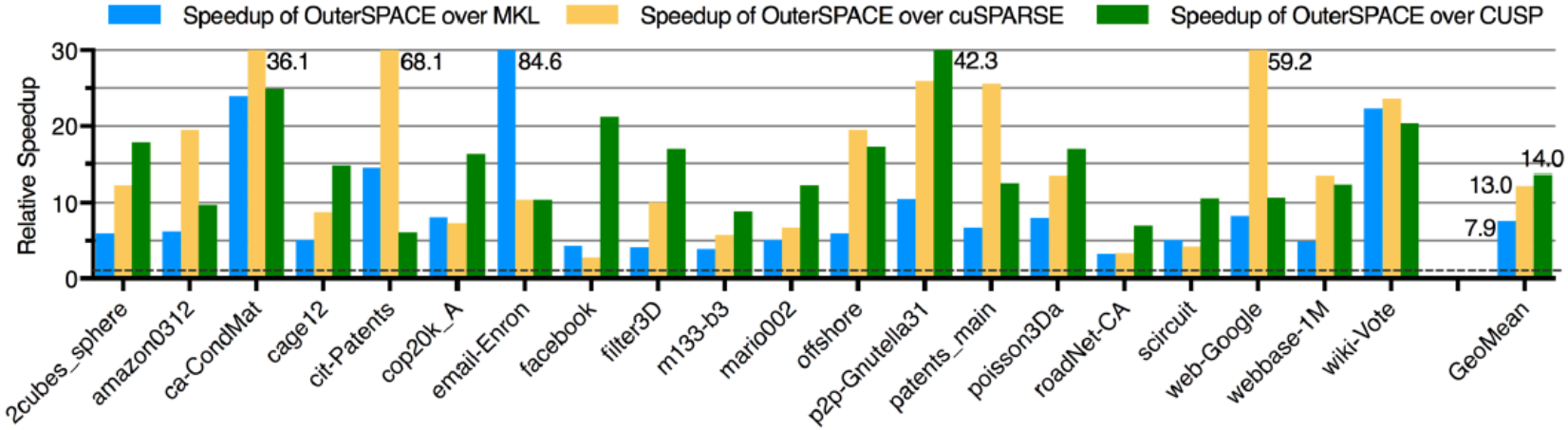
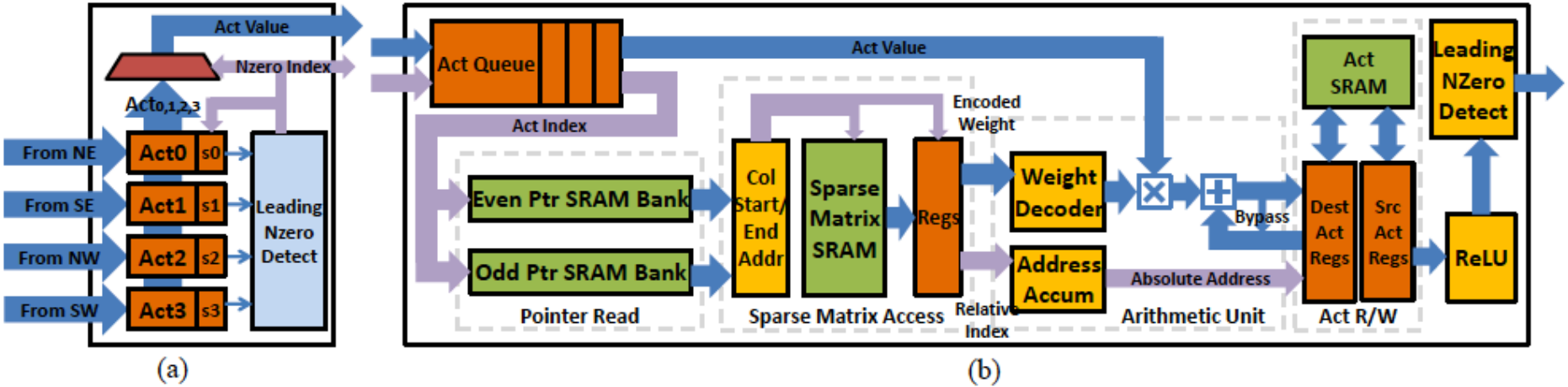


Figure 7: Speedups of OuterSPACE over CPU (Intel MKL) and GPU (cuSPARSE and CUSP).

Sparse Matrix Accelerator [S. Han, ISCA 2016]

- Weight compression + sparse matrix PE



Sparse Matrix Accelerator [S. Han, ISCA 2016]

- Weight compression + sparse matrix PE

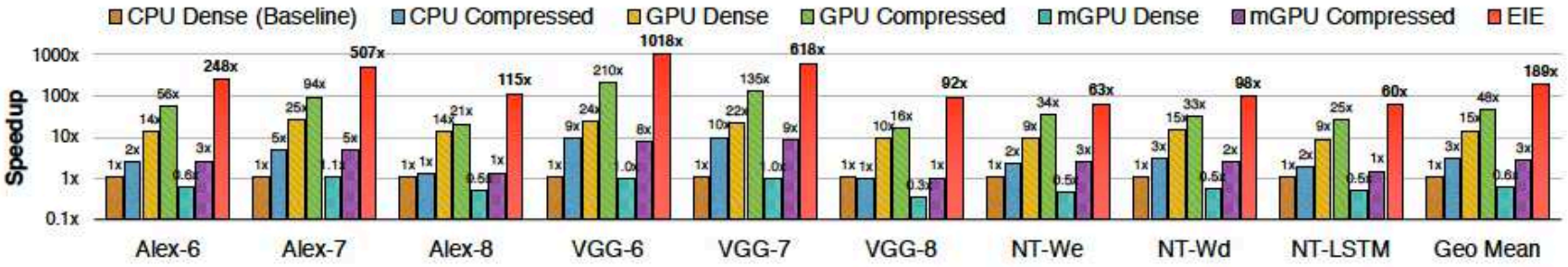


Figure 6. Speedups of GPU, mobile GPU and EIE compared with CPU running uncompressed DNN model. There is no batching in all cases.

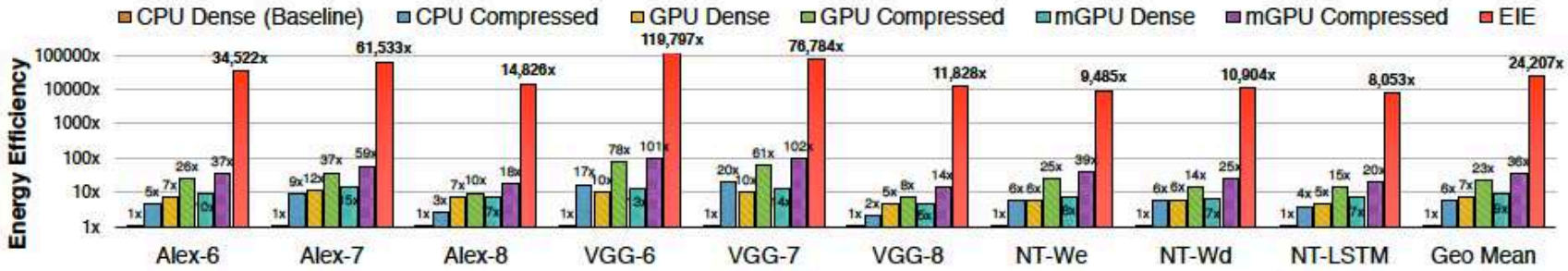


Figure 7. Energy efficiency of GPU, mobile GPU and EIE compared with CPU running uncompressed DNN model. There is no batching in all cases.

Computation in Memory

- "Conv-RAM" [A. Biswas, ISSCC2018]
  - **In-memory compute** for **binarized neural network**

Simplification: Filter weights binary '±1' (with a common scaling factor: α)

Type	Network Variation	Memory Savings (Inference)	Classification Accuracy†
Standard Convolution	*Real-valued Inputs: **X*** ⋮ 0.11 -0.21 .... 0.34 -0.25 0.61 .... 0.52 ⋮ *Real-valued Weights: **W*** ⋮ 0.12 -1.20 .... 0.41 -0.20 0.50 .... 0.68 ⋮	1x	80.2%
Binary-Weight Convolution*	*Real-valued Inputs: **X*** ⋮ 0.11 -0.21 .... 0.34 -0.25 0.61 .... 0.52 ⋮ *Binary Weights: **w*** ⋮ +1 -1 .... +1 -1 +1 .... +1 ⋮	~32x	79.4%

M. Rastegari et al. Arxiv '16

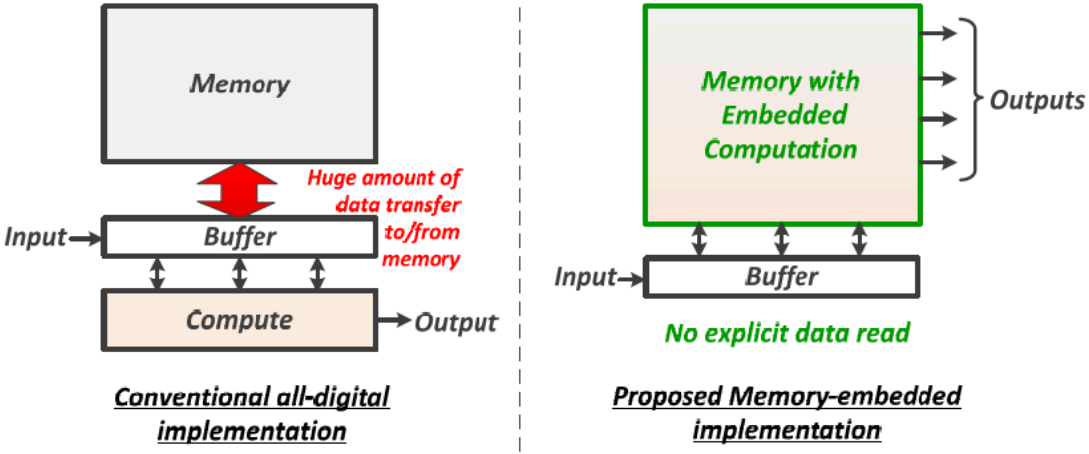
$$Y_k = \sum_i W_{k,i} \times X_i = \alpha_k \sum_i w_{k,i} \times X_i$$

†AlexNet: <1% reduction in accuracy

MAC ($\times, \pm$) replaced by add/sub ($\pm$)

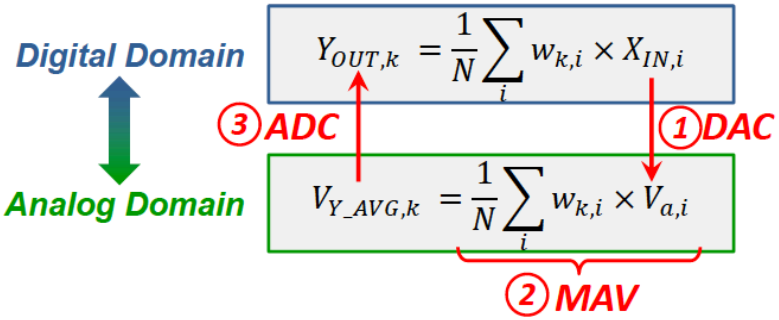
Computation in Memory

- "Conv-RAM" [A. Biswas, ISSCC2018]
  - **In-memory compute** for **binarized neural network**



Embedding computation inside memory

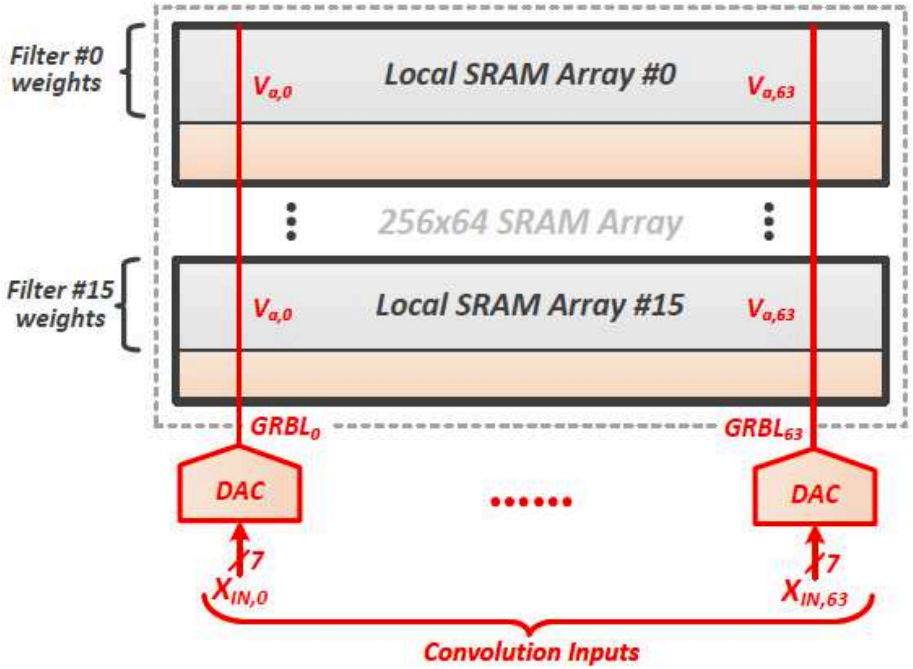
- Less data transfer ⇒ **More Energy-efficient**
- Massively parallel operations ⇒ **Higher Bandwidth**



MAV: multiply and average

Computation in Memory

- "Conv-RAM" [A. Biswas, ISSCC2018]
  - **In-memory compute** for **binarized neural network**



Digital conv input (X_{IN}) to analog voltage (V_a) conversion

$$Y_{OUT,k} = \frac{1}{N}\sum_i w_{k,i} \times X_{IN,i}$$

1 DAC

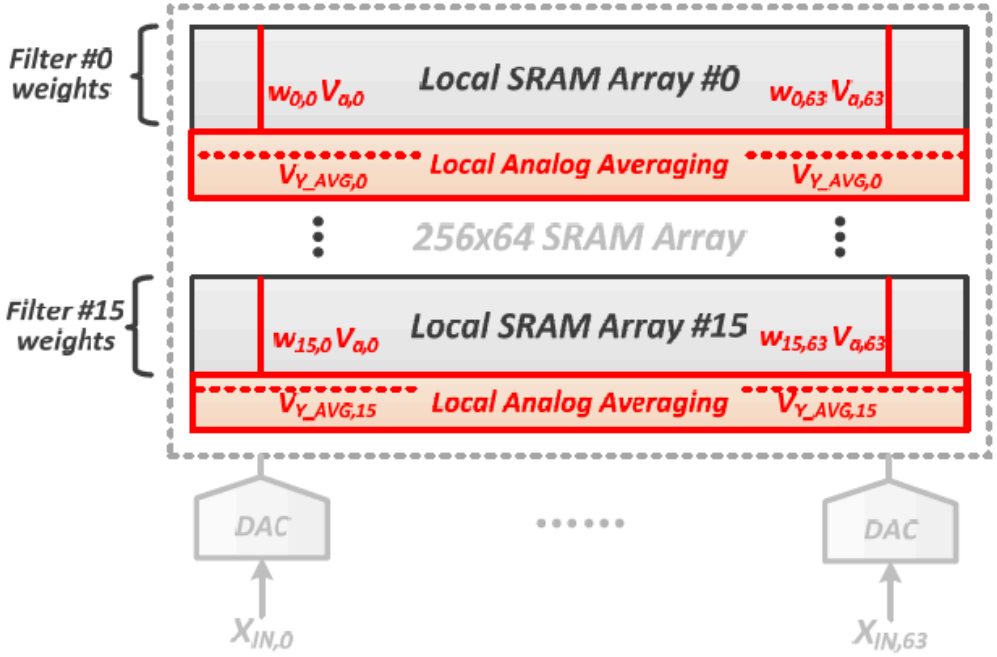
3 ADC

$$V_{Y_AVG,k} = \frac{1}{N}\sum_i w_{k,i} \times V_{a,i}$$

2 MAV

Computation in Memory

- "Conv-RAM" [A. Biswas, ISSCC2018]
  - **In-memory compute** for **binarized neural network**



Row-wise 1-bit weight (*w*) multiply and average (*MAV*)

$$Y_{OUT,k} = \frac{1}{N}\sum_i w_{k,i} \times X_{IN,i}$$

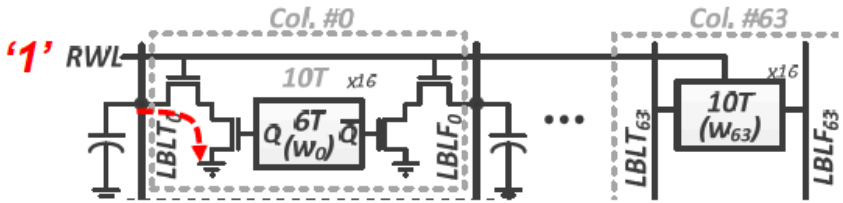
3 ADC

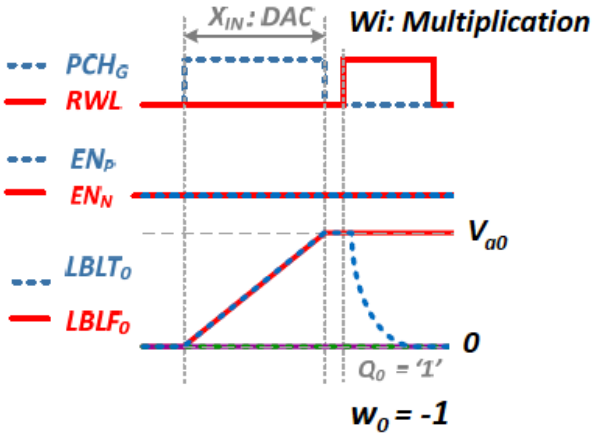
1 DAC

$$V_{Y_AVG,k} = \underbrace{\frac{1}{N}\sum_i w_{k,i} \times V_{a,i}}_{\text{2 MAV}}$$

Computation in Memory

- "Conv-RAM" [A. Biswas, ISSCC2018]
  - **In-memory compute** for **binarized neural network**



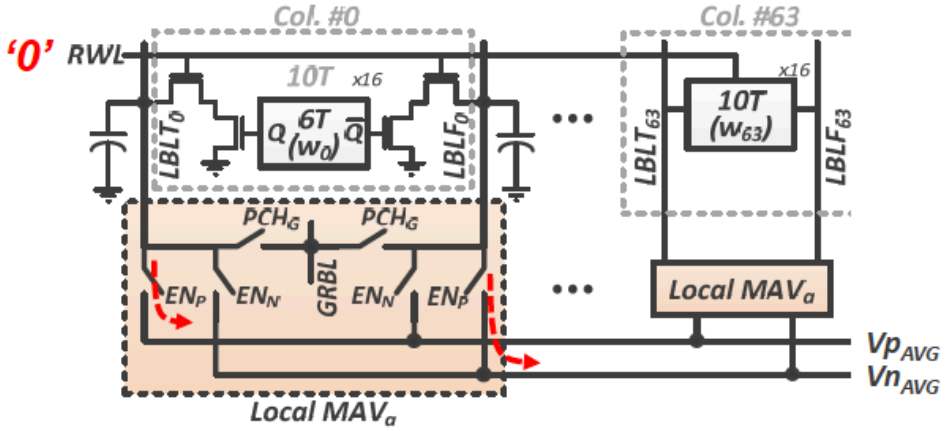


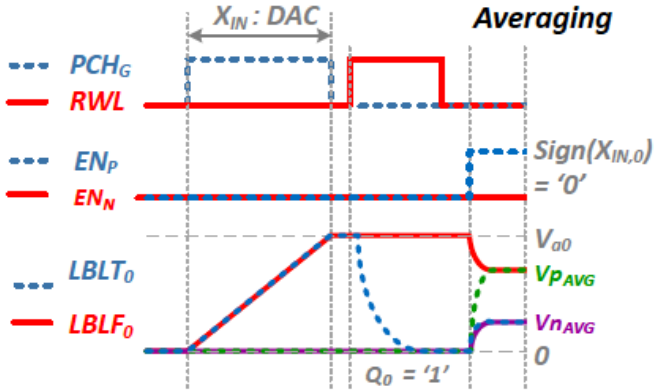
- 1-bit weight multiplication by SRAM bit-cell through local bit-line discharge
  - Q = '0' for w = +1 / '1' for w = -1

$$\Delta V_{LBL,i} = V_{LBLT,i} - V_{LBLF,i} = V_{a,i} \times w_i$$

Computation in Memory

- "Conv-RAM" [A. Biswas, ISSCC2018]
  - **In-memory compute** for **binarized neural network**

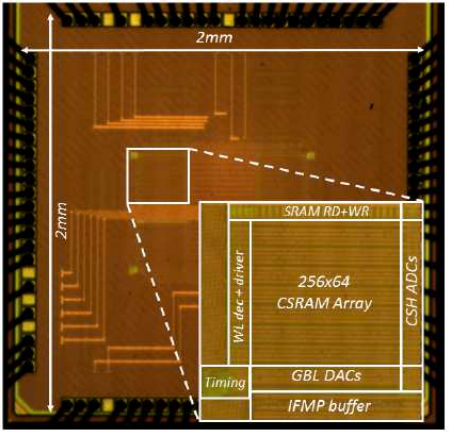




- Charge-sharing of local bit-lines implements averaging
- Separate voltage rails for '+ve' (Vp_{AVG}) and '-ve' (Vn_{AVG}) parts of the average

Computation in Memory

- "Conv-RAM" [A. Biswas, ISSCC2018]
  - **In-memory compute** for **binarized neural network**



	Tech. (nm)	ML Algo., # Classes	Comp. Mode	Classification Accuracy	Input/ Filter	SRAM Size (KB)	Throughput (GOPS)	Energy Eff. (TOPS/W)	FOM
This work	65		Analog	96%	7b/1b	2	10.7	**28.1**	**150.3**
ISSCC '16 [2]	65	CNN, 10	Digital	98.3%	16b	36	64	1.42	2.5
VLSI '16 [3]	40		Digital	98%	6b/4b	144	102	1.75	1.2
Arxiv '16 [5]	65	kNN, 4	Analog	92%	8b	16	10.2	0.98	0.6

$1\ MAV,\ MAC,\ SAD = 2\ OPs.$ $\quad FOM = \frac{Throughput \times Energy\ Eff.}{SRAM\ Size}$

Summary: Energy Efficient DNN HW Architecture

- **Exploit data locality**
  - Data reuse for convolution and FCL
  - More (less) frequent local (remote) accesses
- **Parallelism**
  - Voltage and frequency scaling
- **Weight precision reduction**
  - Less on-chip memory access and lower power MAC
- **Weight compression**
  - Minimal or no off-chip DRAM access
- **Exploit sparsity**
  - Significant portion of activations and weights are zero
- **In-memory compute using specialized memories**

References

- V. Sze, *Hardware Architectures for Deep Neural Networks*, http://eyeriss.mit.edu/tutorial.html
- Y. Chen, *Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks*, ISSCC 2016
- N Jouppi, *In-Datacenter Performance Analysis of a Tensor Processing Unit*, ISCA 2017
- S. Bang, *A 288uW Programmable Deep-Learning Processor with 270KB On-Chip Weight Storage Using Non-Uniform Memory Hierarchy for Mobile Intelligence*, ISSCC2017
- S. Pal, *OuterSPACE: An Outer Product based Sparse Matrix Multiplication Accelerator*, HPCA 2018
- M. Rastegari, *Xnor-net: Imagenet classification using binary convolutional neural networks*, ECCV 2016
- S. Han, *EIE: Efficient Inference Engine on Compressed Deep Neural Network*, ISCA 2016
- S. Han, *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*, ICLR 2016
- A. Biswas, *Conv-RAM: An Energy-Efficient SRAM with Embedded Convolution Computation for Low-Power CNN-Based Machine Learning Applications*, ISSCC 2018
- B. Moons, A 0.3-2.6 TOPS/W Precision-Scalable Processor for Real-Time Large-Scale ConvNets, VLSI2016

Thank you!

Q&A

Google AI

Embedded CNN Inference Engine for On-Device Computer Vision Applications

Bo Wu
Google Inc.
bowu@google.com

Google

Contents

- Applications and challenges: Google Clips and AIY Vision Kit
- Offline CNN model tooling
- On-device CNN inference engine design
- Model optimization for embedded applications

Application: Google Clips

- A novel smart photography device
  - https://store.google.com/us/product/google_clips?hl=en-US
  - Audience: parents, pet owners
- On-device intelligence
  - Real-time capture and cuartion decision
  - CNN models that extract semantic signals for people and pets
  - https://ai.googleblog.com/2018/05/automatic-photography-with-google-clips.html

Application: AIY Vision Kit

- Do-it-yourself intelligent camera
  - One of the products under Google AIY project
  - https://aiyprojects.withgoogle.com/vision/
  - Audience: makers, developers, educators
- On-device CNN models
  - Face detection and ‘joy’ classification
  - Image classification (ImageNet, iNaturalist, dishes)
  - Object detection (person, dog, cat)
  - Support running customized models
  - https://aiyprojects.withgoogle.com/models/

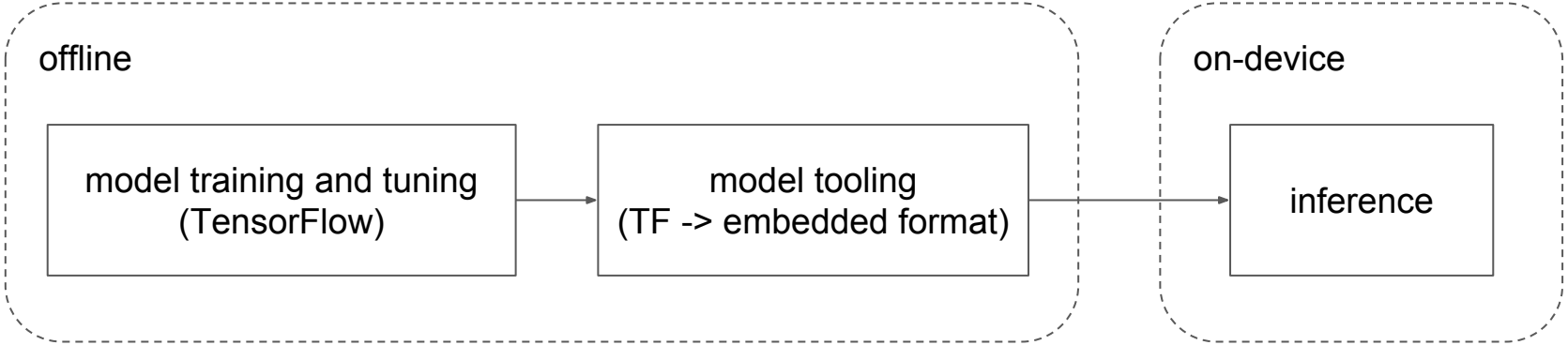
Why on-device intelligence?

- Super strong privacy protection
  - User data are 100% local on-device, and users own the device
  - No cloud side central intelligence is involved
  - Users have full control of what data will be shared eventually
- Real-time processing
  - Real-time interaction is critical for many applications, e.g. automatic photography, robotics.
  - Network latency usually exceed hundreds of milliseconds.
- No internet connection required
  - Intelligence can be used anywhere, including locations where internet connection is poor or inaccessible.

Challenges

- Extremely limited computing resources
  - For example, Myriad chip has 12 SHAVE cores, 2MB CMX memory, 512MB DDR, 1 2-D hardware convolution accelerator, 2 LEON CPUs
- Small power budget
  - Embedded applications usually have limited power source, e.g a battery
  - To achieve x hours of usage per full charge, can only consume y mW on average
- Thermo budget
  - Embedded devices ID put constraints on thermo sink capacity
  - To stay below x °F, peak power consumption can be at most y mW
- Near real-time latency
  - E.g. < 100 ms

Development workflow of our solution



Offline tooling

High level motivation:

Do as much offline preparation as possible before venturing into the wild

Offline tooling: rational

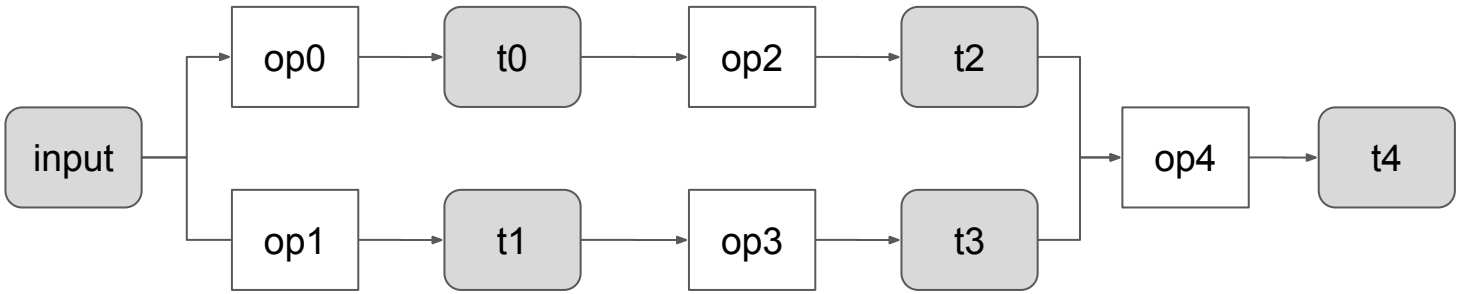
- Offline tooling is a step that converts trained DNN models to a format that can be efficiently processed by an on-device inference engine.
- Offline tooling vs runtime compiling
  - Advantages
    - Remove complexity for on-device inference codes
    - Much easier to debug and tests
    - Allow more specialized on-device engine implementation
  - Disadvantages
    - One more intermediate step for developer
    - Separate tooling implementation and on-device inference engine implementation must be in sync
- Example of offline tooling for mobile: tflite/toco
  - https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/lite/toco/README.md

Offline tooling: graph optimization

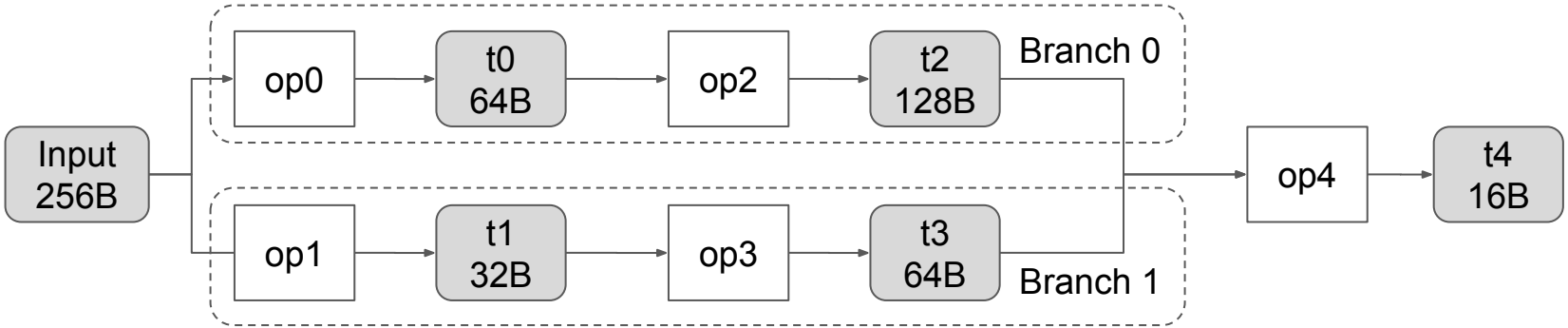
- Remove sub-graphs that are not used during inference
  - Common case: remove training only nodes from a TF graph
  - Less common case: remove irrelevant head layers of multi-purposes models for a certain task
- Fuse layers
  - It could be more efficient to process some operators “together”
  - For example:
    - Conv + Relu + pooling
    - Conv + BatchNorm
- Identify in-place layers
  - Depending on graph structure and operator implementation, some layer can be marked as “in-place”. This help reduce memory cost.

Offline tooling: minimize memory cost (1)

- Goal: during inference, try to keep all the intermediate tensors that have not been consumed in fast memory (CMX) to cut latency. Different orders of layer execution could lead to different peak memory cost. Execution order matters when model has more than one branch.



Offline tooling: minimize memory cost (2)



- Execution order A: op0, op1, op2, op3, op4 (alternating branch 0 and 1)

  peak memory cost: 256 + 64 + 32 **(good)**

- Execution order B: op0, op2, op1, op3, op4 (first branch 0, then branch 1)

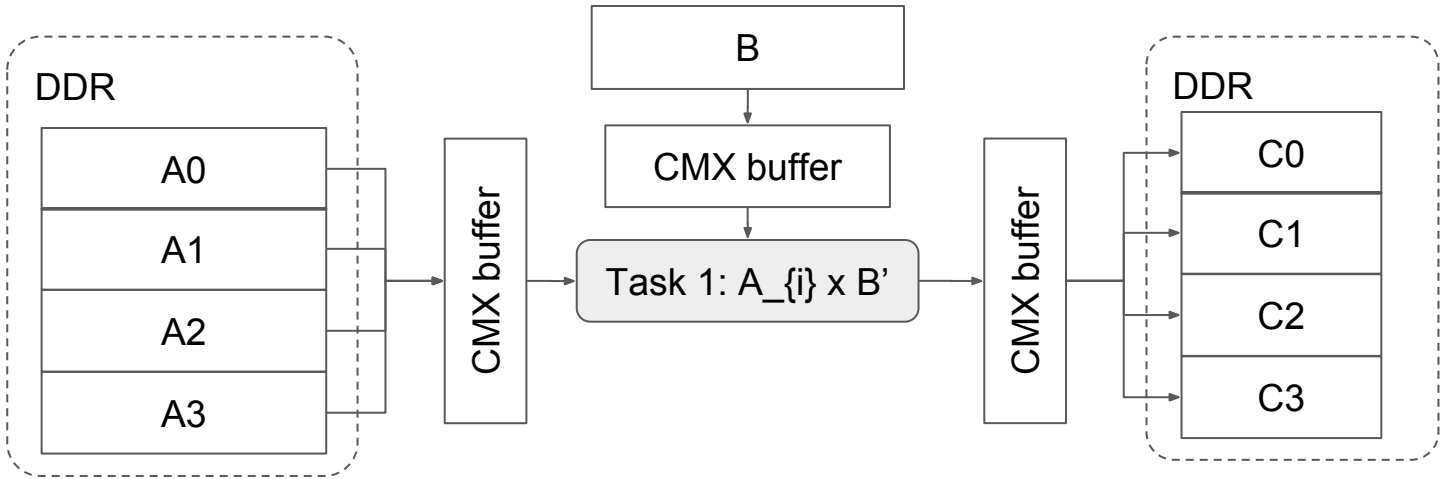
  peak memory cost: 256 + 128 + 32

Offline tooling: minimize memory cost (3)

- Map to DAG optimization problem:
  - One node for each operator
  - Each node has an associated cost corresponding to the operator's memory cost
  - If operator A's output is consumed by operator B, then in the DAG there is a direct link from operator A's node to operator B's node.
  - At any moment, the total cost of the DAG is the sum of the costs of all the nodes that are about to be executed.
  - A node will be freed once its outputs are consumed.
- DAG optimizer to find the best op execution order minimizing peak memory cost.

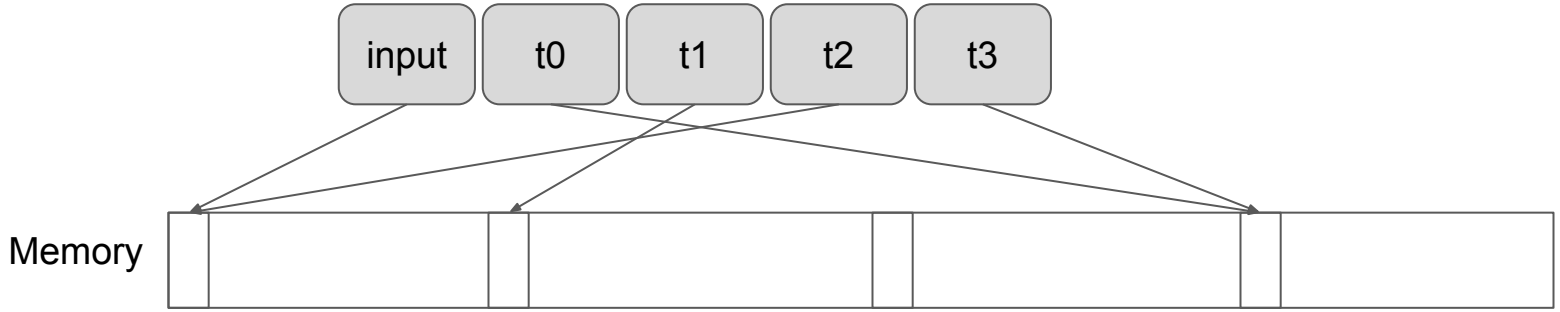
Offline tooling: workaround for big models

- What if the model is too big to fit in the CMX memory budget?
  - Keep intermediate tensors on DDR
  - Divide operation into smaller tasks with smaller input and output that can fit into CMX buffer
  - DMA input / output data between CMX and DDR
  - See below for illustration on GEMM operator



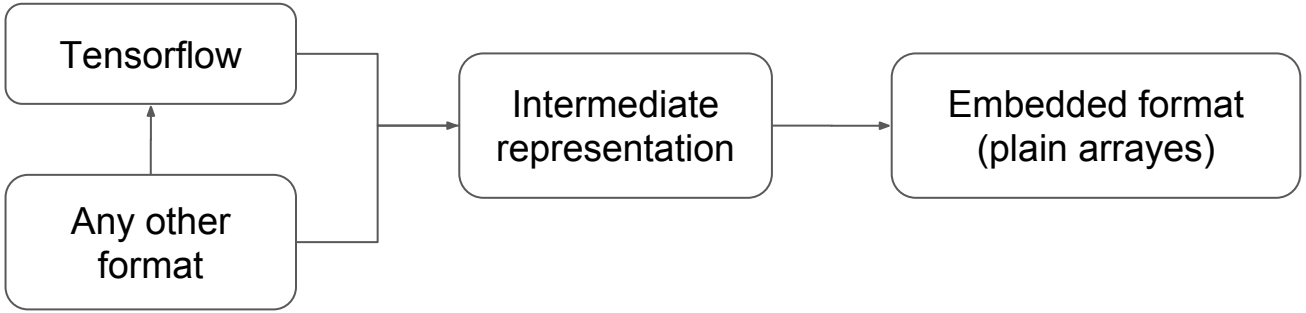
Offline tooling: static memory allocation

- Why static allocation? Because dynamic allocation can fail at runtime.
- A block of memory is reserved for one inference at runtime.
  - The size of the memory block is decided by peak memory cost of the model.
- Offsets of all intermediate tensors during inference are defined during offline tooling
  - memory for intermediate tensors are statically allocated.



Offline tooling: format conversion

- There are a number of public DNN training frameworks with their own model formats: TensorFlow, Caffe, Torch, etc. Essentially they all define some graph data structure.
- On-device inference engine takes plain arrays as input.
  - Array of intermediate tensors, array of parameter tensors, array of operation configurations



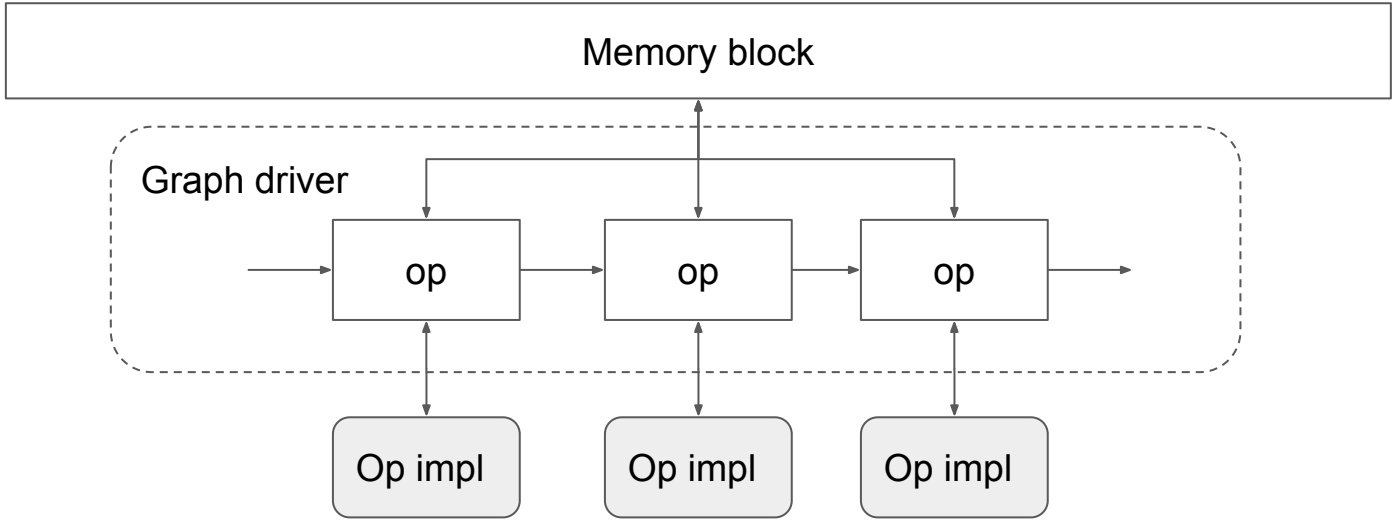
On-device inference engine

We want the on-device inference to be

- Power efficient
- Low memory cost
- Real time
- (Flexibe?)

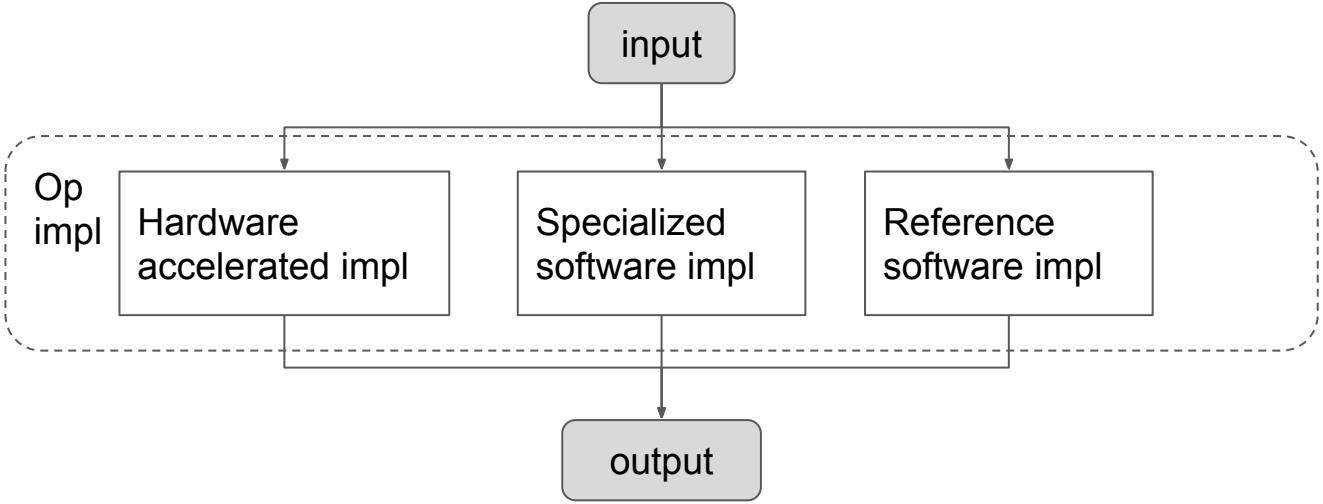
On-device inference: graph driver

- Graph driver is a piece of software that traverses the CNN model graph, invokes the operator of each node, and manages memory of intermediate tensors and temporary buffer.



On-device inference: operator implementation

- Highly optimized, but specialized implementation + less optimized fallback implementation + reference implementation
- Graph driver will decide which operator implementation to run at runtime based on input tensor shape and user settings.



On-device inference: low precision representation

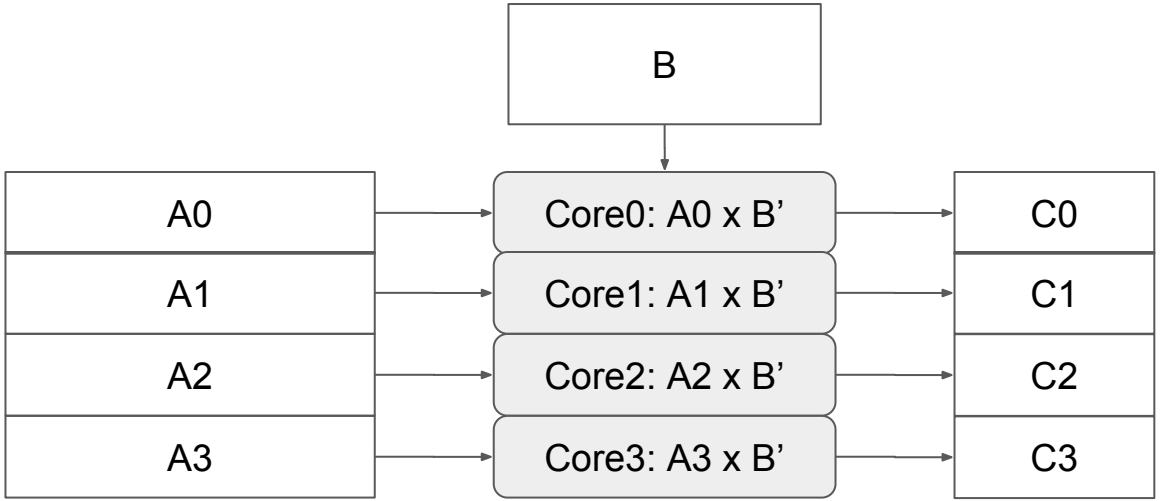
- Fewer bit representation
  - Myriad chip support vectorized operations for 8-bit (fixed point), 16-bit (half float), 32-bit (float).
  - Same vector processor has twice as much as throughput in terms of elements using 16-bit float than 32-bit float.
  - Using lower precision could hurt model accuracy depending on model structure and the way the model is trained.
- Type for storage vs type for computation
  - Lower precision types (8-bit , 16-bit) are used to store intermediate tensors and model parameters
  - Higher precision types (16-bit, 32-bit) are used to accumulate values

On-device inference: issues of low precision types

- Underflow and overflow
  - Half float type range is not that big
    - The minimum positive normal value ≈ 6.10×10^{-5}.
    - The maximum representable value = 65504.
  - Careful implementation to handle underflow gracefully.
  - Design the model to limit the range of activation values, e.g. prefer bounded activation
- Higher numeric errors
  - Careful implementation to minimize numeric errors
    - Kahan summation, hierarchical summation, instead of sequential summation
  - Use higher precision type 32-bit float to accumulate large vectors
- However, some operators are sensitive to numeric errors in inputs.
  - The models are trained to be robust against such inherent errors.
  - It’s a good practice to always double check the statistical behavior.

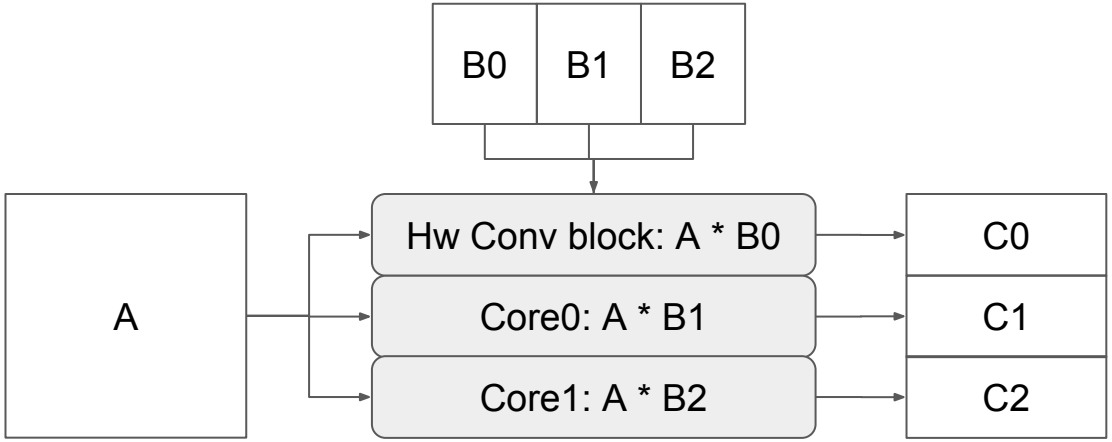
On-device inference: parallelism / concurrency (1)

- Software parallelism within operator: compute one operator with more than one computing cores. Take multi-core GEMM as an example.
- However, the parallelism scalability is sub-linear.



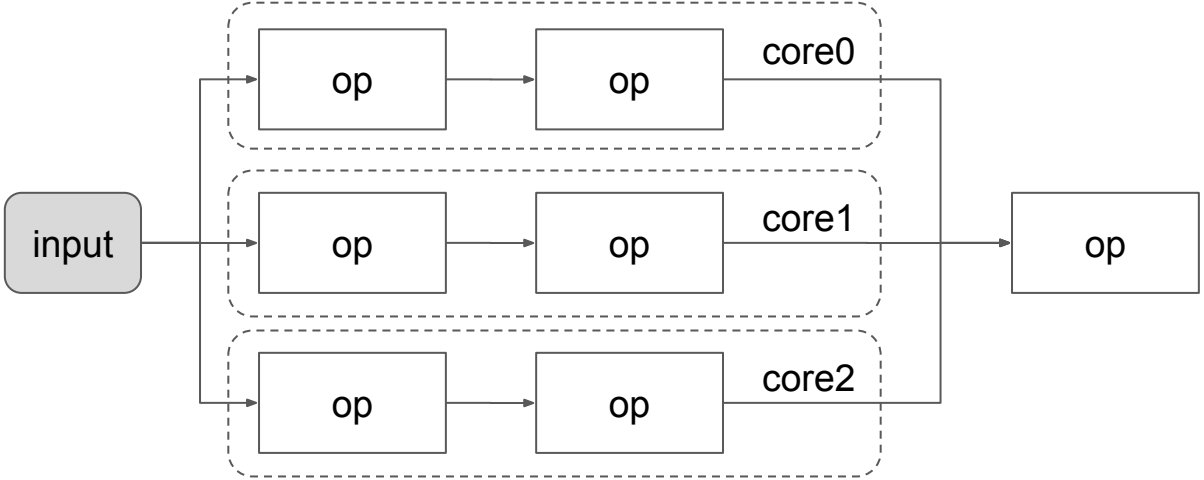
On-device inference: parallelism / concurrency (2)

- Software + hardware parallelism within operator: compute one operator with hardware accelerator and computing cores. For example, Conv can be computed with conv accelerator and computing cores



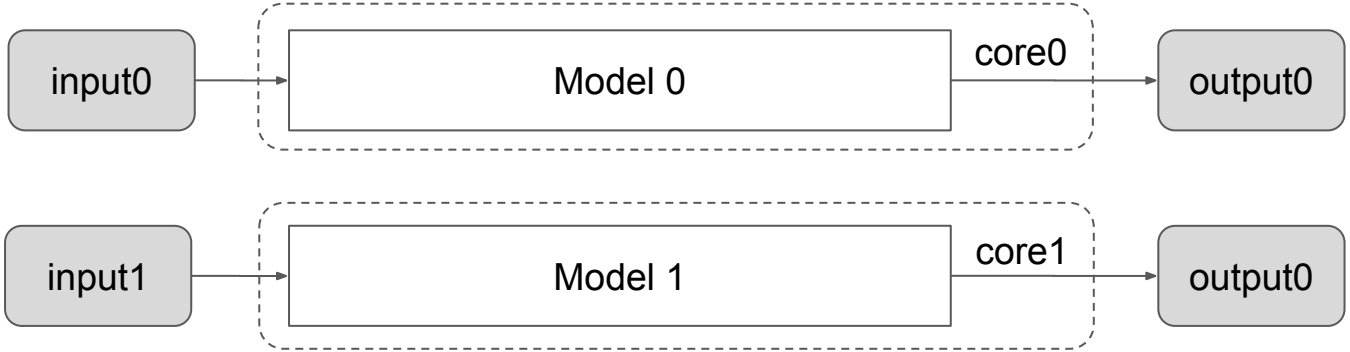
On-device inference: parallelism / concurrency (3)

- Operator level concurrency: compute multiple operators (branches) concurrently with more than one computing cores. Depend on model architecture.



On-device inference: parallelism / concurrency (4)

- Model level concurrency: compute multiple models concurrently with more than one computing cores.



Model optimization

Model training, to deployment is an iterative process, instead of a one way path.

Model optimization: embedded friendly architecture

- Most, if not all, the big popular models architectures, e.g. Inception, VGG, have too high memory and computation (power) cost to be deployed in real embedded applications
  - Too many parameters
  - Too many MACs / Flops
- Model architectures designed for mobile applications, for example
  - MobileNet, see next slide for more
  - SqueezeNet, https://arxiv.org/abs/1602.07360
- We base our models on MobileNet architecture, which has hyper parameters to easily control model size
  - Depth multiplier
  - Resolution multiplier

Model optimization: Mobilenet a quick look

- Mobilenet v1: https://arxiv.org/abs/1704.04861
- Mobilenet v2: https://arxiv.org/abs/1801.04381

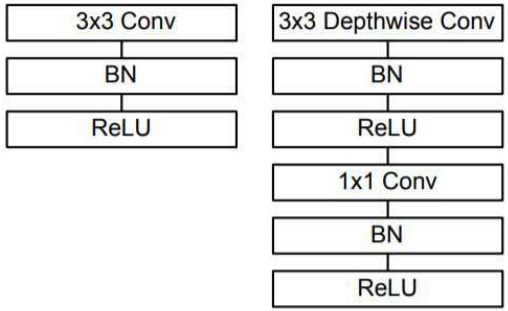
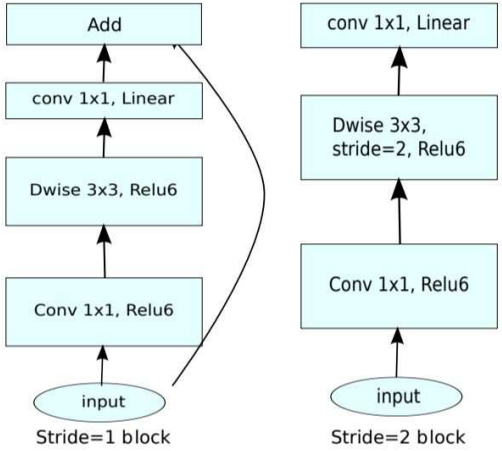


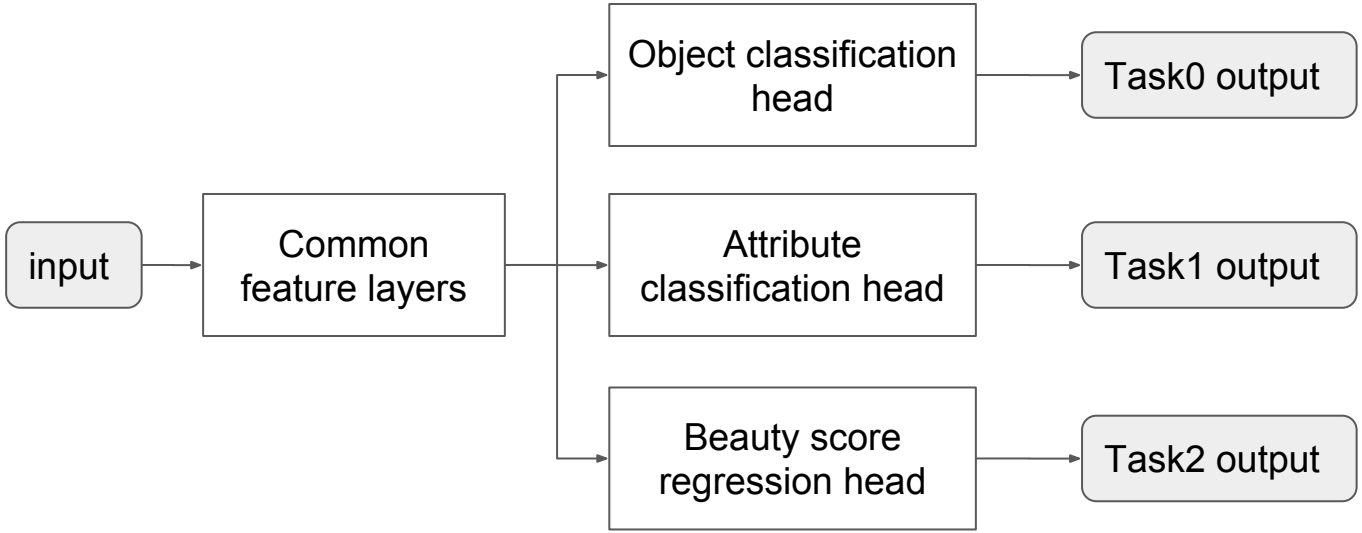
Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.



(d) Mobilenet V2

Model optimization: multi-purpose model

Multi-purpose models have multiple task specific heads share the same feature layers. The feature layers need to be trained for the tasks jointly.



Model optimization: low precision training

- Converting models trained with high precision type to low precision type will result in accuracy loss. Low precision training take precision loss / quantization into account during training stage.
- There are different ways to train models to be low precision / quantization friendly.
  - Tensorflow fixed point model training: https://www.tensorflow.org/performance/quantization

Key takeaways

- Model research and training, hardware accelerator design, and on-device inference engine development is an iterative, closely collaborative process
- Computation efficiency and intelligence accuracy are equally important. Show the accuracy-MAC curve.
- On-device inference engine for embedded applications must be extremely frugal when consuming computing resources. Deploying cloud or mobile CNN runtime to embedded applications is sub-optimal.

Autonomous and Distributed Learning in Large-Scale Infrastructure Systems

Li Shang
University of Colorado Boulder

Large-scale infrastructure systems

Large-scale infrastructure systems serve as the backbone of modern society. Many of these large-scale systems are distributed in nature. System management and maintenance are challenging.

System modeling challenges

- The overall system is a hierarchical composition of numerous self-acting components with similar functionalities.
- The overall system characteristics are aggregated effects of individual sub-system components.
- Effective system modeling and management require detailed component-level modeling.
- Optimized control of individual components often requires a comprehensive view of the overall system.

Limitations of existing system modeling work

- Most of the prior work was led by domain experts, building upon their domain expertise governed by domain-specific physical principles
- The model development process is often driven by simulations or pre-collected data with labeled ground truth.
- Assumptions often need to be made to simplify the system operating environment, thus unable to accurately capture high-order system effects.
- Manual labeling of the collected data is often required, which is time-consuming and error-prone.

Fast-growing data capturing capabilities

Autonomous system learning

- Manual labeling of sensor data is costly, imprecise, and error prone.
- Autonomous learning methods shall capture the internal structure of a system, and derive quantitative models for the target system.
- System evolves over time along with the change of external environment and system aging.
- The system model must evolve accordingly, accurately quantify the root course of such change, and provide high-fidelity prediction of future status.

Distributed system learning

- Many of the infrastructures are located in the remote areas. Sensor data communication remains a challenge.
- Deploying dedicated network infrastructure is costly, and such infrastructures have stringent security requirements.
- System management requires real-time decision making and action taking. Centralized system model is costly and often impractical.
- System modeling must be distributed and in-situ at the individual sub-system component level in order to support real-time control.

Case study: System complexity analysis

Anomaly Type	Anomalies
Visual	discoloration, browning, surface soiling, delamination, partial shading
Thermal	hotspot
Electrical	dark/illuminated current-voltage measurement, line-line, short circuit, open circuit
Others	sensor bias

TABLE I: Common anomalies in solar farms.

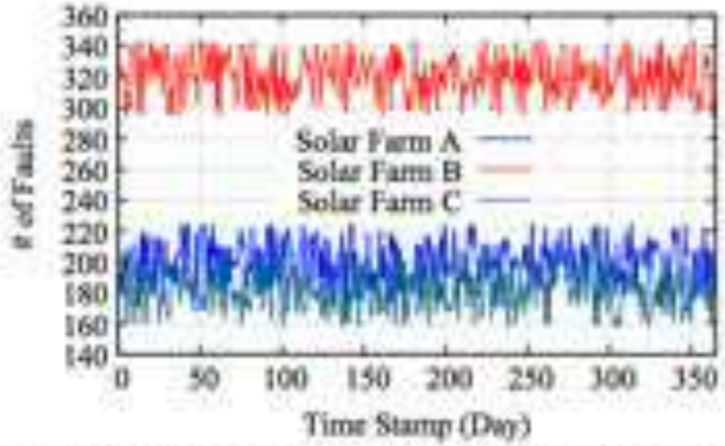


Fig. 1: Solar farm daily fault occurrences.

Case study: System complexity analysis

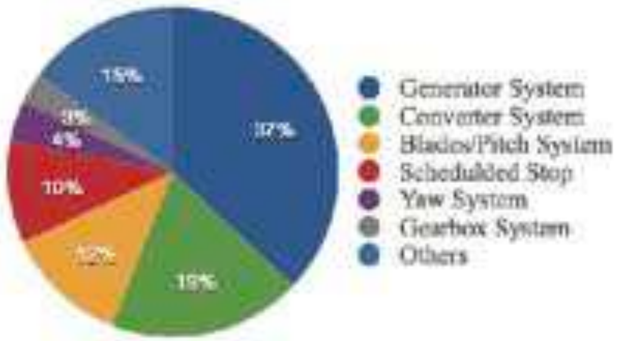


Fig. 2: Common anomalies in wind farms.

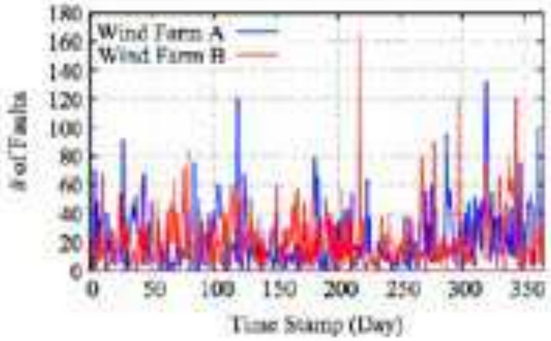


Fig. 3: Wind farm daily fault occurrences.

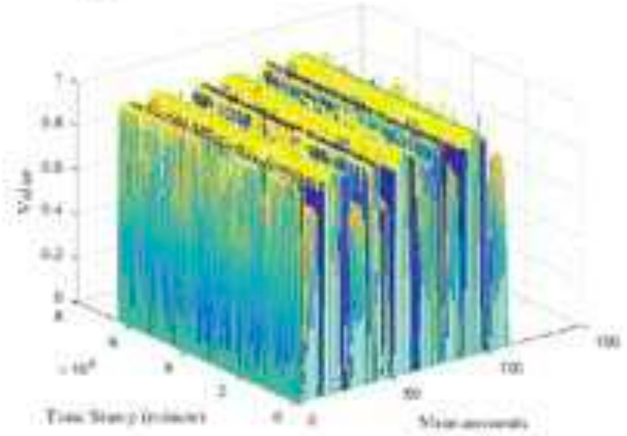


Fig. 4: Measurements in wind turbine.

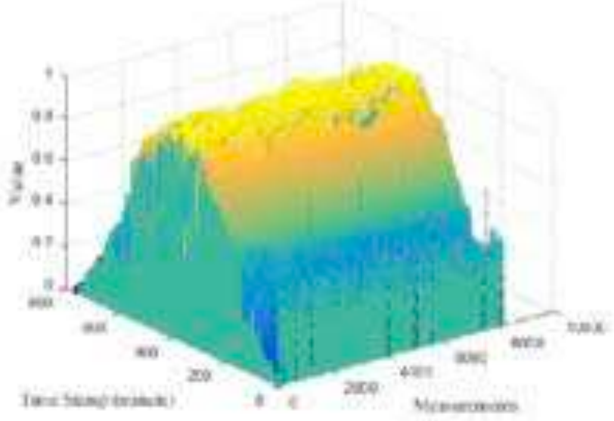


Fig. 5: Measurements in solar farm.

Case study: System redundancy analysis

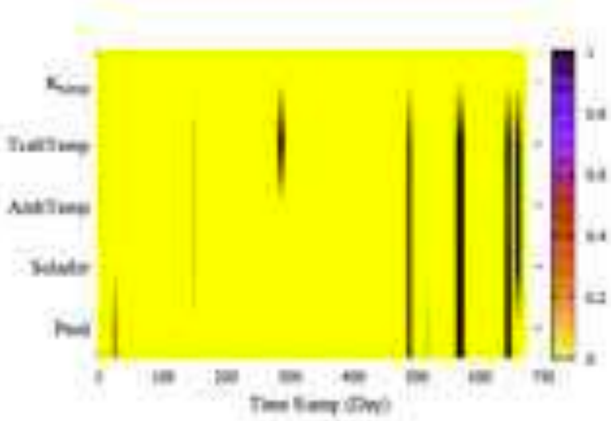


Fig. 6: Missing data statistics.

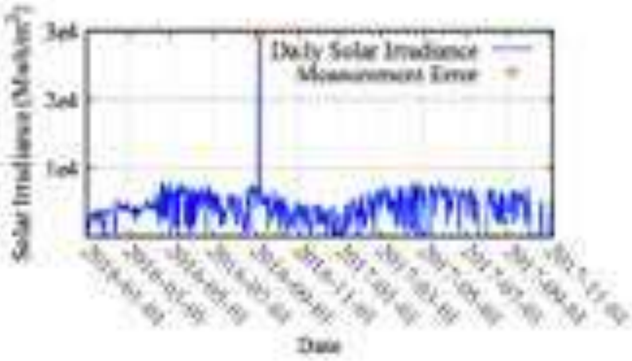


Fig. 7: Data errors and noises.

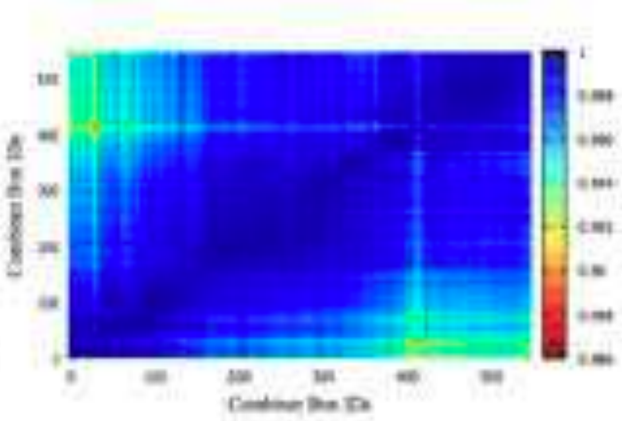


Fig. 8: Data redundancy analysis.

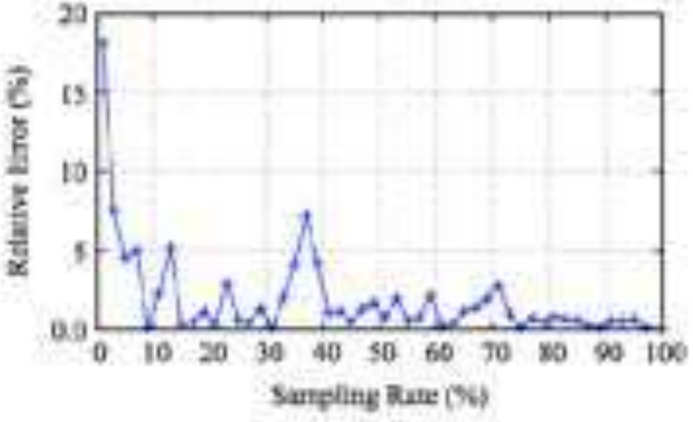


Fig. 9: Spatial variation analysis.

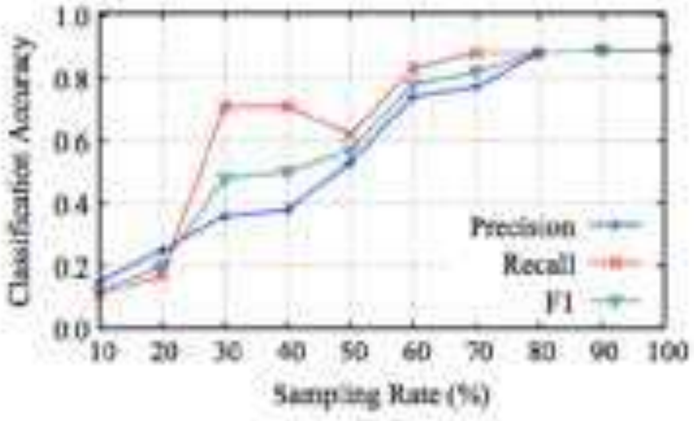
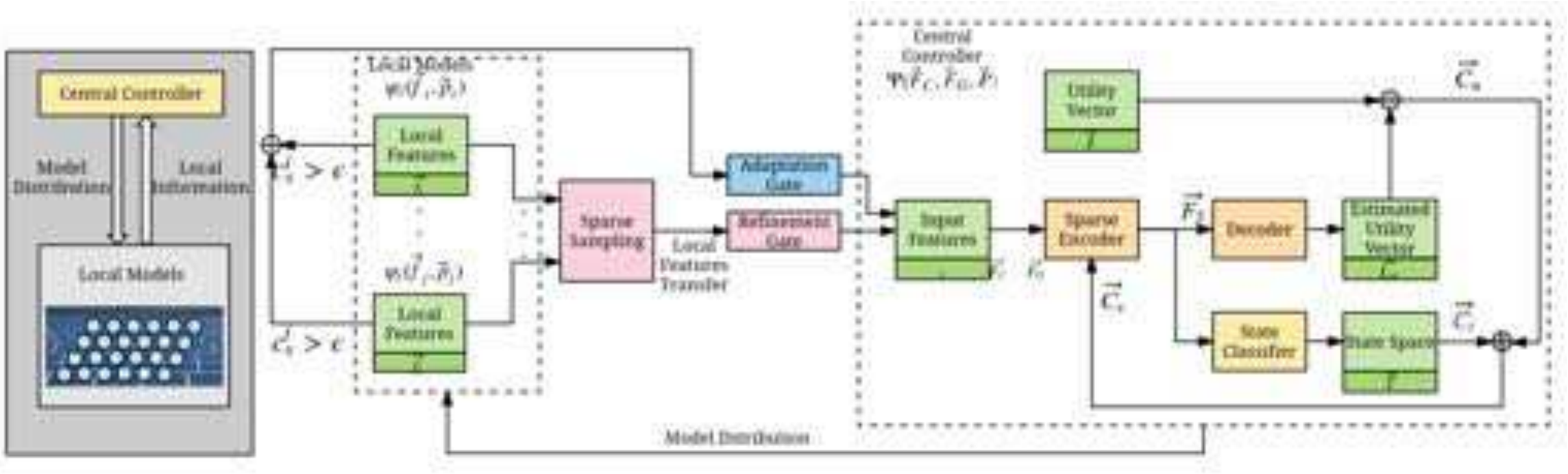


Fig. 10: State classification analysis.

Autonomous and distributed system modeling

- Reinforcement learning to derive an end-to-end system model, in the form of a computational network, distributed among the infrastructure system.

- The end-to-end model network is partitioned into and iteratively refined between two levels, namely a central controller and local models.



End-to-end system model

- The overall system performance or utility: $U(t) = \xi(u_0(t), \ldots, u_{N-1}(t))$

- The overall system state space: $\vec{S} = \{S_0, \ldots, S_{K-1}\}, \forall s_i \in S_j, 0 \leq i < N, 0 \leq j < K$

- The local models are distributed among individual sub-system components. Each local model performs in-situ feature extraction. The local models perform local performance evaluation and state classification of the corresponding local components.

$$G(\vec{x}_i, \phi_i(\vec{x}_i), \nabla\phi_i(\vec{x}_i), \ldots \nabla^k\phi_i(\vec{x}_i)) = 0, \quad \vec{x}_i \in S, 0 \leq i < N$$

$$\phi_i(\vec{x}_i) \approx a(\vec{x}_i) + \psi_i(\vec{f}_i, \vec{p}_i)$$

End-to-end system model

- Central controller: The end-to-end system model consists of a single central controller, which governs the end-to-end system modeling process via reinforcement learning and iterative model refinement and adaptation.

- The central controller is equipped with a model capturing the overall system status. The central controller model is constructed compositionally out of the information collectedly gathered from individual components.

$$\Psi(\vec{F}_C, \vec{F}_G, \vec{P}) = \psi_i(\vec{f_i}, \vec{p_i}), \text{where } \forall j \neq i, \vec{f_j} = 0$$

- The central controller is used to assess the performance of the overall system. Local models are used to monitor the run-time operation of individual components and support component-level in-situ real-time critical control activities.

Reward functions

- Reward functions: Reinforcement learning is used to govern the modeling process and automatically classify system states.

- Central controller reward function is based on the overall system modeling accuracy in order to guide the information collection process.

$$R_C(t) = U_C(\Psi(\vec{F}_C, \vec{F}_G, \vec{P}), t) - U_M(t),$$

- Local model reward function is based on the accuracy of individual local model to guide the information delivery process.

$$r_i(t) = u_i(\psi(\vec{f}_i, \vec{F}_{G_i}, \vec{p}_i)) - u_{M_i}(t).$$

Iterative sparse feature sampling

- The end-to-end system model is collaboratively developed by the central controller and local models via distributed iterative model refinement.

- Targeting large-scale distribute environment, it is then critical to minimize the data transactions between the central controller and local models.

- Iterative sparse feature sampling is governed by the reward functions through stochastic gradient descent search. The iterative sparse sampling process continues until the system modeling accuracy of the central controller reaches the required threshold.

$$\nabla R_C(t) = \nabla U_C(\Psi(\vec{F}_C, \vec{F}_G, \vec{P}), t), \forall F^j \in \{\vec{F}_C, \vec{F}_G\}, 0 < j < M$$

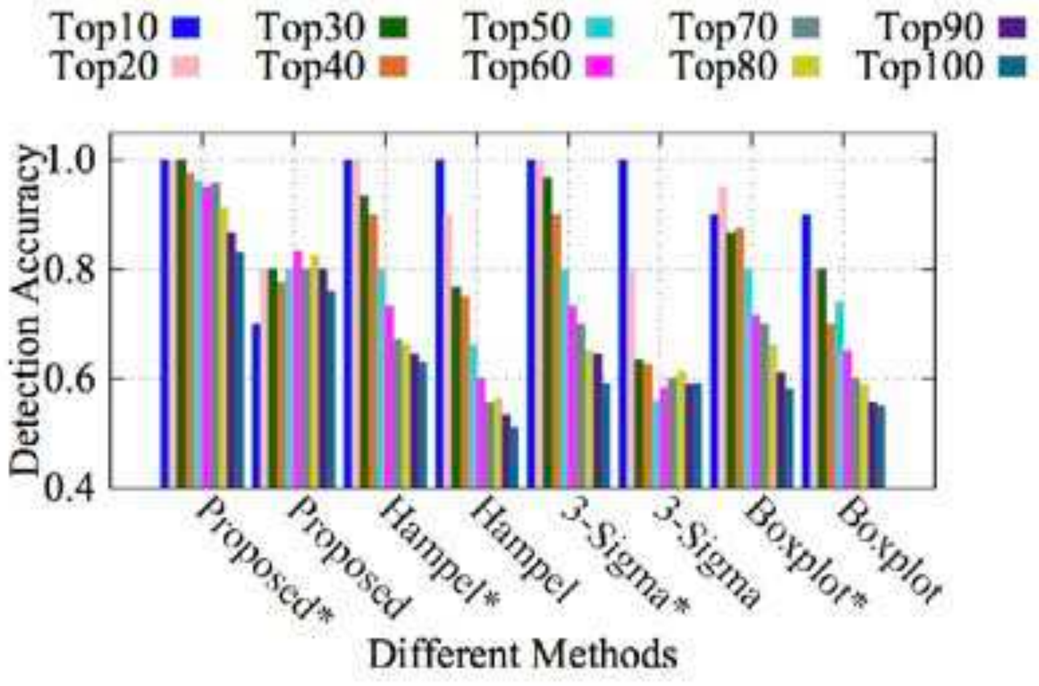
Model refinement and adaptation

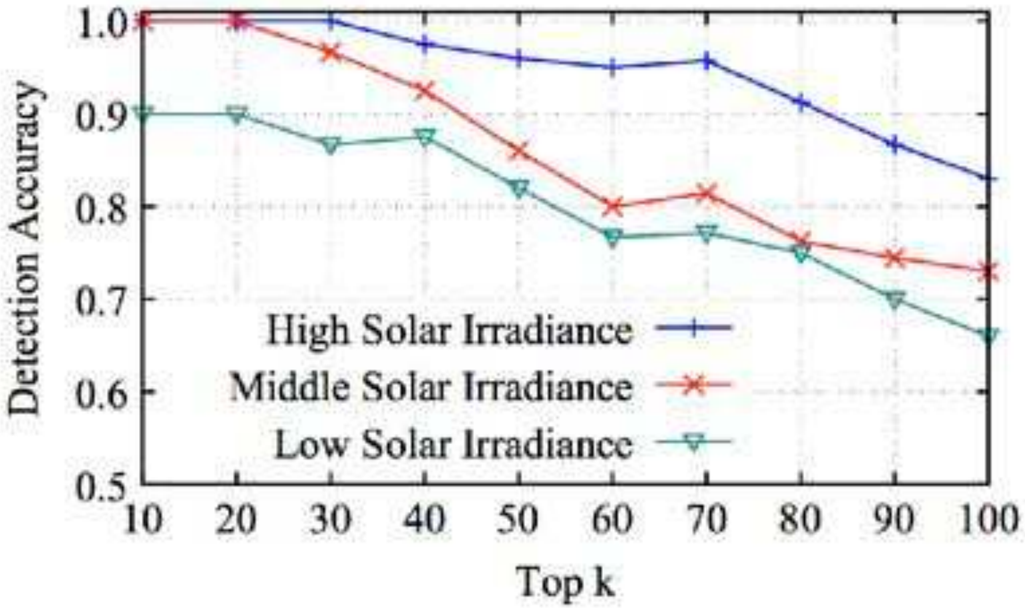
- Model refinement is to train and fine-tune the hyper-parameters tailored to individual components using their local feature data.

- Model deputation is to reevaluate the performance of utility prediction and system state classification.

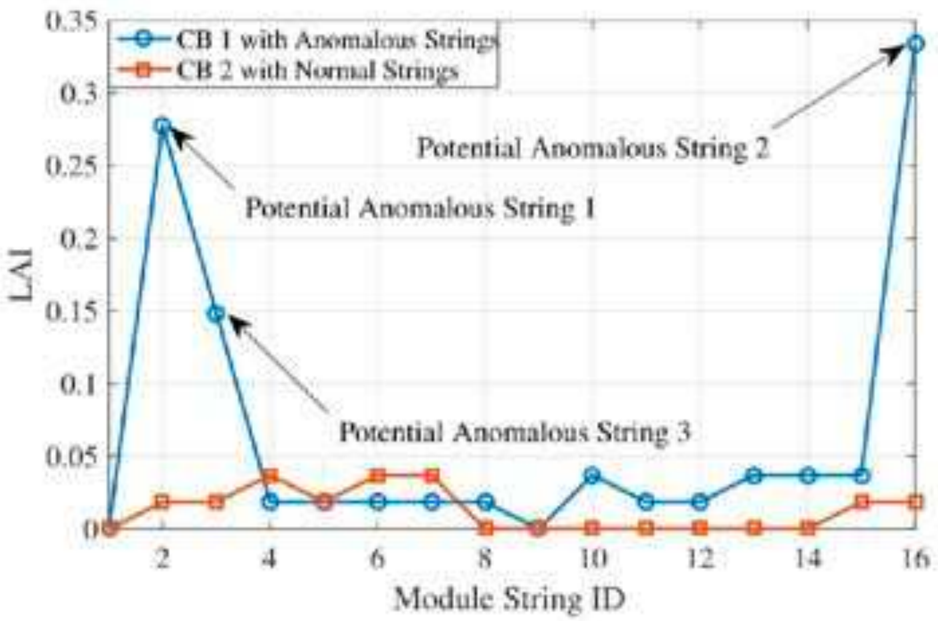
- Sparse encoding and deciding model is used for feature reduction. System state classification model is a set of weak model candidates, initially, each of which has certain level of state estimation capability, and can be learned to collectively produce the best state estimation results via minimizing the cost function.

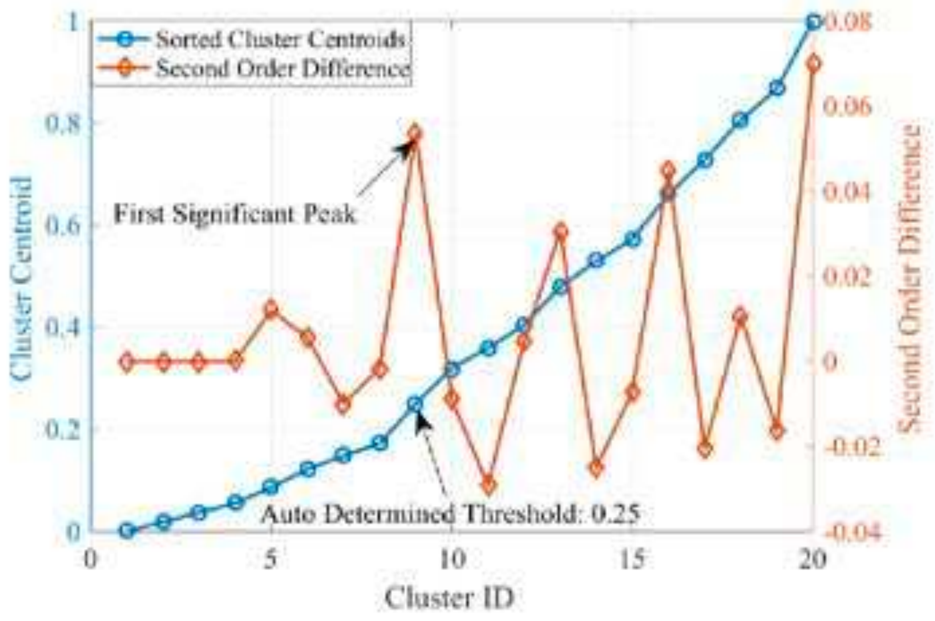
Case study: Anomaly detection evaluation



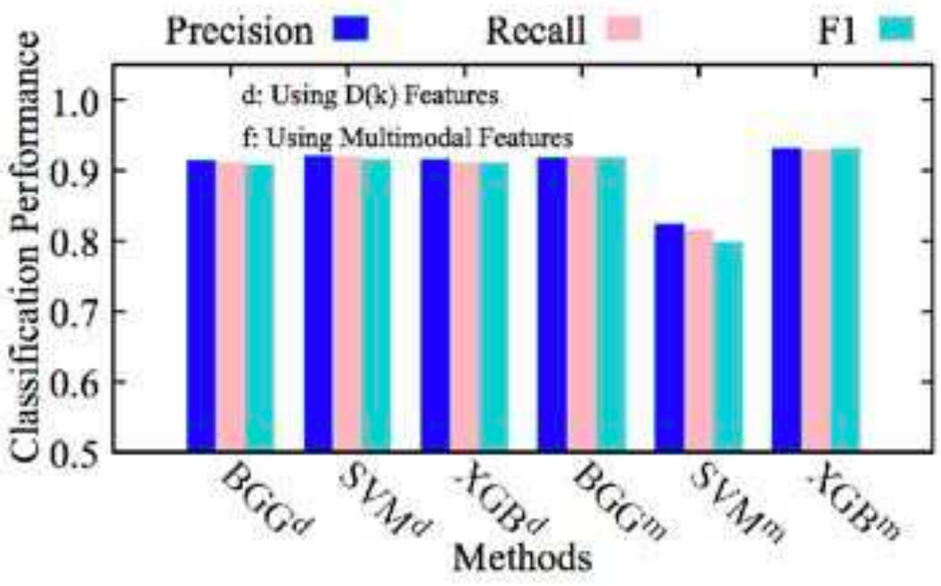


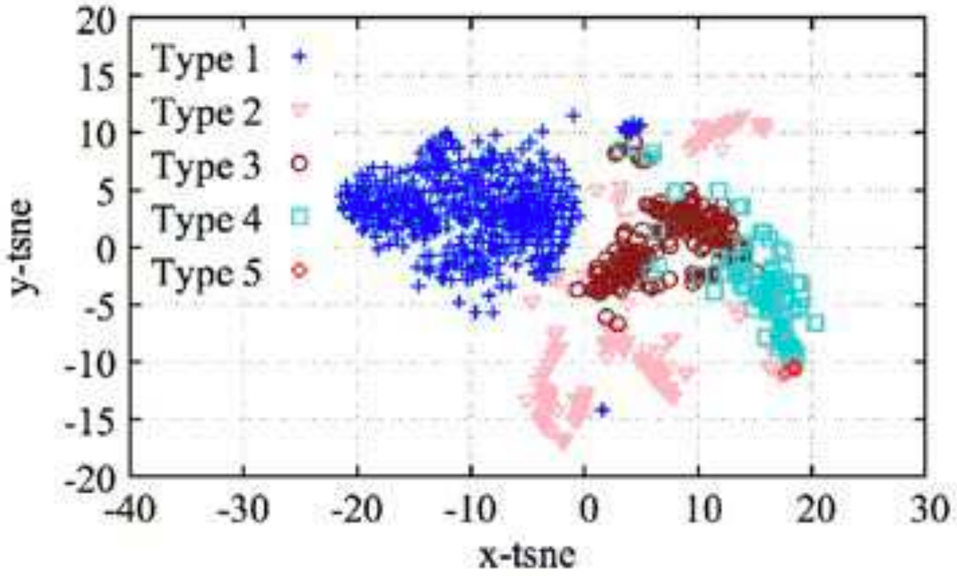
Case study: Anomaly detection evaluation



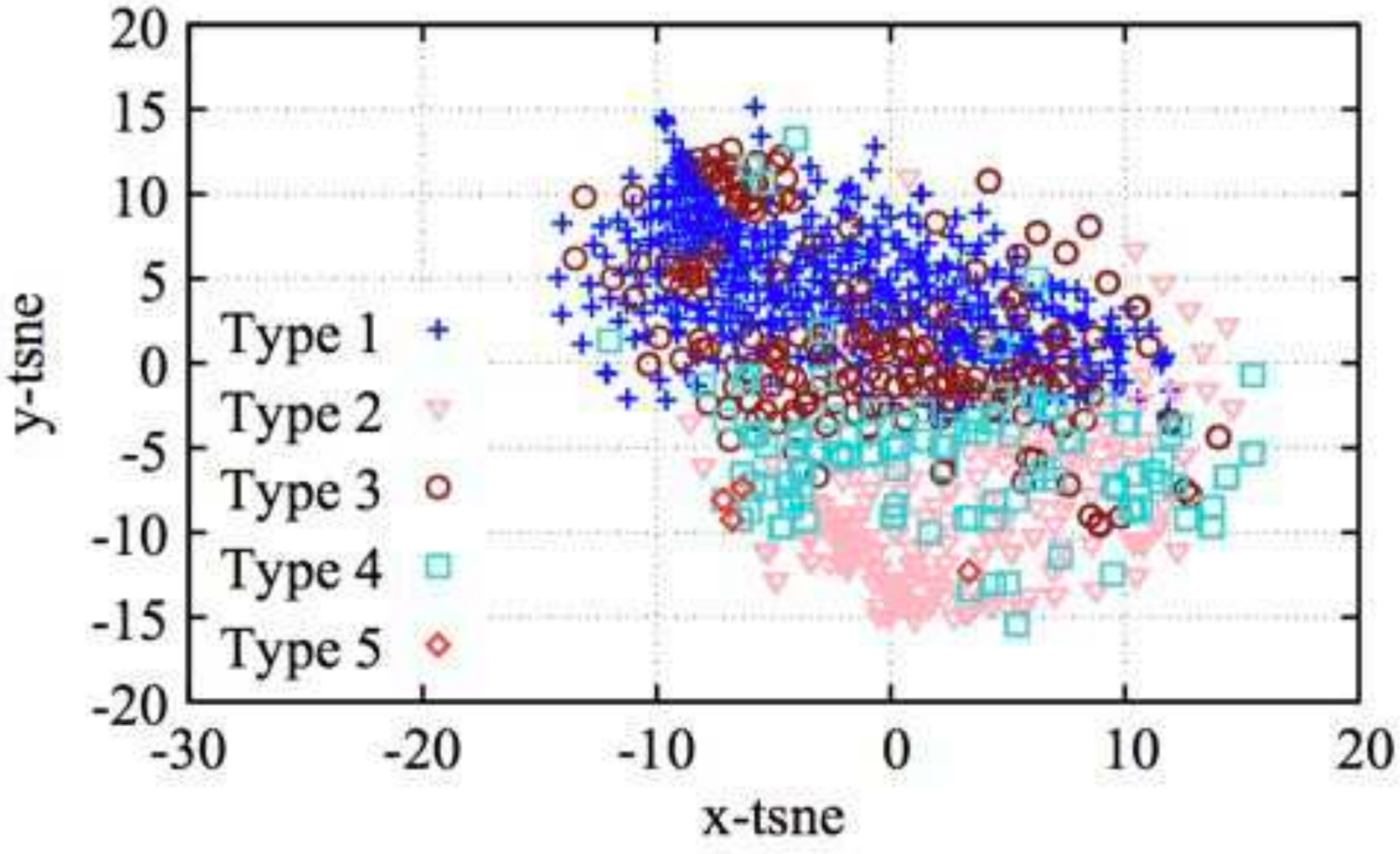


Case study: Anomaly classification evaluation





Case study: Anomaly classification evaluation



Challenges revisit

- What are the fundamental system modeling challenges for large-scale infrastructure systems?
- How to model large-scale infrastructure systems using detailed but noisy sensing data with limited labels?
- How to connect the needs and increasing amounts of data in real-world infrastructure systems with research community?