Optimizing Dynamic Neural Networks with Brainstorm

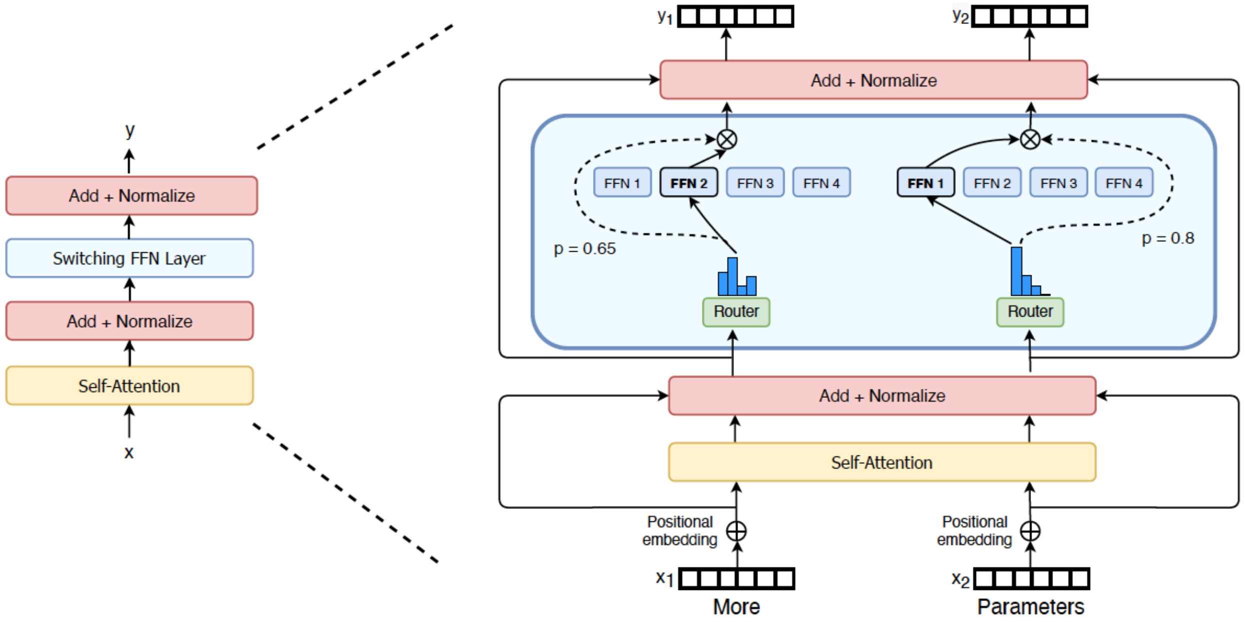
OSDI’23

Weihao Cui, Zhenhua Han, Lingji Ouyang, Yichuan Wang, Ningxin Zheng, Lingxiao Ma, Yuqing Yang, Fan Yang, Jilong Xue, Jili Qiu, Lidong Zhou, Quan Chen, Haisheng Tan and Minyi Guo

SJTU, MSRA, USTC

Background

Dynamic Neural Networks

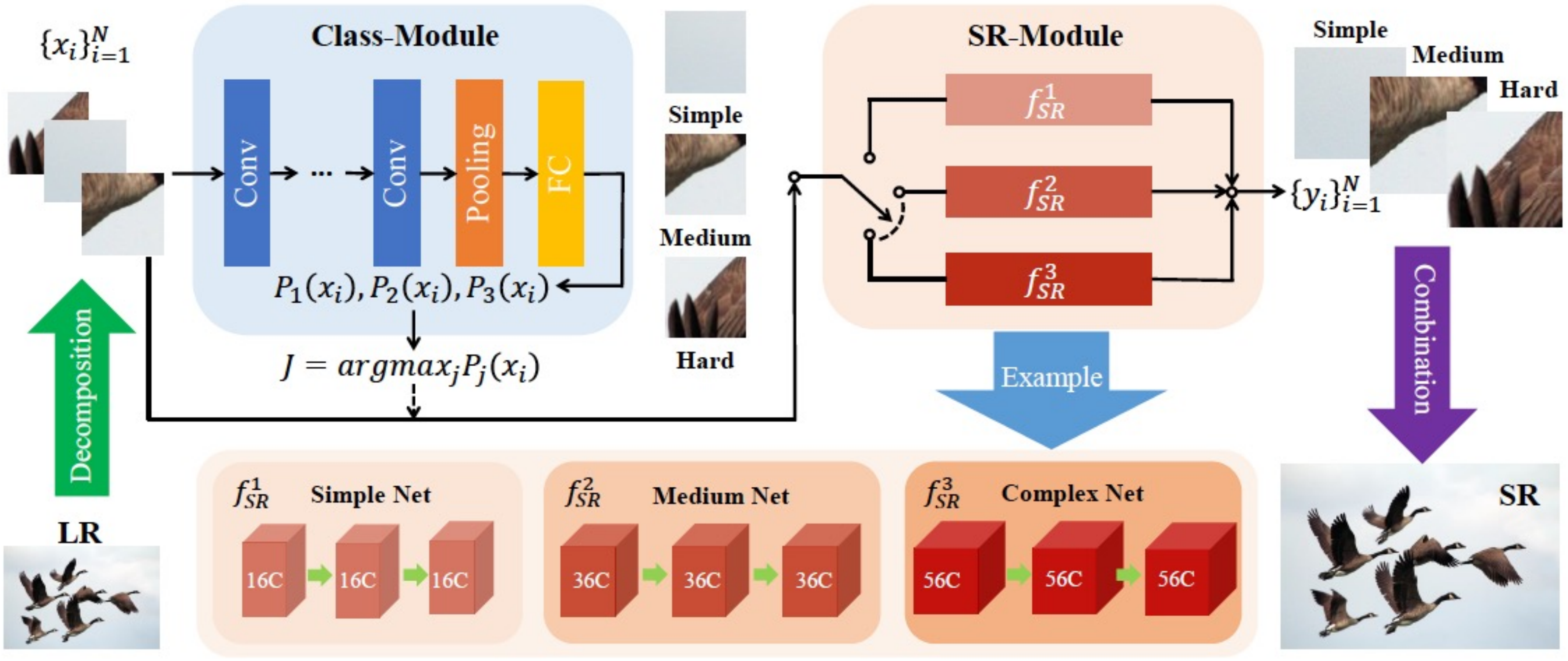


Mixture-of-Experts

[JMLR'22 SwitchTransformer]

Background

Dynamic Neural Networks

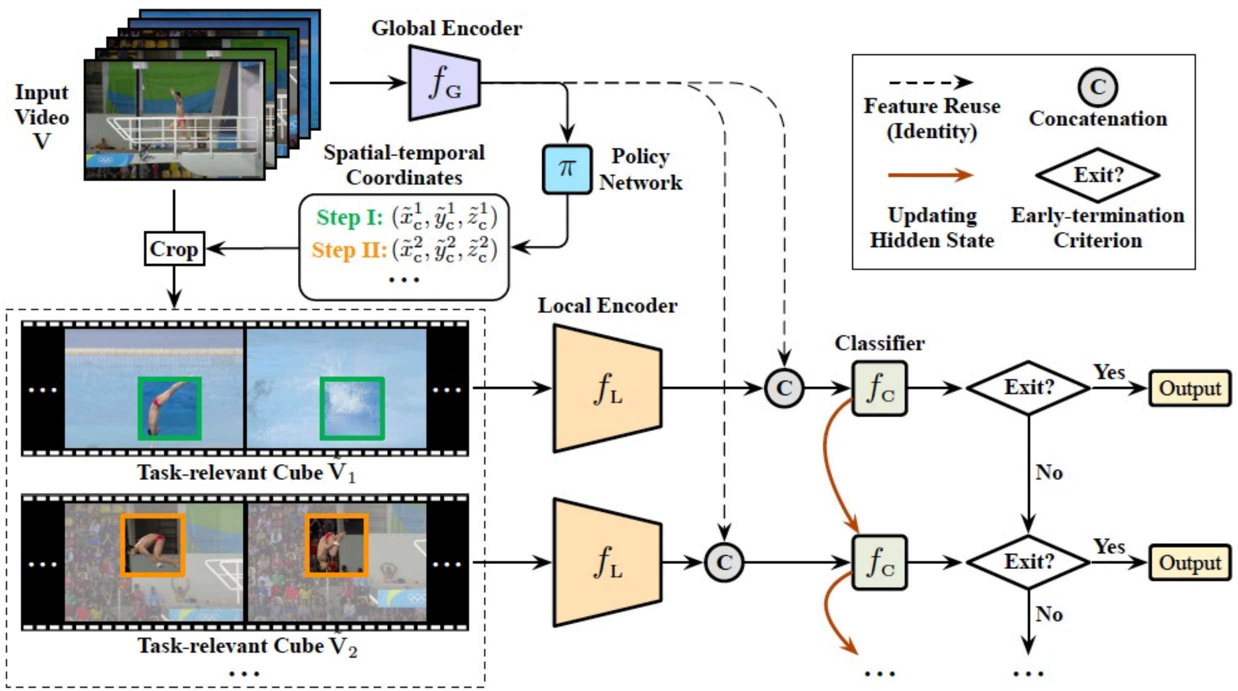


First Classify and Then Execution

[CVPR'21 ClassSR]

Background

Dynamic Neural Networks

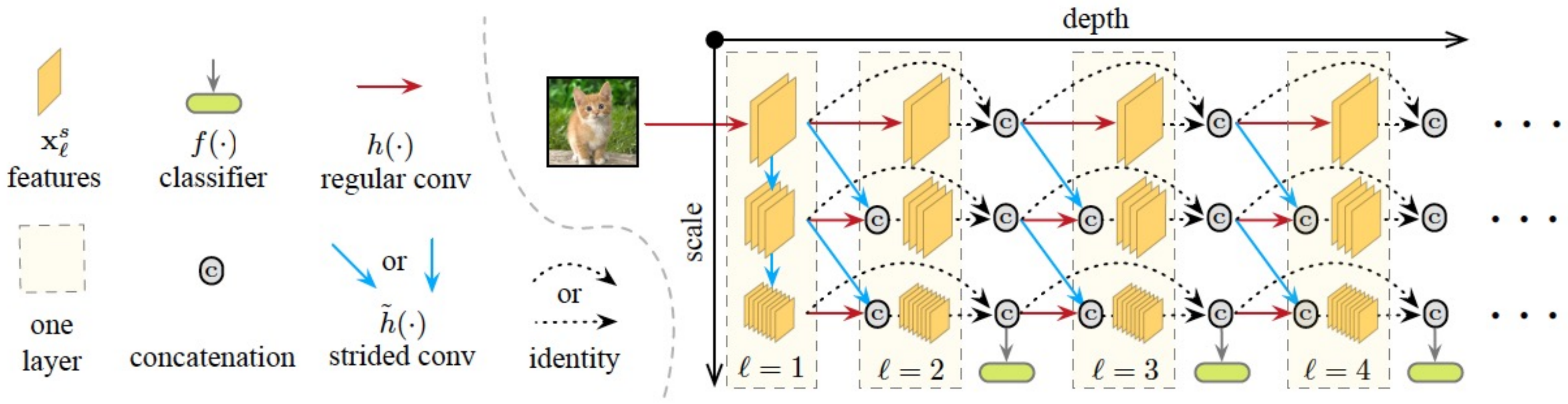


Temporal and Spatial Data Filtering

[ICCV'21 AdaFocus]

Background

Dynamic Neural Networks



Early Exits

[ICLR'18 MSDNet]

Background

Optimization of Deep Learning Program

```
A = t.placeholder((1024, 1024))
B = t.placeholder((1024, 1024))
k = t.reduce_axis((0, 1024))
C = t.compute((1024, 1024), lambda y, x:
                t.sum(A[k, y] * B[k, x], axis=k))
s = t.create_schedule(C.op)
```

```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

+ Loop Tiling

```
yo, xo, ko, yi, xi, ki = s[C].tile(y, x, k, 8, 8, 8)
```

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

+ Cache Data on Accelerator Special Buffer

```
CL = s.cache_write(C, vdla.acc_buffer)
AL = s.cache_read(A, vdla.inp_buffer)
# additional schedule steps omitted …
```

+ Map to Accelerator Tensor Instructions

```
s[CL].tensorize(yi, vdla.gemm8x8)
```

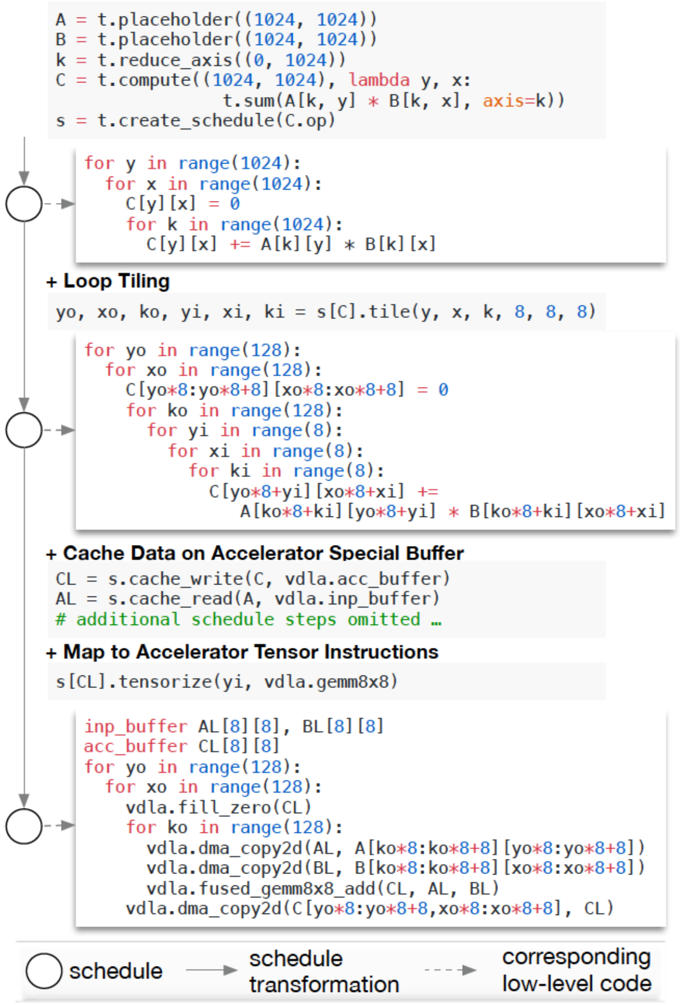
```
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
  for xo in range(128):
    vdla.fill_zero(CL)
    for ko in range(128):
      vdla.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])
      vdla.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])
      vdla.fused_gemm8x8_add(CL, AL, BL)
    vdla.dma_copy2d(C[yo*8:yo*8+8,xo*8:xo*8+8], CL)
```

schedule — schedule transformation — corresponding low-level code

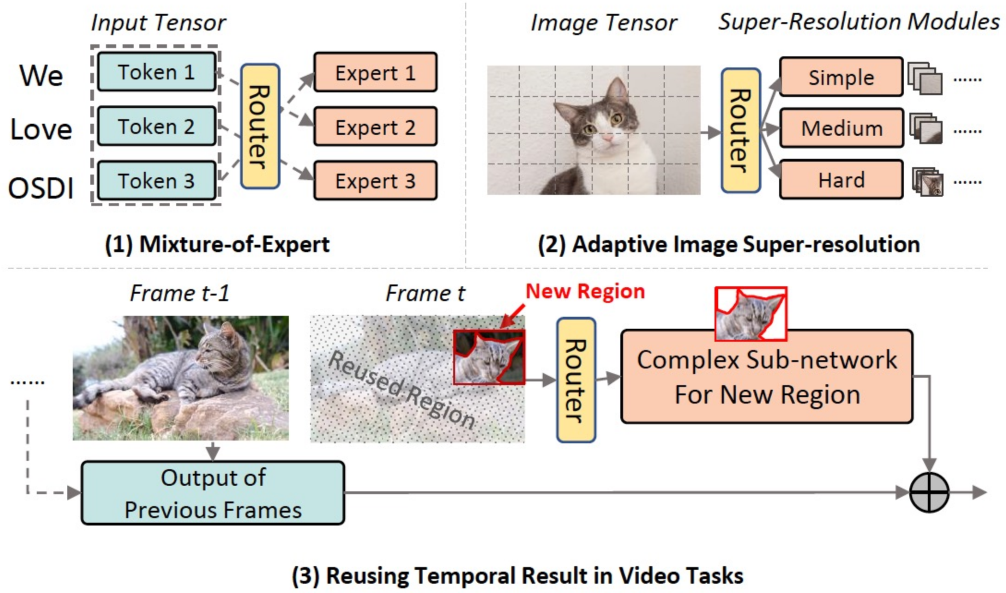
[OSDI'18 TVM]

Background

Optimization of Deep Learning Program



It's hard to utilize these existing optimizations on dynamic neural networks.



Background

Optimization of Deep Learning Program

```
A = t.placeholder((1024, 1024))
B = t.placeholder((1024, 1024))
k = t.reduce_axis((0, 1024))
C = t.compute((1024, 1024), lambda y, x:
            t.sum(A[k, y] * B[k, x], axis=k))
s = t.create_schedule(C.op)
```

```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

+ Loop Tiling

```
yo, xo, ko, yi, xi, ki = s[C].tile(y, x, k, 8, 8, 8)
```

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

+ Cache Data on Accelerator Special Buffer

```
CL = s.cache_write(C, vdla.acc_buffer)
AL = s.cache_read(A, vdla.inp_buffer)
# additional schedule steps omitted …
```

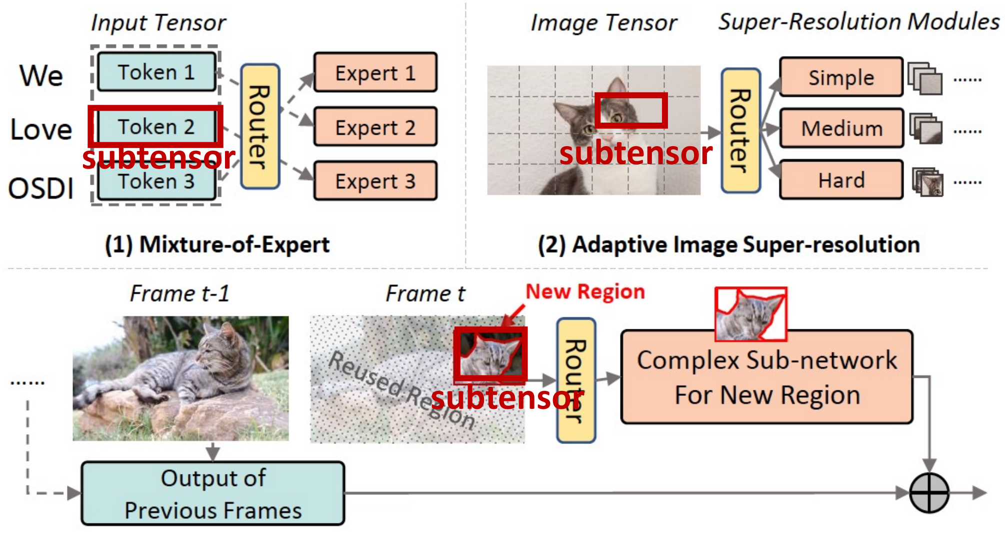
+ Map to Accelerator Tensor Instructions

```
s[CL].tensorize(yi, vdla.gemm8x8)
```

```
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
  for xo in range(128):
    vdla.fill_zero(CL)
    for ko in range(128):
      vdla.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])
      vdla.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])
      vdla.fused_gemm8x8_add(CL, AL, BL)
    vdla.dma_copy2d(C[yo*8:yo*8+8,xo*8:xo*8+8], CL)
```

schedule — schedule transformation — corresponding low-level code

It's hard to utilize these existing optimizations on dynamic neural networks.



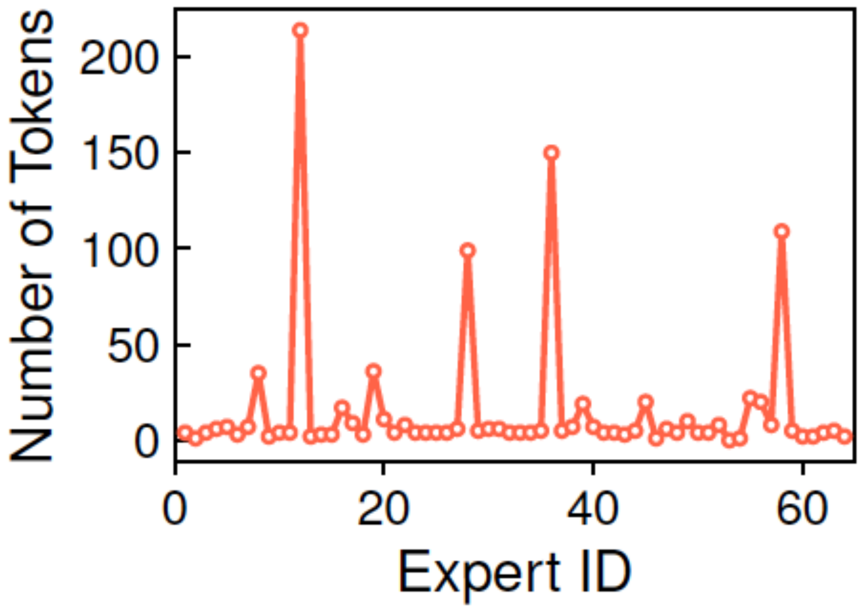
(3) Reusing Temporal Result in Video Tasks

Existing networks are all developed by Tensor-centric deep learning framework and the DL compiler only can optimize in tensor-level rather than subTensor-level in dynamic NNs.

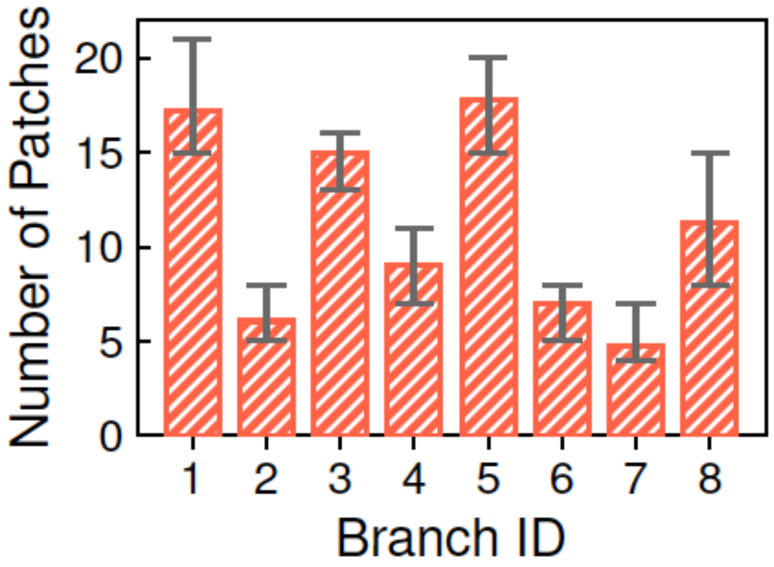
Motivation

Dynamic Optimization Opportunities

Study on SwitchTransformer



Study on LiveSR



Phenomenon: Imbalanced distribution of dispatched values for different experts.

Opportunity:

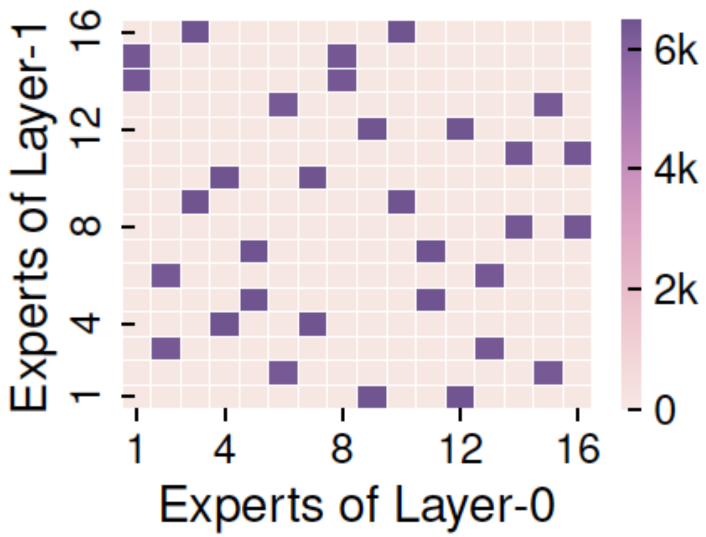
Tune efficient kernels to fit their shape to load distribution;

Horizontally fuse parallel branches for concurrent execution.

Motivation

Dynamic Optimization Opportunities

Study on TaskMoE



Phenomenon: Branch activation of two consecutive layers is correlated.

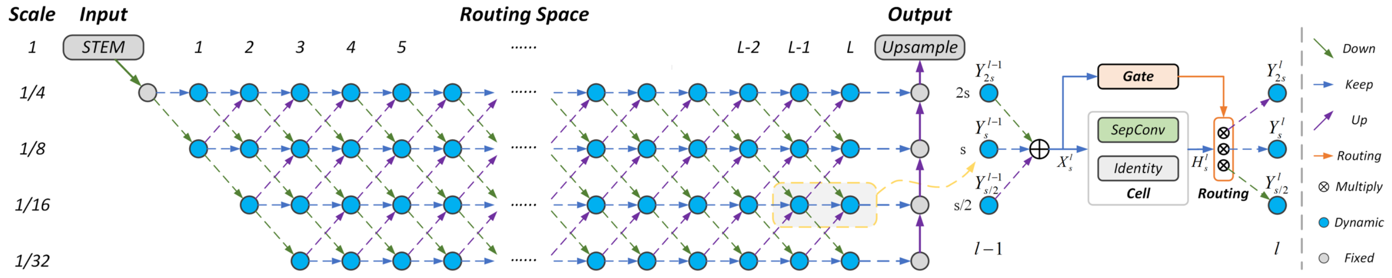
Opportunity:

Place highly related experts on the same GPU to save inter-GPU communication.

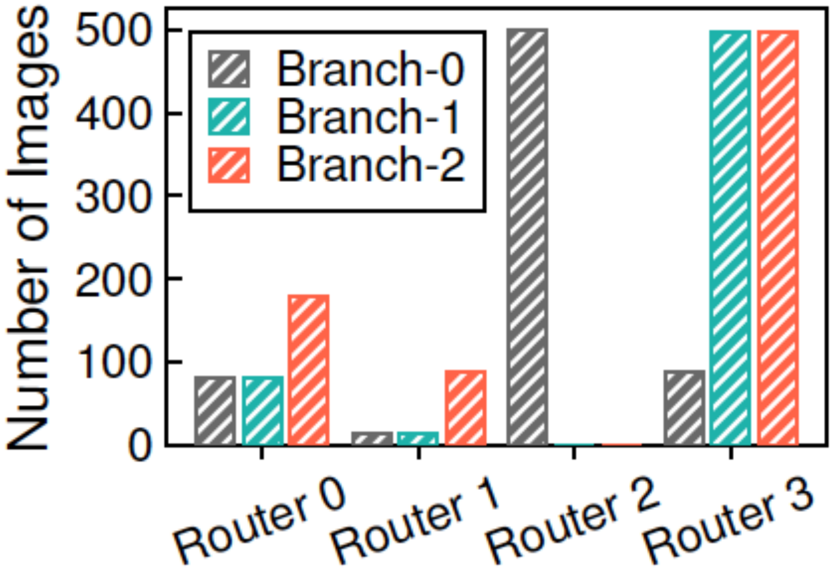
[EMNLP'21 TaskMoE]

Motivation

Dynamic Optimization Opportunities



Study on DynamicRouting



Phenomenon: Large time spent on routing but many routers have a biased distribution.

Opportunity:

Skipping routing computation to reduce routing overhead;

Preload weight to GPU memory for overlapping weight loading cost.

[CVPR'20 DynamicRouting]

Motivation

Dynamic Optimization Opportunities

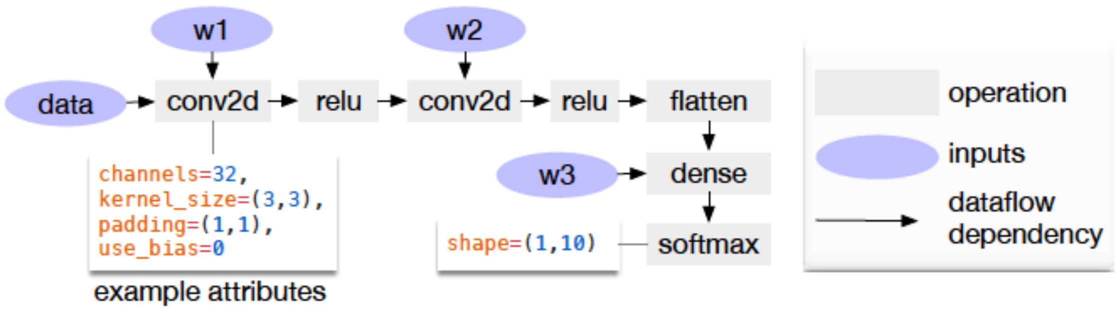
Opportunity:

1. Tune efficient kernels to fit their shape to load distribution;
2. Horizontally fuse parallel branches for concurrent execution;
3. Place highly related experts on the same GPU to save inter-GPU communication;
4. Skipping routing computation to reduce routing overhead;
5. Preload weight to GPU memory for overlapping weight loading cost.

Motivation

Misaligned Programming Model

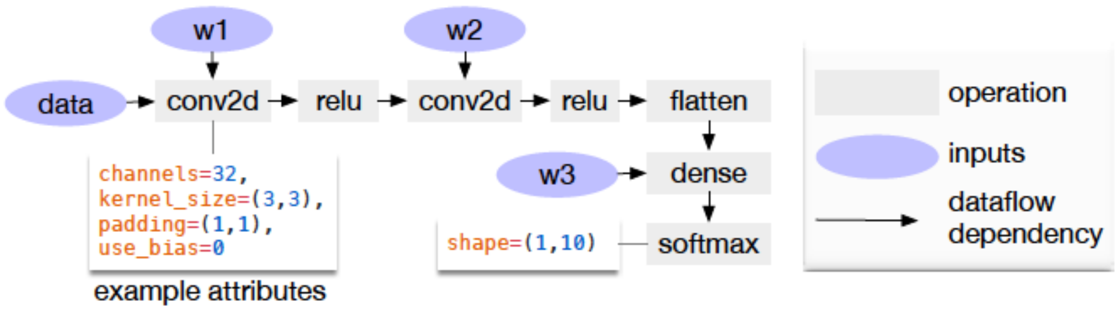
Existing Deep Learning Framwork: Tensor-centric.



Motivation

Misaligned Programming Model

Existing Deep Learning Framwork: Tensor-centric.

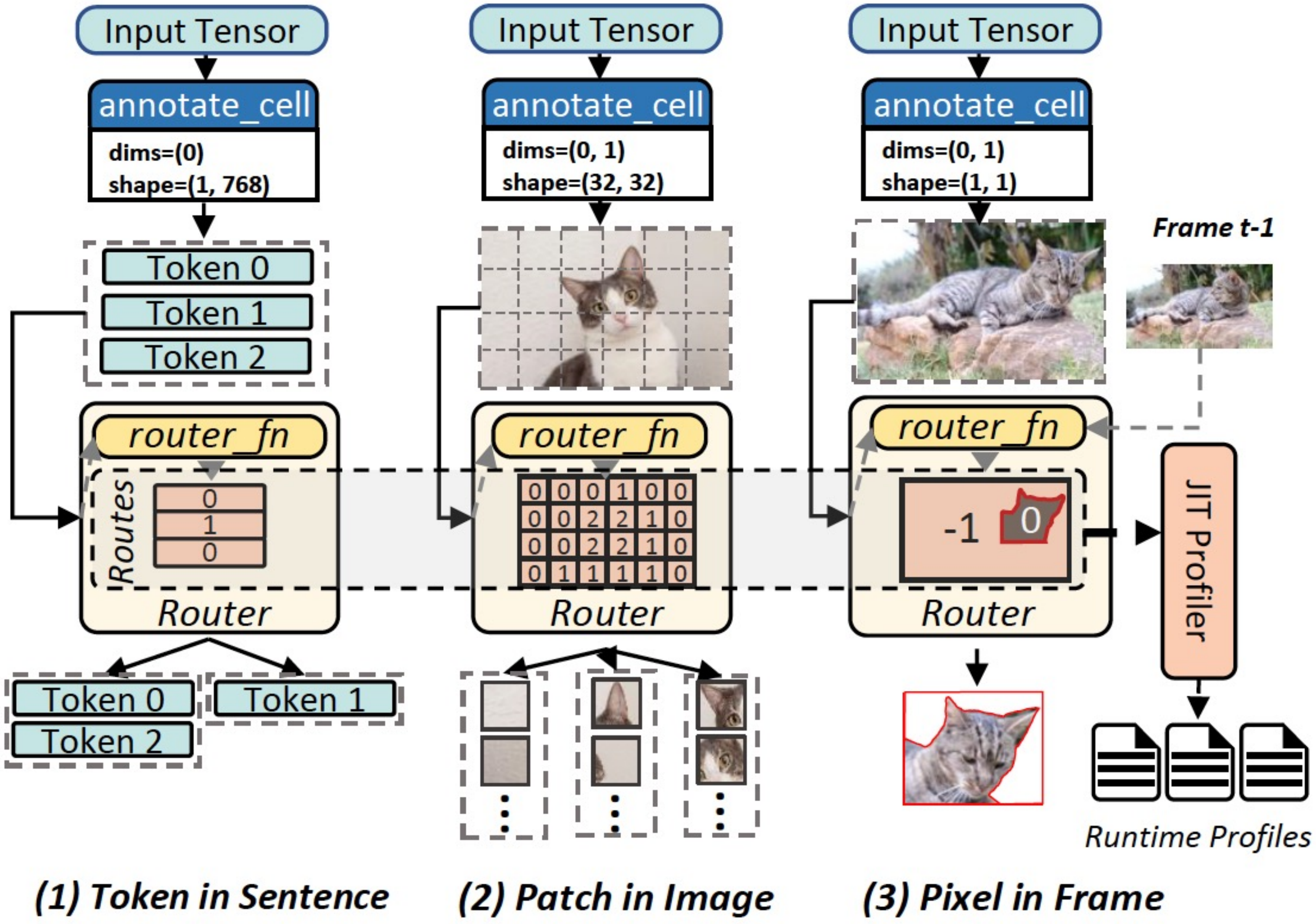


In dynamic NNs, the realistic computation happen in subtensors.

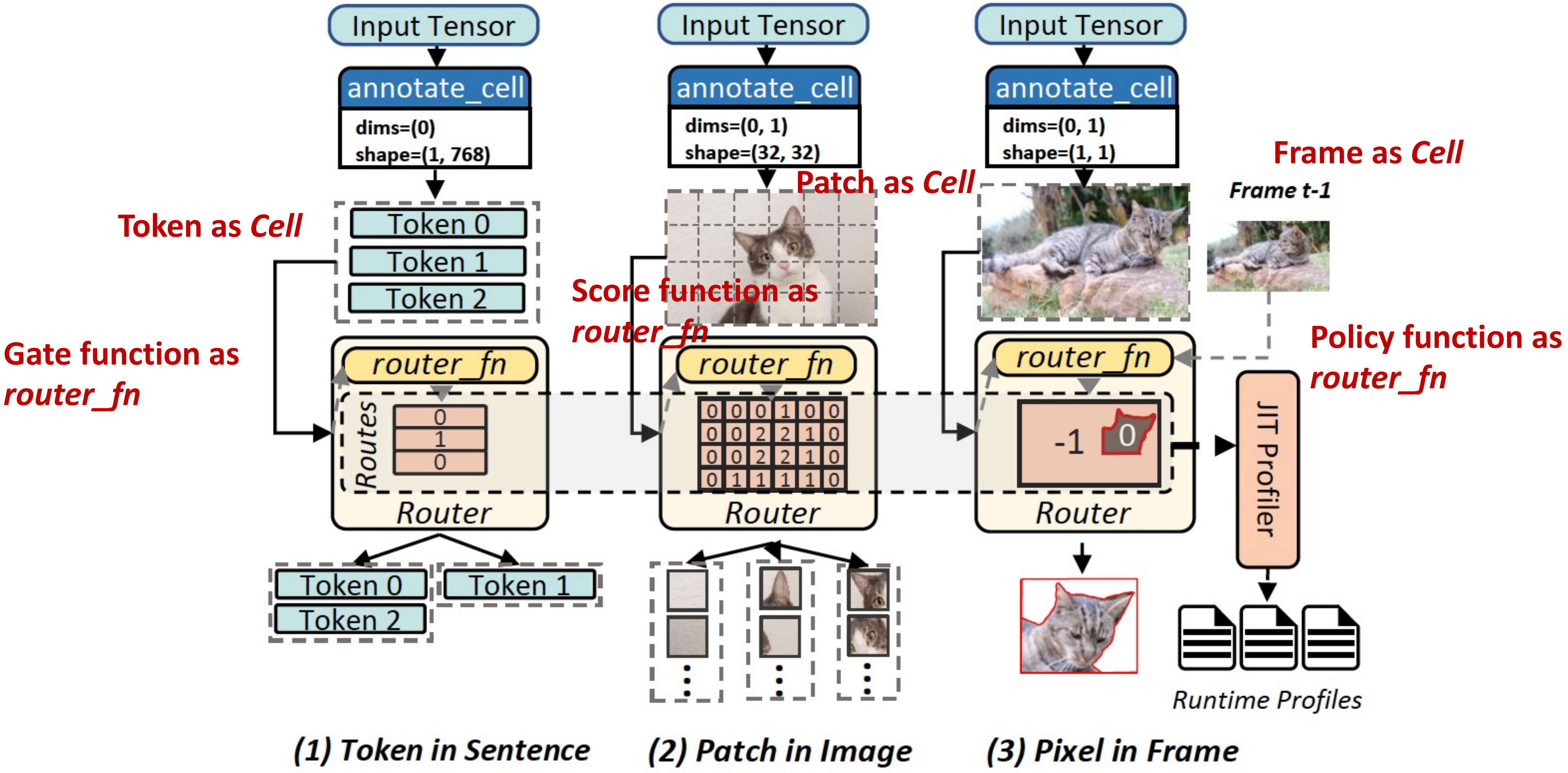
Performance: Hard to analyze and optimize in runtime.

Usability: Hard to program dynamic NNs.

Cell and *Router* Abstraction

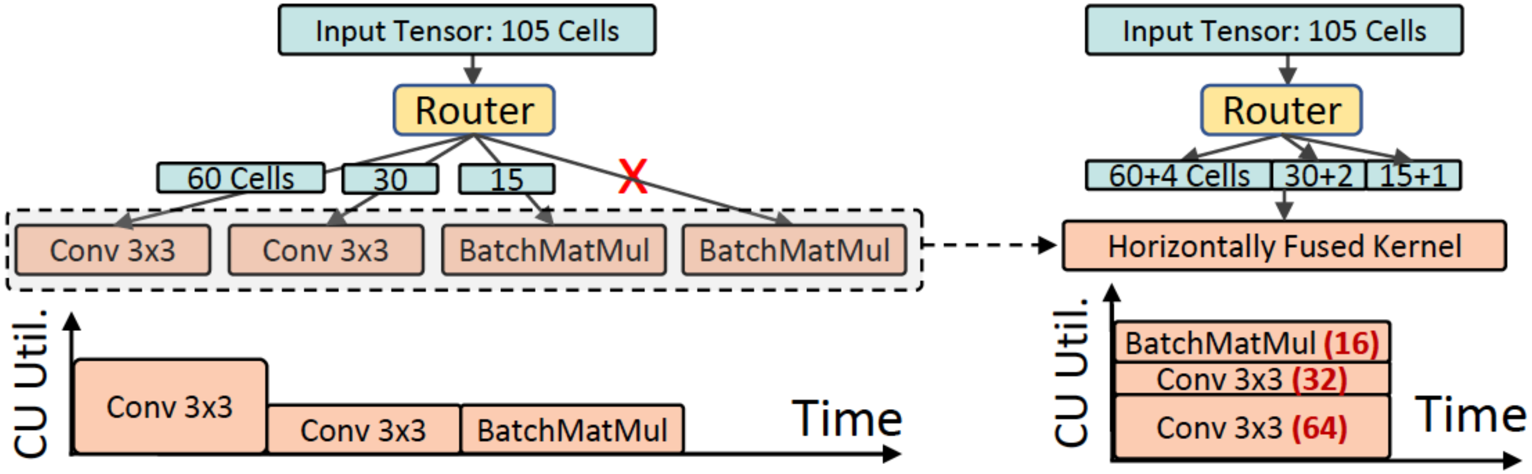


Cell and *Router* Abstraction



Dynamic Optimizations

Dynamic Horizontal Fusion

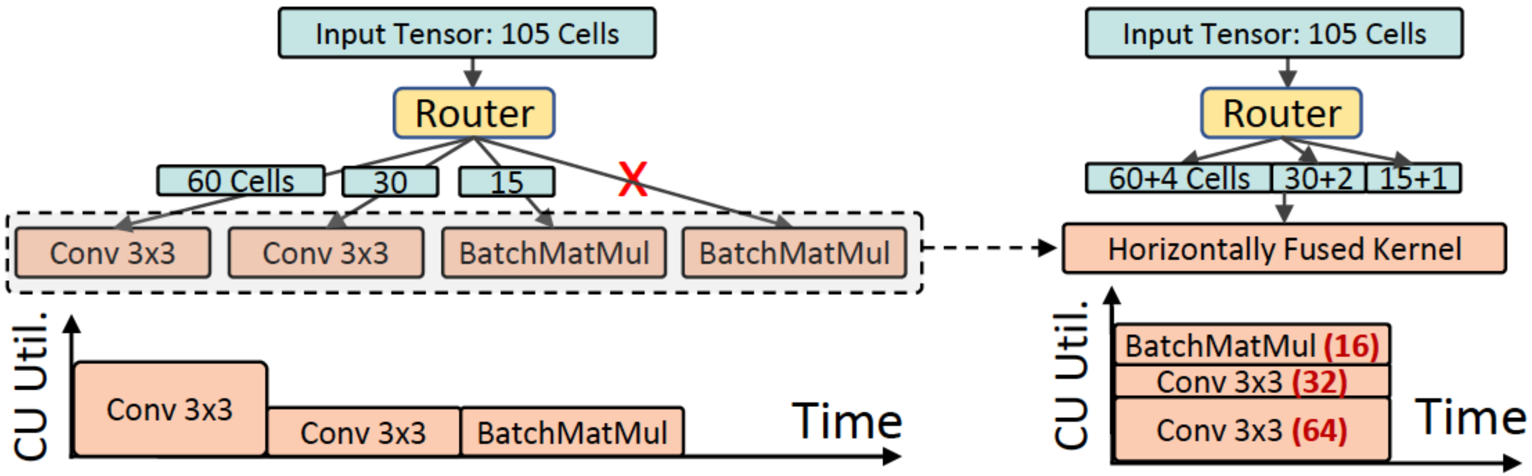


(1) sequential execution (2) horizontally fused

Supports dynamically and sparsely activated branches.

Dynamic Optimizations

Dynamic Horizontal Fusion



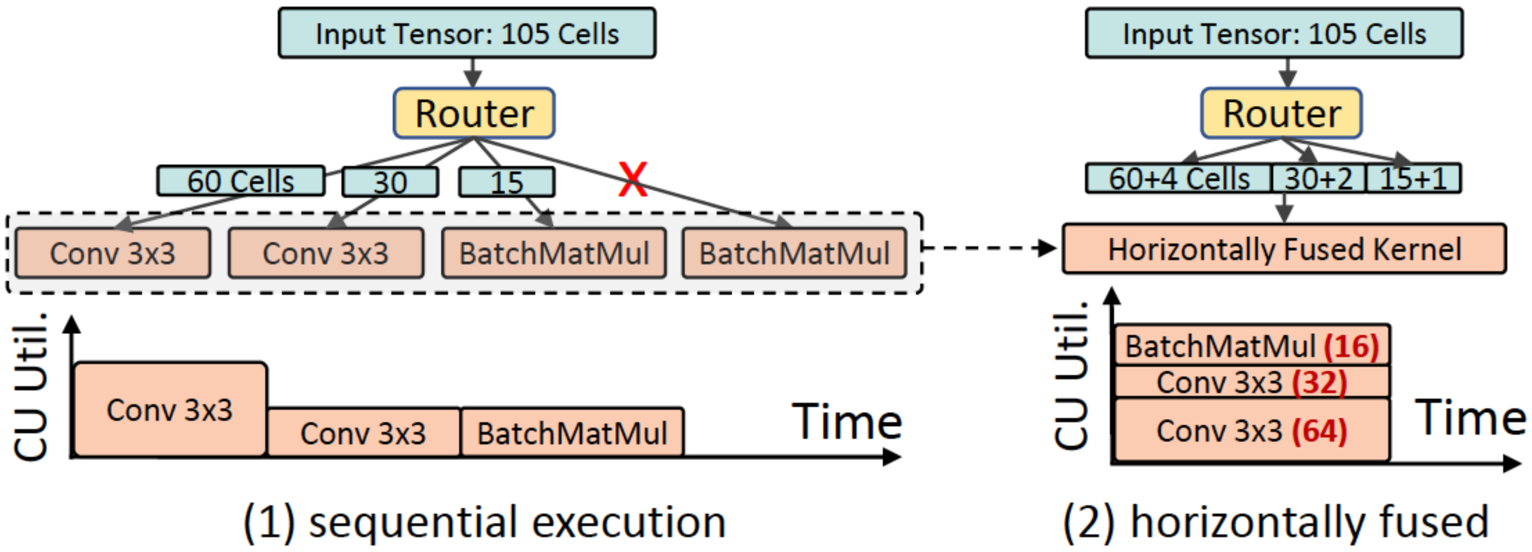
(1) sequential execution (2) horizontally fused

Supports dynamically and sparsely activated branches.

Before inference, tuning various kernels with different shape.

Dynamic Optimizations

Dynamic Horizontal Fusion

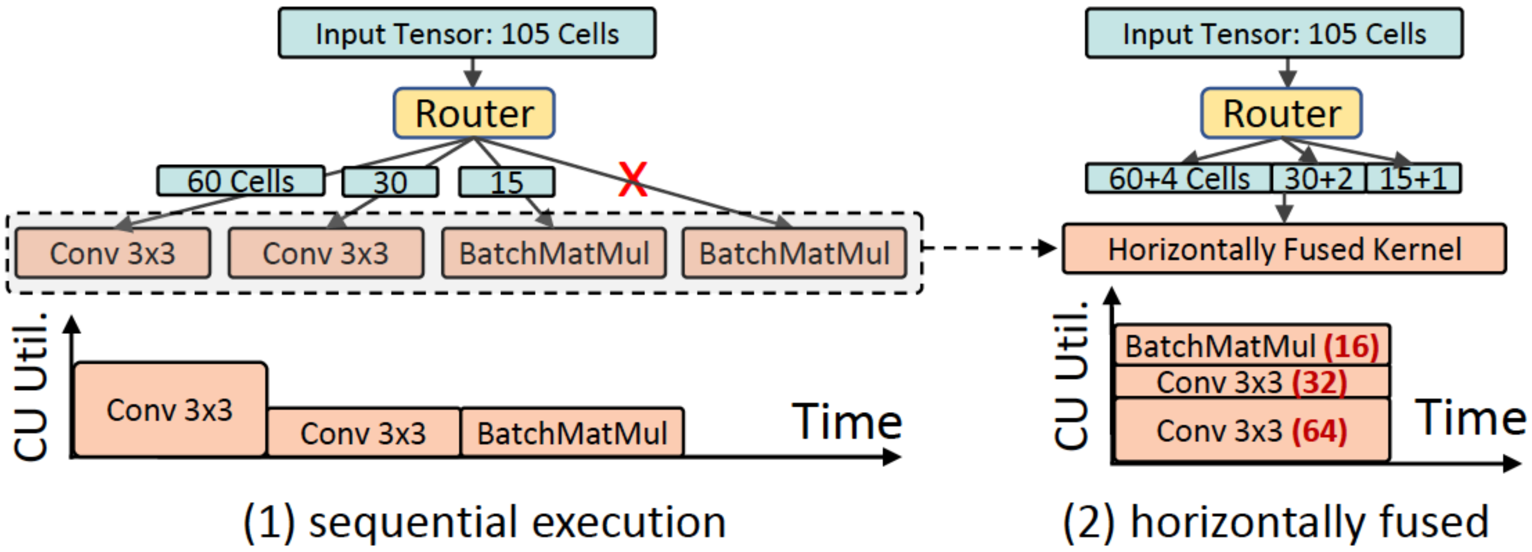


Supports dynamically and sparsely activated branches.

Before inference, tuning various kernels with different shape.
At inferenece, pads the input of each branch to the nearest tuned kernel.

Dynamic Optimizations

Dynamic Horizontal Fusion

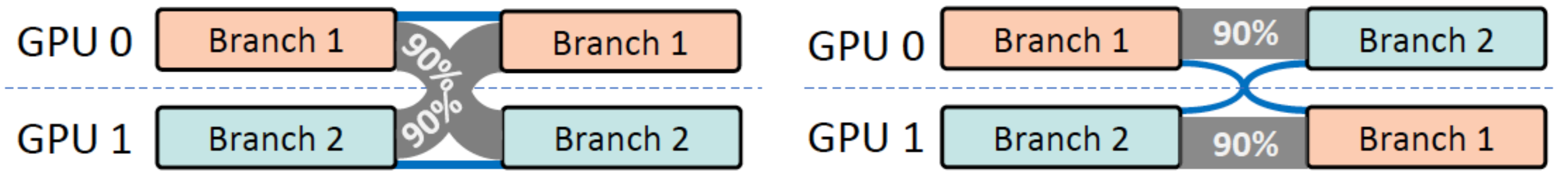


Supports dynamically and sparsely activated branches.

Before inference, tuning various kernels with different shape.
At inferenece, pads the input of each branch to the nearest tuned kernel.
The dynamically fused GPU kernel only uses the weights of activated branches.

Dynamic Optimizations

Profile-guided Model Placement



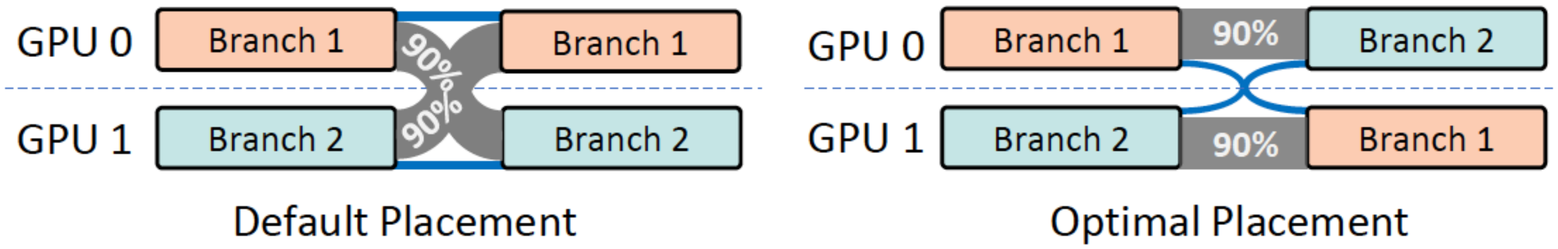
Default Placement

Optimal Placement

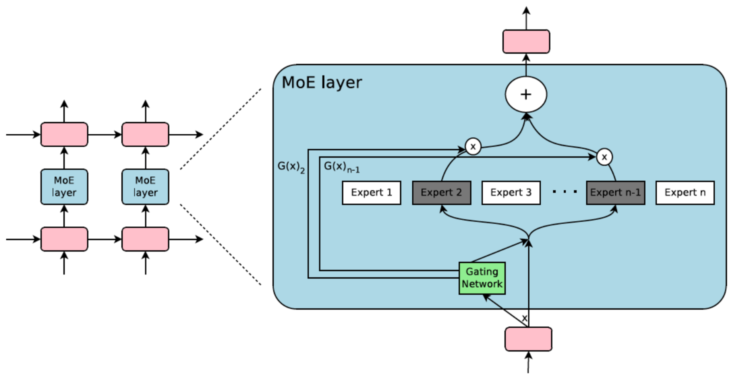
Co-locate correlated sub-networks on the same GPU to reduce inter-GPU communication.

Dynamic Optimizations

Profile-guided Model Placement



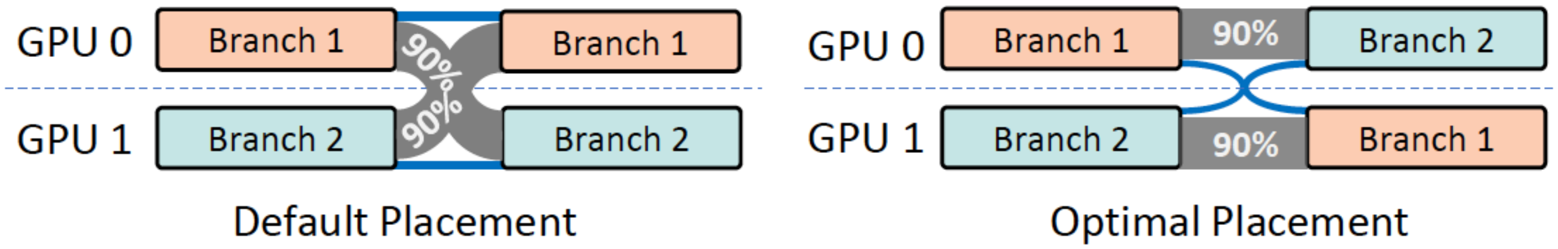
Co-locate correlated sub-networks on the same GPU to reduce inter-GPU communication.



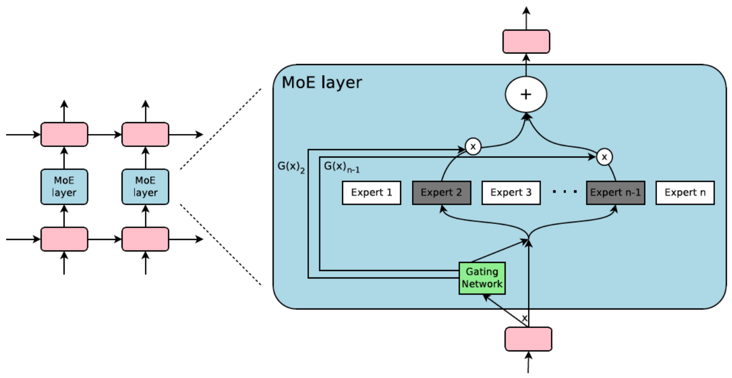
In MoE layers, layer-wise dependence exists which implies a placement constraint that all Cells of a sentence should be gathered at the same GPU.

Dynamic Optimizations

Profile-guided Model Placement



Co-locate correlated sub-networks on the same GPU to reduce inter-GPU communication.

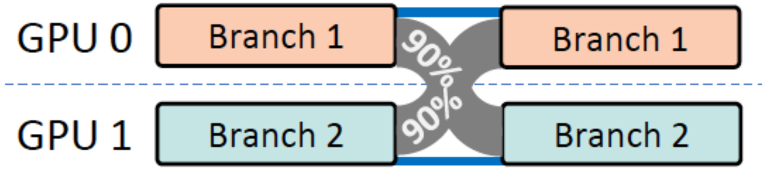


In MoE layers, layer-wise dependence exists which implies a placement constraint that all Cells of a sentence should be gathered at the same GPU.

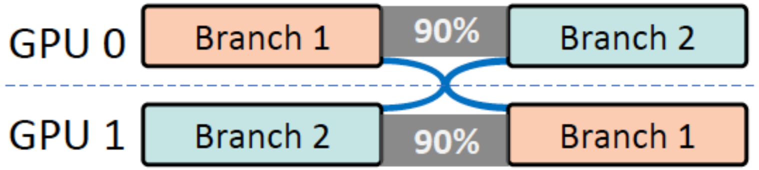
Need to achieve these constraints by static dataflow analysis.

Dynamic Optimizations

Profile-guided Model Placement

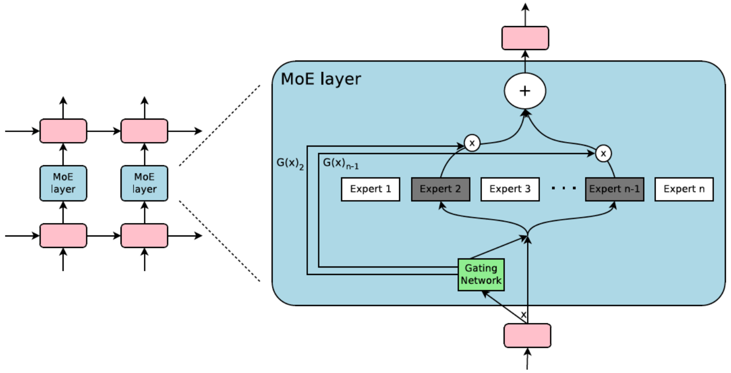


Default Placement



Optimal Placement

Co-locate correlated sub-networks on the same GPU to reduce inter-GPU communication.



In MoE layers, layer-wise dependence exists which implies a placement constraint that all Cells of a sentence should be gathered at the same GPU.

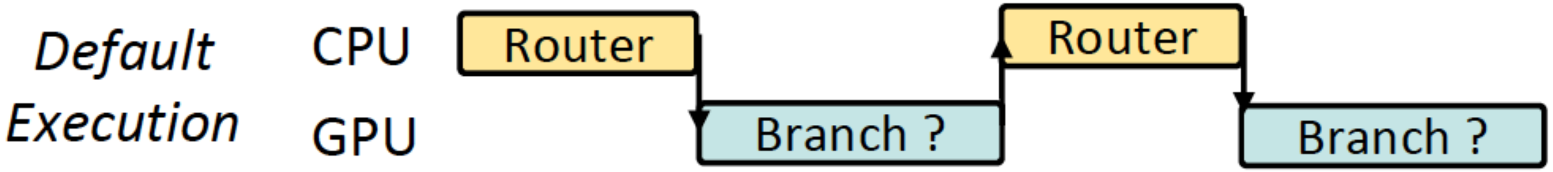
Need to achieve these constraints by static dataflow analysis.

Moreover, utilizing imbalance workloads for different branches that combining a strong branch and a weak branch can help balance the workloads around all GPUs.

Dynamic Optimizations

Speculative Routing

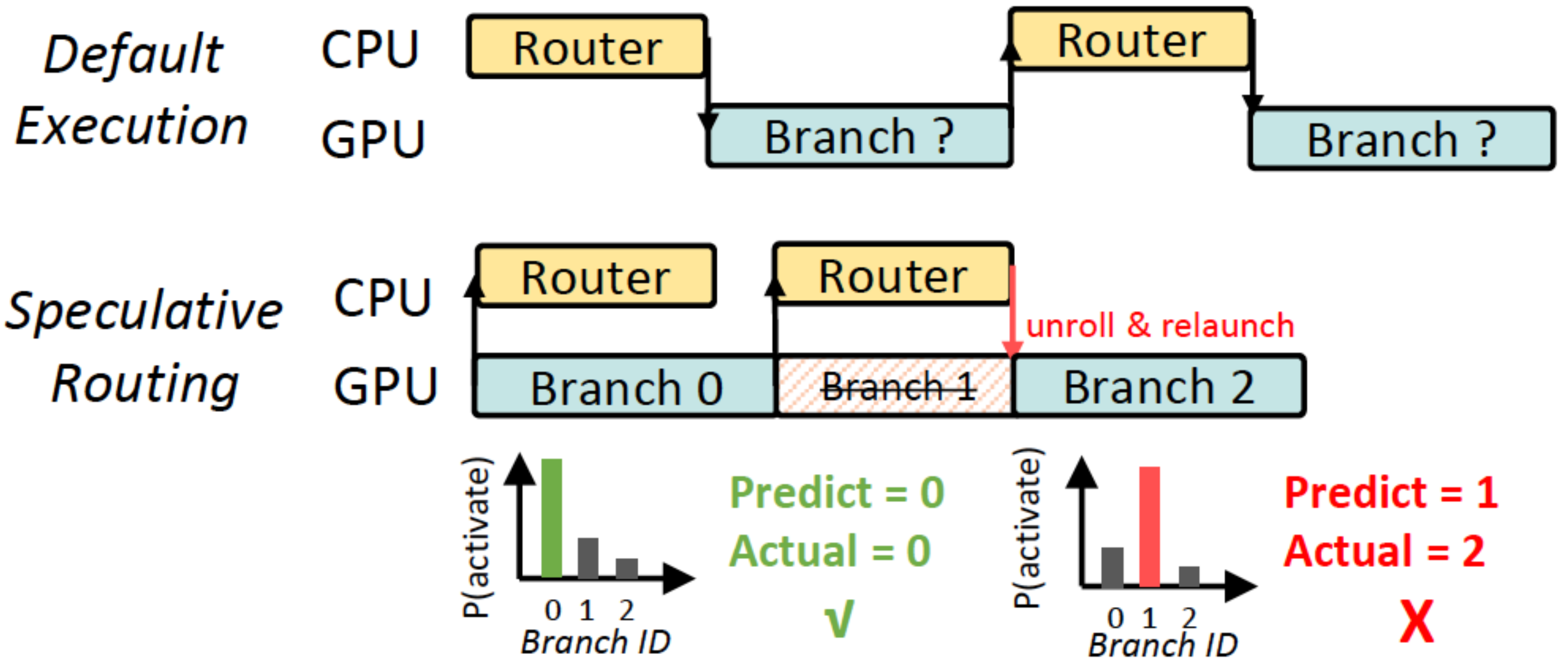
Routing require CPU processing and incur CPU-GPU synchronization overhead.



Dynamic Optimizations

Speculative Routing

Routing require CPU processing and incur CPU-GPU synchronization overhead.

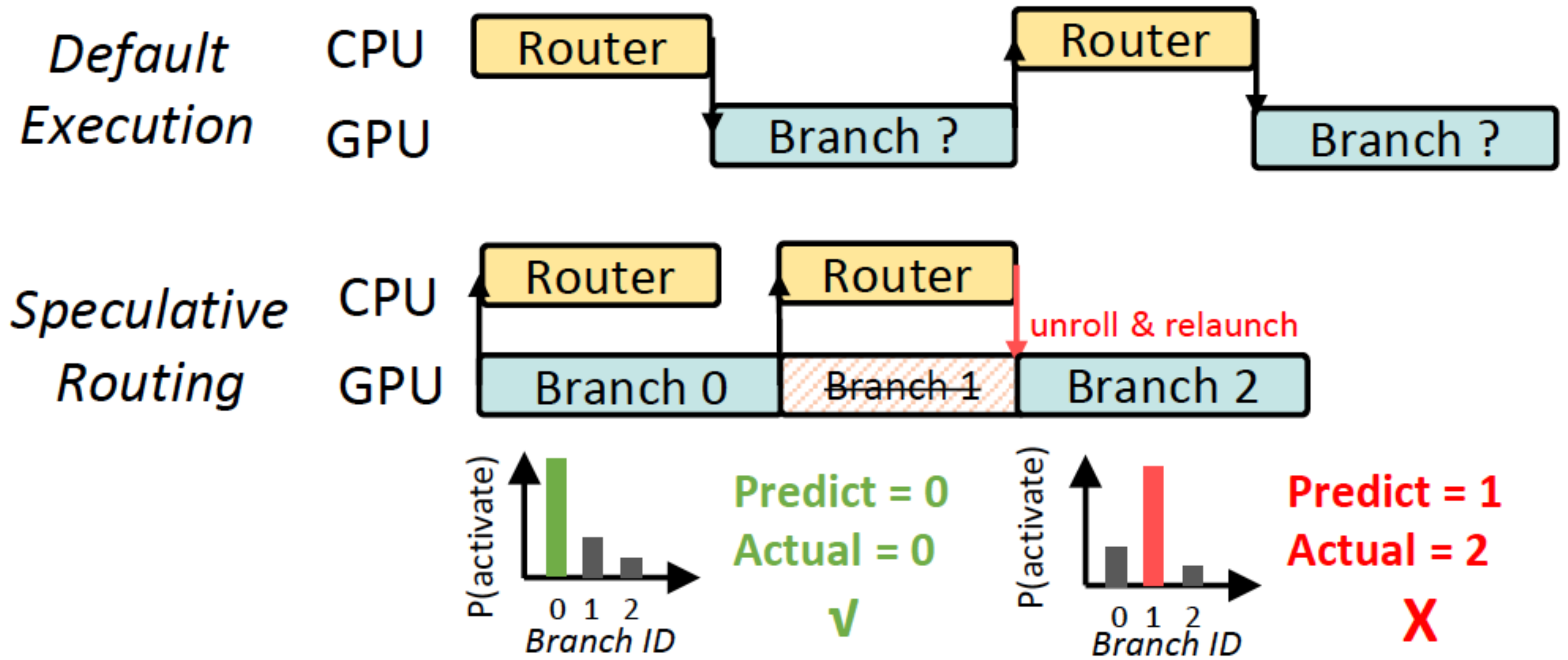


Predict the routing decisions of Router in advance based on statistical profiles and skip *router_fn* to hide the routing overhead.

Dynamic Optimizations

Speculative Routing

Routing require CPU processing and incur CPU-GPU synchronization overhead.

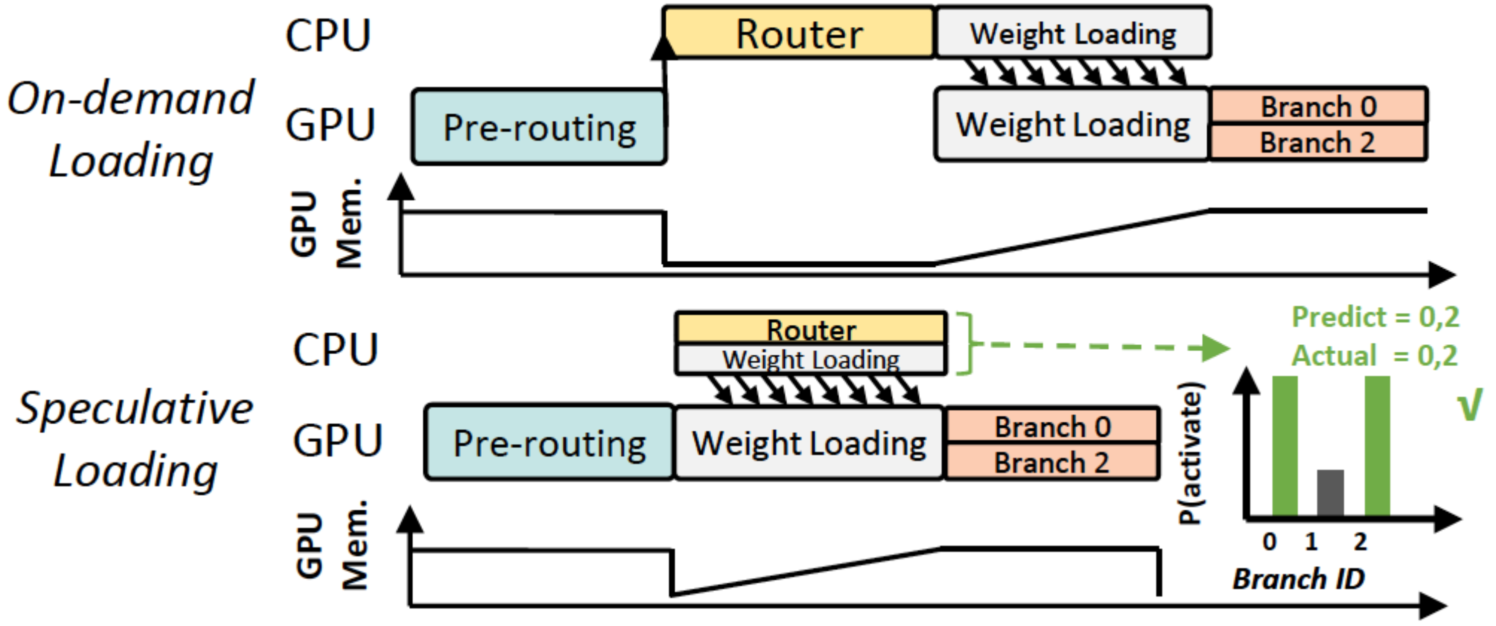


Predict the routing decisions of Router in advance based on statistical profiles and skip *router_fn* to hide the routing overhead.

When misprediction happens, the model execution will be unrolled to re-execute the correct branch with negligible misprediction overhead.

Dynamic Optimizations

Speculative Weight Preloading



Similar to speculative routing, leverages the statistical profiles of branch activation distribution to speculatively preload weights of branches that can be activated with a high probability.

It fall back to on-demand loading with negligible overhead when the predictive preloading misses.

Dynamic Optimizations

Opportunity:

1. Tune efficient kernels to fit their shape to load distribution;
2. Horizontally fuse parallel branches for concurrent execution; → **Dynamic horizontal fusion**
3. Place highly related experts on the same GPU to save inter-GPU communication; → **Profile-guided model placement**
4. Skipping routing computation to reduce routing overhead; → **Speculative routing**
5. Preload weight to GPU memory for overlapping weight loading cost. → **Speculative weight preloading**

Dynamic Optimizations

Opportunity:

1. Tune efficient kernels to fit their shape to load distribution; → **Static kernel tuning.**
2. Horizontally fuse parallel branches for concurrent execution; → **Dynamic horizontal fusion**
3. Place highly related experts on the same GPU to save inter-GPU communication; → **Profile-guided model placement**
4. Skipping routing computation to reduce routing overhead; → **Speculative routing**
5. Preload weight to GPU memory for overlapping weight loading cost. → **Speculative weight preloading**

Dynamic Optimizations

Opportunity:

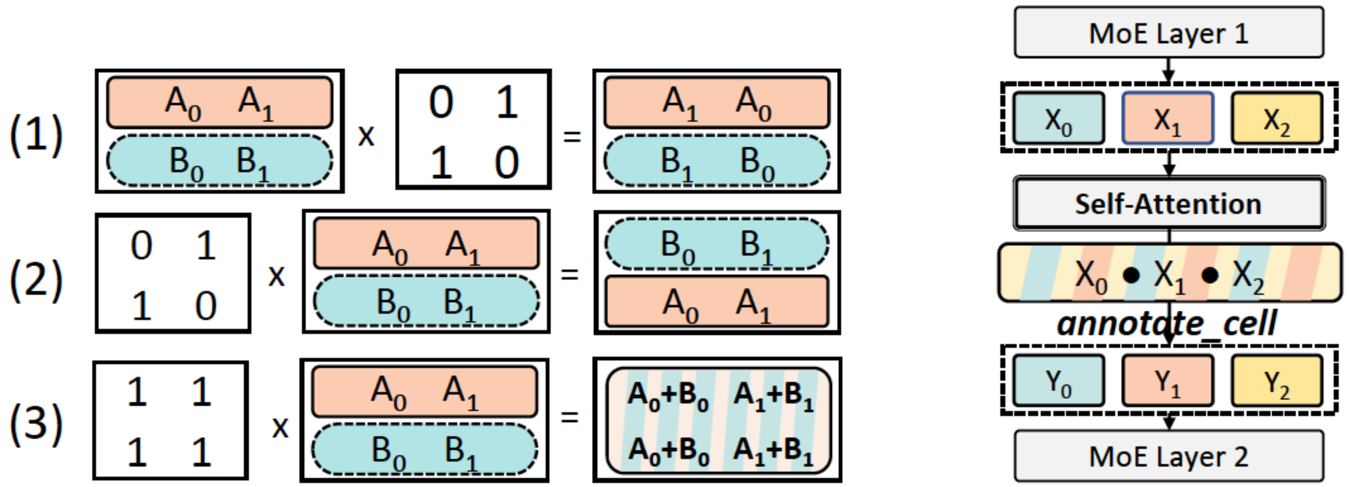
1. Tune efficient kernels to fit their shape to load distribution; → **Static kernel tuning.**
2. Horizontally fuse parallel branches for concurrent execution; → **Dynamic horizontal fusion**
3. Place highly related experts on the same GPU to save inter-GPU communication; → **Profile-guided model placement**
4. Skipping routing computation to reduce routing overhead; → **Speculative routing**
5. Preload weight to GPU memory for overlapping weight loading cost. → **Speculative weight preloading**

How to do analysis to achieve profiles for these optimizations?

Tracing Cell-level Dataflow

Static Cell-level Dataflow

Uses symbolic execution at Cell-level to extract finer-grained relations in ahead-of-time compiling.



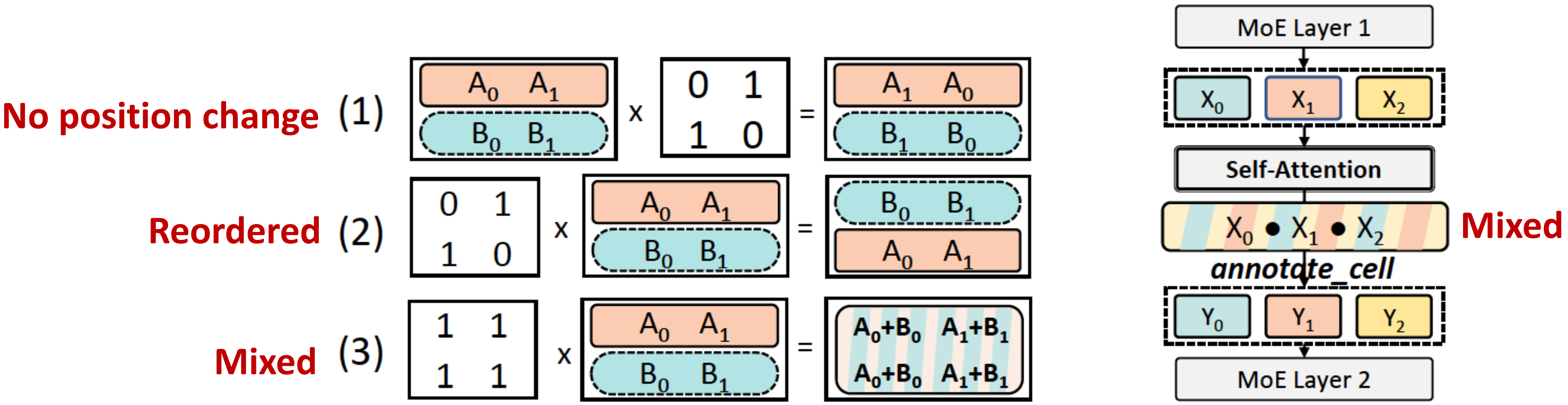
(a) Matrix multiplication

(b) Self-attention

Tracing Cell-level Dataflow

Static Cell-level Dataflow

Uses symbolic execution at Cell-level to extract finer-grained relations in ahead-of-time compiling.



(a) Matrix multiplication

(b) Self-attention

By checking the results of symbolic computation, Brainstorm understands how Cells are transmitted in static operators.

Tracing Cell-level Dataflow

Dynamic Cell-level Dataflow

Trace the necessary information in *router_fn*.

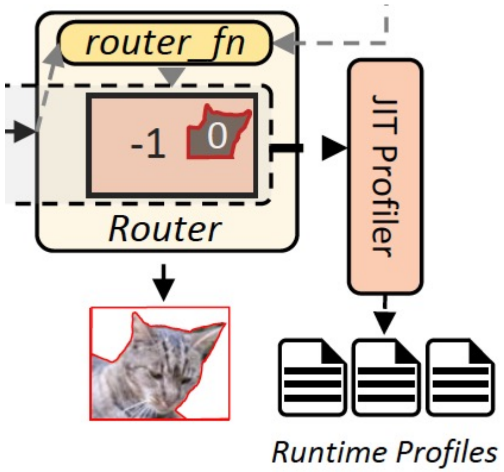
Tracing Cell-level Dataflow

Dynamic Cell-level Dataflow

Trace the necessary information in *router_fn*.

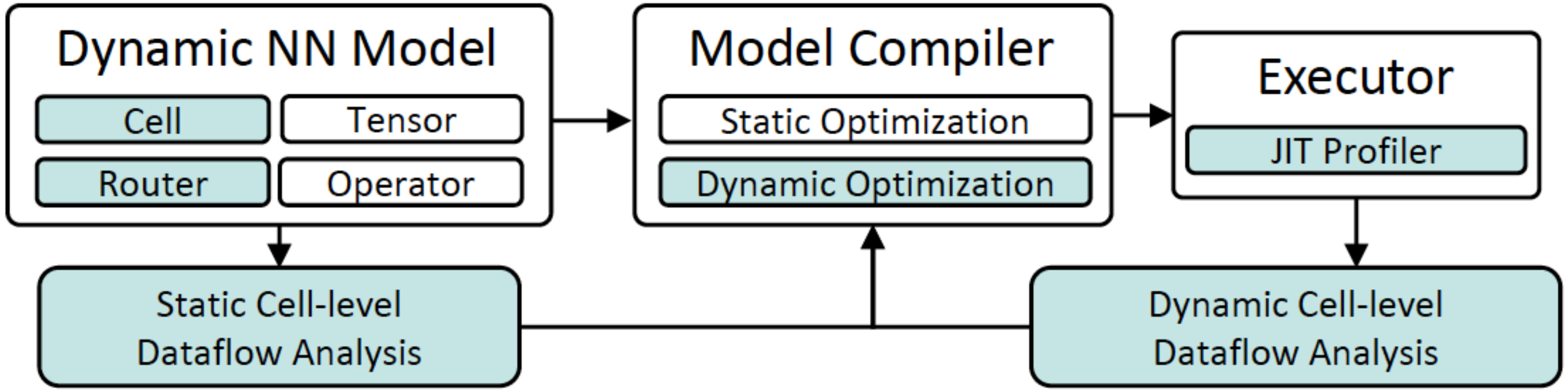
When each time a Router is called, Brainstorm records its routing decision into a buffer.

Brainstorm has a separate thread to stream the buffer to a profile file.



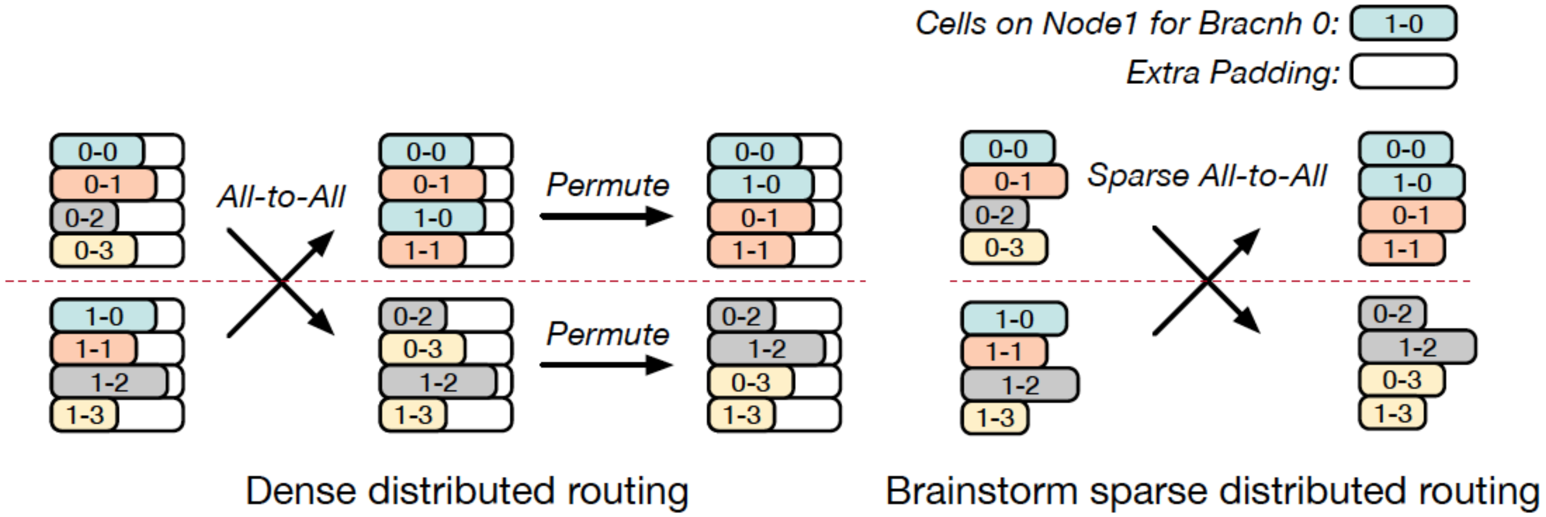
Implementation

System Architecture



Implementation

Sparse Communication



Implementation

More Optimizations

Efficient *Cell* Routing : Use a custom **GPU kernel** to rearrange Cells inside a tensor according to the routing decisions and **routing in parallel.**

Excessive candidates for kernel fusion: Each multiple branches fused kernel **comprising several potential candidates.**

Evaluation

Setup

Single-GPU server: A100 (80GB) with AMD-EPYC-7V13 CPUs.

Multi-GPU server: V100 (32GB) x 8 with Intel Xeon E5-2690 v4 CPUs.

Model	Dataset	Fusion	Place	Route	Load
Switch [14]	MNLI [40]	✔			
TaskMoE [27]	Synthetic		✔		
SwinV2-MoE [41]	ImageNet22k [42]		✔		
LiveSR	Iowa-DOT [43]	✔			
DRouting [28]	Cityscapes [44]			✔	✔
MSDNet [1]	Imagenet [42]	✔		✔	

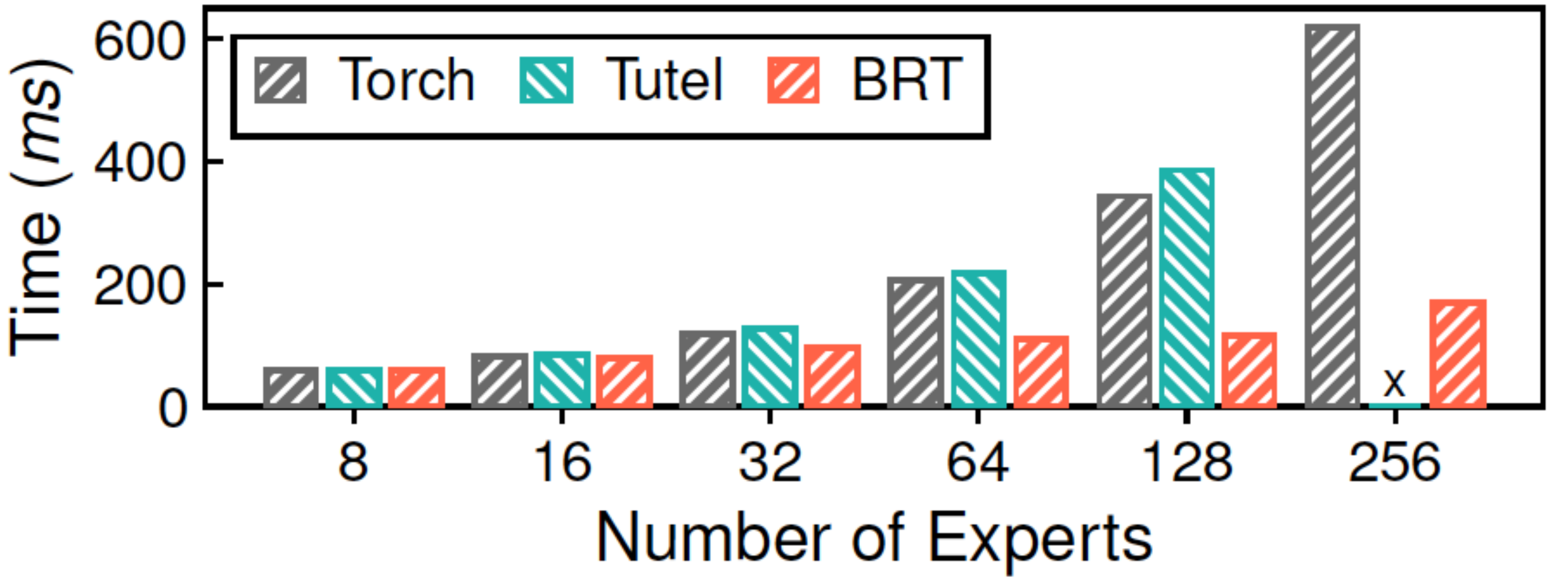
Baseline:

PyTorch for all included vertical fusion.
Tutel for MoE (optimized in routing and supports parallel execution).

Evaluation

End-to-end Model Execution

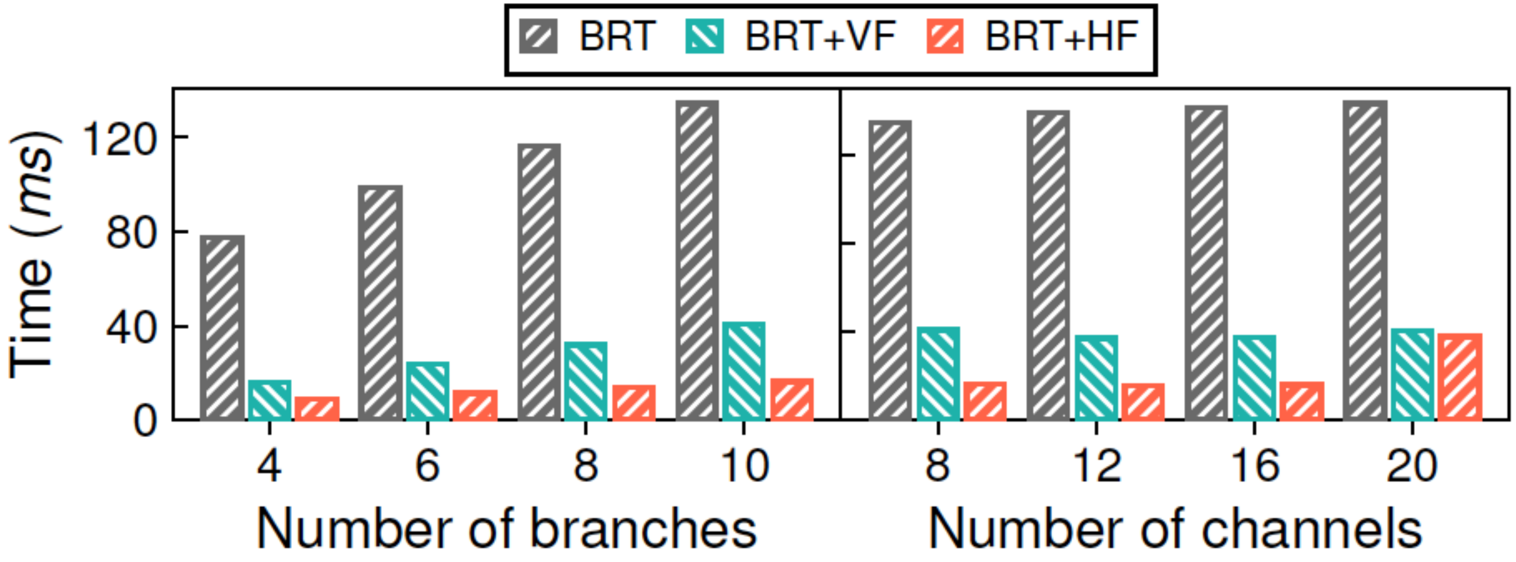
SwitchTransformer



Evaluation

End-to-end Model Execution

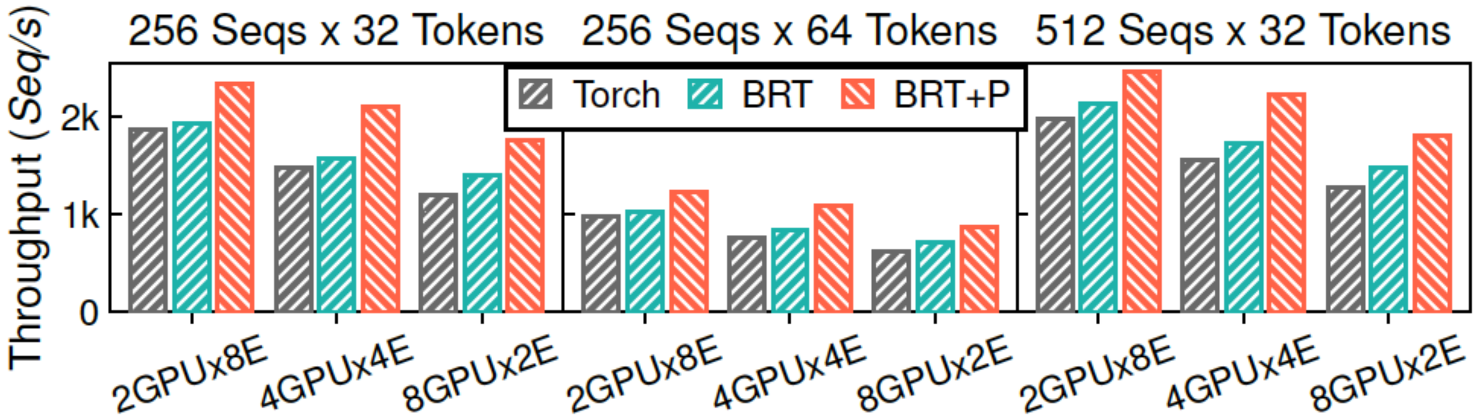
LiveSR



Evaluation

End-to-end Model Execution

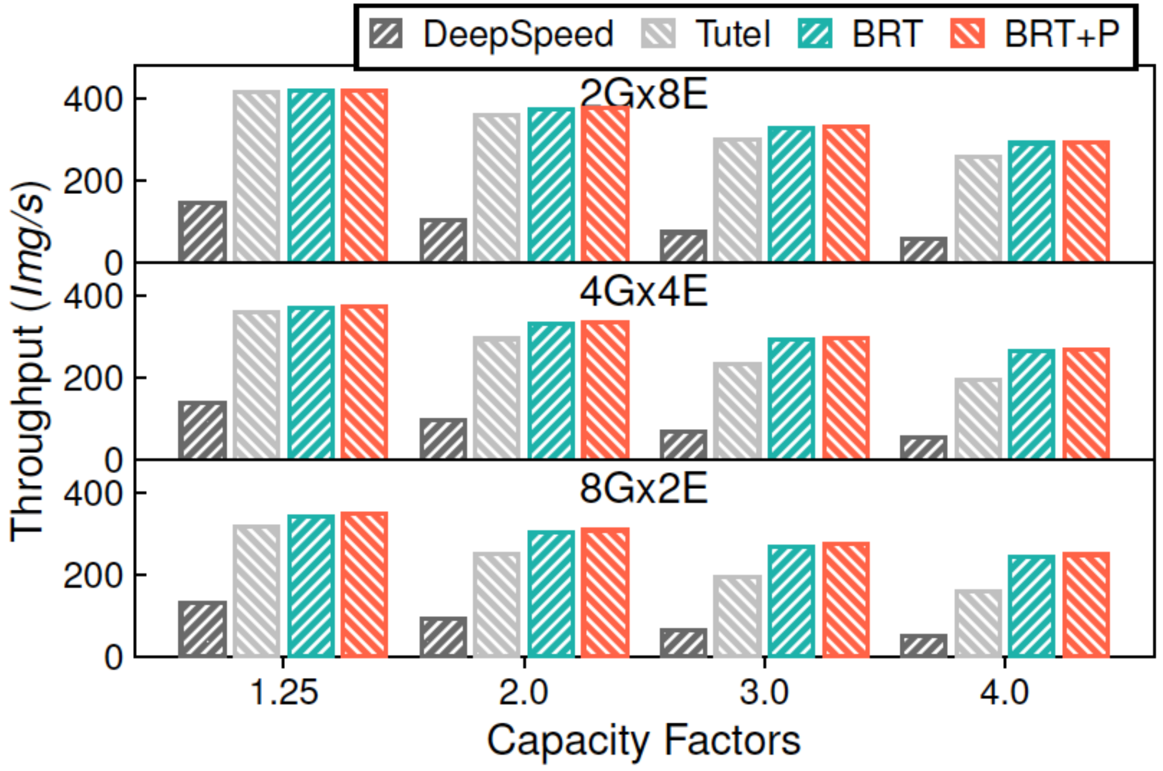
TaskMoE



Evaluation

End-to-end Model Execution

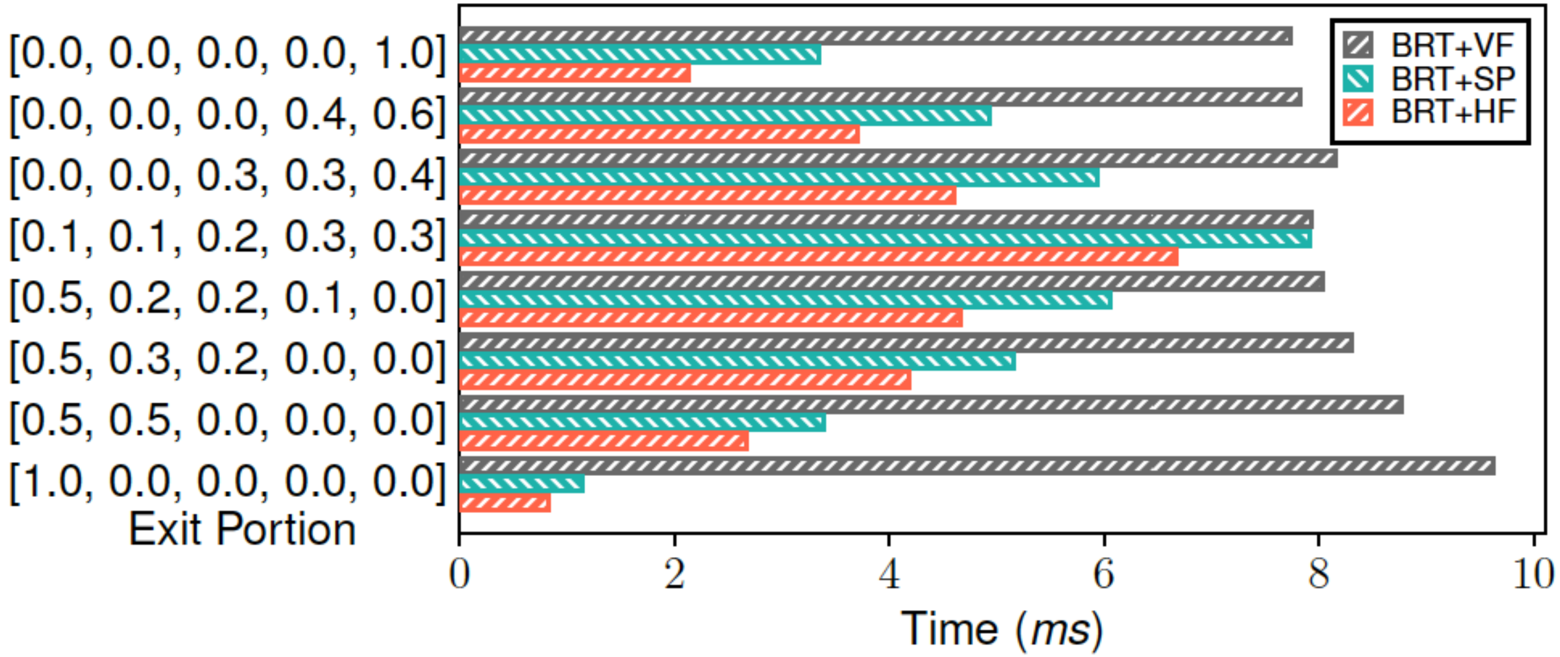
SwinV2MoE



Evaluation

End-to-end Model Execution

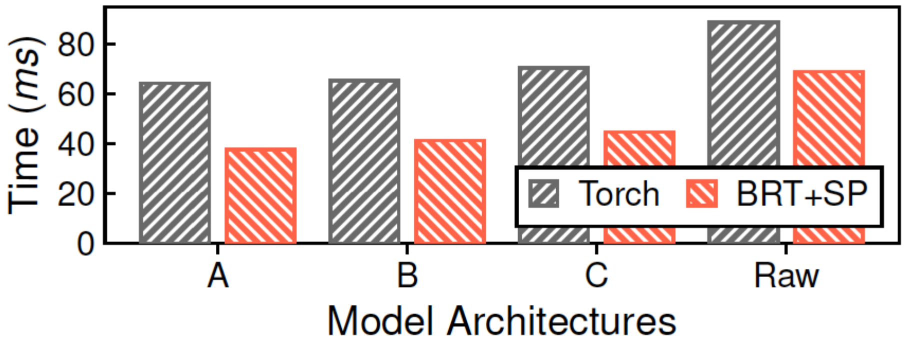
MSDNet

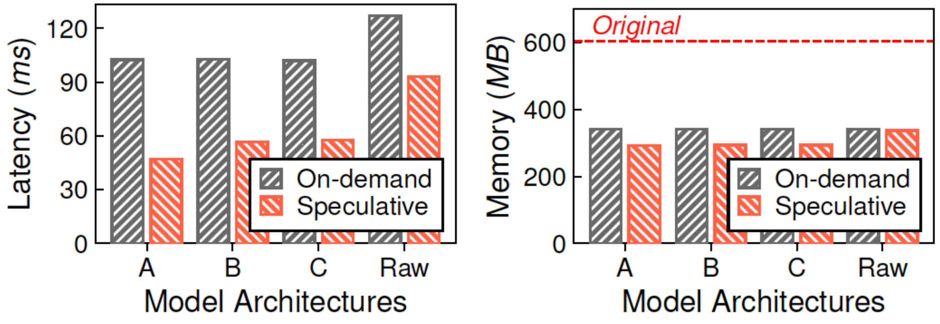


Evaluation

End-to-end Model Execution

DynamicRouting



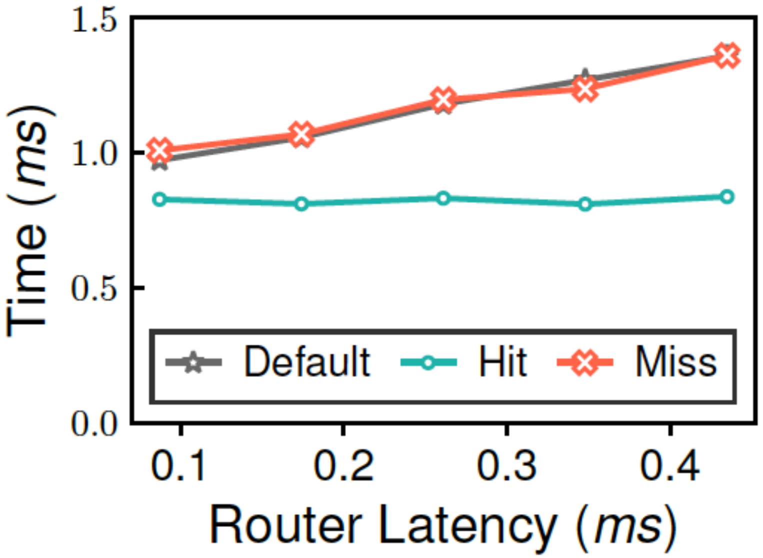


(a) Latencies (b) Memory Consumption

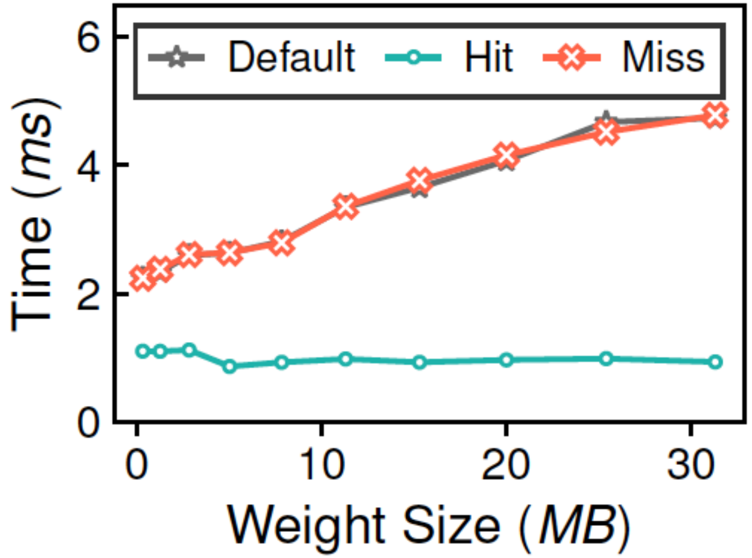
On-demand: Only loads the weight of a branch after the routing decision is made.

Evaluation

Speculative Effectiveness



(a) Variable *Router* latencies



(b) Variable weight sizes

Evaluation

Usability

Porting pretrained model into Brainstorm.

Model	Switch	TaskMoE	SwinV2-MoE
LOC	12	24	14
Model	**LiveSR**	**DRouting**	**MSDNet**
LOC	6	18	14

Evaluation

Usability

Porting pretrained model into Brainstorm.

Model	Switch	TaskMoE	SwinV2-MoE
LOC	12	24	14
Model	LiveSR	DRouting	MSDNet
LOC	6	18	14

```
376 >   def combine_addmask(self, sr_list, num_h, num_w, h, w, patch_size, step, type): ...
457         return sr_img
458
```

Actually, there are more LoC tweaking the original model.

Conclusion and Thoughts

- The first deep learning framework for optimizing the execution of dynamic neural network.
- Speculative optimizations are fancy in optimizing the neural network execution and effective.
- Full of compiling.
- Not general and hard to become a popular framework like TVM.
- Tedious programming work to extend the framework.

Thank You!

May 10, 2024

Presented by Mengyang Liu