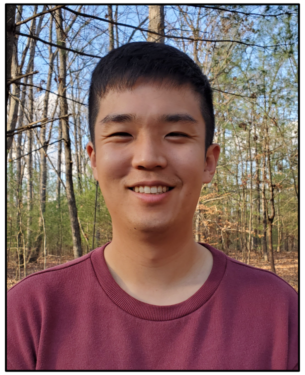
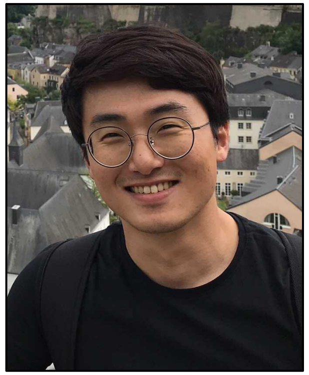
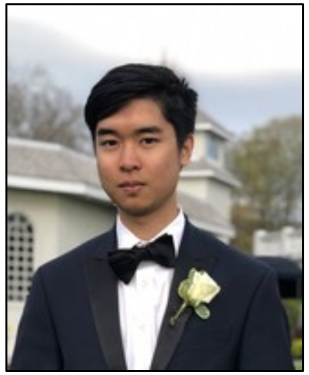
Collage: Automated Integration of Deep Learning Backends

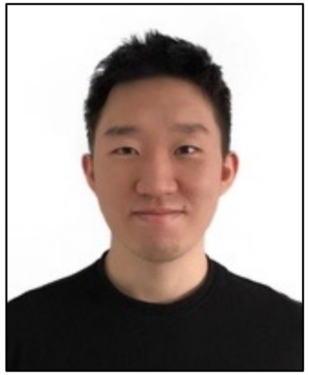
Byungsoo Jeon*[1], Sunghyun Park*[2], Peiyuan Liao[1,4], Sheng Xu[3], Tianqi Chen[1,2], Zhihao Jia[1]

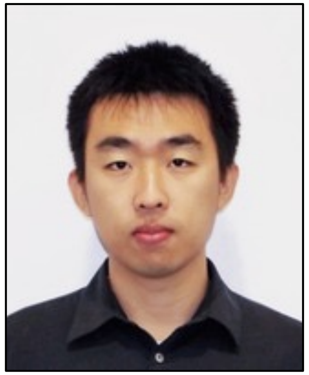
[1]Carnegie Mellon University, [2]OctoML, [3]Amazon Web Services, [4]Praxis Pioneering







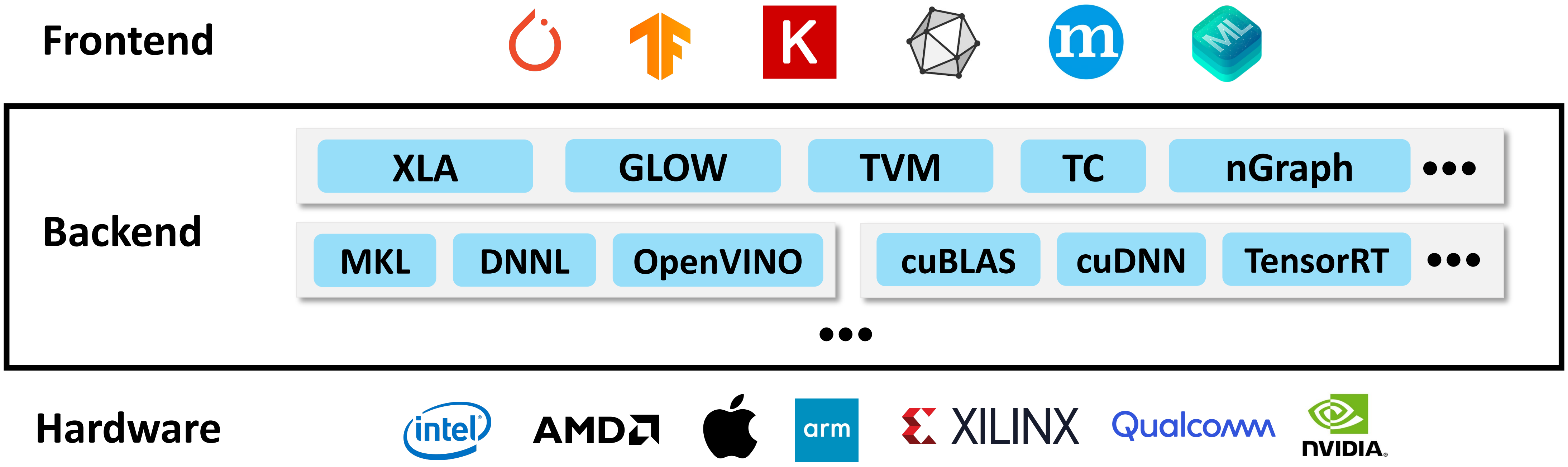




Deep Learning (DL) Backend

Backend

- a software library or a runtime framework that takes DL workloads as inputs and generates an optimized low-level target code



Frontend

Backend

XLA | GLOW | TVM | TC | nGraph | •••

MKL | DNNL | OpenVINO

cuBLAS | cuDNN | TensorRT | •••

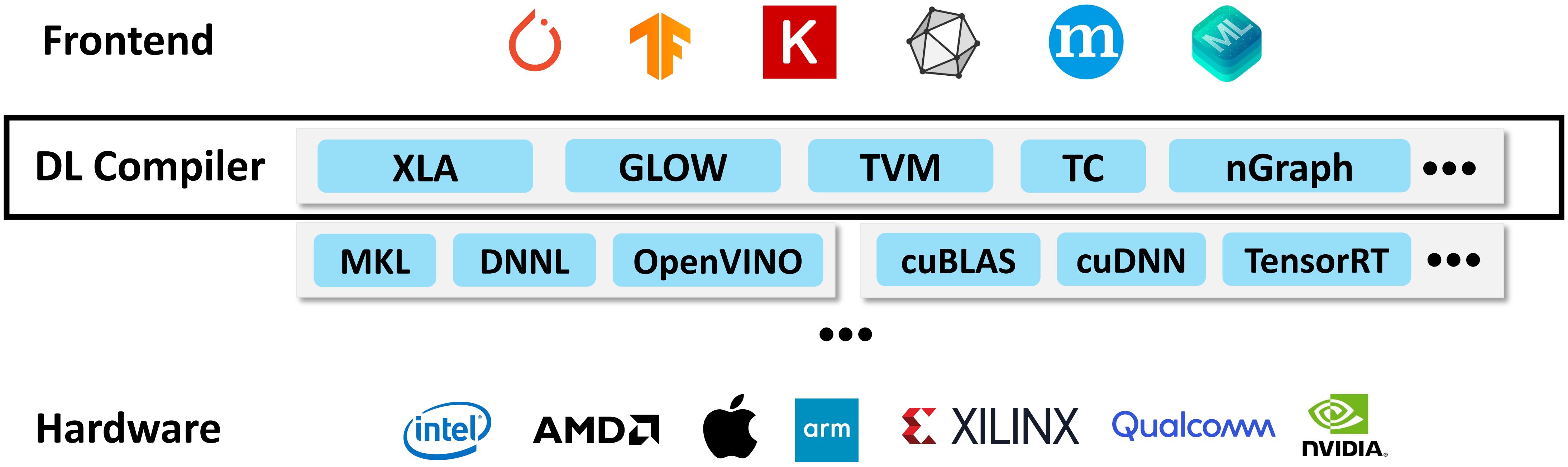
•••

Hardware

Deep Learning (DL) Backend

Backend

- a software library or a runtime framework that takes DL workloads as inputs and generates an optimized low-level target code



Frontend

DL Compiler: XLA | GLOW | TVM | TC | nGraph | •••

MKL | DNNL | OpenVINO

cuBLAS | cuDNN | TensorRT | •••

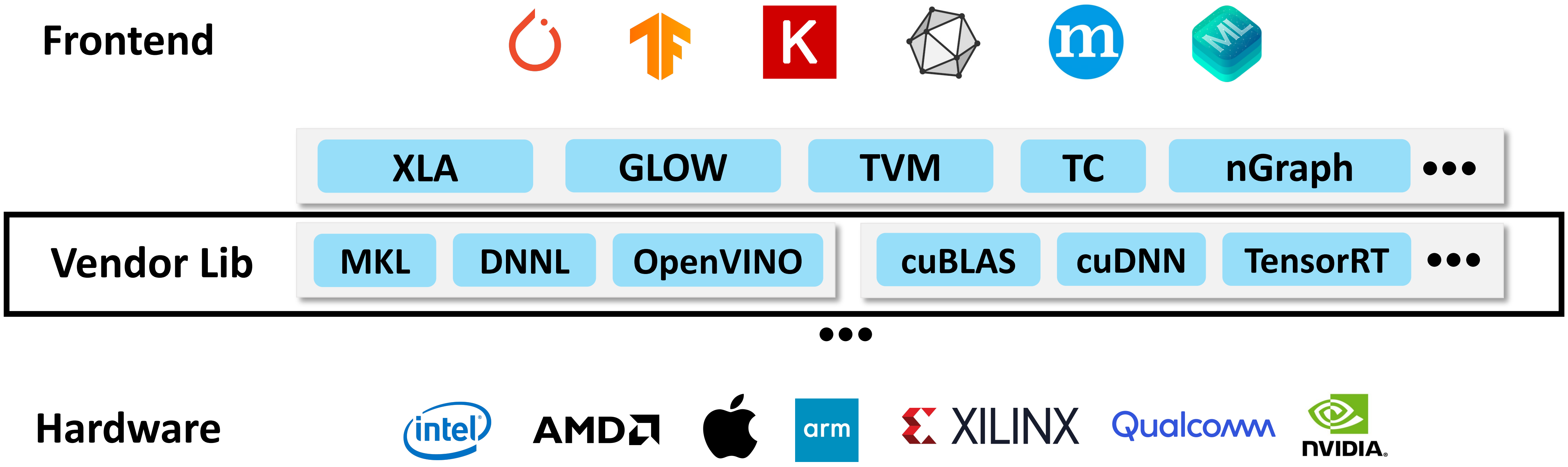
•••

Hardware

Deep Learning (DL) Backend

Backend

- a software library or a runtime framework that takes DL workloads as inputs and generates an optimized low-level target code



Frontend

XLA GLOW TVM TC nGraph •••

Vendor Lib

MKL DNNL OpenVINO

cuBLAS cuDNN TensorRT •••

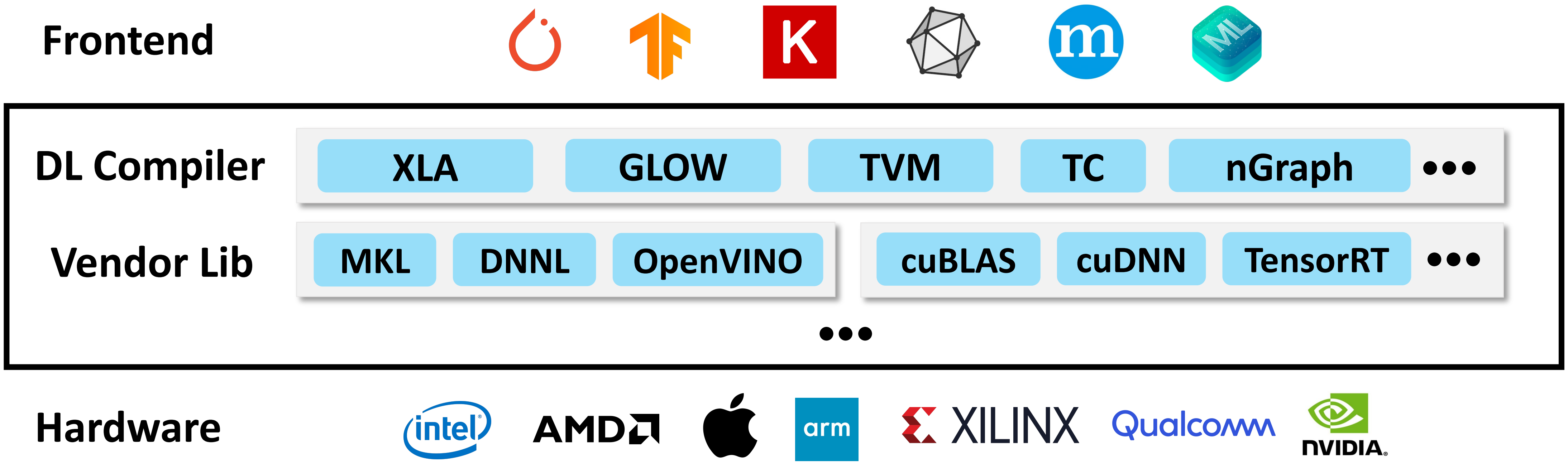
•••

Hardware

Observation: Diversified DL Backends

DL backends are highly diversified and evolving fast

- Each backend has its own coverage (e.g., HW, DL operators) and strength



Frontend

DL Compiler: XLA | GLOW | TVM | TC | nGraph | •••

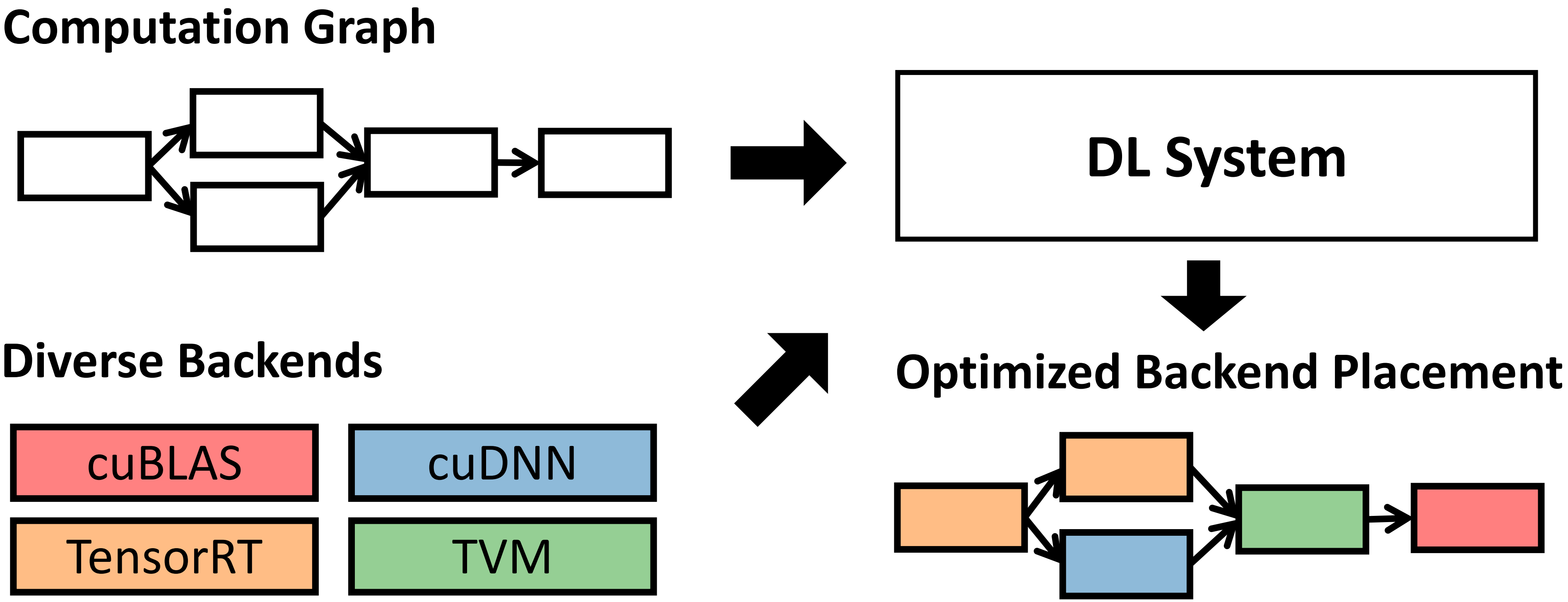
Vendor Lib: MKL | DNNL | OpenVINO | cuBLAS | cuDNN | TensorRT | •••

•••

Hardware

Problem: Backend Integration

Backend Integration = Backend Register + Backend Placement



Computation Graph

DL System

Diverse Backends

cuBLAS

cuDNN

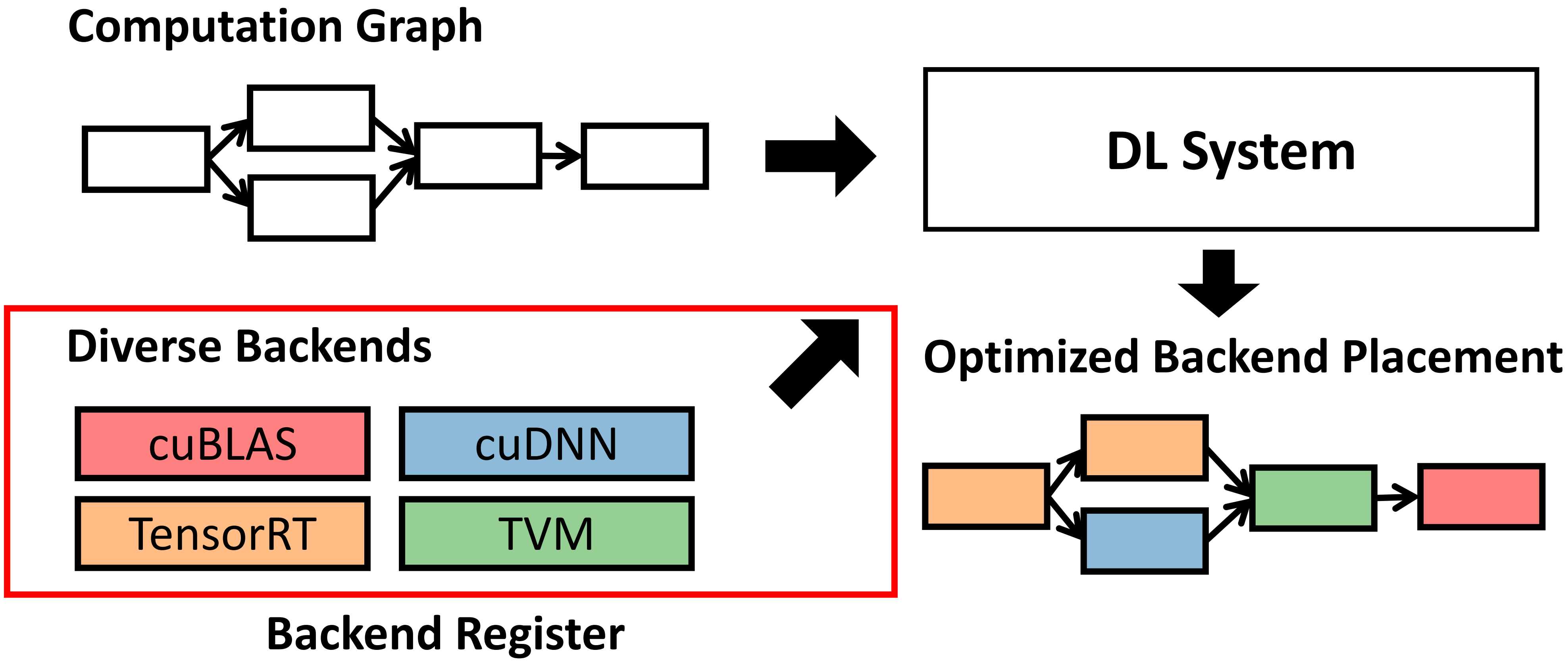
TensorRT

TVM

Optimized Backend Placement

Problem: Backend Integration

Backend Integration = **Backend Register** + Backend Placement



Computation Graph

DL System

Diverse Backends

cuBLAS

cuDNN

TensorRT

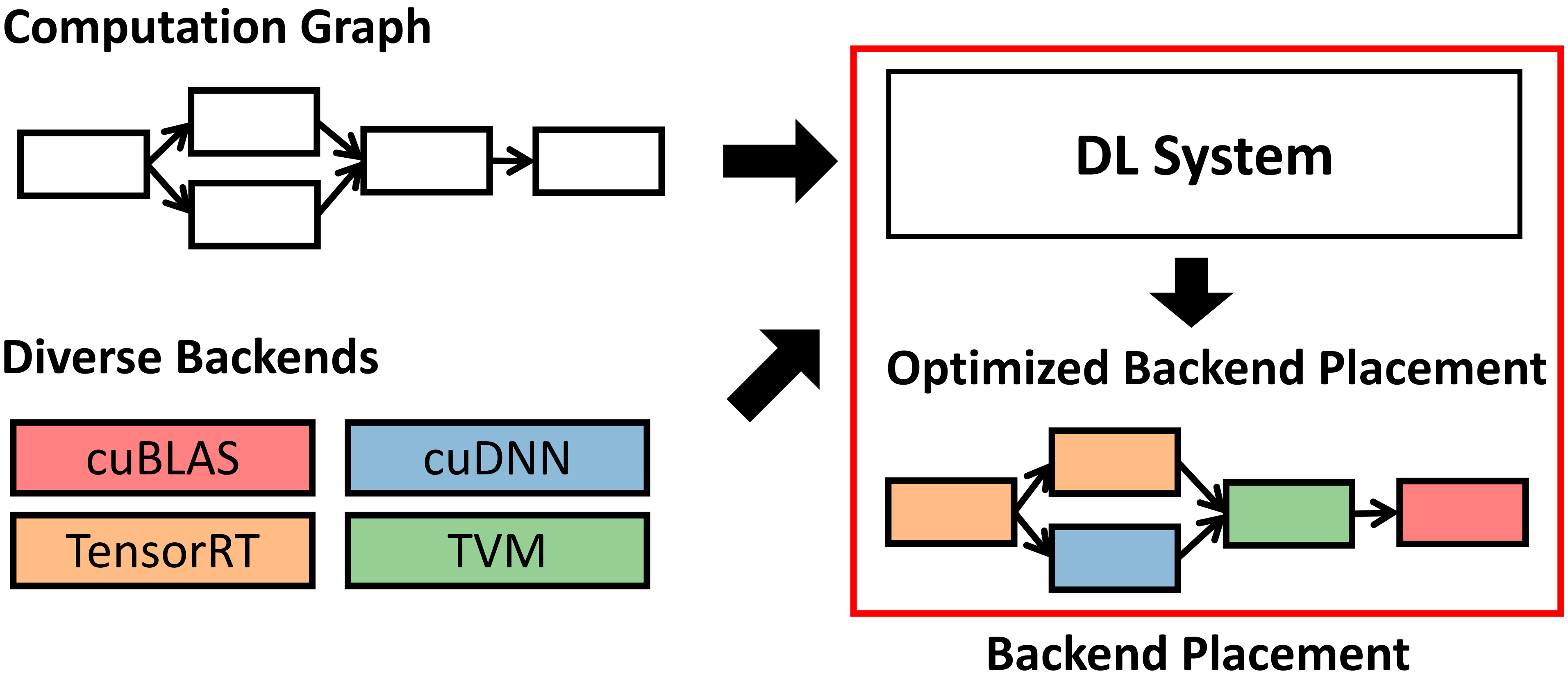
TVM

Optimized Backend Placement

Backend Register

Problem: Backend Integration

Backend Integration = Backend Register + **Backend Placement**



Computation Graph

Diverse Backends

cuBLAS

cuDNN

TensorRT

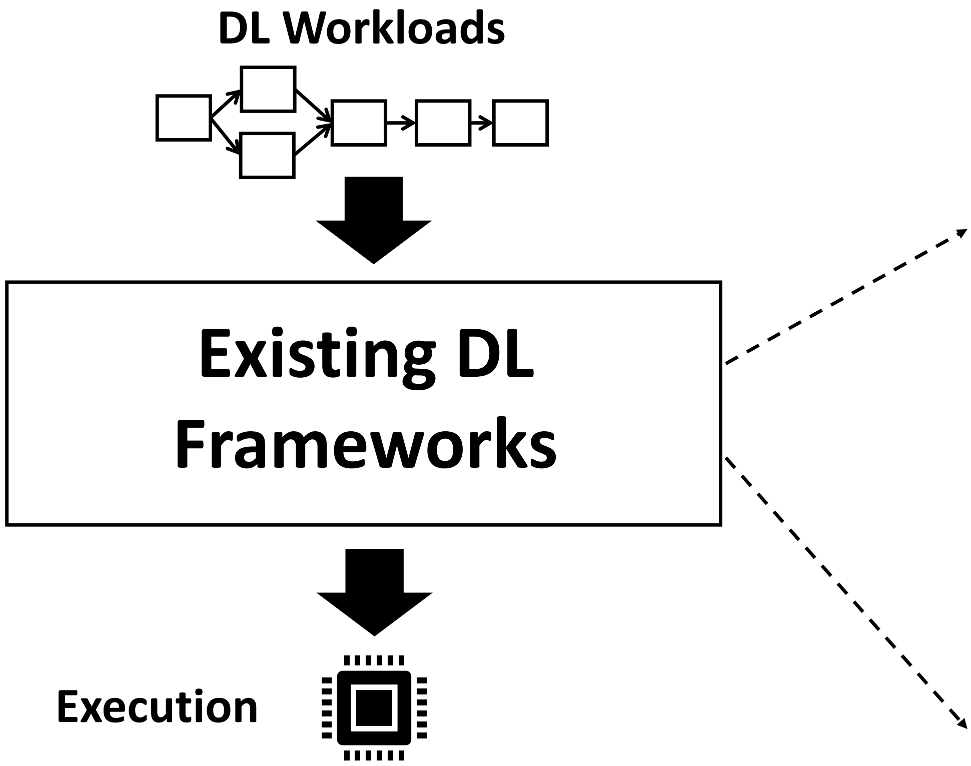
TVM

DL System

Optimized Backend Placement

Backend Placement

Existing Approach: Manual Backend Integration

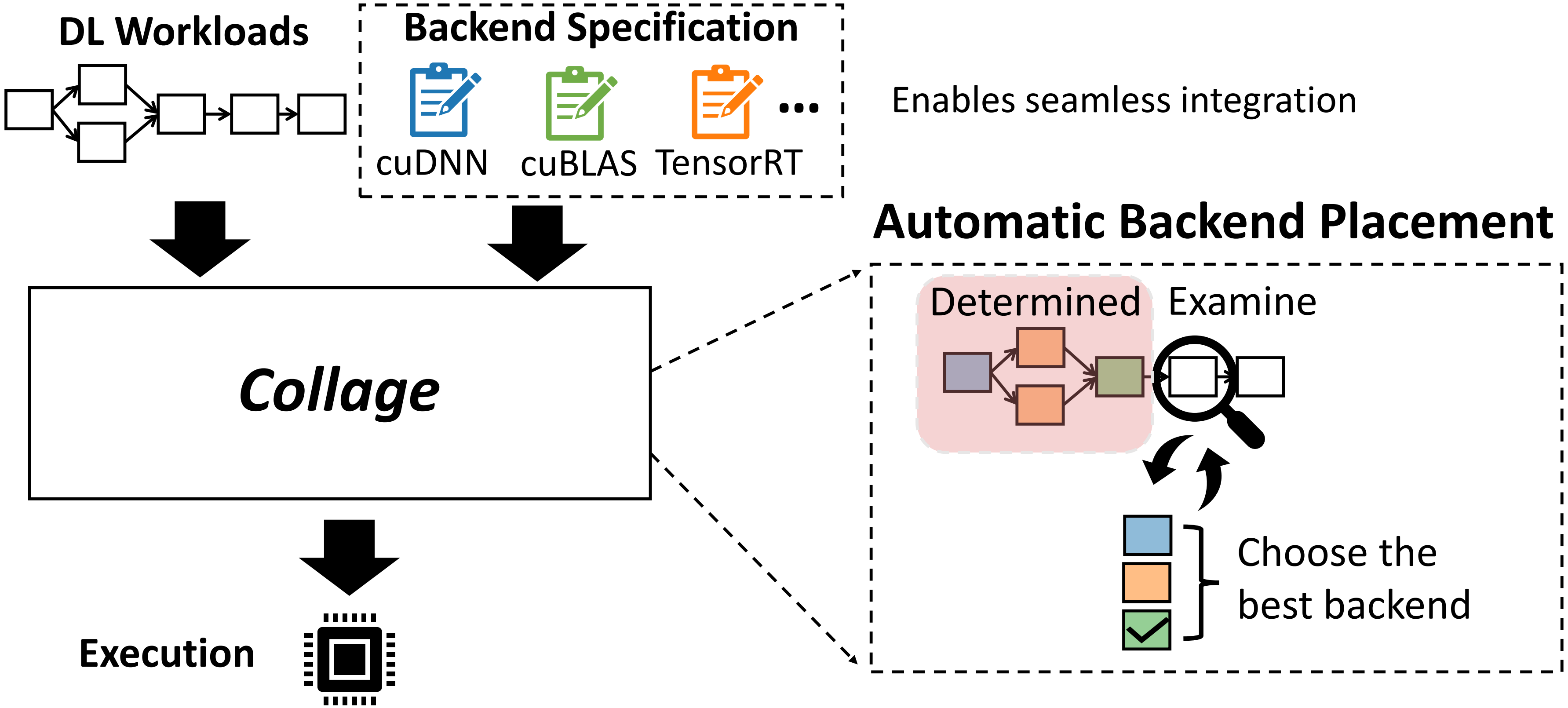


Rule-based Heuristics

```
If op == “conv2d”:
  If cudnn_enabled:
    lower_to_cudnn_kernel
  Else if …
Else if op == “batch_matmul”:
  If cublas_enabled:
    lower_to_cublas_kernel
   ….
```

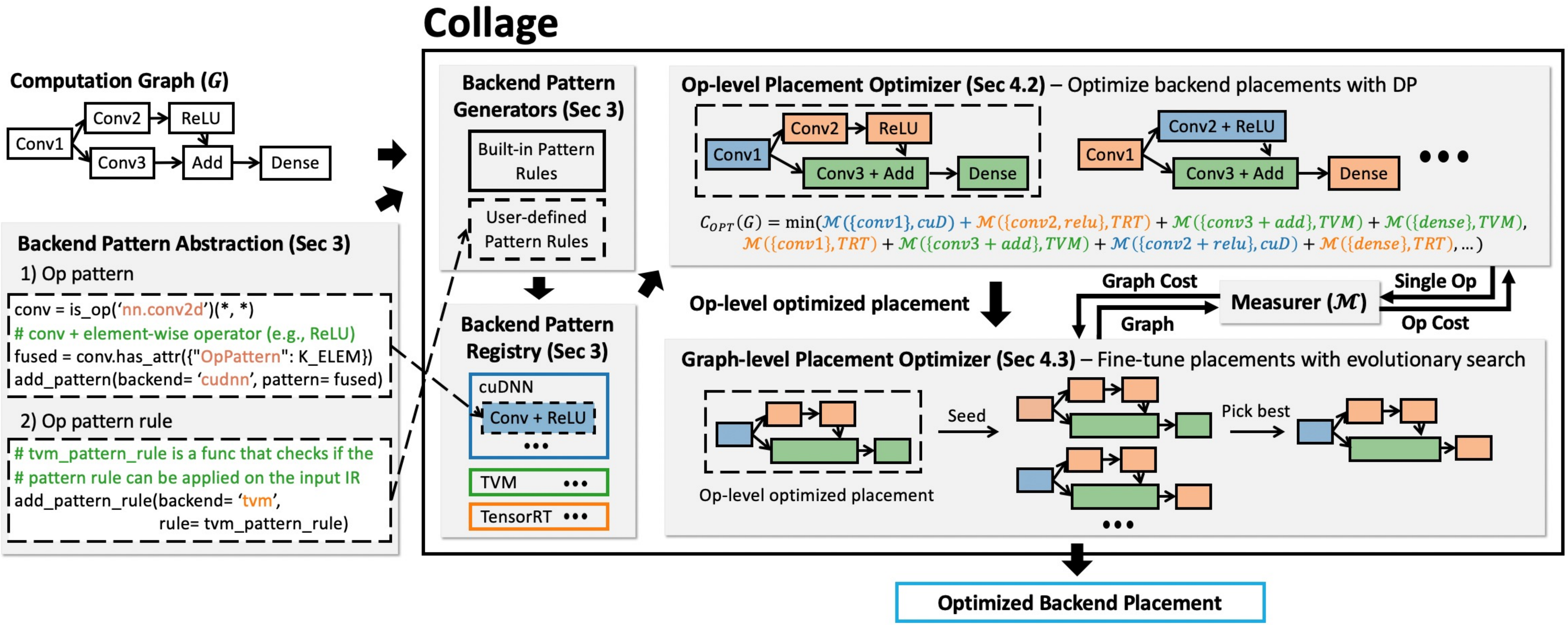
- ○ Heuristics are often sub-optimal and susceptible to be outdated
- ○ Direct code modification to the DL framework is required

Our Approach: Automated Backend Integration



- It eliminates manual efforts to design heuristics and change codes
- It provides fast and stable performance across different models and hardwares

System Overview



Collage

Computation Graph (G)

Backend Pattern Abstraction (Sec 3)

1) Op pattern

```
conv = is_op('nn.conv2d')(*, *)
# conv + element-wise operator (e.g., ReLU)
fused = conv.has_attr({"OpPattern": K_ELEM})
add_pattern(backend= 'cudnn', pattern= fused)
```

2) Op pattern rule

```
# tvm_pattern_rule is a func that checks if the
# pattern rule can be applied on the input IR
add_pattern_rule(backend= 'tvm',
                 rule= tvm_pattern_rule)
```

Backend Pattern Generators (Sec 3)

Built-in Pattern Rules

User-defined Pattern Rules

Backend Pattern Registry (Sec 3)

Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP

$$C_{OPT}(G) = \min(\mathcal{M}(\{conv1\}, cuD) + \mathcal{M}(\{conv2, relu\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{dense\}, TVM), \mathcal{M}(\{conv1\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{conv2 + relu\}, cuD) + \mathcal{M}(\{dense\}, TRT), \ldots)$$

Op-level optimized placement

Graph Cost

Graph

Measurer ($\mathcal{M}$)

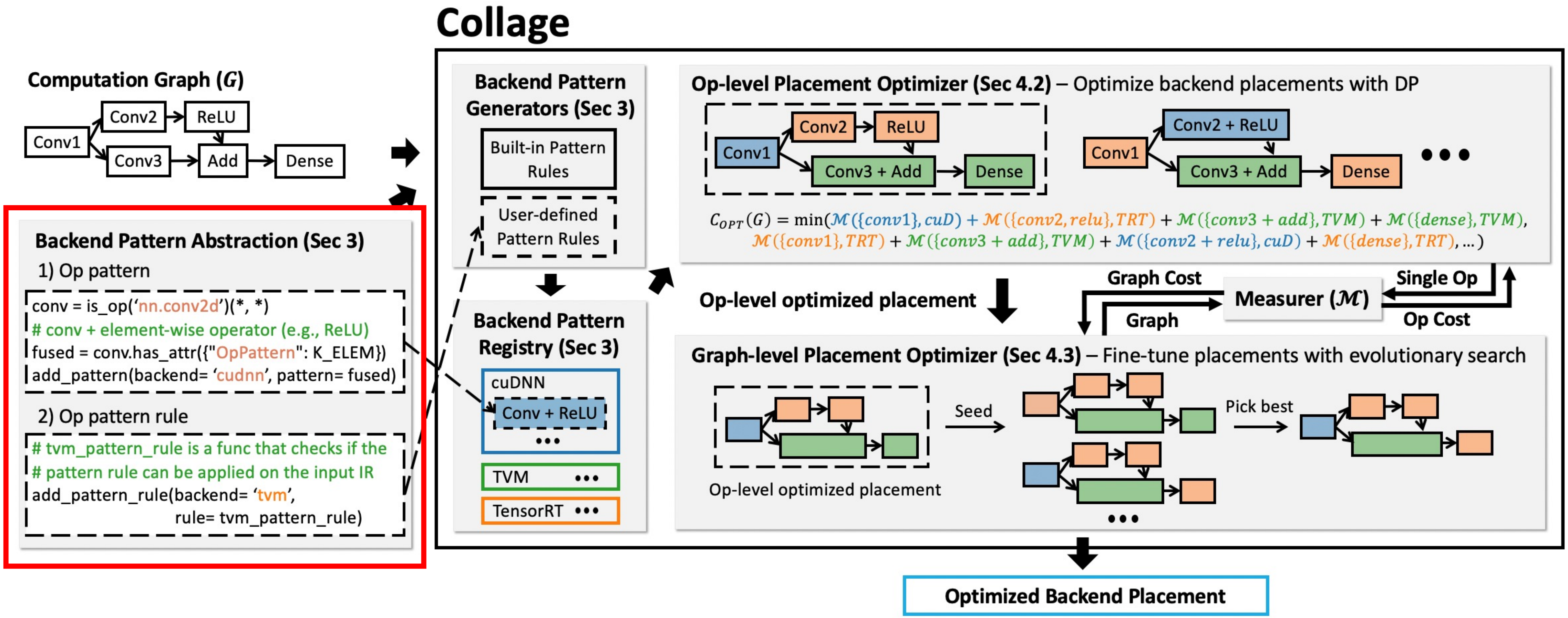
Single Op

Op Cost

Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search

Optimized Backend Placement

Overview

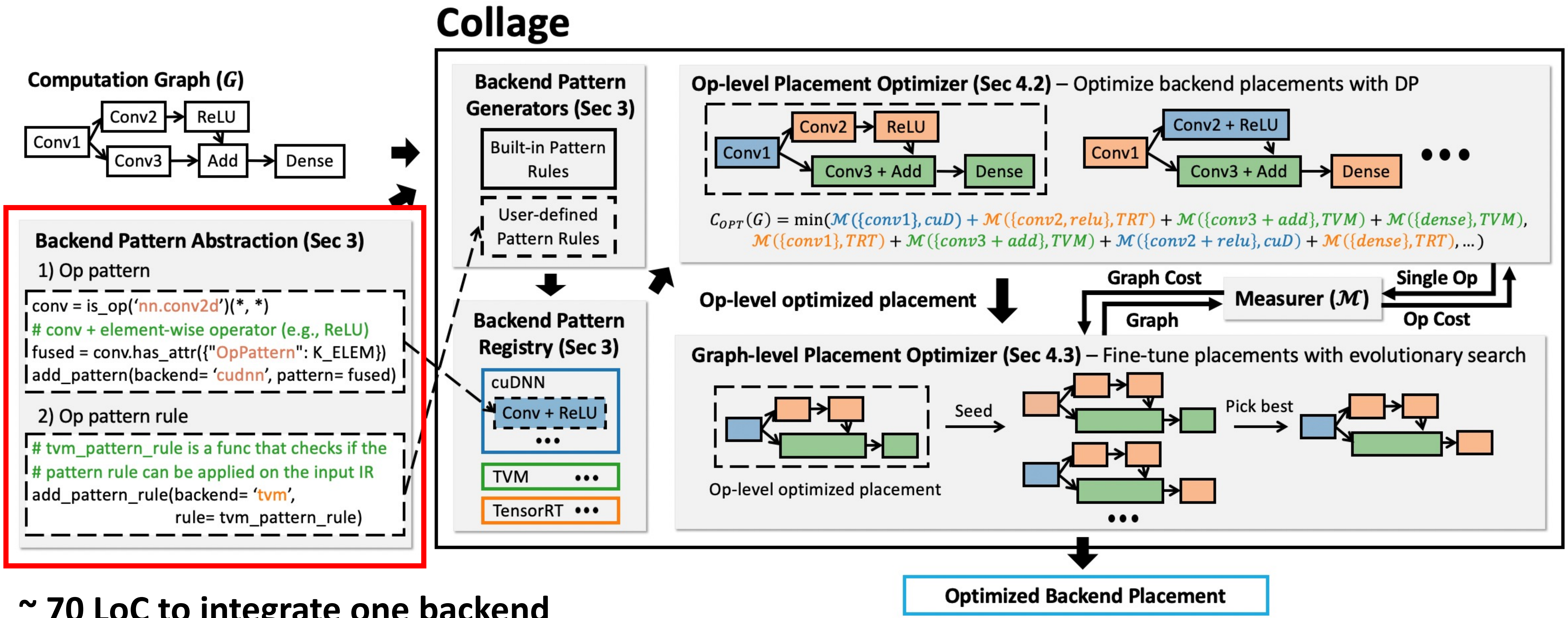

Collage
Computation Graph (G)
Conv1
Conv2
ReLU
Conv3
Add
Dense
Backend Pattern Abstraction (Sec 3)
1) Op pattern
2) Op pattern rule
Backend Pattern Generators (Sec 3)
Built-in Pattern Rules
User-defined Pattern Rules
Backend Pattern Registry (Sec 3)
cuDNN
Conv + ReLU
•••
TVM •••
TensorRT •••
Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP
Conv1
Conv2
ReLU
Conv3 + Add
Dense
Conv1
Conv2 + ReLU
Conv3 + Add
Dense
•••
Op-level optimized placement
Graph Cost
Graph
Measurer ($\mathcal{M}$)
Single Op
Op Cost
Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search
Seed
Pick best
Op-level optimized placement
•••
Optimized Backend Placement

```
conv = is_op('nn.conv2d')(*, *)
# conv + element-wise operator (e.g., ReLU)
fused = conv.has_attr({"OpPattern": K_ELEM})
add_pattern(backend= 'cudnn', pattern= fused)
```

```
# tvm_pattern_rule is a func that checks if the
# pattern rule can be applied on the input IR
add_pattern_rule(backend= 'tvm',
                        rule= tvm_pattern_rule)
```

$$C_{OPT}(G) = \min(\mathcal{M}(\{conv1\}, cuD) + \mathcal{M}(\{conv2, relu\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{dense\}, TVM), \mathcal{M}(\{conv1\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{conv2 + relu\}, cuD) + \mathcal{M}(\{dense\}, TRT), \ldots)$$

Overview



Collage

Computation Graph (G)

Backend Pattern Abstraction (Sec 3)

1) Op pattern

```
conv = is_op('nn.conv2d')(*, *)
# conv + element-wise operator (e.g., ReLU)
fused = conv.has_attr({"OpPattern": K_ELEM})
add_pattern(backend= 'cudnn', pattern= fused)
```

2) Op pattern rule

```
# tvm_pattern_rule is a func that checks if the
# pattern rule can be applied on the input IR
add_pattern_rule(backend= 'tvm',
                 rule= tvm_pattern_rule)
```

Backend Pattern Generators (Sec 3)

Backend Pattern Registry (Sec 3)

Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP

$$C_{OPT}(G) = \min(\mathcal{M}(\{conv1\}, cuD) + \mathcal{M}(\{conv2, relu\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{dense\}, TVM), \mathcal{M}(\{conv1\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{conv2 + relu\}, cuD) + \mathcal{M}(\{dense\}, TRT), \dots)$$

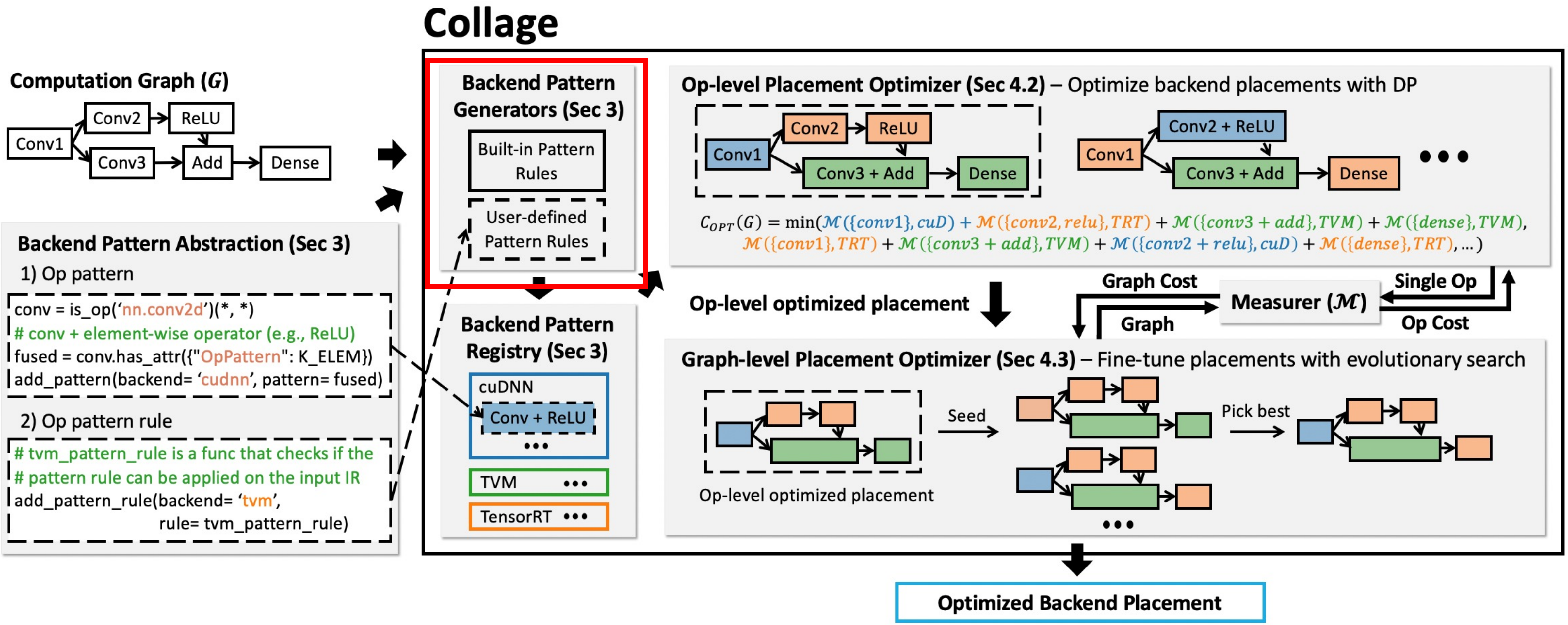
Op-level optimized placement

Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search

Optimized Backend Placement

~ 70 LoC to integrate one backend

Overview


Collage
Computation Graph (G)
Conv1
Conv2
ReLU
Conv3
Add
Dense
Backend Pattern Abstraction (Sec 3)
1) Op pattern
2) Op pattern rule
Backend Pattern Generators (Sec 3)
Built-in Pattern Rules
User-defined Pattern Rules
Backend Pattern Registry (Sec 3)
cuDNN
Conv + ReLU
•••
TVM •••
TensorRT •••
Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP
Conv1
Conv2
ReLU
Conv3 + Add
Dense
Conv1
Conv2 + ReLU
Conv3 + Add
Dense
•••
Op-level optimized placement
Graph Cost
Graph
Measurer (M)
Single Op
Op Cost
Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search
Seed
Pick best
Op-level optimized placement
•••
Optimized Backend Placement

1) Op pattern

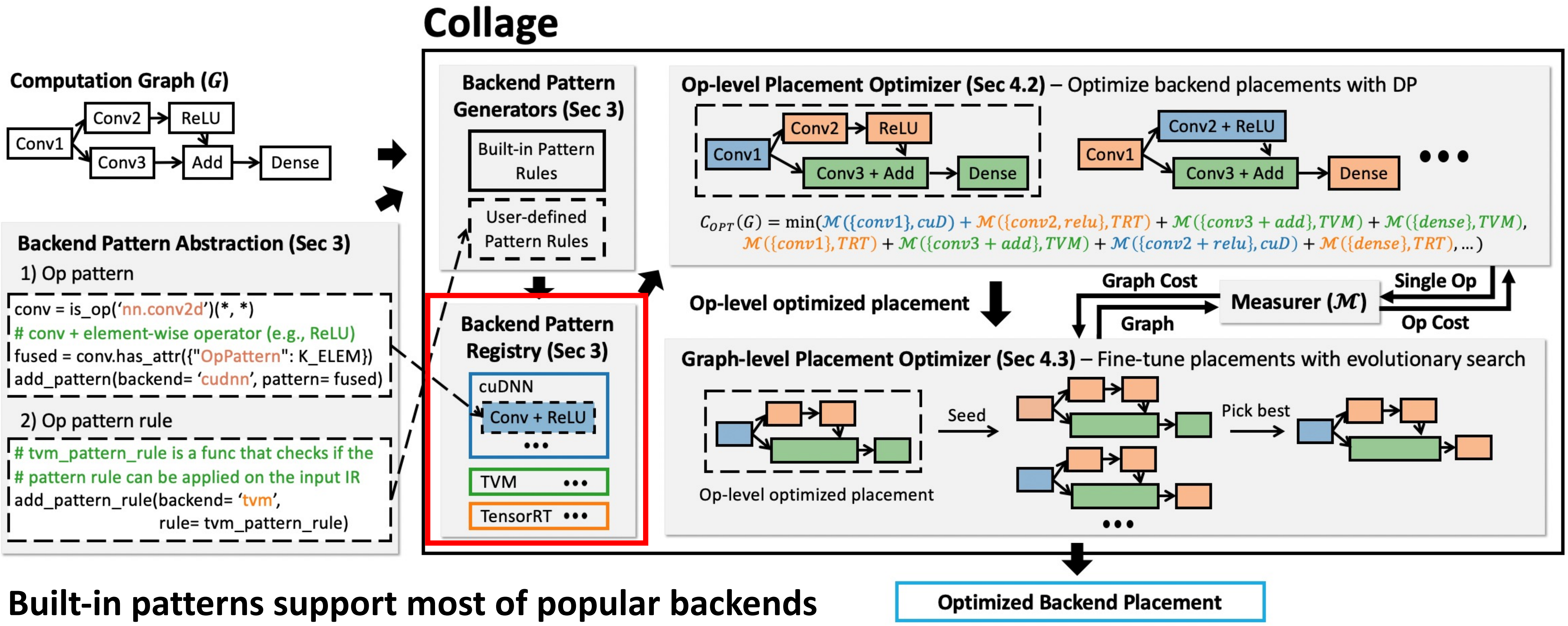
```
conv = is_op('nn.conv2d')(*, *)
# conv + element-wise operator (e.g., ReLU)
fused = conv.has_attr({"OpPattern": K_ELEM})
add_pattern(backend= 'cudnn', pattern= fused)
```

2) Op pattern rule

```
# tvm_pattern_rule is a func that checks if the
# pattern rule can be applied on the input IR
add_pattern_rule(backend= 'tvm',
                 rule= tvm_pattern_rule)
```

$$C_{OPT}(G) = \min(\mathcal{M}(\{conv1\}, cuD) + \mathcal{M}(\{conv2, relu\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{dense\}, TVM),\ \mathcal{M}(\{conv1\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{conv2 + relu\}, cuD) + \mathcal{M}(\{dense\}, TRT), \ldots)$$

Overview


Collage
Computation Graph (G)
Conv1
Conv2
ReLU
Conv3
Add
Dense
Backend Pattern Abstraction (Sec 3)
1) Op pattern
2) Op pattern rule
Backend Pattern Generators (Sec 3)
Built-in Pattern Rules
User-defined Pattern Rules
Backend Pattern Registry (Sec 3)
cuDNN
Conv + ReLU
•••
TVM •••
TensorRT •••
Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP
Conv1
Conv2
ReLU
Conv3 + Add
Dense
Conv1
Conv2 + ReLU
Conv3 + Add
Dense
•••
Op-level optimized placement
Graph Cost
Graph
Measurer ($\mathcal{M}$)
Single Op
Op Cost
Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search
Seed
Pick best
Op-level optimized placement
•••
Optimized Backend Placement

```
conv = is_op('nn.conv2d')(*, *)
# conv + element-wise operator (e.g., ReLU)
fused = conv.has_attr({"OpPattern": K_ELEM})
add_pattern(backend= 'cudnn', pattern= fused)
```

```
# tvm_pattern_rule is a func that checks if the
# pattern rule can be applied on the input IR
add_pattern_rule(backend= 'tvm',
                 rule= tvm_pattern_rule)
```

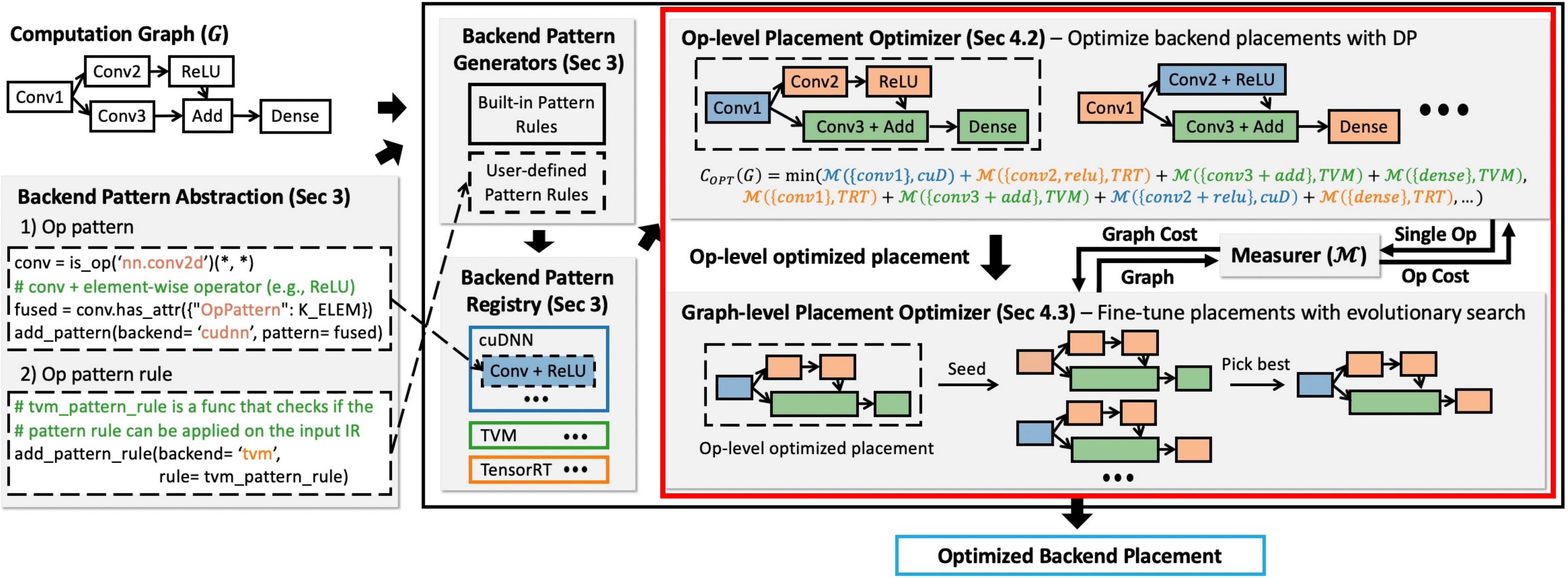
$$C_{OPT}(G) = \min(\mathcal{M}(\{conv1\}, cuD) + \mathcal{M}(\{conv2, relu\}, TRT) + \mathcal{M}(\{conv3+add\}, TVM) + \mathcal{M}(\{dense\}, TVM),\ \mathcal{M}(\{conv1\}, TRT) + \mathcal{M}(\{conv3+add\}, TVM) + \mathcal{M}(\{conv2+relu\}, cuD) + \mathcal{M}(\{dense\}, TRT), \ldots)$$

Built-in patterns support most of popular backends (e.g., cuDNN, cuBLAS, TensorRT, TVM, MKL, etc.)

Overview

Collage

Automated Two-level Optimizer



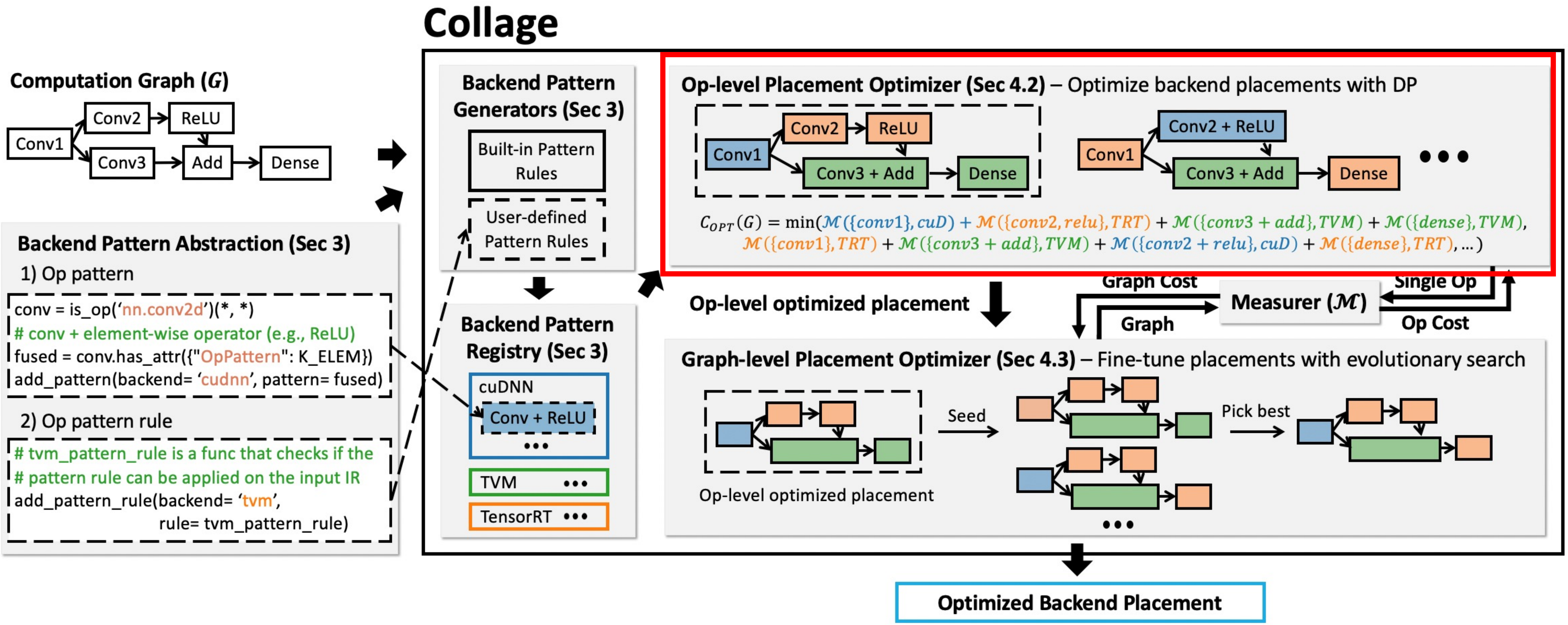
Computation Graph (G)
Conv1
Conv2
ReLU
Conv3
Add
Dense
Backend Pattern Abstraction (Sec 3)
1) Op pattern
2) Op pattern rule
Backend Pattern Generators (Sec 3)
Built-in Pattern Rules
User-defined Pattern Rules
Backend Pattern Registry (Sec 3)
cuDNN
Conv + ReLU
•••
TVM •••
TensorRT •••
Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP
Conv1
Conv2
ReLU
Conv3 + Add
Dense
Conv1
Conv2 + ReLU
Conv3 + Add
Dense
•••
Op-level optimized placement
Graph Cost
Graph
Measurer (M)
Single Op
Op Cost
Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search
Seed
Pick best
Op-level optimized placement
•••
Optimized Backend Placement

```
conv = is_op('nn.conv2d')(*, *)
# conv + element-wise operator (e.g., ReLU)
fused = conv.has_attr({"OpPattern": K_ELEM})
add_pattern(backend= 'cudnn', pattern= fused)
```

```
# tvm_pattern_rule is a func that checks if the
# pattern rule can be applied on the input IR
add_pattern_rule(backend= 'tvm',
                 rule= tvm_pattern_rule)
```

$$C_{OPT}(G) = \min(\mathcal{M}(\{conv1\}, cuD) + \mathcal{M}(\{conv2, relu\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{dense\}, TVM), \mathcal{M}(\{conv1\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{conv2 + relu\}, cuD) + \mathcal{M}(\{dense\}, TRT), \ldots)$$

Overview



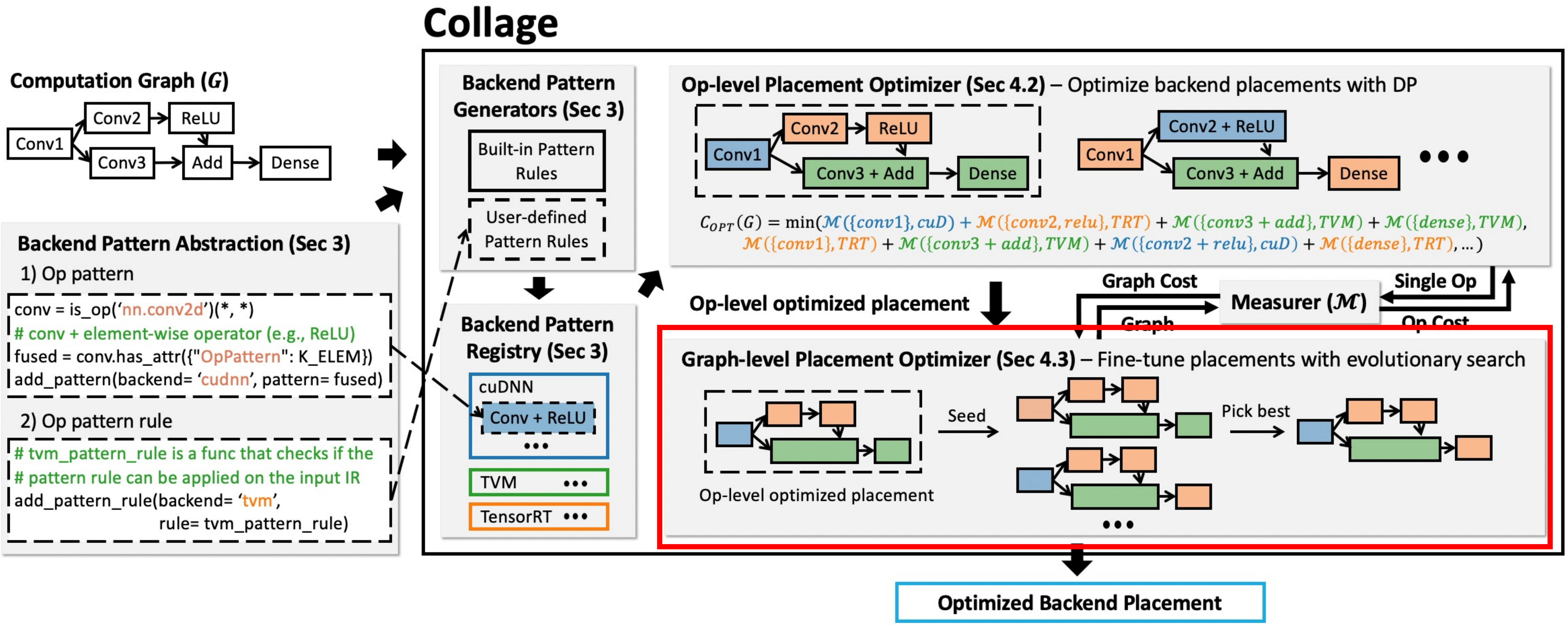
Collage
Computation Graph (*G*)
Conv1
Conv2
ReLU
Conv3
Add
Dense
Backend Pattern Abstraction (Sec 3)
1) Op pattern
2) Op pattern rule
Backend Pattern Generators (Sec 3)
Built-in Pattern Rules
User-defined Pattern Rules
Backend Pattern Registry (Sec 3)
cuDNN
Conv + ReLU
•••
TVM •••
TensorRT •••
Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP
Conv1
Conv2
ReLU
Conv3 + Add
Dense
Conv1
Conv2 + ReLU
Conv3 + Add
Dense
•••
Op-level optimized placement
Graph Cost
Single Op
Measurer (ℳ)
Graph
Op Cost
Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search
Seed
Pick best
Op-level optimized placement
•••
Optimized Backend Placement

```
conv = is_op('nn.conv2d')(*, *)
# conv + element-wise operator (e.g., ReLU)
fused = conv.has_attr({"OpPattern": K_ELEM})
add_pattern(backend= 'cudnn', pattern= fused)
```

```
# tvm_pattern_rule is a func that checks if the
# pattern rule can be applied on the input IR
add_pattern_rule(backend= 'tvm',
                 rule= tvm_pattern_rule)
```

$$C_{OPT}(G) = \min(\mathcal{M}(\{conv1\}, cuD) + \mathcal{M}(\{conv2, relu\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{dense\}, TVM), \mathcal{M}(\{conv1\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{conv2 + relu\}, cuD) + \mathcal{M}(\{dense\}, TRT), \ldots)$$

Overview



Collage

Computation Graph (G)

Conv1 → Conv2 → ReLU → Add → Dense; Conv1 → Conv3 → Add

Backend Pattern Abstraction (Sec 3)

1) Op pattern

```
conv = is_op('nn.conv2d')(*, *)
# conv + element-wise operator (e.g., ReLU)
fused = conv.has_attr({"OpPattern": K_ELEM})
add_pattern(backend= 'cudnn', pattern= fused)
```

2) Op pattern rule

```
# tvm_pattern_rule is a func that checks if the
# pattern rule can be applied on the input IR
add_pattern_rule(backend= 'tvm',
                    rule= tvm_pattern_rule)
```

Backend Pattern Generators (Sec 3)

Built-in Pattern Rules

User-defined Pattern Rules

Backend Pattern Registry (Sec 3)

cuDNN

Conv + ReLU

•••

TVM •••

TensorRT •••

Op-level Placement Optimizer (Sec 4.2) – Optimize backend placements with DP

Conv1, Conv2, ReLU, Conv3 + Add, Dense

Conv1, Conv2 + ReLU, Conv3 + Add, Dense

•••

$$C_{OPT}(G) = \min(\mathcal{M}(\{conv1\}, cuD) + \mathcal{M}(\{conv2, relu\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{dense\}, TVM), \mathcal{M}(\{conv1\}, TRT) + \mathcal{M}(\{conv3 + add\}, TVM) + \mathcal{M}(\{conv2 + relu\}, cuD) + \mathcal{M}(\{dense\}, TRT), \ldots)$$

Op-level optimized placement

Graph Cost

Measurer ($\mathcal{M}$)

Single Op

Graph

Op Cost

Graph-level Placement Optimizer (Sec 4.3) – Fine-tune placements with evolutionary search

Op-level optimized placement

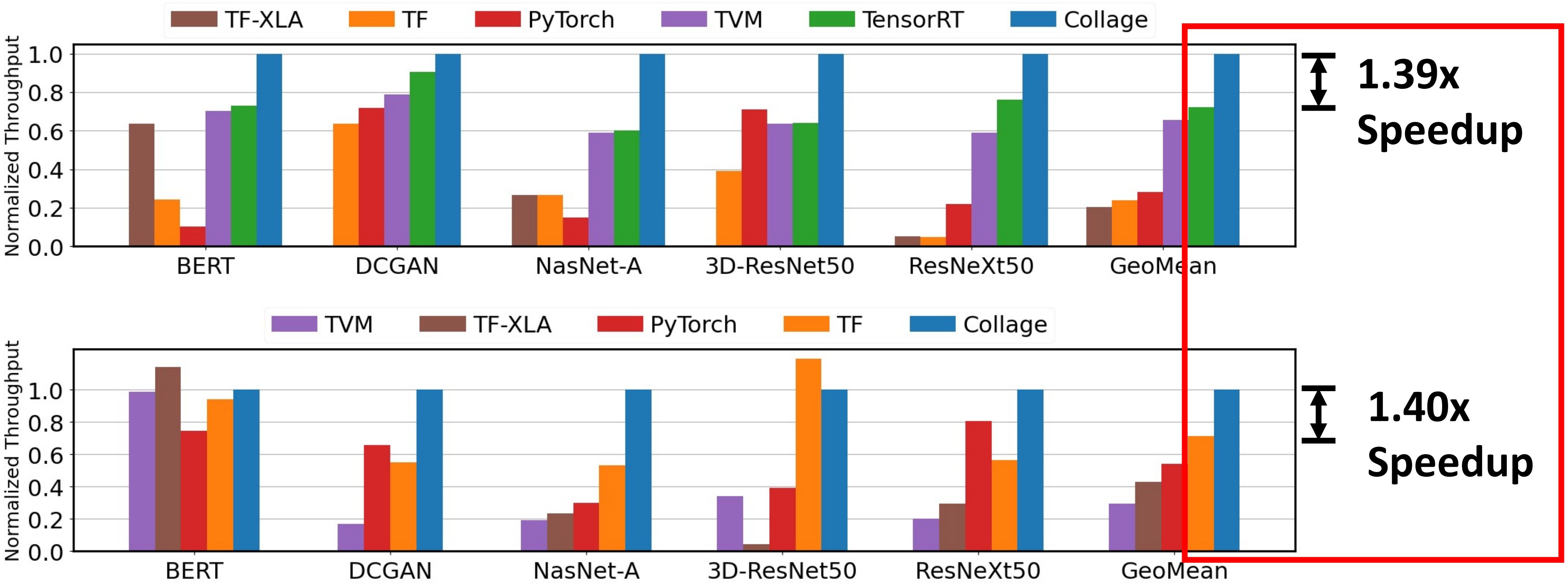
Seed

Pick best

•••

Optimized Backend Placement

End-to-end Evaluation: NVIDIA V100, Intel Xeon

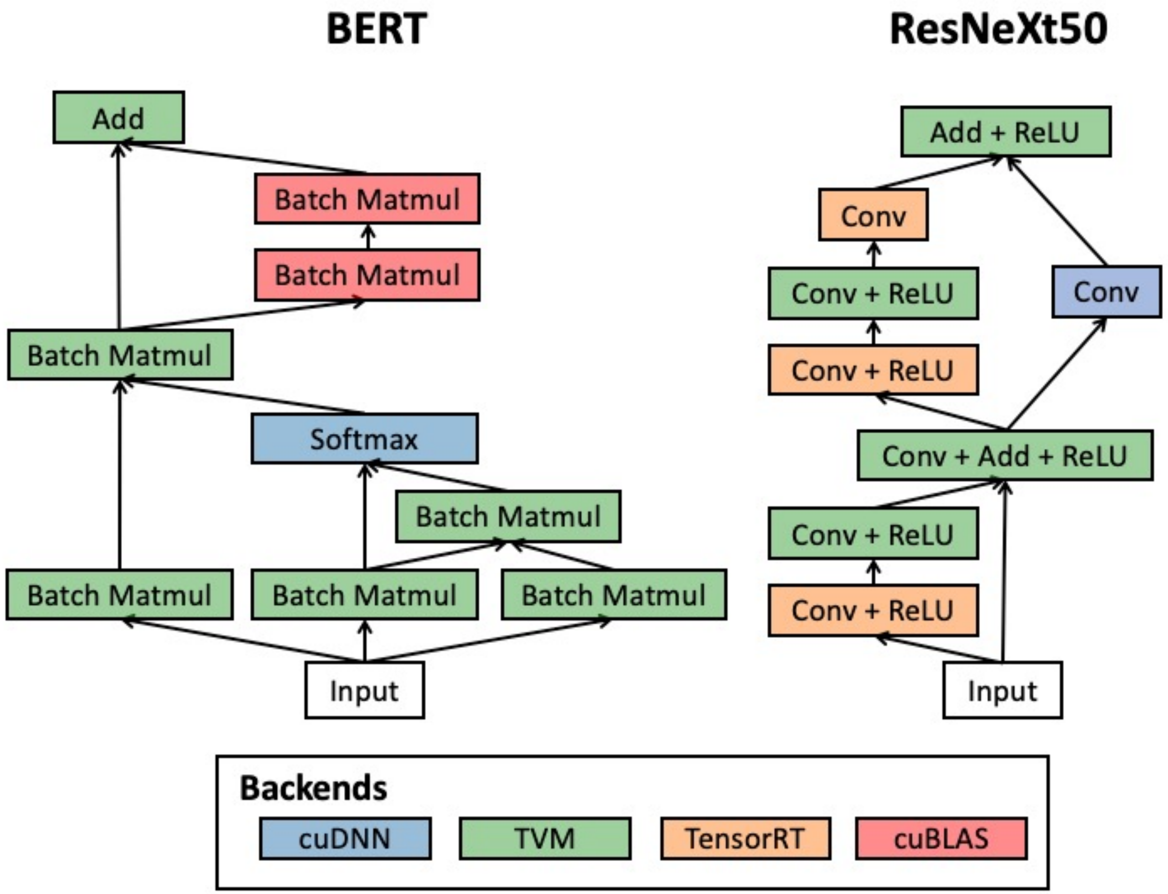


1.39x Speedup

1.40x Speedup

- Stable performance across different networks and hardwares

Optimized Backend Placement



- *Collage* leverages unique strength of each backend
- Collage maps same type of operators to different backends based on the performance landscape
- *Collage* employs diverse operator fusion patterns

References

Arxiv Paper: https://arxiv.org/abs/2111.00655

Code: https://github.com/cmu-catalyst/collage