

# PyTorch and CEDR: Enabling Deployment of Machine Learning Models on Heterogeneous Computing Systems

**Umut Suluhan<sup>1</sup>**, Serhan Gener<sup>1</sup>, Alexander Fusco<sup>1</sup>,  
Fatih Ugurdag<sup>2</sup>, and Ali Akoglu<sup>1</sup>

<sup>1</sup>University of Arizona, <sup>2</sup>Ozyegin University



COLLEGE OF ENGINEERING  
Electrical & Computer  
Engineering



Cloud and Autonomic Computing Center

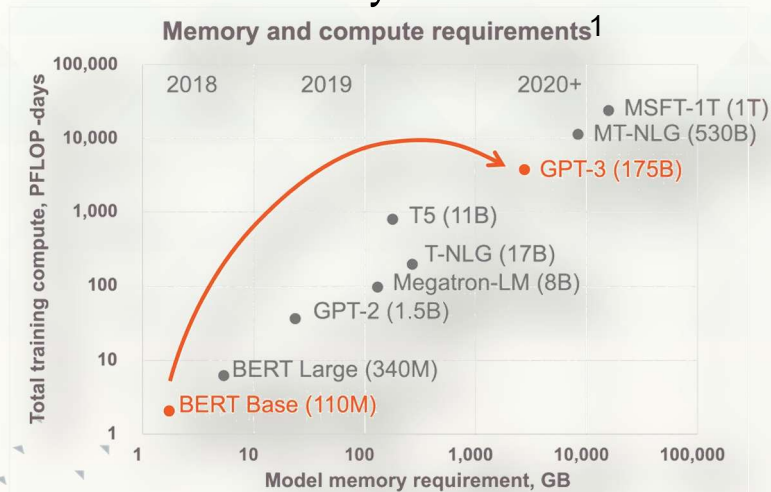


# Goal and Motivation

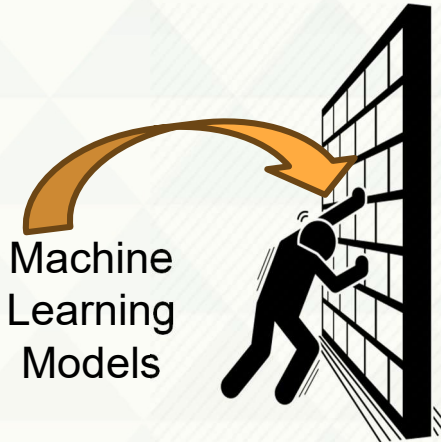
PyTorch offers productive GPU-based deployment experience



A steep increase in computation & memory demand



Machine Learning Models

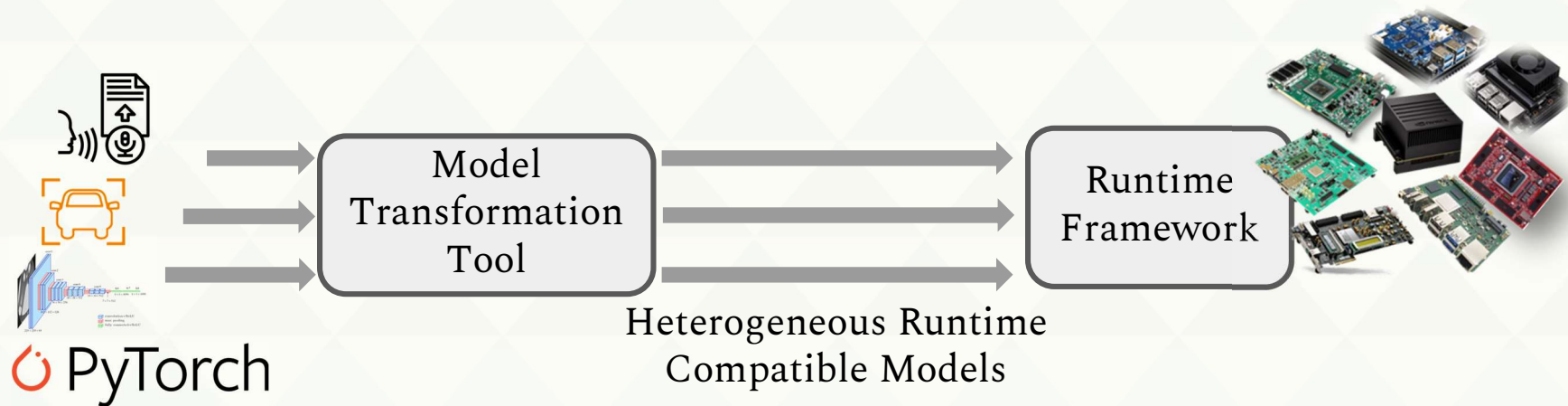


No path forward for deploying ML models on systems offering balance between throughput and energy efficiency

Aim: Productive PyTorch model deployment on heterogeneous SoCs

- hardware agnostic application development
- balance trade-off between throughput and energy efficiency
- explore SoC configurations for PyTorch based workflows

# Technical Contributions

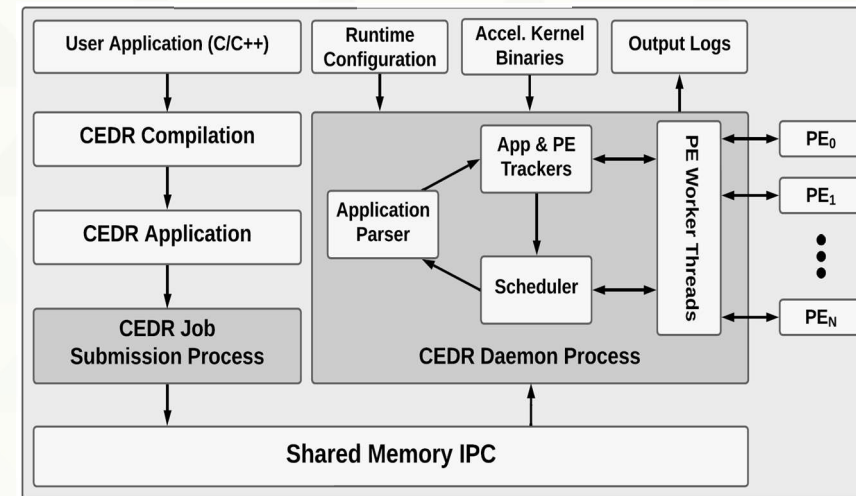


***Hardware agnostic model development and deployment experience for PyTorch developers across rich set of off-the-shelf SoC platforms***



# CEDR<sup>1</sup> – A Compiler-Integrated, Extensible DSSoC Runtime

- Open source<sup>2</sup>, unified environment for programming and execution on heterogenous SoCs
- **Key features:**
  - Provides users with an abstraction layer through APIs
  - Refactors applications into a sequence of hardware agnostic function calls
  - Generates application representation that allows run time system invoke each function call on its supported processing elements
  - Flexible to execute arbitrary, interleaved workloads on various accelerators
  - Portable across off the shelf SoC platforms
  - Avoids requiring users to become hardware experts



*Need for a transformation tool that can translate  
PyTorch models into a CEDR compatible application representation*



COLLEGE OF ENGINEERING  
Electrical & Computer  
Engineering

[1] J. Mack et al. "CEDR-API: Productive, Performant Programming of Domain-Specific Embedded Systems," IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2023,

<https://doi.org/10.1109/IPDPSW59300.2023.00016>

[2] Project Homepage: <https://ua-rcl.github.io/projects/cedr/>



# Transformation Flow: Step 1



```
1 def conv2d(in_channel, out_channel, kernel_size,
2           stride, padding):
3     return nn.Sequential(
4         nn.Conv2d(in_channel, out_channel,
5                   kernel_size, stride, padding)
6         nn.ReLU())
7 class Model(nn.Module):
8     def __init__(self):
9         self.conv_relu = conv2d(3, 8, 3)
10        self.linear = nn.Linear(32, 16)
11        self.maxpool = nn.MaxPool2d(2, 2)
12    def forward(self, x):
13        conv1 = self.conv_relu(x)
14        pool1 = self.maxpool(conv1)
15        linear1 = self.linear(pool1)
16        return linear1
```

STEP 1

```
{"sequential": "yes",
"name": "conv_relu",
"length": 2,
"0": {"name":
"conv_relu_0",
"type": "Conv2d",
"in_channels": 3,
"out_channels": 8,
"kernel_size": [3,3]
},
"1": {"name":
"conv_relu_1",
"type": "ReLU" }}

{"sequential": "no",
"name": "linear",
"type": "Linear",
"in_features": 32,
"out_features": 16}

{"sequential": "no",
"name": "maxpool",
"type": "MaxPool2d"}
```

- Extracts information regarding each layer
- Distinguishes layers from each other with distinct attributes



# Transformation Flow: Step 2



```
1 def conv2d(in_channel, out_channel, kernel_size,
2           stride, padding):
3     return nn.Sequential(
4         nn.Conv2d(in_channel, out_channel,
5                   kernel_size, stride, padding)
6         nn.ReLU())
7 class Model(nn.Module):
8     def __init__(self):
9         self.conv_relu = conv2d(3, 8, 3)
10        self.linear = nn.Linear(32, 16)
11        self.maxpool = nn.MaxPool2d(2, 2)
12    def forward(self, x):
13        conv1 = self.conv_relu(x)
14        pool1 = self.maxpool(conv1)
15        linear1 = self.linear(pool1)
16        return linear1
```

STEP 1

```
{
  "sequential": "yes",
  "name": "conv_relu",
  "length": 2,
  "0": {
    "name": "conv_relu_0",
    "type": "Conv2d",
    "in_channels": 3,
    "out_channels": 8,
    "kernel_size": [3, 3]
  },
  "1": {
    "name": "conv_relu_1",
    "type": "ReLU"
  }
},
{
  "sequential": "no",
  "name": "linear",
  "type": "Linear",
  "in_features": 32,
  "out_features": 16
},
{
  "sequential": "no",
  "name": "maxpool",
  "type": "MaxPool2d"
}
```

STEP 2

```
{
  "name": "conv_relu",
  "input": "x",
  "output": "conv1",
  "next": "maxpool",
  "sequential": "yes",
  "length": 2,
  "0": {
    "name": "conv_relu_0",
    "type": "Conv2d",
    "in_channels": 3,
    "out_channels": 8,
    "kernel_size": [3, 3]
  },
  "1": {
    "name": "conv_relu_1",
    "type": "ReLU"
  }
},
{
  "name": "maxpool",
  "input": "conv1",
  "output": "pool1",
  "next": "linear",
  "sequential": "no",
  "type": "Maxpool2d"
},
{
  "name": "linear",
  "input": "pool1",
  "output": "linear1",
  "next": "NONE",
  "sequential": "no",
  "type": "Linear"
}
```

- Obtains DAG based on input-output relationship between layers
- Allows building C++ based model while respecting the dataflow and layer attributes



# Transformation Flow: Step 3

```
{ "name": "conv_relu",  
  "input": "x",  
  "output": "conv1",  
  "next": "maxpool",  
  "sequential": "yes",  
  "length": 2,  
  "0": { "name":  
    "conv_relu_0",  
    "type": "Conv2d",  
    "in_channels": 3,  
    "out_channels": 8,  
    "kernel_size": [3,3] },  
  "1": { "name":  
    "conv_relu_1",  
    "type": "ReLU" }}  
  
{ "name": "maxpool",  
  "input": "conv1",  
  "output": "pool1",  
  "next": "linear",  
  "sequential": "no",  
  "type": "Maxpool2d"}  
  
{ "name": "linear",  
  "input": "pool1",  
  "output": "linear1",  
  "next": "NONE",  
  "sequential": "no",  
  "type": "Linear" }
```

STEP 3

```
1 Conv2d* conv_relu_0 = new Conv2d(3, 8, 3);  
2 ReLU* relu = new ReLU();  
3 Linear* linear = new Linear(32, 16);  
4 Maxpool2d* maxpool = new Maxpool2d();  
5 Module* module = new Module();  
6 module->add(*conv_relu_0);  
7 module->add(*linear);  
8 module->filter_assign();  
9 x = conv_relu_0->forward(x);  
10 Tensor3D* conv1 = relu->forward(x);  
11 Tensor3D* pool1 = maxpool->forward(conv1);  
12 Tensor2D* linear1 = linear->forward(pool1);
```

- Maps each layer in the DAG to equivalent C++ implementation
- Serves as a baseline model for replacing key kernels with hardware agnostic CEDR compatible API calls



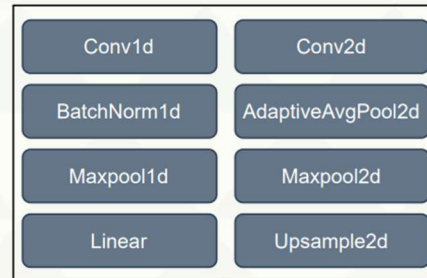


# Transformation Flow: Step 4

```

1 Conv2d* conv_relu_0 = new Conv2d(3, 8, 3);
2 ReLU* relu = new ReLU();
3 Linear* linear = new Linear(32, 16);
4 Maxpool2d* maxpool = new Maxpool2d();
5 Module* module = new Module();
6 module->add(*conv_relu_0);
7 module->add(*linear);
8 module->filter_assign();
9 x = conv_relu_0->forward(x);
10 Tensor3D* conv1 = relu->forward(x);
11 Tensor3D* pool1 = maxpool->forward(conv1);
12 Tensor2D* linear1 = linear->forward(pool1);

```



Function Inlining

STEP 4

```

1 void conv(float *in, float *filter, float *bias, float *out, int height, int width,
2           int kernel_size, int filter_number, int in_channel){
3     for(int i = 0; i < filter_number; i++){
4         // Memory allocation operations
5         for(int j = 0; j < in_channel; j++){
6             float *conv_output = (float *) malloc (sizeof (float) * height * width);
7             CEDR_CONV_2D(&(in[j * height * width]), height, width,
8                         (filter[(i * in_channel + j) * kernel_size * kernel_size]),
9                         kernel_size, conv_output);
10            CEDR_ZIP(&conv_output, &(out[(i * height * width)]),
11                   &(out[(i * height * width)]), height * width / 2, ZIP_ADD);
12            &(out[(i * height * width)]), height * width / 2, ZIP_ADD);}}}
13 void linear(float *in, float *filter, float *bias, float *out, int channel,
14            int in_channel, int out_channel){
15     float *filter_t = (float *) malloc (sizeof (float) * in_channel * out_channel);
16     transpose_linear_weight(filter, filter_t, out_channel, in_channel);
17     // Tensor manipulations
18     CEDR_GEMM(in, filter_t, out, channel, in_channel, out_channel);
19     // Tensor manipulations}

```

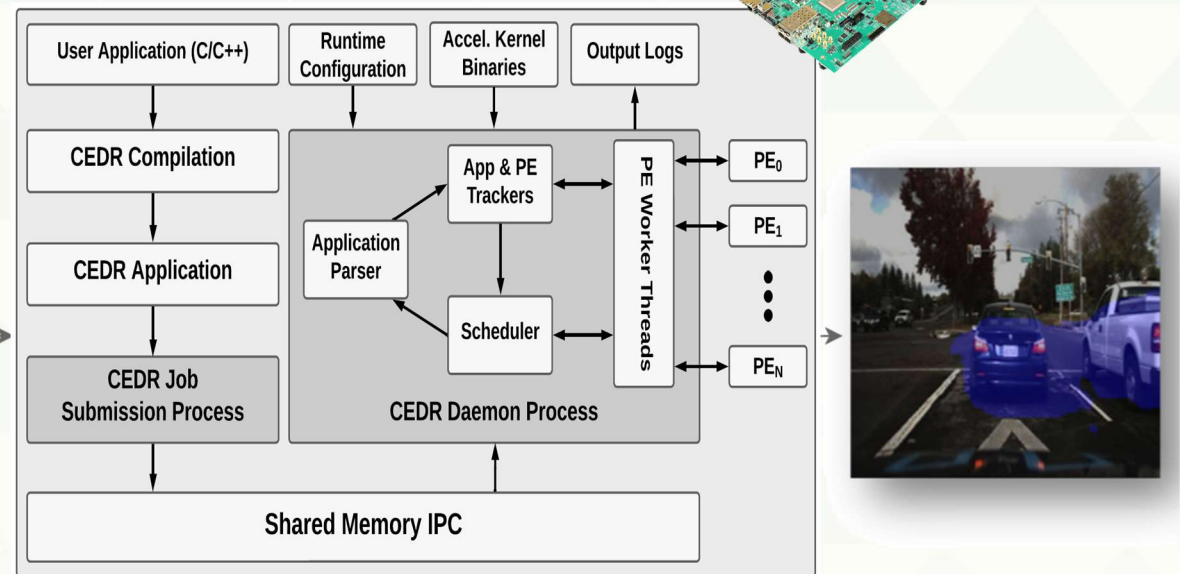
- Key kernels are replaced with CEDR compatible hardware agnostic API calls through function inlining
- Generates final C++ model with API implementations that can be compiled and executed with CEDR.





# Transformation Flow: Step 5

```
1 void conv(float *in, float *filter, float *bias, float *out, int height, int width,  
2         int kernel_size, int filter_number, int in_channel){  
3     for(int i = 0; i < filter_number; i++){  
4         // Memory allocation operations  
5         for(int j = 0; j < in_channel; j++){  
6             float *conv_output = (float *) malloc (sizeof (float) * height * width);  
7             CEDR_CONV_2D(&(in[j] * height * width)), height, width,  
8                 (filter[((i * in_channel + j) * kernel_size * kernel_size))),  
9                 kernel_size, conv_output);  
10            CEDR_ZIP(&conv_output, &(out[(i * height * width))),  
11                &(out[(i * height * width))), height * width / 2, ZIP_ADD);  
12            &(out[(i * height * width))), height * width / 2, ZIP_ADD);}}}  
13 void linear(float *in, float *filter, float *bias, float *out, int channel,  
14            int in_channel, int out_channel){  
15     float*filter_t = (float*) malloc (sizeof (float) * in_channel * out_channel);  
16     transpose_linear_weight(filter, filter_t, out_channel, in_channel);  
17     // Tensor manipulations  
18     CEDR_GEMM(in, filter_t, out, channel, in_channel, out_channel);  
19     // Tensor manipulations}
```



- C++ model with API calls is compiled and prepared for execution on heterogenous SoC
- Integrated scheduler makes task to processing element mapping decisions based on current state of system resources and performance goals.



# Experimental Setup





## Hardware Composition

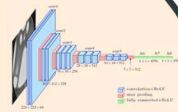
- 3 CPUs
- Accelerators (Conv2D, FFT, ZIP)



Xilinx Zynq UltraScale+  
MPSoC ZCU102

## Workload Composition

 Object Detection  
Visual Geometry Group  
 Speech Classification  
Wifi-TX   
Pulse Doppler 



## Scheduling Heuristics

- Earliest Finish Time (EFT)
- Earliest Time to Finish (ETF)
- Heterogeneous Earliest Finish Time (HEFT-RT<sup>1</sup>)

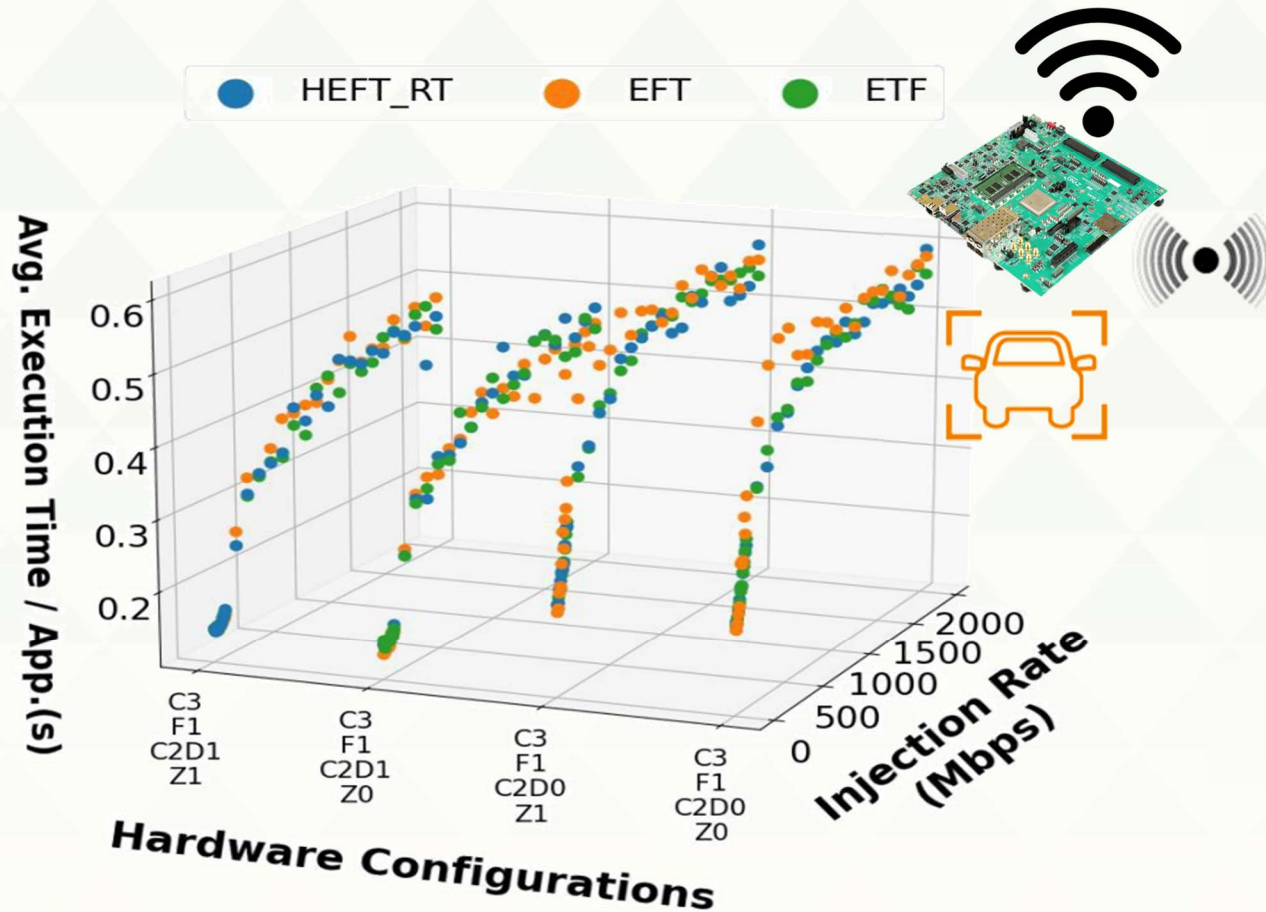


COLLEGE OF ENGINEERING

Electrical & Computer  
Engineering

[1] J. Mack et al. "Performant, multi-objective scheduling of highly interleaved task graphs on heterogeneous system on chip devices," IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 9, pp. 2148–2162, 2022  
<https://doi.org/10.1109/TPDS.2021.3135876>

# Cross-Domain Applications

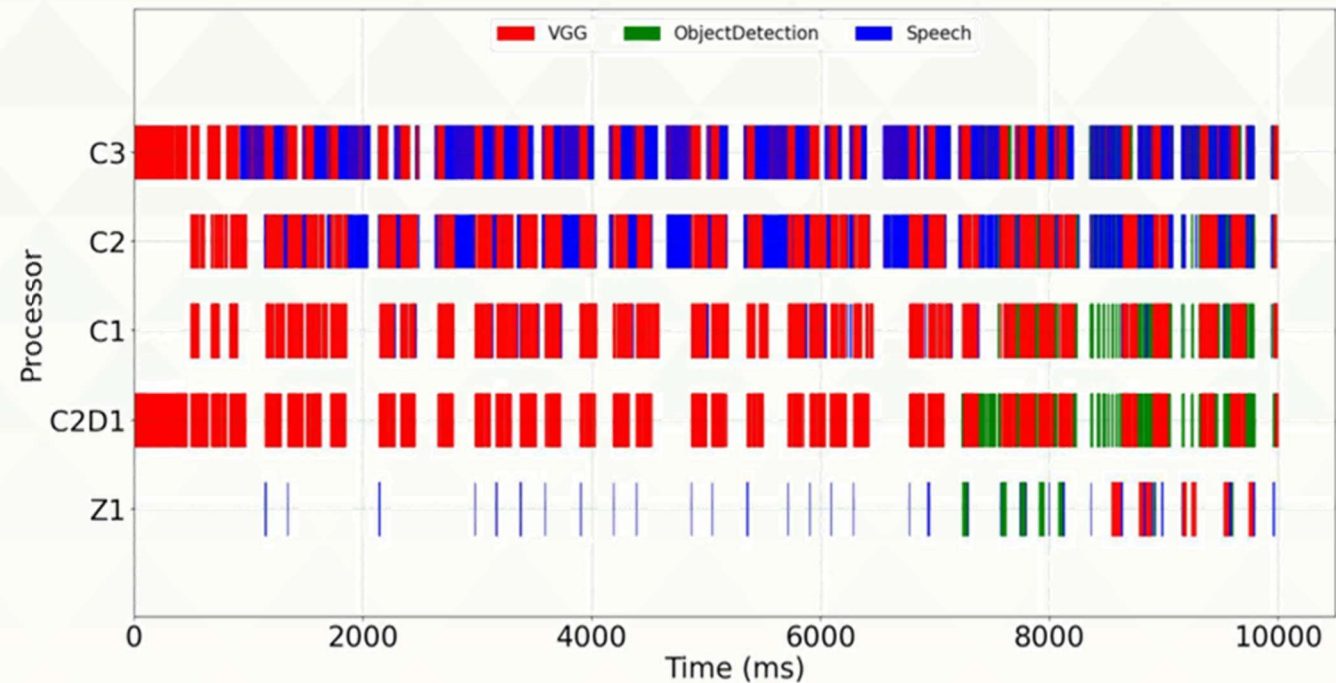
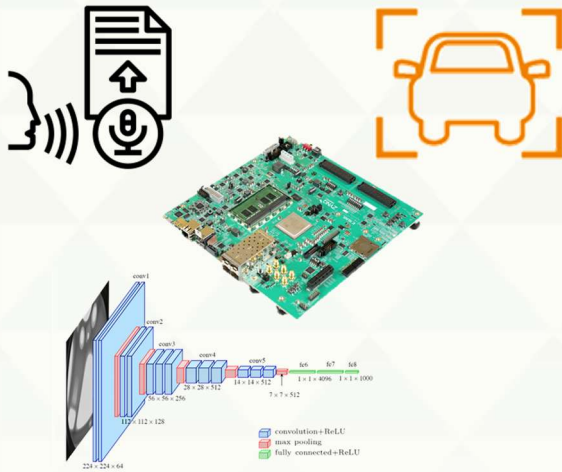


- Saturation trend with respect to workload complexity (injection rate): oversubscribed system
- Resource rich configuration saturates latest
- Execution time reduces with the increased degree of heterogeneity



# PyTorch Applications

CEDR can execute multiple models concurrently on the target SoC in dynamically arriving workload scenarios

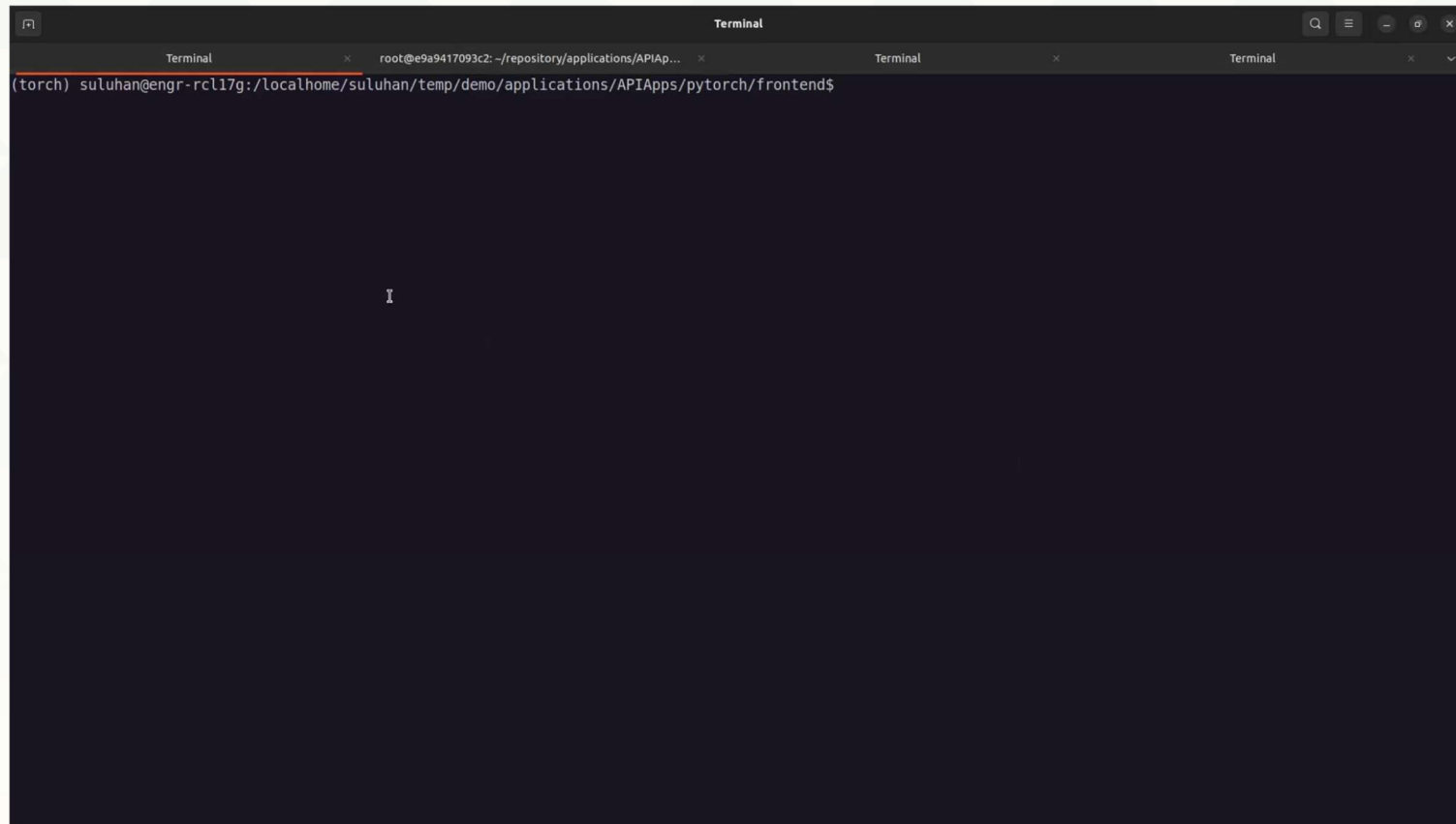


First 10 seconds of the Gantt chart for multiple PyTorch applications running on 3 CPUs, 1 C2D, and 1 ZIP accelerator with EFT scheduler





# Demo



A terminal window with a dark background and light text. The title bar at the top says "Terminal". Below the title bar, there are several tabs, one of which is active and shows the path `root@e9a9417093c2: ~/repository/applications/APIAp...`. The main area of the terminal shows a command prompt `(torch) suluhan@engr-rc117g:/localhome/suluhan/temp/demo/applications/APIApps/pytorch/frontend$` with a cursor blinking at the end of the line.



# Related Work

Other works in the literature offer system-level solutions and allow running neural network workloads on heterogeneous SoCs:

- Zhong et al.<sup>1</sup> leverages FPGA accelerators and NEON engine cores to offload convolution workloads, targeting experienced engineers
- Shea et al.<sup>2</sup> designs a hardware accelerator specifically for neural network workloads and introduces heterogeneity-aware scheduler
- Dagli et al.<sup>3</sup> implements layer-level design time scheduling, aiming to strike a balance between energy and performance trade-offs.

***Proposed transformation tool is unique when coupled with CEDR as other methods fall short in terms of programmer productivity and can not cope with dynamically arriving workload scenarios***



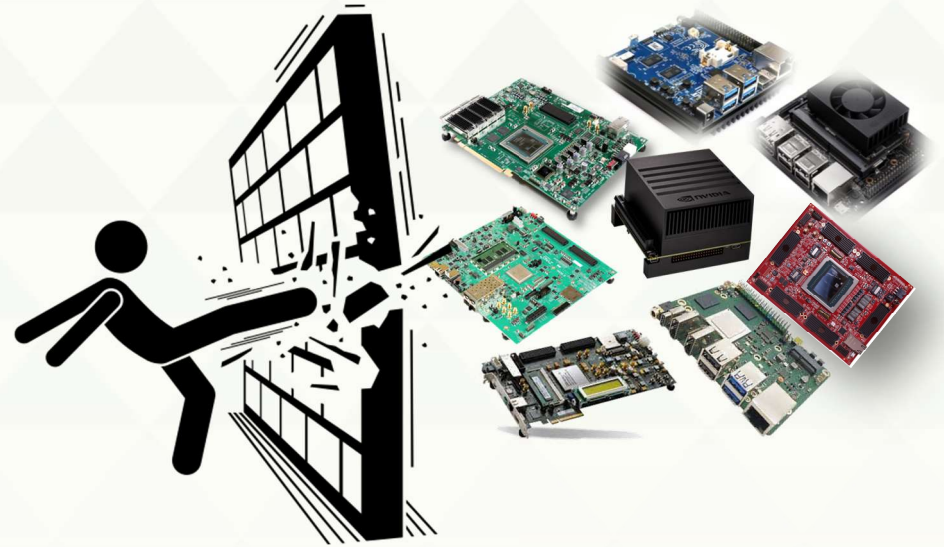
[1] G. Zhong et al. "Synergy: An hw/sw framework for high throughput cnns on embedded heterogeneous soc," ACM Transactions on Embedded Computing Systems (TECS), vol. 18, no. 2, pp. 1-23, 2019.

[2] C. Shea et al. "Heterogeneous scheduling of deep neural networks for low-power real-time designs," ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 15, no. 4, pp. 1-31, 2019.

[3] I. Dagli et al. "Axonn:energy-aware execution of neural network inference on multi-accelerator heterogeneous socs," Proceedings of the 59th ACM/IEEE Design Automation Conference, 2022, pp. 1069-1074.

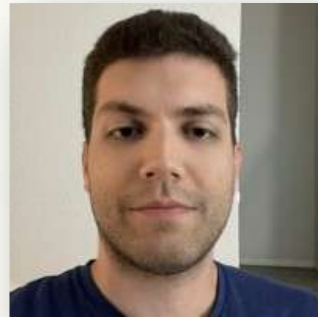
# Conclusions and Future Work

- For the first time, PyTorch application developers have access to FPGA-based execution without having to become hardware experts
  - balance trade-off between throughput and energy efficiency
  - explore SoC configurations for dynamic workloads
- **Next Step**
  - Generalize the framework for supporting wide range of ML models.



# Thank you

- Questions?
- Links
  - Website: <https://ua-rcl.github.io/projects/cedr/>
  - Source code: <https://github.com/UA-RCL/CEDR/>
- Contact
  - Umut Suluhan: [suluhan@arizona.edu](mailto:suluhan@arizona.edu)



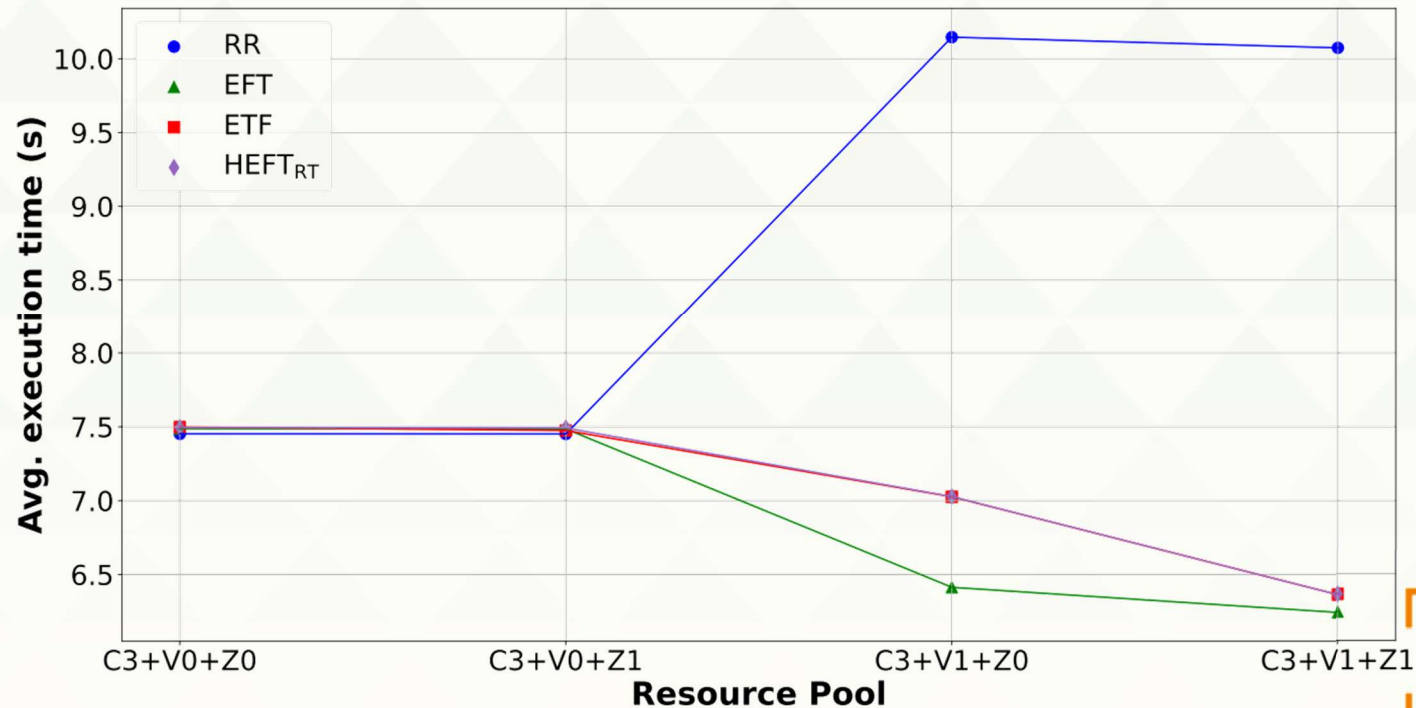


# BACKUP



# Results - BACKUP

- Sophisticated schedulers make better scheduling decisions with the increase in resource pool since it considers execution time of all processing elements.



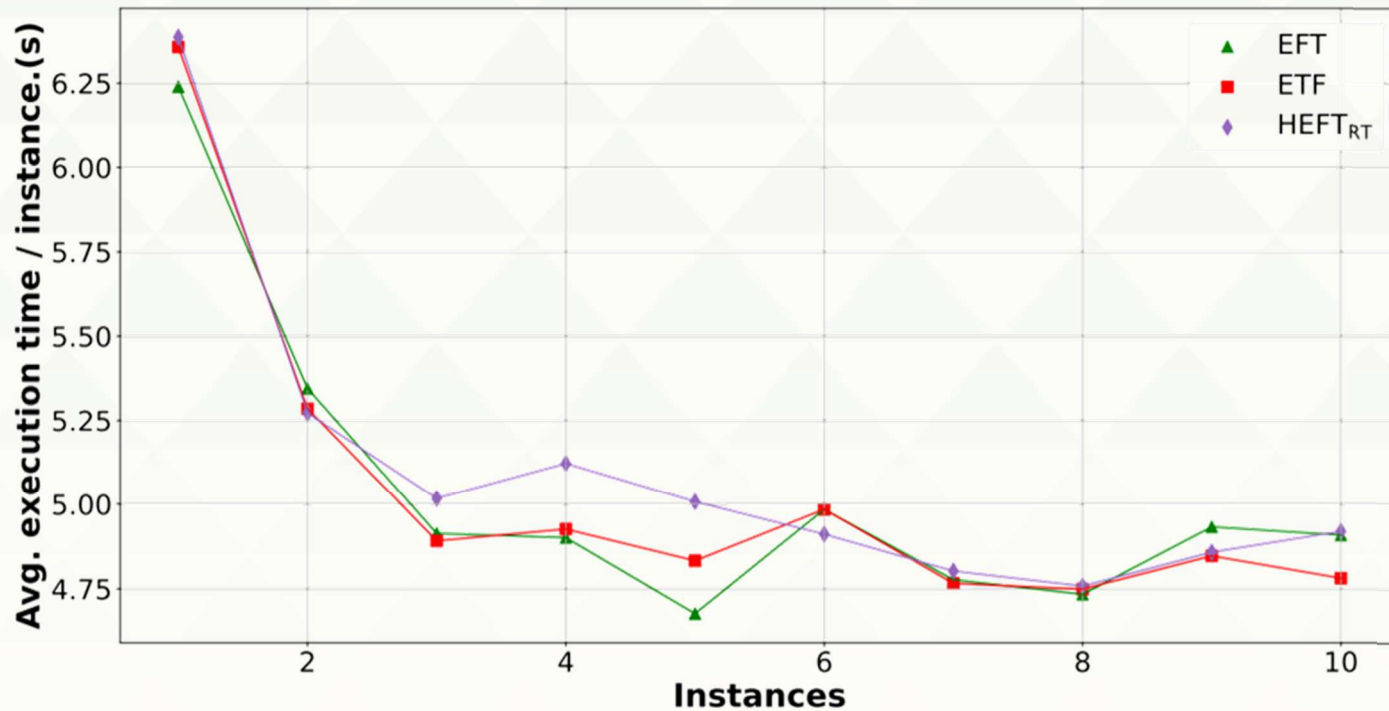
Scheduler and Resource Design-Space Exploration



COLLEGE OF ENGINEERING  
Electrical & Computer  
Engineering

# Results - BACKUP

- Rich resource pool enables system to overlap executions of applications leading to faster execution per instance with the increase in the number of instances running simultaneously.



Scheduler comparison for increasing number of instances

