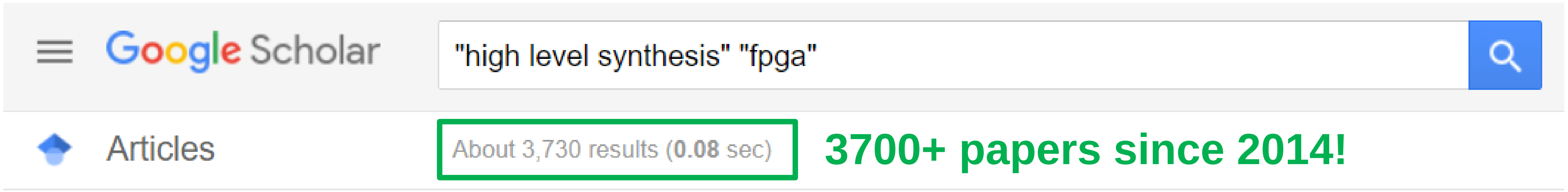
Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs

Yuan Zhou, Udit Gupta, Steve Dai, Ritchie Zhao, Nitish Srivastava, Hanchen Jin, Joseph Featherston, Yi-Hsiang Lai, Gai Liu, Gustavo Angarita Velasquez, Wenping Wang, Zhiru Zhang

Computer Systems Lab
Electrical and Computer Engineering
Cornell University

Increasing Deployment of HLS for FPGAs


Google Scholar

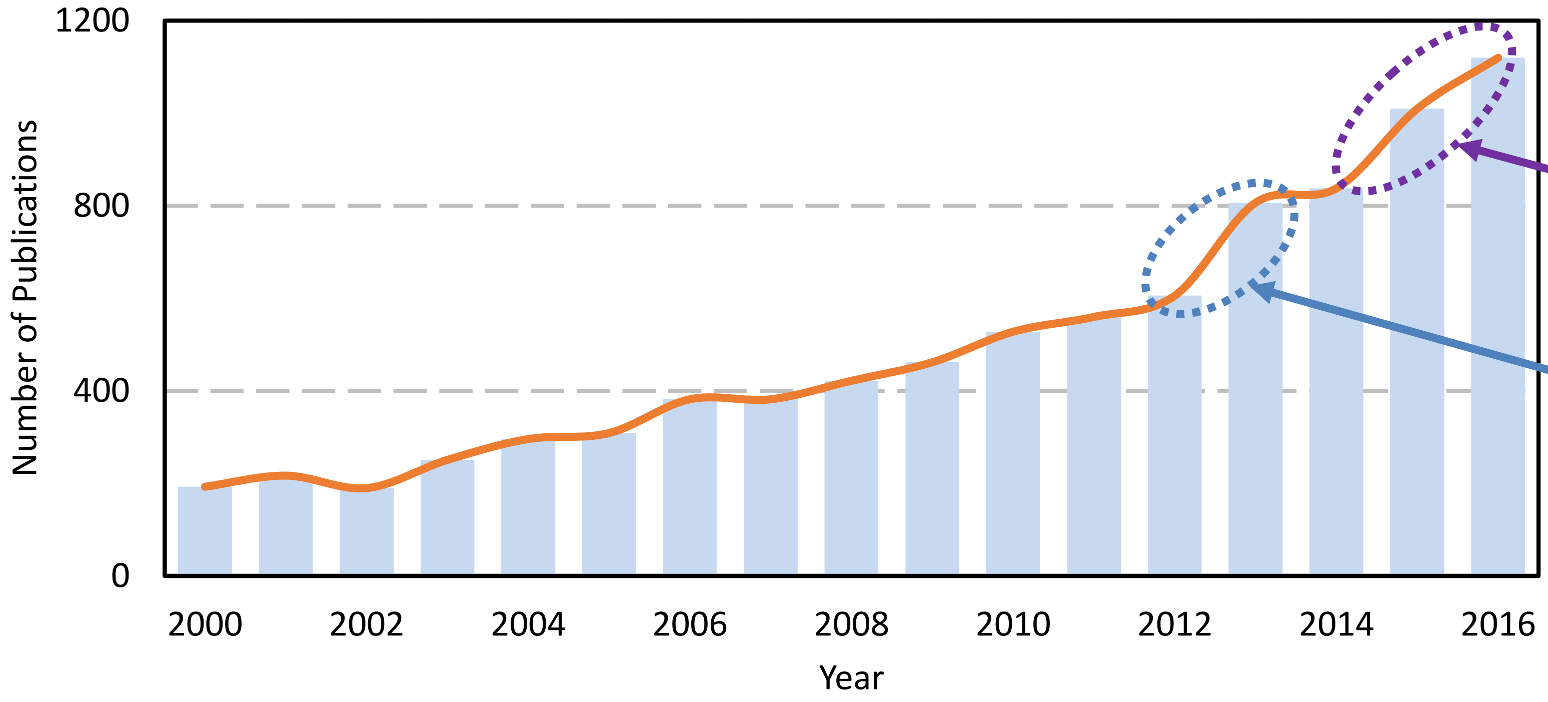
"high level synthesis" "fpga"

Articles

About 3,730 results (**0.08** sec)

3700+ papers since 2014!

- Google scholar trend on "HLS for FPGAs"



High demand for deep learning acceleration on FPGAs

Software-programmable FPGA SoCs becoming available

Need for Realistic HLS Benchmarks

- **Need for testing a rich set of synthesis directives in modern HLS tools**
  - Loop unrolling, pipelining, array partitioning, …

SDAccel™ Environment SDSoC™ Environment Intel® HLS Compiler Altera® SDK for OpenCL™ LegUp HLS PandA Bambu HLS

- **Need for evaluating new classes of HLS optimization techniques**
  - Fast design space exploration [Shao et.al, ISCA'14] [Zhao et.al, ICCAD'17] …
  - Automatic memory banking & reuse [Wang et.al, FPGA'14] [Zhou et.al, FPGA'17] …
  - Polyhedral loop transformations [Pouchet et.al, FPGA'13] [Zhang et.al, FPGA'15] …

    …

Related Work

- Popular benchmark suites used by the HLS community

Benchmark Suite	Language	Primary Focus	#Benchmarks	Avg. Kernel Length (LOC)	User Directives Applied
CHStone [Hara et.al, JIP'08]	C	Synthesizability check	12	707	/
Rodinia [Che et.al, IISWC'09]	OpenCL	GPU benchmarking	21	356	/
PolyBench [Pouchet, 2010]	C	Polyhedral analysis	30	29	/
MachSuite [Reagen et.al, IISWC'14]	C/C++	Kernel selection & optimization	19	88	Unrolling, pipelining, array partitioning, functional unit selection
Spector [Gautier et.al, FPT'16]	OpenCL	Design space exploration	9	204	Unrolling, pipelining, OpenCL vector type & workgroup size

- Existing HLS benchmarks typically lack
  1. Large complex applications with realistic design constraints
  2. Optimized HLS implementation on real FPGA devices

Rosetta: A Realistic HLS Benchmark Suite

Realistic applications with optimizations and constraints

- Complementary to existing benchmark suites

Retargetable to different FPGA platforms and HLS tools

- Specified in multiple programming languages

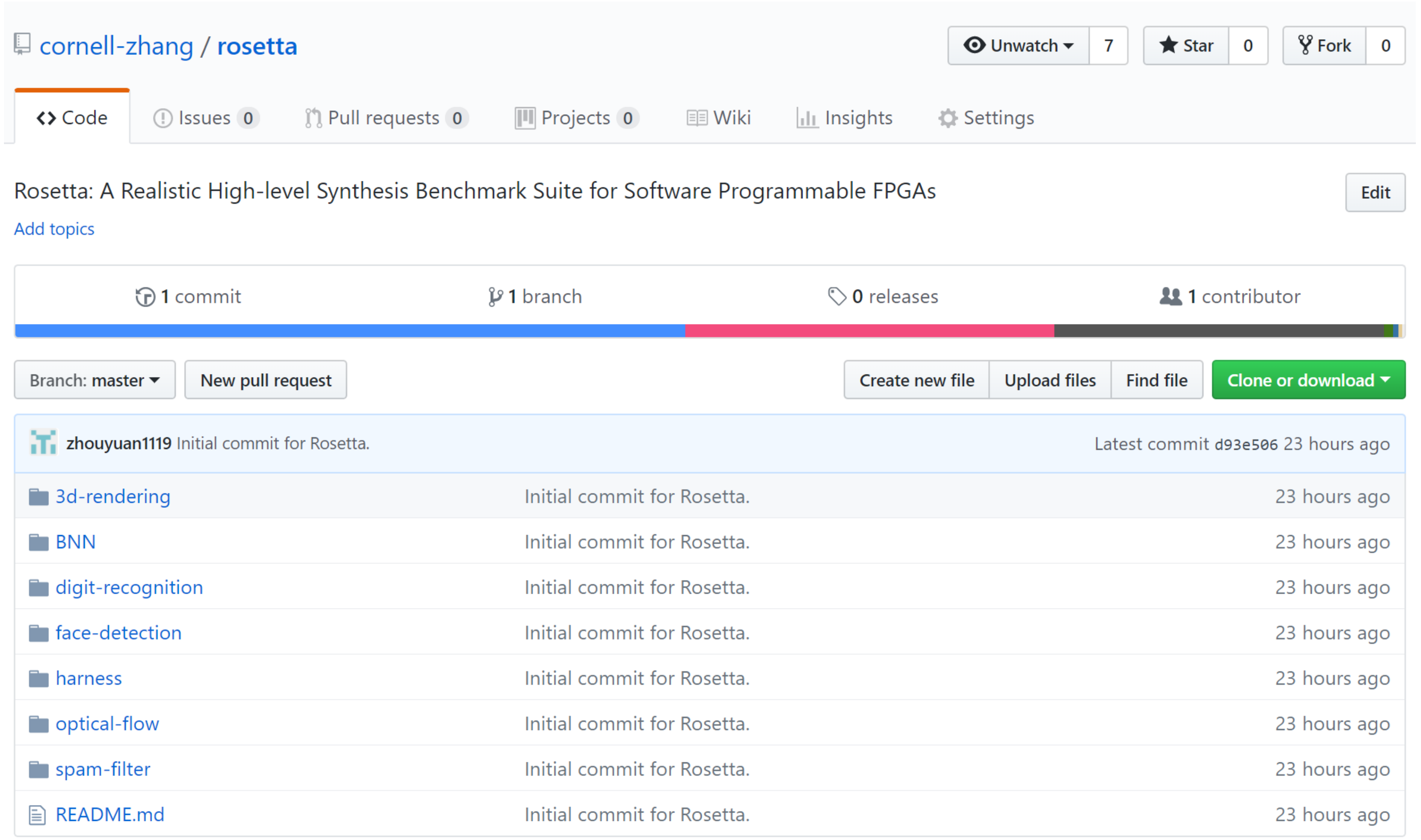
Reducible to sub-kernels for micro-benchmarking

Rosetta gets the name following the convention of a plethora of "stone" benchmark suites. It also symbolizes that our benchmarks are specified in multiple languages (i.e., C++, OpenCL) and useful for evaluating HLS across different tools and platforms.

Figure source: https://discoveringegypt.com/egyptian-video-documentaries/mystery-of-the-rosetta-stone/

Open-Source Release on GitHub

- github.com/cornell-zhang/rosetta



Rosetta: A Realistic High-level Synthesis Benchmark Suite for Software Programmable FPGAs

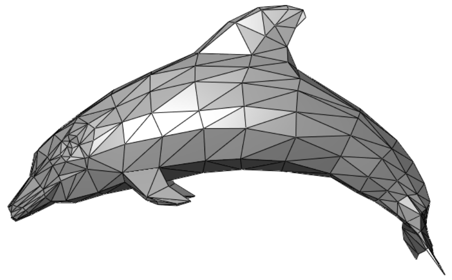
1 commit | 1 branch | 0 releases | 1 contributor

zhouyuan1119 Initial commit for Rosetta. Latest commit d93e506 23 hours ago

3d-rendering	Initial commit for Rosetta.	23 hours ago
BNN	Initial commit for Rosetta.	23 hours ago
digit-recognition	Initial commit for Rosetta.	23 hours ago
face-detection	Initial commit for Rosetta.	23 hours ago
harness	Initial commit for Rosetta.	23 hours ago
optical-flow	Initial commit for Rosetta.	23 hours ago
spam-filter	Initial commit for Rosetta.	23 hours ago
README.md	Initial commit for Rosetta.	23 hours ago

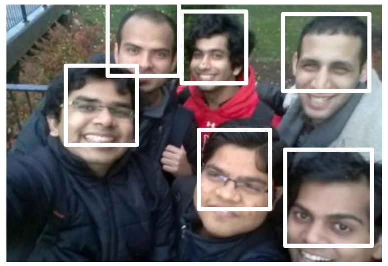
Rosetta Overview

3D Rendering

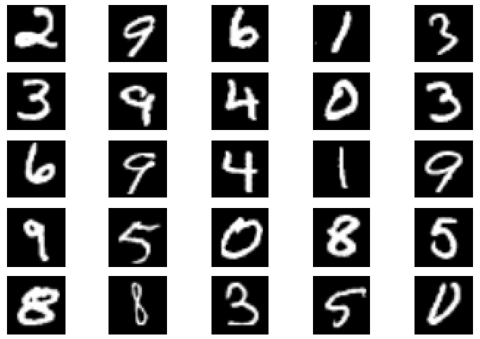


Optical Flow

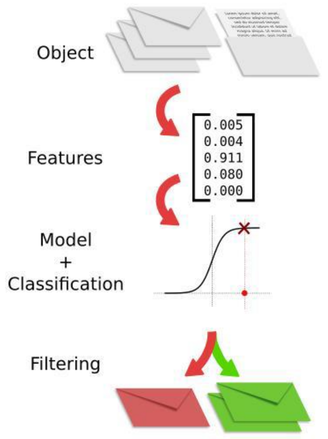
Face Detection



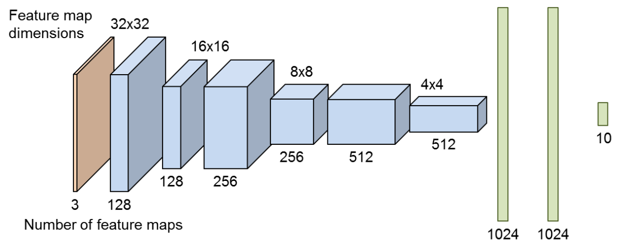
Digit Recognition



Spam Filtering



Binarized Neural Network



Rosetta Overview

- Programming languages

- Current targets

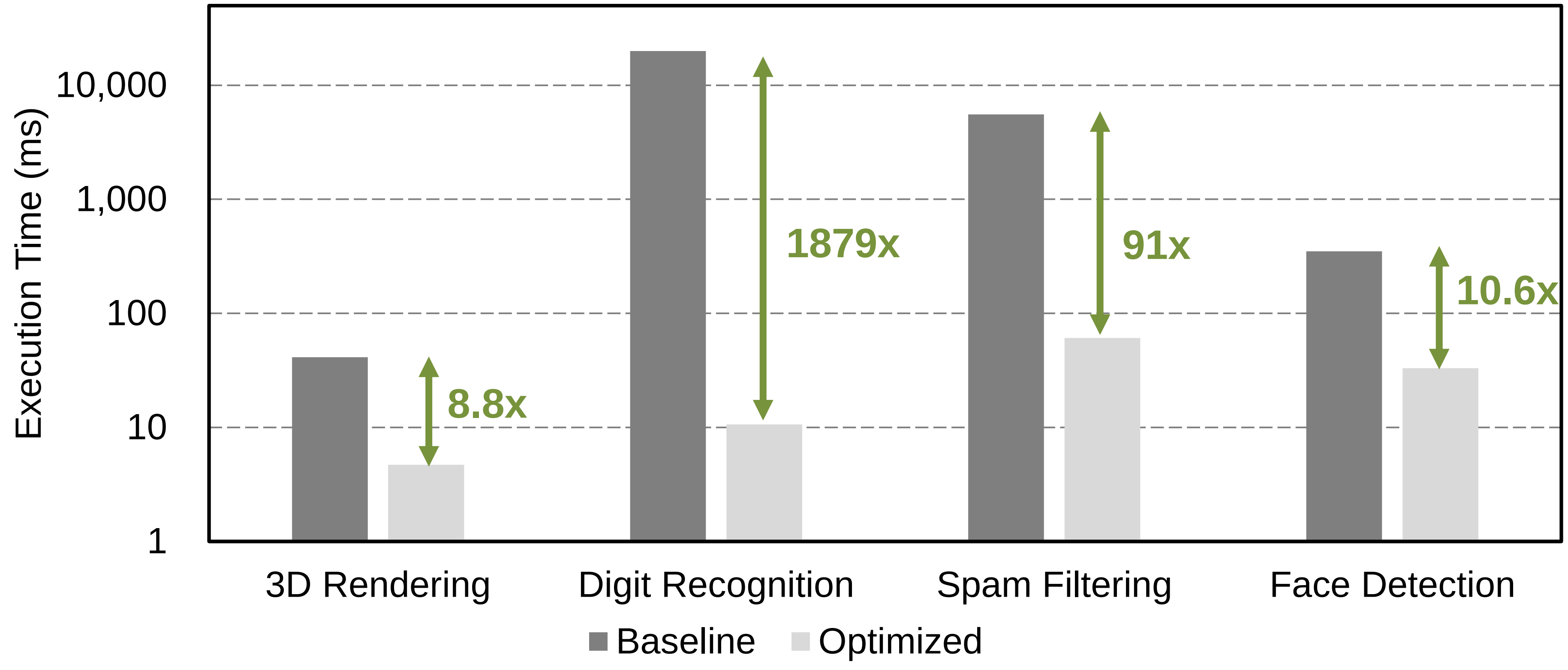
Cloud FPGA (AWS F1)

Embedded FPGA (Xilinx ZC706)

- Synthesizable baseline designs also provided

Software Baseline vs. Optimized Version

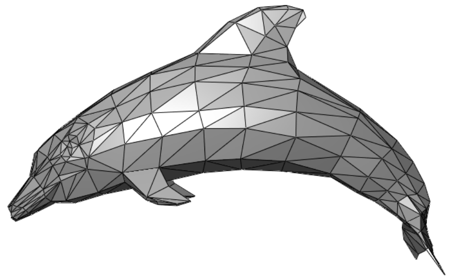
- Execution times on ZC706



Rosetta is also useful as a design tutorial on usage of HLS optimization directives

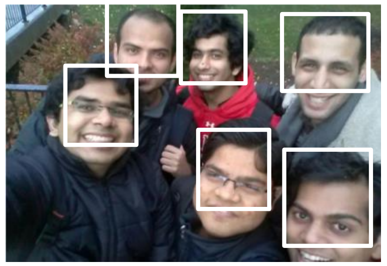
Rosetta Overview

3D Rendering

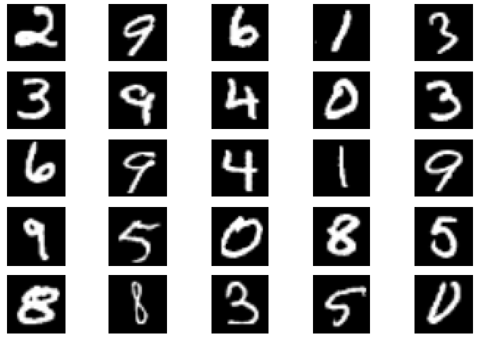


Optical Flow

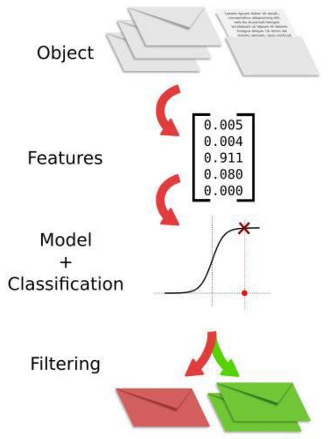
Face Detection



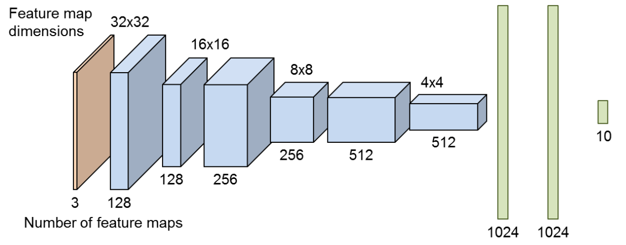
Digit Recognition



Spam Filtering

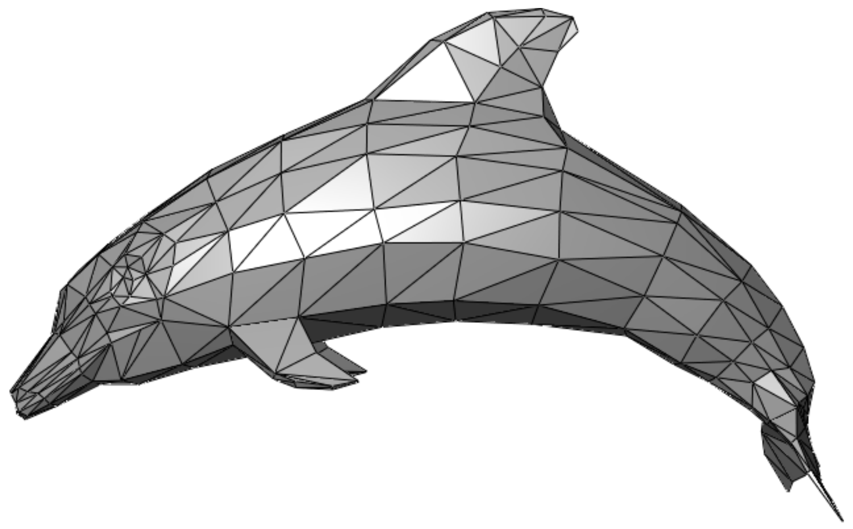


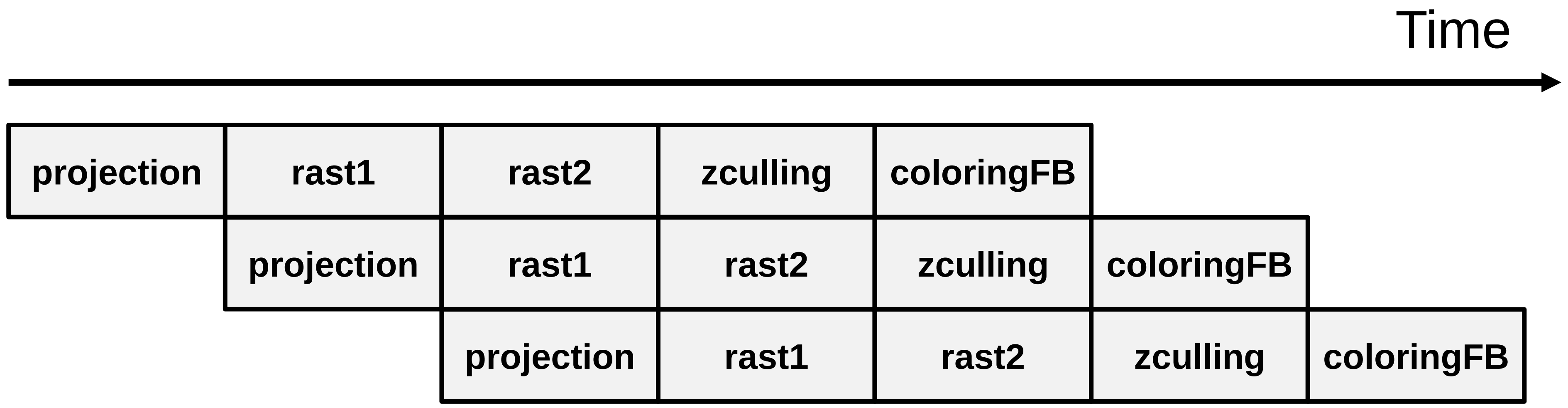
Binarized Neural Network



3D Rendering

- Render images from 3D triangle meshes
- Performance constraint: real-time
- Dataflow optimization

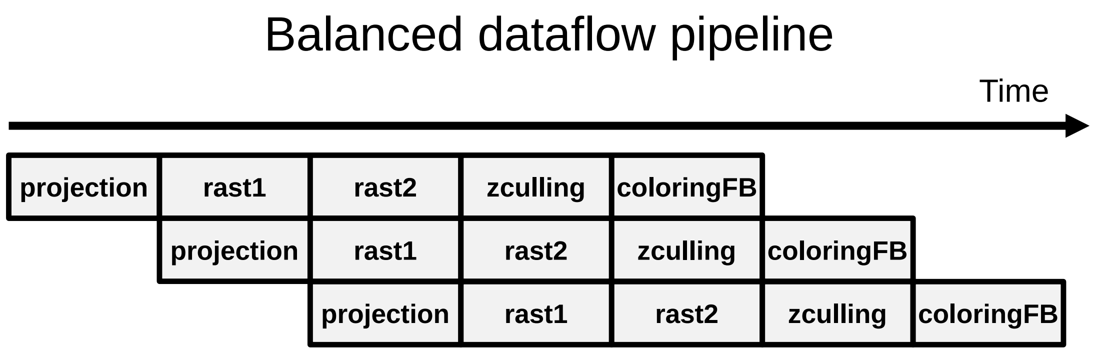


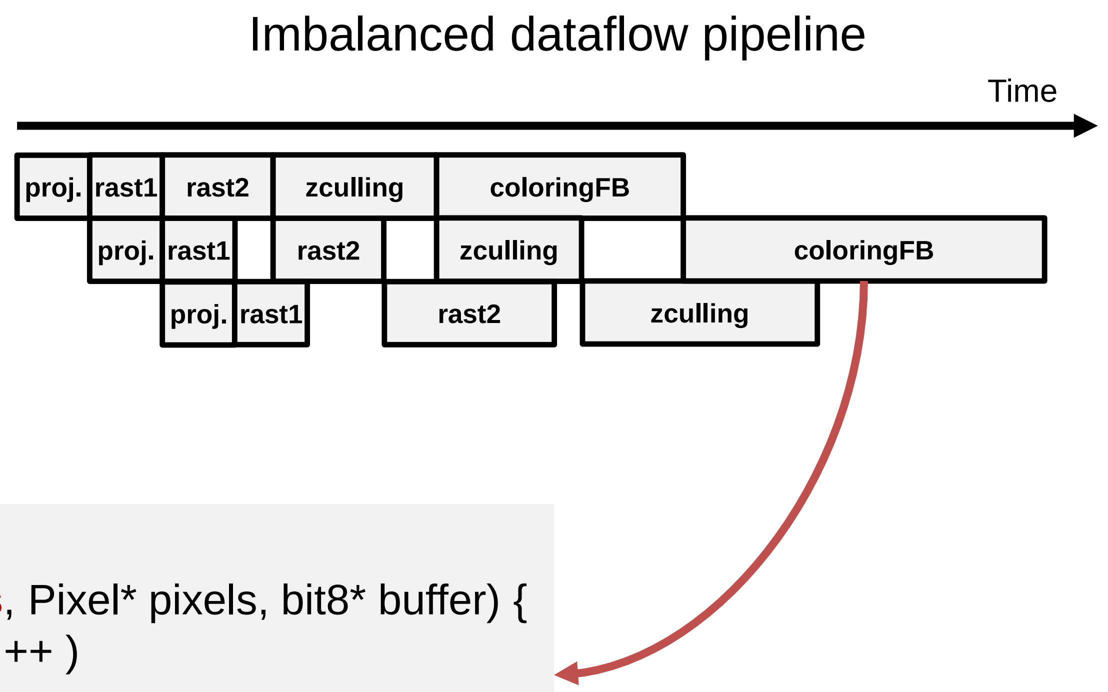


Dolphin figure source: https://opentechschool-brussels.github.io/intro-to-webGL-and-shaders/log1_graphic-pipeline

Challenge: Stage Balancing with Data-Dependent Loops

Pipeline stalls negatively affect performance





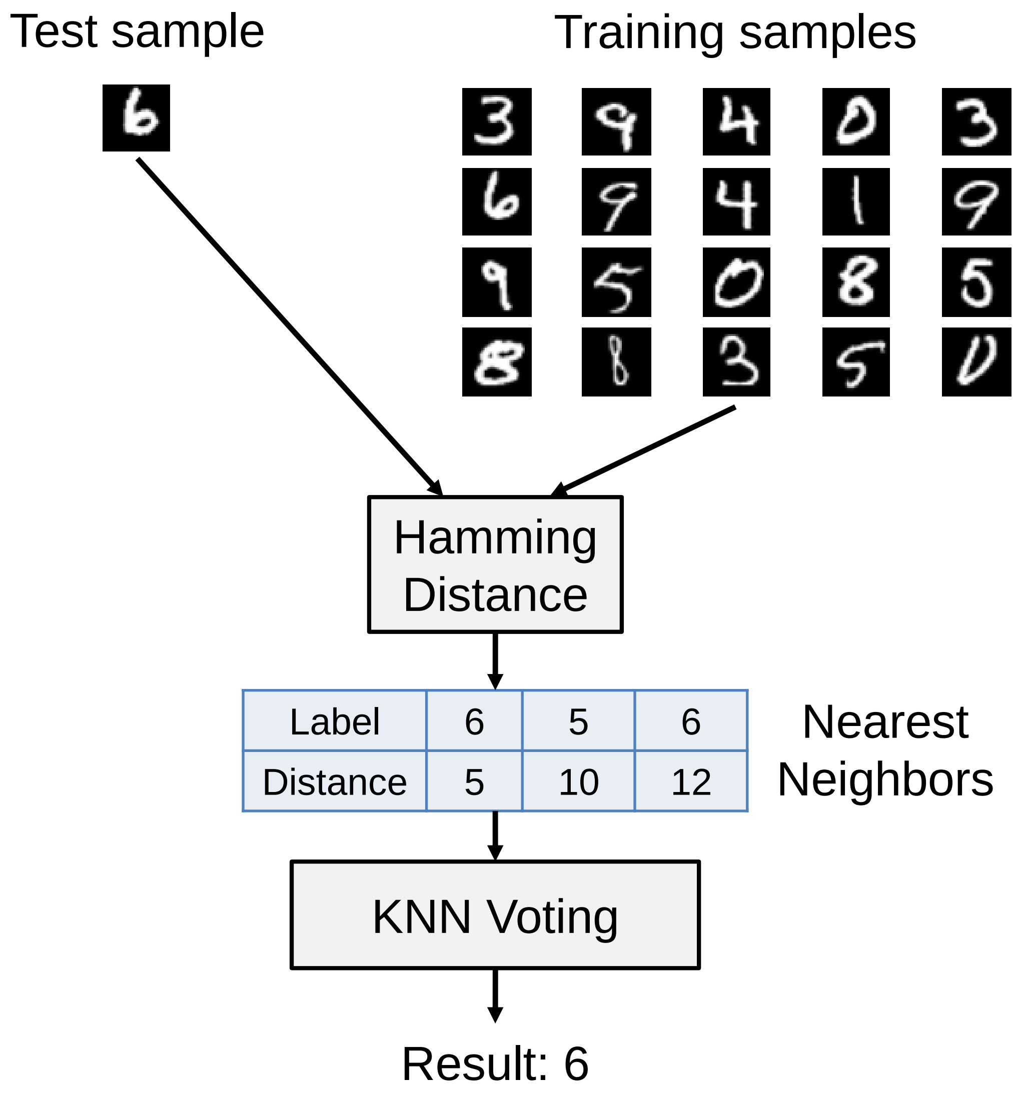
Latency is data-dependent

Difficult to balance the pipeline stages without profiling

```
// color the frame buffer
void coloringFB( bit16 n_pixels, Pixel* pixels, bit8* buffer) {
 for ( bit16 i = 0; i < n_pixels; i ++ )
   #pragma pipeline
   buffer[ pixels[i].x * MAX_Y + pixels[i].y ] = pixels[i].color;
}
```

Digit Recognition

- K-nearest-neighbor digit recognition
  - Downsampled, binarized MNIST subset
    - 14x14 resolution => 196 bits per digit
- Compute kernels
  - Hamming distance calculation: bit-level operation intensive
  - Sorting-based KNN voting



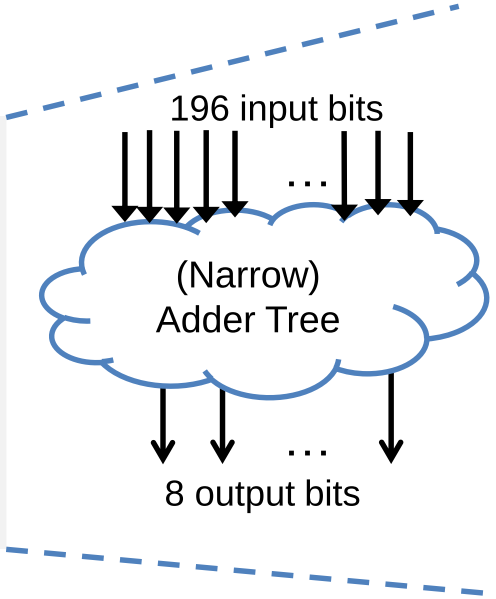
Label	6	5	6
Distance	5	10	12

MNIST image source : http://andrew.gibiansky.com/blog/machine-learning/k-nearest-neighbors-simplest-machine-learning/

Challenge: Area/Delay Estimation for Bitwise Operations

- Popcount of a wide integer: a subroutine in hamming distance kernel

```
bit8 popcount(bit196 digit)
{
  #pragma pipeline
  bit8 ones = 0;
  for (int  i = 0; i < 196; i++)
    ones += digit[i];
  return ones;
}
```



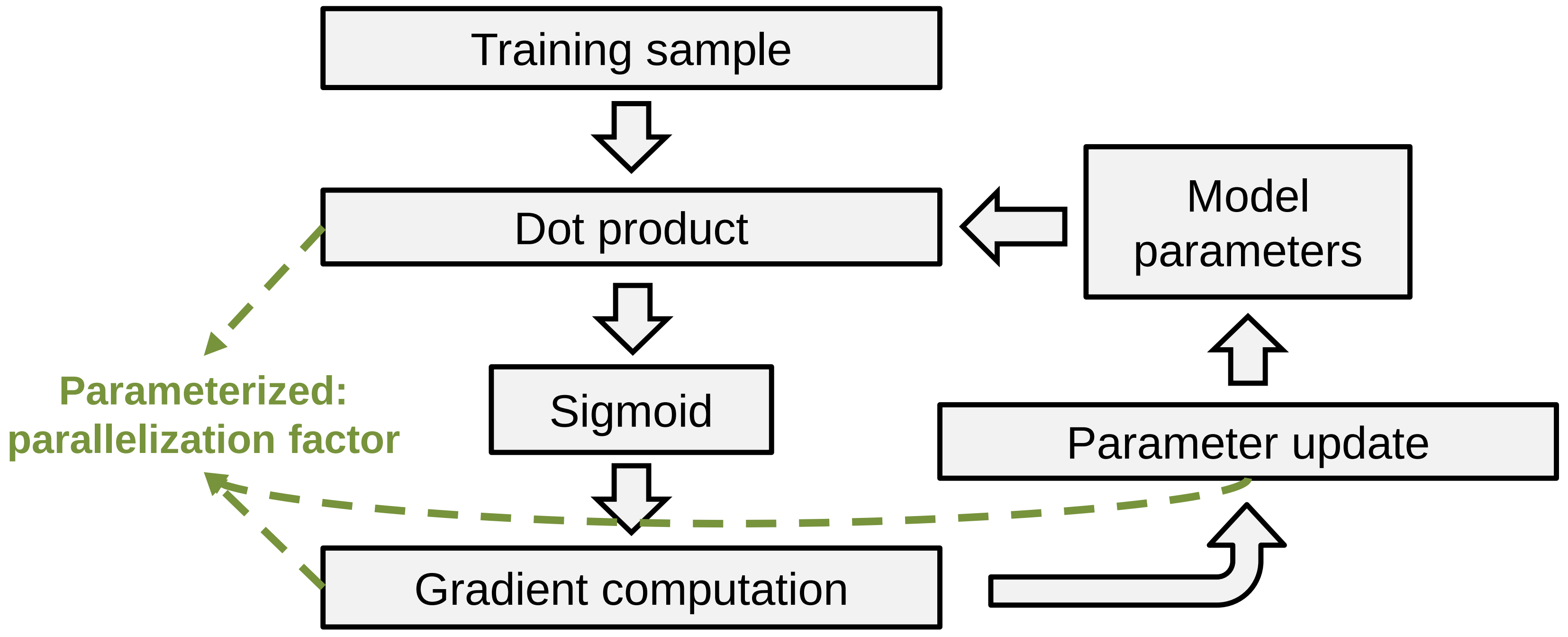
HLS delay estimate: 5.88 ns
Post PAR: 2.88 ns

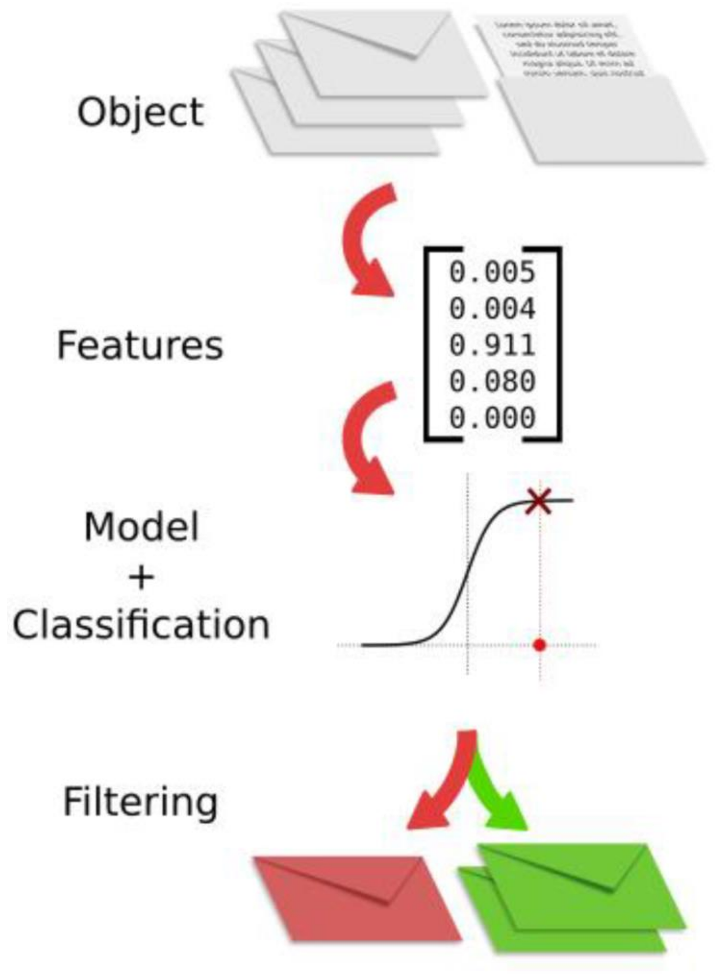
Resource Estimate	LUT	FF
HLS	1505	69
Post PAR	245	68

Observation: Due to lack of awareness of post-RTL optimization, HLS may overestimate delay & resource of bit-level ops, resulting in suboptimal QoR

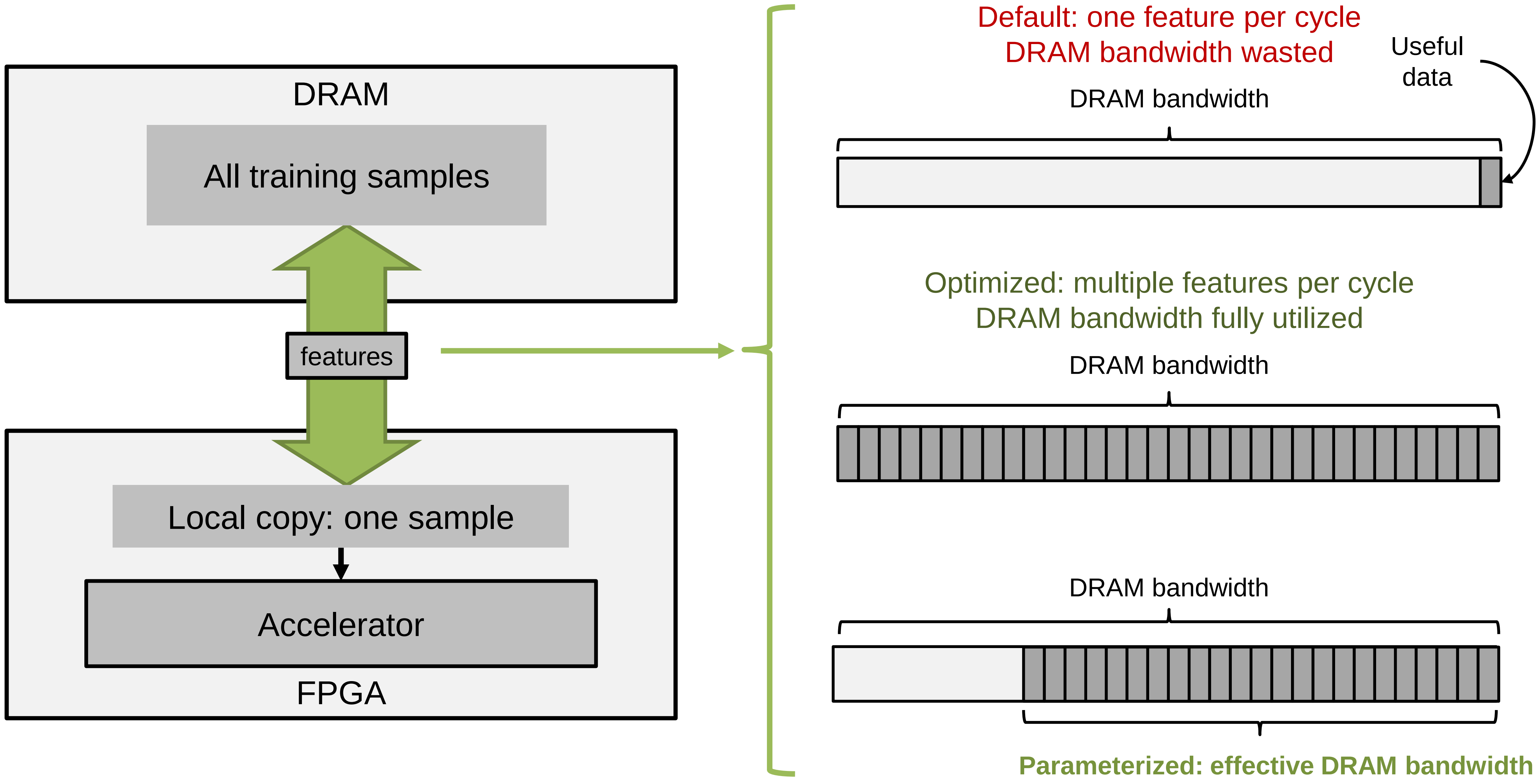
Spam Filtering

- Train a logistic regression model for email classification
  - Stochastic gradient descent (SGD)
- Representative kernels in machine learning
  - Dot product, vector addition, sigmoid function

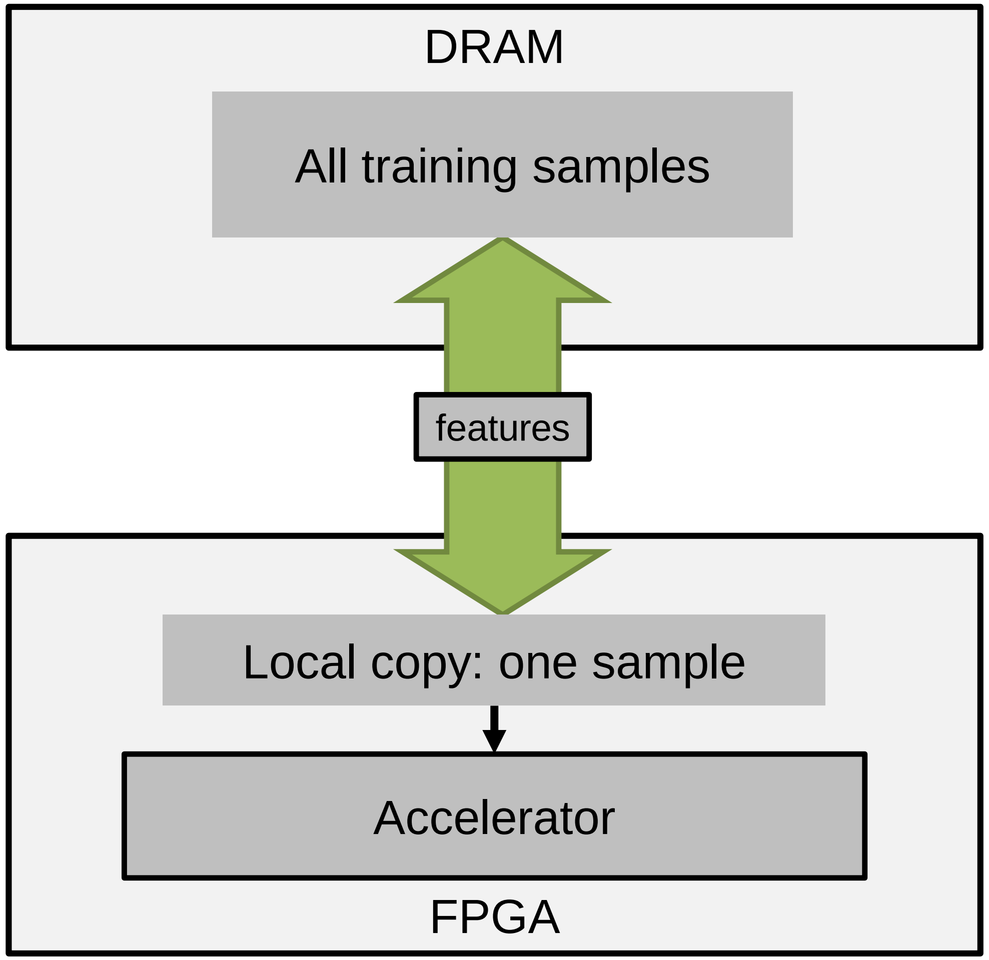


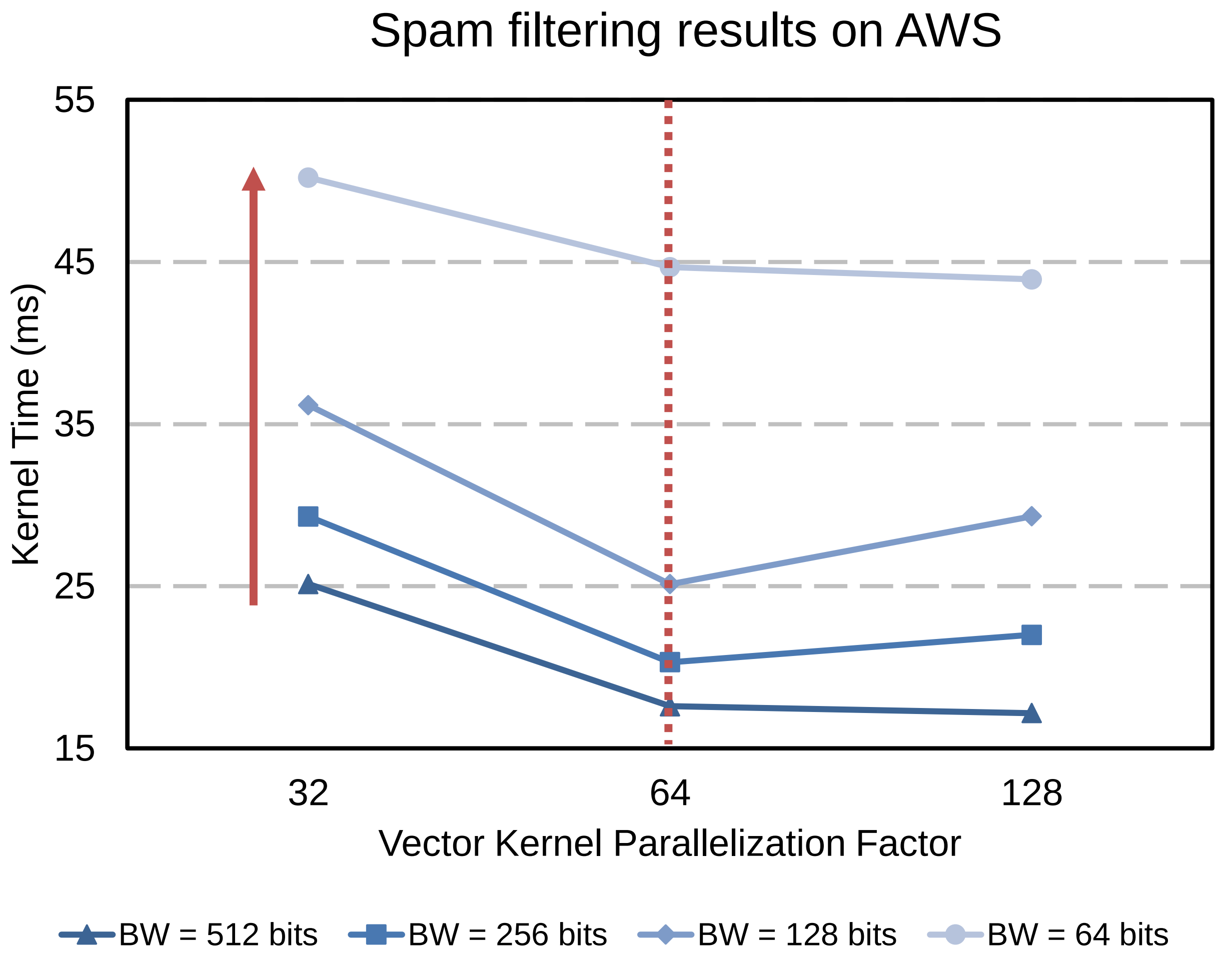


Challenge: Saturating Memory Bandwidth



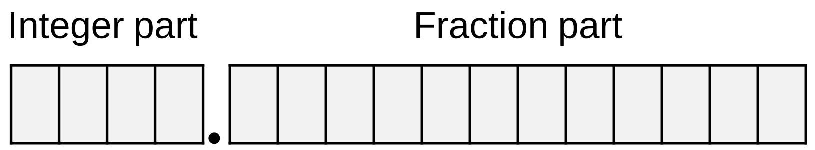
A Memory-bound Application

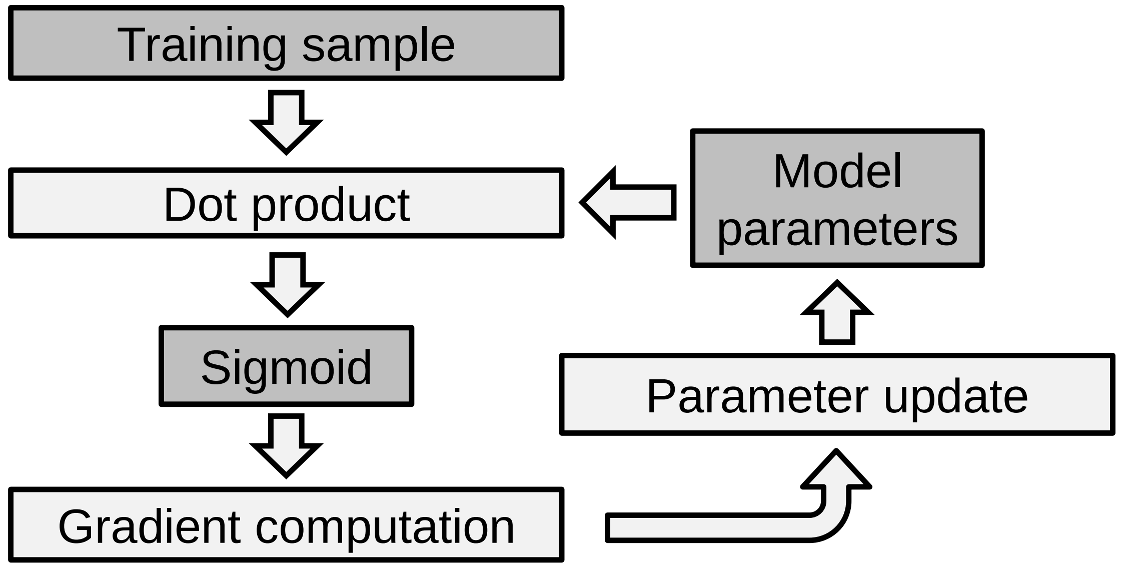




Challenge: Tuning Fixed-point Bitwidth

- Datatype customization with fixed-point types
  - Training samples
  - Parameter vector
  - Sigmoid values
  - Other intermediate results

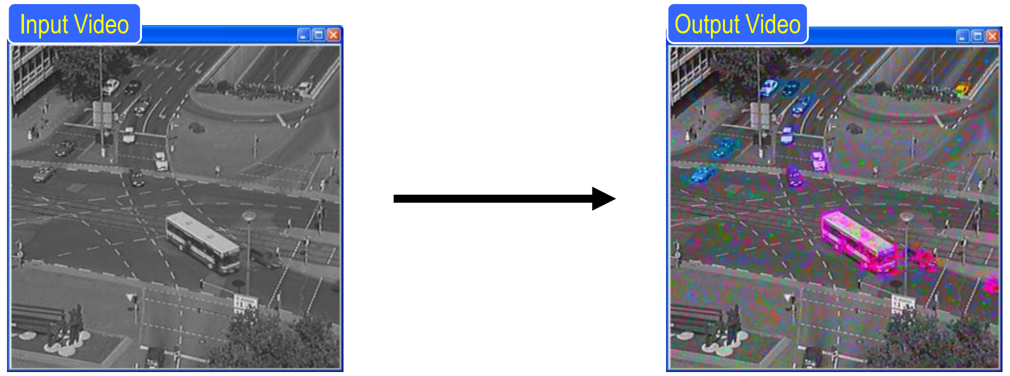




Require careful tuning of the bitwidths for a good trade-off between model accuracy, throughput and area

Optical Flow

- Computing/estimating motion field from time-varying image intensity

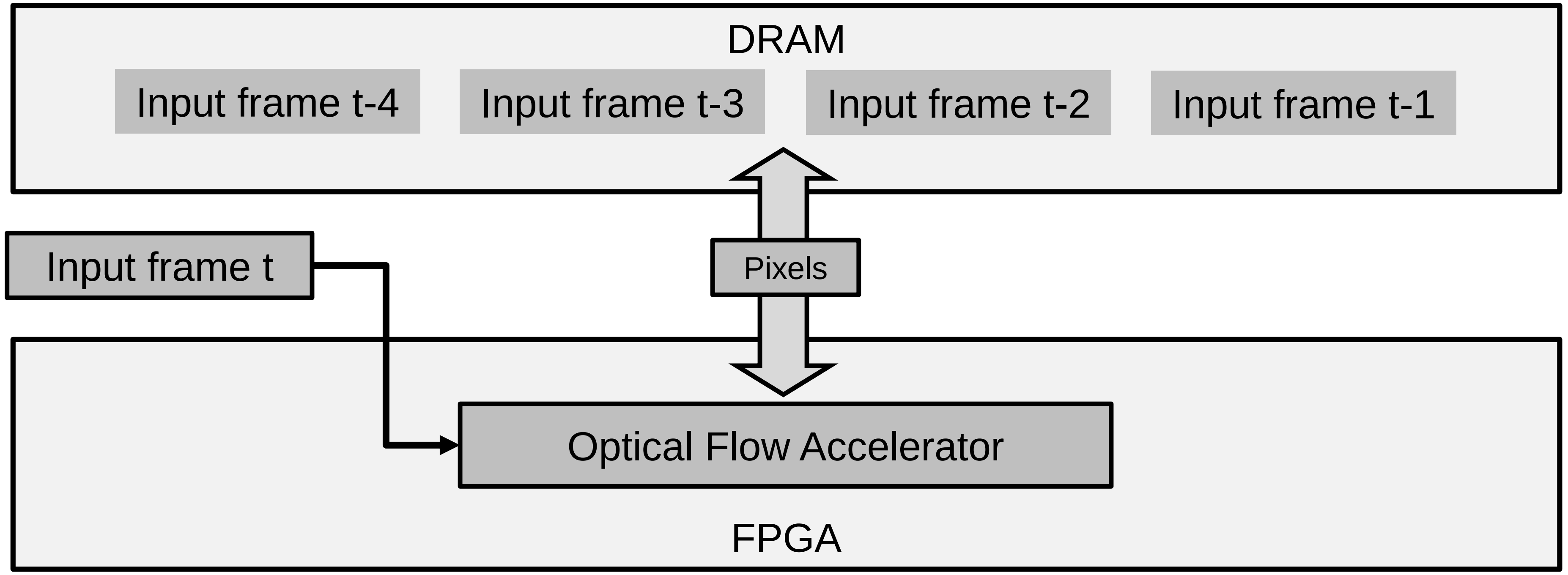


- Compute kernels: Eight-stage of image filtering
  - 1D and 2D stencil access patterns

- Performance constraint: 30 frames/s

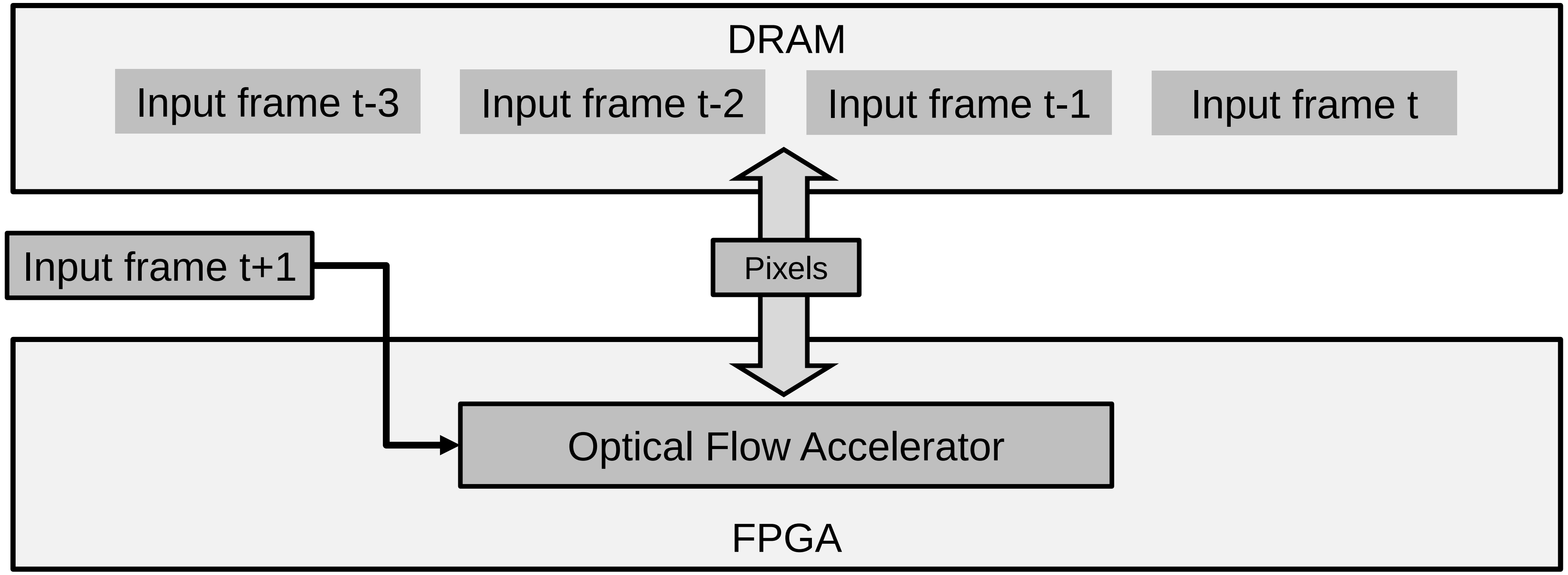
Challenge: Data Reuse Across Frames

- Data reuse in the time dimension
  - Optical flow kernel reads from previous four frames
- Buffer previous image frames in DRAM



Challenge: Data Reuse Across Frames

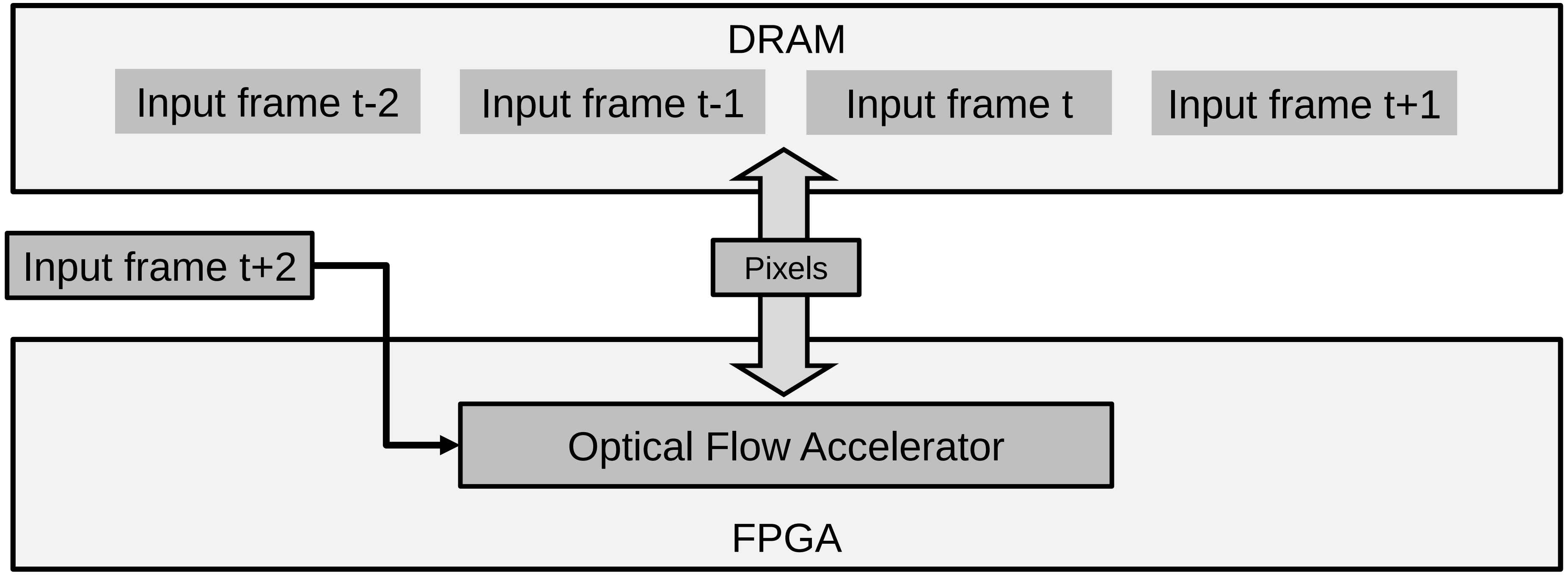
- Data reuse in the time dimension
  - Optical flow kernel reads from previous four frames
- Buffer previous image frames in DRAM



Challenge: Data Reuse Across Frames

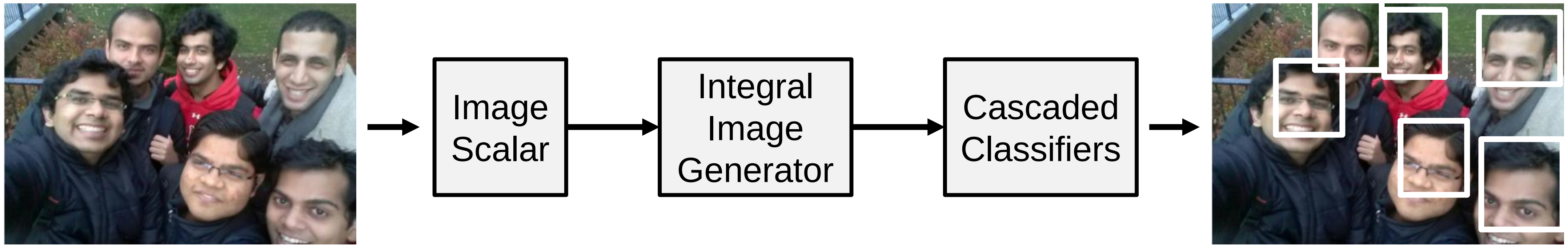
- Data reuse in the time dimension
  - Optical flow kernel reads from previous four frames
- Buffer previous image frames in DRAM

Automatic reuse technique required



Face Detection [Srivastava et.al, FPGA'17]

- Viola-Jones face detection algorithm



- Compute kernels
  - Image scaling (memory intensive)
  - Cascaded classifiers (integer arithmetic intensive)
- Performance constraint: 30 frames/s

Figure adopted from: N. Srivastava, FPGA'17 presentation

Challenge: Memory Banking with Irregular Access Patterns

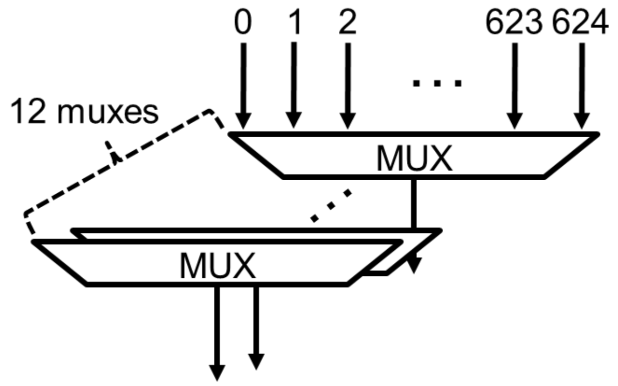
- Irregular memory access pattern of classifiers

```
int cascadeClassifier( Pixel IntImg[][], …) {
  for ( i = 0; i < 25; i++ ) {
    …
    for ( j = 0; j < stages_array[i] ; j++ ) {
      #pragma pipeline
      ….
      c[0] = IntImg[r0.y][r0.x];
      c[1] = IntImg[r0.y][r0.x + r0.w];
      ….
      c[10] = IntImg[pt.y + r2.y + r2.h][pt.x + r2.x];
      c[11] = IntImg[pt.y + r2.y + r2.h][pt.x + r2.x + r2.w];
      ….
      }
} }
```

Memory address have no obvious patterns

- Apply a customized partitioning scheme
  - Led to a trace-based memory banking work [Zhou et.al, FPGA'17]

Default: Twelve 625x1 muxes
170K LUTs, fail timing



Optimized: Two-level mux network
16K LUTs, timing met

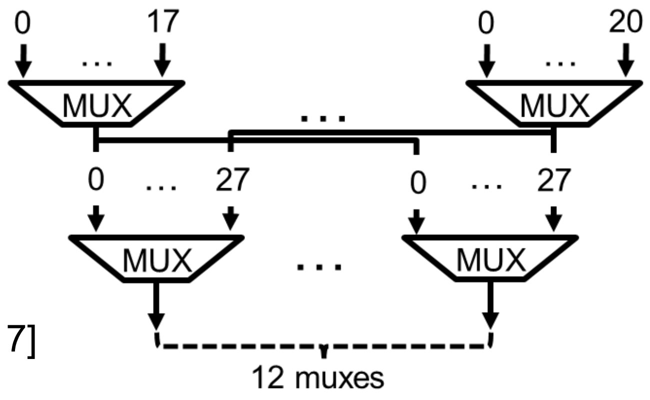
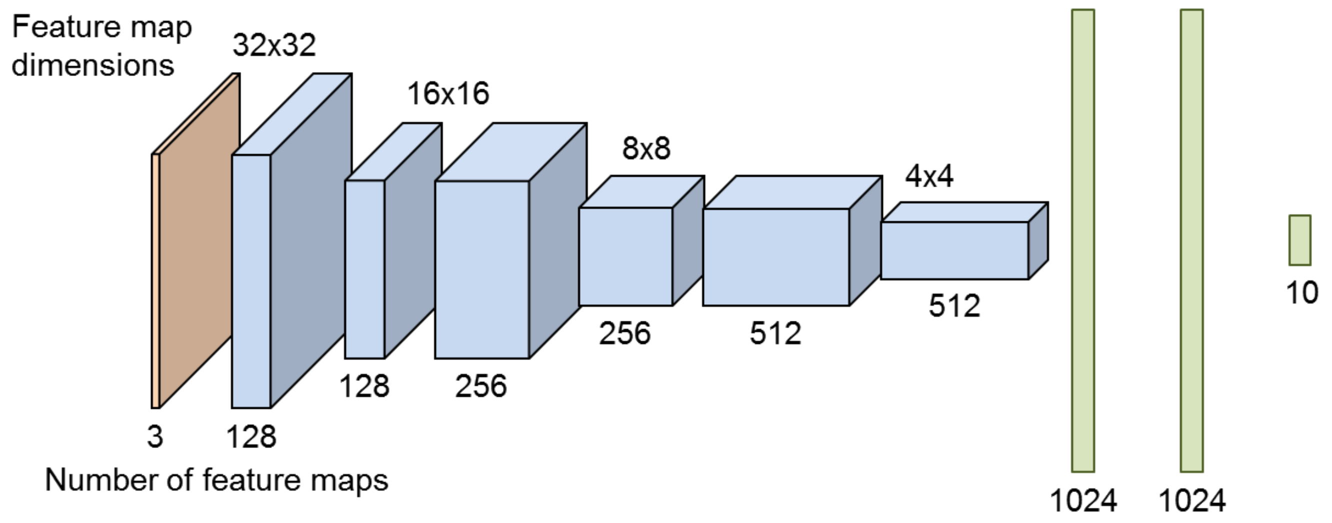


Figure adopted from: Y. Zhou, FPGA'17 presentation

Binarized Neural Network [Zhao et.al, FPGA'17]

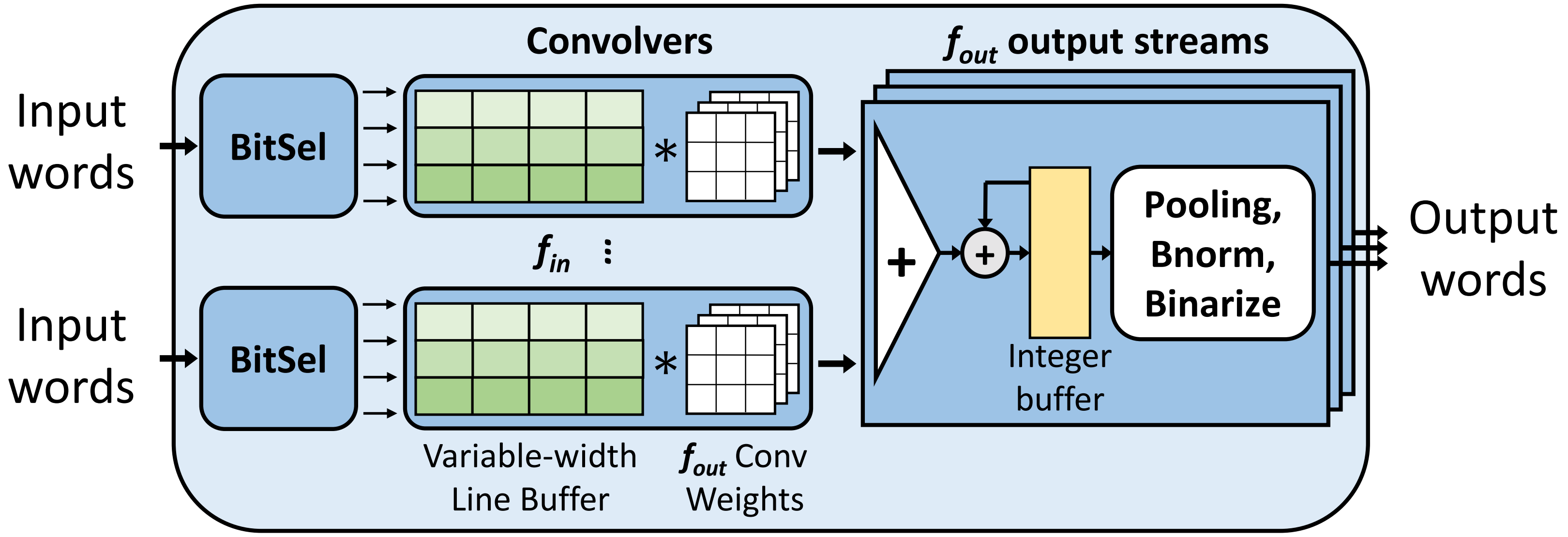
- Based on CIFAR-10 BNN model [Courbariaux et.al, arXiv'16]



- Binarized convolution/dense layers
  - Bitwise operation intensive
  - Convolution layers are compute-bound
  - Dense layers are memory-bound

Figure source: R. Zhao, FPGA'17 presentation

Challenge: Shared-Layer Compute Engine



Many diverse sources of parallelism ➡ Effective parallelization of the accelerator

Different network layer types and sizes ➡ Flexibility in accelerator architecture

Large data size and slow communication ➡ Data reuse and storage sharing

Difficult for manual HLS design!

Experimental Results

- Results on ZC706

Benchmark	#LUTs	#FFs	#BRAMs	#DSPs	Runtime (ms)	Throughput
3D Rendering	8893	12471	48	11	4.7	213 frames/s
Digit Recognition[1]	41238	26468	338	1	10.6	189k digits/s
Spam Filtering[2]	12678	22134	69	224	60.8	370k samples/s
Optical Flow	42878	61078	54	454	24.3	41.2 frames/s
Face Detection	62688	83804	121	79	33.0	30.3 frames/s
BNN[3]	46899	46760	102	4	4995.2	200 images/s

1: K=3, PAR_FACTOR=40. 2: Five epochs, PAR_FACTOR=32, VDWIDTH=64. 3: Eight convolvers, 1000 test images.

- Results on AWS F1

Benchmark	#LUTs	#FFs	#BRAMs	#DSPs	Runtime (ms)	Throughput	Performance-cost Ratio
3D Rendering	6763	7916	36	11	4.4	227 frames/s	**496k frames/$**
Digit Recognition[1]	39971	33853	207	0	11.1	180k digits/s	**393M digits/$**
Spam Filtering[2]	7207	17434	90	224	25.1	728k samples/s	**1.6G samples/$**
Optical Flow	38094	63438	55	484	8.4	119 frames/s	**260k frames/$**
Face Detection	48217	54206	92	72	21.5	46.5 frames/s	**101k frames/$**

1: K=3, PAR_FACTOR=40. 2: Five epochs, PAR_FACTOR=32, VDWIDTH=512.

Conclusions and Future Work

- Rosetta can serve as
  - A realistic benchmark suite for the HLS research community
    - Complex applications with rich compute/memory characteristics
    - Synthesizable baseline designs exposing opportunities to new HLS automation
  - A useful design tutorial for HLS/FPGA users
    - Optimized implementations meeting realistic design constraints

- Future work
  - Incorporate more applications from different fields
  - Provide OpenCL version for each benchmark
  - Continue to optimize the applications

Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs

Yuan Zhou, Udit Gupta, Steve Dai, Ritchie Zhao, Nitish Srivastava, Hanchen Jin, Joseph Featherston, Yi-Hsiang Lai, Gai Liu, Gustavo Angarita Velasquez, Wenping Wang, Zhiru Zhang

github.com/cornell-zhang/rosetta

Cornell University