NNStreamer: Stream Processing Paradigm for Neural Networks, Toward Efficient Development and Execution of On-Device AI Applications

MyungJoo Ham[1] Ji Joong Moon[1] Geunsik Lim[1] Wook Song[1] Jaeyun Jung[1] Hyoungjoo Ahn[1] Sangjung Woo[1]
Youngchul Cho[1] Jinhyuck Park[2] Sewon Oh[1] Hong-Seok Kim[1,3]
[1]Samsung Research, Samsung Electronics
[2]Biotech Academy, Samsung BioLogics
[3]Left the affiliation

Contents

- Background of the Project
- Related Works
- Componets Design
- Evaluation with 2 Apps
- Conclusion

Background of the Project

- On-device AI
  - Avoid data privacy and protection issues
  - Reduce data transmission latency
  - Reduce operating cost
- Situation
  - Limited computing power
  - Short response time or high processing rates
  - Use sensors incurring data stream with high bandwidth

Background of the Project

- Other Challenges
  - A neural network may process multiple input streams
  - Multiple neural networks may process streams simultaneously
  - Network may share input
  - The topology of pipelines may be dynamic
  - ...

Background of the Project

- Stream processing paradigm
  - GStreamer
    - highly modular architecture
    - fully open source software
    - compatible with various operating systems and hardware architecture

Related Works

- Multimedia Stream Processing in Practice
  - GStreamer
  - FFmpeg
    - for Linus, is not modular, Gstreamer can embed FFmpeg as plugin
  - StageFright
    - for Android, not flexible enough for other generic Linux distributions
  - AVFoundation
    - for iOS and macOS, cannot alter and use because it is proprietary software and supports proprietary platforms only
  - DirectShow
    - for Windows, cannot alter and use because it is proprietary software and supports proprietary platforms only

Related Works

- Stream Processing in General
  - StreamCloud
    - addressed the bottleneck of a single input node in a stream pipeline, provides a scalable and elastic stream framework to process large data input
  - Kumar, Apache Flink
    - help manage large data streams efficiently while do not provide a general stream processing framework for neural networks and complex topology
  - Gstream, Nvidia Tesla architecture
    - applying streaming framework for GPUs fits well with general architectures of GPUs

Related Works

- Stream Processing for Neural Networks
  - Nvidia DeepStream
  - Intel Media SDK
  - Pexip Infinity
  - RidgeRun
    - do not support neural networks with arbitrary input and output streams
- Tensor
  - not limiting streams to conventional media types(audio/video/text)
  - allowing outputs of neural networks to be inputs of other networks

Componets Design

- Basic Components
  - NNFW(Nerual Network FrameWork) filter
  - stream sink
  - Converters
  - preprocessors

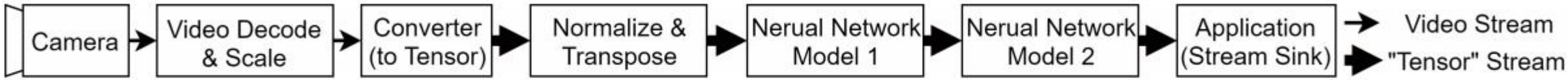

Figure 1: Basic linear pipeline with two neural networks and single input and output without path manipulations

Componets Design

- Isodimensional Stream Path Control
  - Tee
  - Mux
  - Demux
  - Recurrence Helper

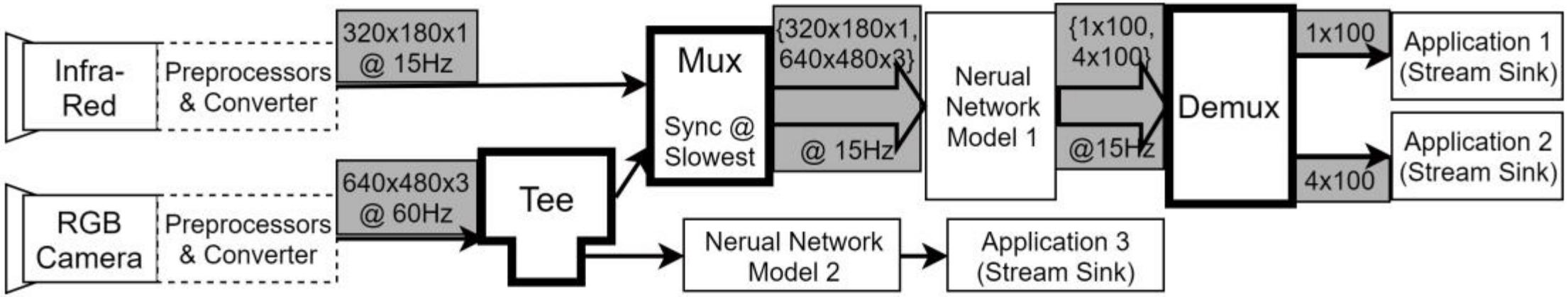

Figure 2: Pipeline with isodimensional stream path controls: Tee, Mux, and Demux

Componets Design

- Isodimensional Stream Path Control
  - Tee
  - Mux
  - Demux
  - Recurrence Helper

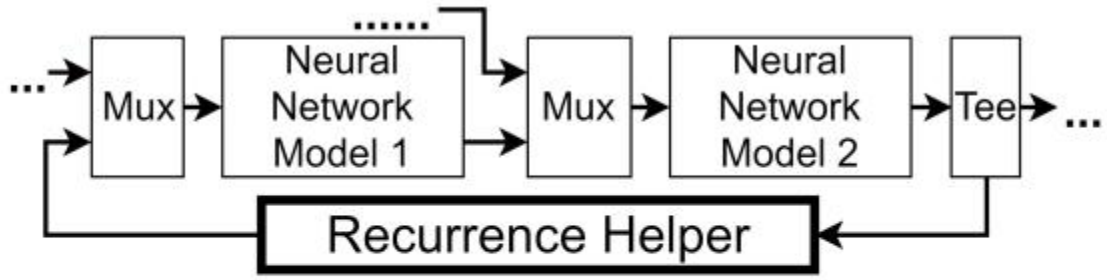

Figure 3: Pipeline with external recurrences

Componets Design

- Non-isodimensional Stream Path Control
  - Merge
  - Split
  - Aggregator(long short-term memory)

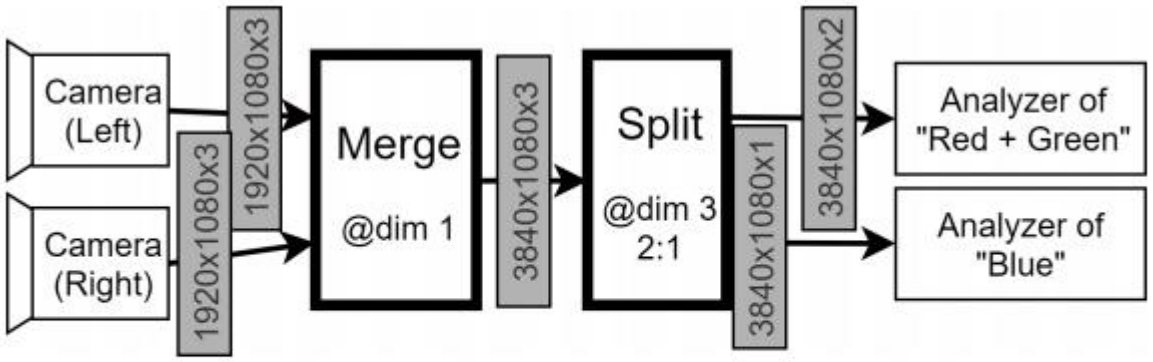

Figure 4: Pipeline with Merge and Split

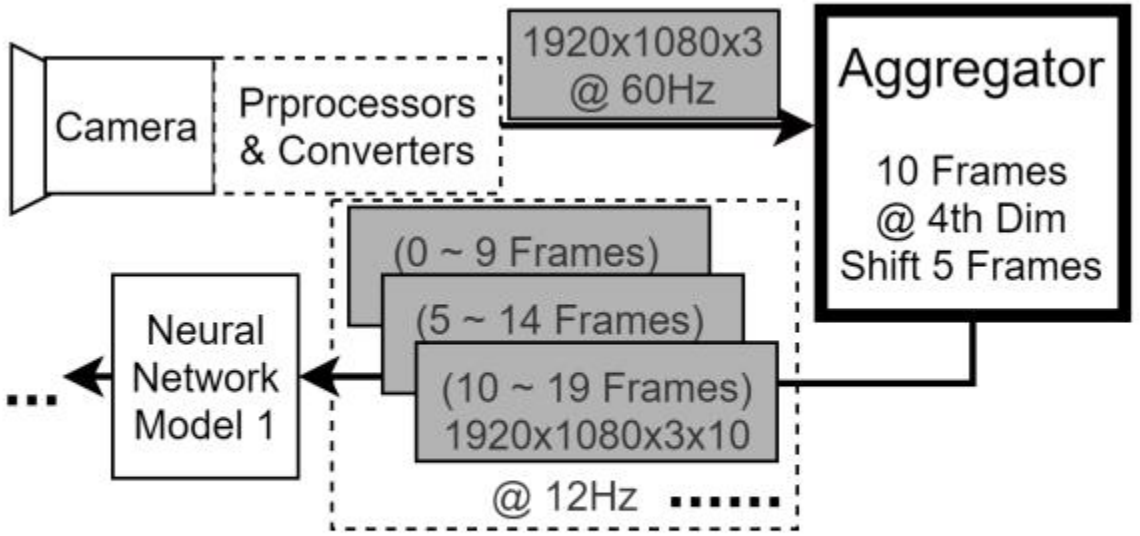

Figure 5: LSTM pipeline with Aggregator

Componets Design

- Other Requirements
  - Valve
  - Switch
  - Queue
  - …

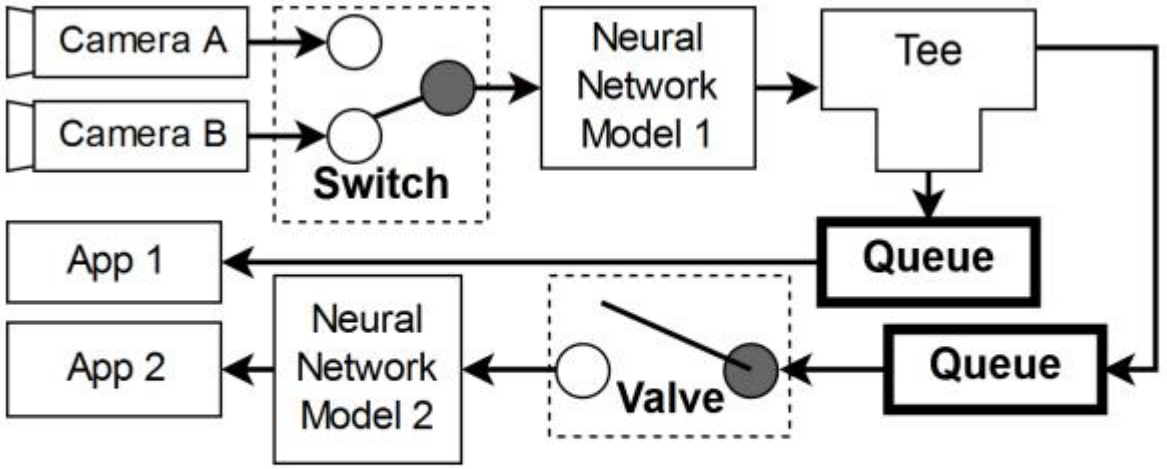

Figure 6: Pipeline with queues, a switch, and a valve

Evaluation with 2 Apps

- Activity Recognition Sensor(ARS)
- Multi-Task Cascaded Convolutional Networks(MTCNN)——ROS

	A / Artik 530s	B / Odroid-XU4	C / 8890	D / PC
SoC	Nexell S5P4418	Samsung Exynos 5422	Samsung Exynos 8890	Intel i7-7700
ISA	armv7 (32 bits)	armv7 (32 bits)	armv8 (64 bits)	x64 (64 bits)
CPU Microarchitecture	4 CA9	4 CA15 + 4 CA7	4 Mongoose + 4 CA53	4 Kaby-Lake
CPU Clock Speed	1.2 GHz	2 GHz / 1.5GHz	2.3 GHz / 1.6 GHz	3.6 GHz
DRAM Capacity	1 GiB DDR3	2 GiB LPDDR3	4 GiB LPDDR4	16 GiB DDR4
DRAM Throughput[1]	0.51 / 0.75	2.62 / 4.08	5.86 / 6.40	18.54 / 13.78
OS & Linux Kernel	Ubuntu 16.04 / 4.4	Tizen 5.0 / 4.1	Tizen 5.0 / 3.18	Ubuntu 16.04 / 4.15

Table 1: Specification of experimented devices. CA denotes Cortex-A. Linux kernels are supplied by the vendors.

Evaluation with 2 Apps

- Activity Recognition Sensor(ARS)

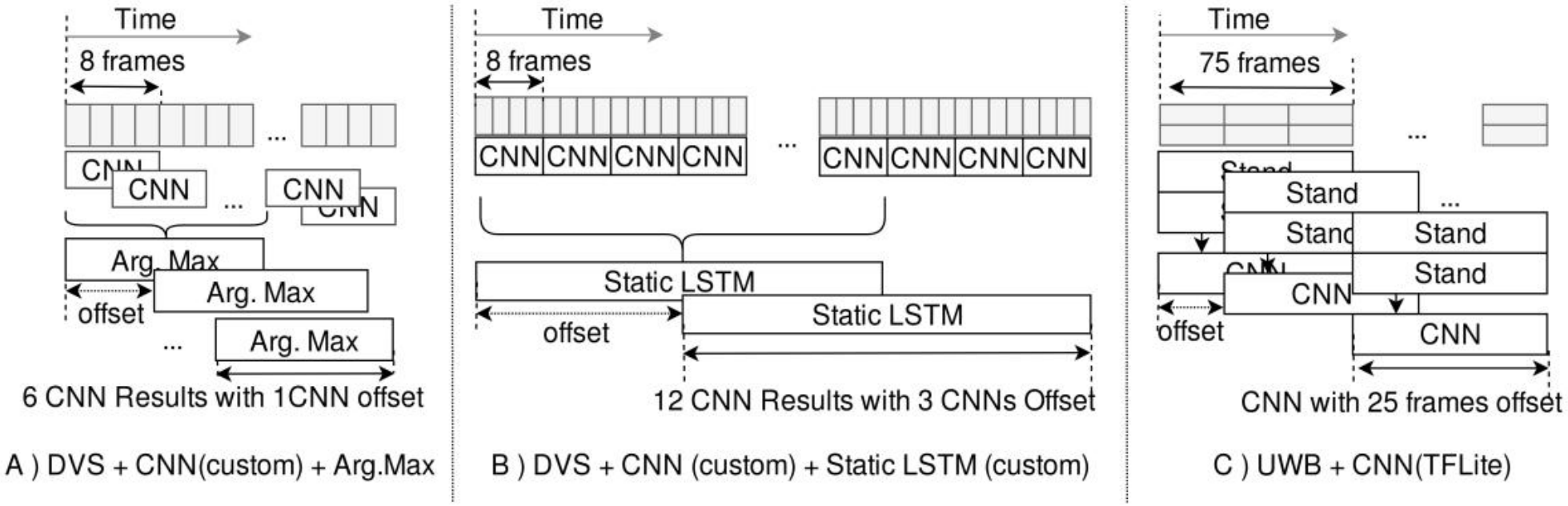

Figure 9: Neural network algorithms of activity recognition sensors (ARS)

Evaluation with 2 Apps

- Activity Recognition Sensor(ARS)

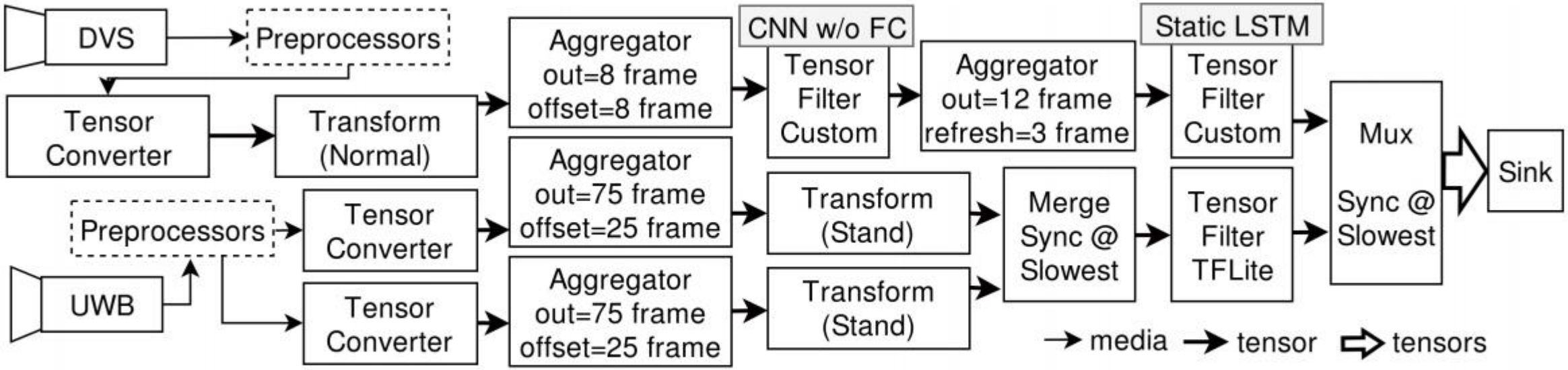

Figure 10: ARS Pipeline. The top half, B of Figure 9, processes DVS. The bottom half, C of FIgure 9, processes UWB.

Evaluation with 2 Apps

- Activity Recognition Sensor(ARS)

Row	Metric	A) DVS Arg Max		B) DVS LSTM		C) UWB		Improvement
		Control	*Nns*	Control	*Nns*	Control	*Nns*	by *Nns* (%)
1	LOC	364	169	359	186	383	1	-
2	Total mmap amount (MiB)	145	80	161	96	142	58	47.5
3	Max # threads	12	9	17	9	16	8	-
4	CPU (%), ∞ input rate	31.18	28.50	56.52	49.50	44.00	46.50	5.4
5	Output FPS, ∞ input rate	46.0	59.4	2.5	3.2	9.3	25.5	65.5
6	Output FPS / CPU (%), ∞ input rate	1.48	2.08	0.044	0.065	0.211	0.548	75.0
7	CPU (%), 30 FPS input rate	29.38	17.35	38.95	28.50	22.10	5.50	43.5

Table 2: Experimental results of ARS with 1100 input frames. FPS denotes frames per second. Nns denotes *nnstreamer*.

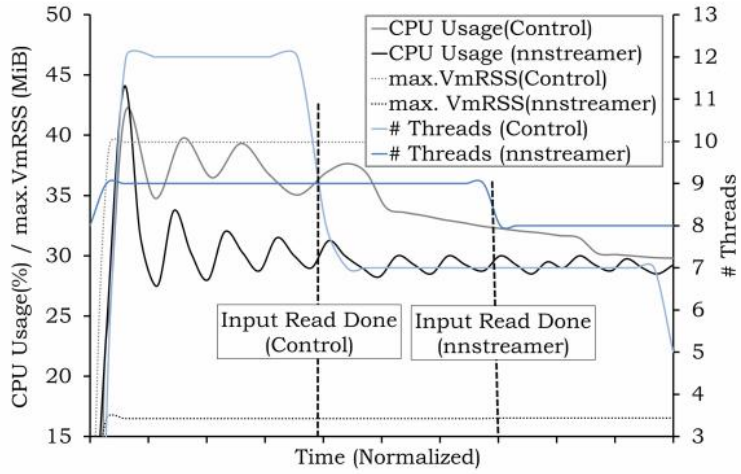

Figure 11: Resource utilization over time

Evaluation with 2 Apps

- Multi-Task Cascaded Convolutional Networks(MTCNN)

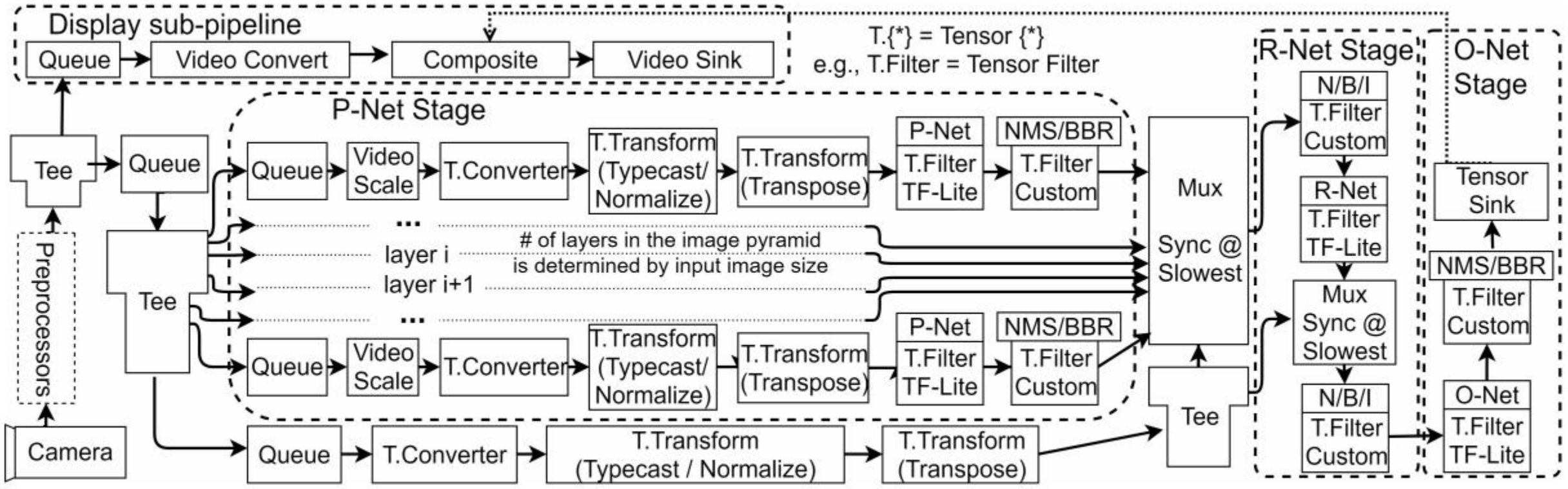

Figure 12: Pipeline Topology of *nnstreamer* MTCNN. N/B/I denotes NMS/BBR/Image Patch Generation

Evaluation with 2 Apps

- Multi-Task Cascaded Convolutional Networks(MTCNN)

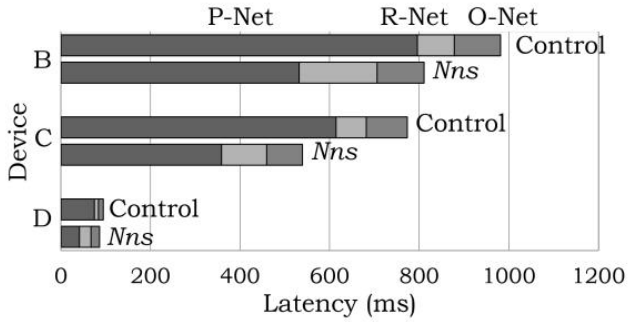

Figure 13: Breakdown of the face detection latency.

Row	Metric	Input Rate	B / Odroid-XU4 Control	B / Odroid-XU4 *Nns*	C / 8890 Control	C / 8890 *Nns*	D / PC Control	D / PC *Nns*	Improvement by *Nns* (%)
1	End-to-end latency (ms)	1 FPS	981.78	811.00	704.49	539.64	94.28	85.91	15.35
2	Output rate (FPS)	30 FPS	1.01	1.73	1.48	4.02	10.41	13.76	83.21
3	Avg. CPU usage (%)	30 FPS	31.85	84.60	31.15	87.10	12.90	48.74	-
4	Memory usage (MiB)	30 FPS	136	307	129	474	248	440	-

Table 4: MTCNN performance with 1000 input frames. FPS denotes frames per second. Nns denotes *nnstreamer*.

Conclusion

- 联系实际，有明确的应用背景和需求（三星的项目）
- 基于前人的基础（GStreamer），明确增加的功能和方向
- 对相关工作的调研总结比较全面，其他流处理工具没有撰写类似的文章
- 实验中针对两个应用设计特定的流程架构，对于使用其他工具的我们也具有一定参考价值

- 主要提升体现在输出帧率的提高，但是对于资源的消耗也增加，实际贡献有限
- 强调on-device AI，不使用边缘设备的GPU，较难满足实时性的需求