

---

# **AMD MIGraphX Documentation**

***Release 2.9.0***

**Advanced Micro Devices, Inc.**

**May 18, 2024**



# CONTENTS

<b>1</b>	<b>Contributing to this documentation</b>	<b>3</b>
<b>2</b>	<b>Index and search</b>	<b>5</b>
<b>3</b>	<b>What is MIGraphX?</b>	<b>7</b>
<b>4</b>	<b>MIGraphX driver</b>	<b>9</b>
4.1	Commands . . . . .	9
4.2	Options . . . . .	9
4.3	Usage . . . . .	10
4.4	Driver options . . . . .	20
<b>5</b>	<b>Contributing to MIGraphX</b>	<b>23</b>
5.1	Performing basic operations . . . . .	23
5.2	Sample programs . . . . .	25
5.3	Data types . . . . .	26
5.4	Operators . . . . .	38
5.5	Program . . . . .	100
5.6	Targets . . . . .	104
5.7	Quantization . . . . .	106
5.8	Passes . . . . .	107
5.9	Matchers . . . . .	111
5.10	Tools . . . . .	113
5.11	Environment Variables . . . . .	115
<b>6</b>	<b>C++ API reference</b>	<b>121</b>
6.1	shape . . . . .	121
6.2	argument . . . . .	124
6.3	target . . . . .	124
6.4	program . . . . .	125
6.5	quantize . . . . .	127
6.6	parse_onnx . . . . .	128
6.7	load . . . . .	129
6.8	save . . . . .	129
<b>7</b>	<b>Python API reference</b>	<b>131</b>
7.1	shape . . . . .	131
7.2	dynamic_dimension . . . . .	132
7.3	argument . . . . .	133
7.4	target . . . . .	134
7.5	module . . . . .	134

7.6	program . . . . .	135
7.7	op . . . . .	136
7.8	parse_onnx . . . . .	136
7.9	parse_tf . . . . .	137
7.10	load . . . . .	137
7.11	save . . . . .	138
<b>8</b>	<b>License</b>	<b>139</b>
	<b>Python Module Index</b>	<b>141</b>
	<b>Index</b>	<b>143</b>

Welcome to the MIGraphX docs home page! To learn more, see [What is MIGraphX?](#).

Our documentation is structured as follows:

API reference

- [C++ API reference](#)
- [Python API reference](#)

Command-line tool

- [MIGraphX driver](#)

Tutorial

- [Contributing to MIGraphX](#)



## CONTRIBUTING TO THIS DOCUMENTATION

We welcome collaboration! If you'd like to contribute to our documentation, you can find instructions on our [Contribute to ROCm docs](#) page. Known issues are listed on [GitHub](#). Licensing information for all ROCm components is listed on our [Licensing](#) page.





## INDEX AND SEARCH

- `genindex`
- `search`



## WHAT IS MIGRAPHX?

AMD MIGraphX is AMD's graph inference engine that accelerates machine learning model inference. The optimized execution engine is useful for deep learning neural networks.

You can utilize MIGraphX functions using *C++ APIs*, *Python APIs*, and the *command-line tool migraphx-driver*.



## MIGRAPHX DRIVER

The MIGraphX driver is a command-line tool that allows you to utilize many of the MIGraphX core functions without having to write a program. It can read, compile, run, and test the performance of a model with randomized data.

It is installed by default when you install MIGraphX. You can find it in `/opt/rocm/bin/migraphx-driver` or in `AMDMIGraphX/build/bin/migraphx-driver` after building the source code.

### 4.1 Commands

The table below summarizes the MIGraphX driver commands.

Table 4.1: commands

Command	Description
op	Prints all operators of MIGraphX when followed by the option <code>--list</code> or <code>-l</code>
params	Prints the input and output parameter shapes
run	Compiles, allocates parameters, evaluates, and prints input graph
read	Loads and prints input graph
compile	Compiles and prints input graph
verify	Runs reference and GPU implementations and checks outputs for consistency
perf	Compiles and runs input graph followed by printing the performance report

### 4.2 Options

The table below summarizes the various options to be used with the *MIGraphX driver commands*. To learn which options can be used with which commands, see the *MIGraphX driver options*.

Table 4.2: commands

Option	Description
<code>-help</code>   <code>-h</code>	Prints help section.
<code>-model &lt;resnet50 inceptionv3 alexnet&gt;</code>	Loads one of the three default models.
<code>-onnx</code>	Loads the file as an ONNX graph.
<code>-tf</code>	Loads the file as a tensorflow graph.
<code>-migraphx</code>	Loads the file as a migraphx graph.
<code>-migraphx-json</code>	Loads the file as a migraphx JSON graph.
<code>-batch</code>	Sets batch size for a static model. Sets the batch size at runtime for a dynamic batch model.
<code>-nhwc</code>	Treats tensorflow format as nhwc.

continues on next page

Table 4.2 – continued from previous page

<code>-nchw</code>	Treats tensorflow format as nchw.
<code>-skip-unknown-operators</code>	Skips unknown operators when parsing and continues to parse.
<code>-trim   -t</code>	Trims instructions from the end.
<code>-optimize   -O</code>	Optimizes read
<code>-graphviz   -g</code>	Prints a graphviz representation
<code>-brief</code>	Makes the output brief
<code>-cpp</code>	Prints the program in .cpp format
<code>-json</code>	Prints the program in .json format
<code>-text</code>	Prints the program in .txt format
<code>-binary</code>	Prints the program in binary format
<code>-output   -o</code>	Writes output in a file
<code>-fill0</code>	Fills parameter with 0s
<code>-fill1</code>	Fills parameter with 1s
<code>-input-dim</code>	Sets static dimensions of a parameter
<code>-dyn-input-dim</code>	Sets dynamic dimensions of a parameter
<code>-default-dyn-dim</code>	Sets default dynamic dimension
<code>-gpu</code>	Compiles on the GPU
<code>-cpu</code>	Compiles on the CPU
<code>-ref</code>	Compiles on the reference implementation
<code>-enable-offload-copy</code>	Enables implicit offload copying
<code>-disable-fast-math</code>	Disables fast math optimization
<code>-exhaustive-tune</code>	Enables exhaustive search to find the fastest kernel
<code>-fp16</code>	Quantizes for fp16
<code>-int8</code>	Quantizes for int8
<code>-fp8</code>	Quantize for Float8E4M3FNUZ type
<code>-rms-tol</code>	Sets tolerance for the RMS error (Default: 0.001)
<code>-atol</code>	Sets tolerance for elementwise absolute difference (Default: 0.001)
<code>-rtol</code>	Sets tolerance for elementwise relative difference (Default: 0.001)
<code>-per-instruction   -i</code>	Verifies each instruction
<code>-reduce   -r</code>	Reduces program and verifies
<code>-iterations   -n</code>	Sets the number of iterations to run for perf report
<code>-list   -l</code>	Lists all the MIGraphX operators

## 4.3 Usage

This section demonstrates the usage of MIGraphX driver tool with some commonly used options. Note that these examples use a simple MNIST ConvNet as the input graph for demonstration purposes as models of higher complexity generate considerably larger outputs in most cases.

### 4.3.1 Option: op

```
$ /opt/rocm/bin/migraphx-driver op -list
```

View Output

```
@literal
@param
@return
abs
```

(continues on next page)

(continued from previous page)

```
acos
acosh
add
argmax
argmin
as_shape
asin
asinh
atan
atanh
batch_norm_inference
broadcast
capture
ceil
check_context::migraphx::gpu::context
clip
concat
contiguous
convert
convolution
cos
cosh
deconvolution
div
dot
elu
equal
erf
exp
flatten
floor
gather
gpu::abs
gpu::acos
gpu::acosh
gpu::add
gpu::add_clip
gpu::add_gelu
gpu::add_gelu_new
gpu::add_relu
gpu::add_tanh
gpu::argmax
gpu::argmin
gpu::asin
gpu::asinh
gpu::atan
gpu::atanh
gpu::batch_norm_inference
gpu::ceil
gpu::clip
gpu::concat
gpu::contiguous
```

(continues on next page)

(continued from previous page)

```
gpu::conv_bias
gpu::conv_bias_relu
gpu::convert
gpu::convolution
gpu::cos
gpu::cosh
gpu::deconv
gpu::div
gpu::elu
gpu::equal
gpu::erf
gpu::exp
gpu::floor
gpu::gather
gpu::gelu
gpu::gelu_new
gpu::gemm
gpu::greater
gpu::layernorm
gpu::leaky_relu
gpu::less
gpu::log
gpu::logsoftmax
gpu::lrn
gpu::max
gpu::min
gpu::mul
gpu::mul_add
gpu::mul_add_relu
gpu::pad
gpu::pooling
gpu::pow
gpu::prelu
gpu::quant_convolution
gpu::quant_gemm
gpu::recip
gpu::record_event
gpu::reduce_max
gpu::reduce_mean
gpu::reduce_min
gpu::reduce_prod
gpu::reduce_sum
gpu::relu
gpu::rnn_var_sl_last_output
gpu::rnn_var_sl_shift_output
gpu::rnn_var_sl_shift_sequence
gpu::round
gpu::rsqrt
gpu::set_stream
gpu::sigmoid
gpu::sign
gpu::sin
```

(continues on next page)



(continued from previous page)

```
gpu::sinh
gpu::softmax
gpu::sqdiff
gpu::sqrt
gpu::sub
gpu::tan
gpu::tanh
gpu::triadd
gpu::triadd_clip
gpu::triadd_relu
gpu::triadd_sigmoid
gpu::triadd_tanh
gpu::wait_event
greater
gru
hip::allocate
hip::copy
hip::copy_from_gpu
hip::copy_to_gpu
hip::hip_allocate_memory
hip::hip_copy_literal
identity
im2col
leaky_relu
less
load
log
logsoftmax
lrn
lstm
max
min
mul
multibroadcast
neg
outline
pad
pooling
pow
prelu
quant_convolution
quant_dot
recip
reduce_max
reduce_mean
reduce_min
reduce_prod
reduce_sum
ref::batch_norm_inference
ref::convolution
ref::deconvolution
ref::dot
```

(continues on next page)

(continued from previous page)

```
ref::elu
ref::im2col
ref::leaky_relu
ref::logsoftmax
ref::lrn
ref::op
ref::pad
ref::pooling_average
ref::pooling_max
ref::quant_convolution
ref::rnn_var_sl_last_output
ref::softmax
relu
reshape
rnn
rnn_last_cell_output
rnn_last_hs_output
rnn_var_sl_last_output
rnn_var_sl_shift_output
rnn_var_sl_shift_sequence
round
rsqrt
scalar
sigmoid
sign
sin
sinh
slice
softmax
sqdiff
sqrt
squeeze
sub
tan
tanh
transpose
undefined
unknown:
unsqueeze
```

### 4.3.2 Option: params

```
$ /opt/rocm/bin/migraphx-driver params simple_graph.pb
```

View Output

```
Reading: simple_graph.pb
x: float_type, {1, 28, 28}, {784, 28, 1}
```

### 4.3.3 Option: run (ONNX file input)

```
$ /opt/rocm/bin/migraphx-driver run -onnx simple_graph.onnx
```

View Output

```
Compiling ...
Reading: simple_graph.onnx
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
->{256}, {1}
@2 = hip::hip_copy_literal[id=@literal:1] -> float_type, {784, 128}, {128, 1}
x:0 = @param:x:0 -> float_type, {1, 28, 28}, {784, 28, 1}
@3 = reshape[dims={-1, 784}](x:0) -> float_type, {1, 784}, {784, 1}
@4 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}
@5 = gpu::gemm[alpha=1,beta=0](@3,@2,@4) -> float_type, {1, 128}, {128, 1}
@6 = hip::hip_copy_literal[id=@literal:0] -> float_type, {128}, {1}
@7 = hip::hip_copy_literal[id=@literal:2] -> float_type, {10}, {1}
@8 = hip::hip_copy_literal[id=@literal:3] -> float_type, {128, 10}, {10, 1}
@9 = multibroadcast[output_lens={1, 128}](@6) -> float_type, {1, 128}, {0, 1}
@10 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}
@11 = gpu::add_relu(@5,@9,@10) -> float_type, {1, 128}, {128, 1}
@12 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}
@13 = gpu::gemm[alpha=1,beta=0](@11,@8,@12) -> float_type, {1, 10}, {10, 1}
@14 = multibroadcast[output_lens={1, 10}](@7) -> float_type, {1, 10}, {0, 1}
@15 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}
@16 = gpu::add(@13,@14,@15) -> float_type, {1, 10}, {10, 1}
#output_0 = @param:#output_0 -> float_type, {1, 10}, {10, 1}
@17 = gpu::softmax[axis=1](@16,#output_0) -> float_type, {1, 10}, {10, 1}
@18 = @return(@17)

Allocating params ...
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
->{256}, {1}
@2 = hip::hip_copy_literal[id=@literal:1] -> float_type, {784, 128}, {128, 1}
x:0 = @param:x:0 -> float_type, {1, 28, 28}, {784, 28, 1}
@3 = reshape[dims={-1, 784}](x:0) -> float_type, {1, 784}, {784, 1}
@4 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}
@5 = gpu::gemm[alpha=1,beta=0](@3,@2,@4) -> float_type, {1, 128}, {128, 1}
@6 = hip::hip_copy_literal[id=@literal:0] -> float_type, {128}, {1}
@7 = hip::hip_copy_literal[id=@literal:2] -> float_type, {10}, {1}
@8 = hip::hip_copy_literal[id=@literal:3] -> float_type, {128, 10}, {10, 1}
@9 = multibroadcast[output_lens={1, 128}](@6) -> float_type, {1, 128}, {0, 1}
@10 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}
@11 = gpu::add_relu(@5,@9,@10) -> float_type, {1, 128}, {128, 1}
@12 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}
@13 = gpu::gemm[alpha=1,beta=0](@11,@8,@12) -> float_type, {1, 10}, {10, 1}
@14 = multibroadcast[output_lens={1, 10}](@7) -> float_type, {1, 10}, {0, 1}
@15 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}
@16 = gpu::add(@13,@14,@15) -> float_type, {1, 10}, {10, 1}
#output_0 = @param:#output_0 -> float_type, {1, 10}, {10, 1}
@17 = gpu::softmax[axis=1](@16,#output_0) -> float_type, {1, 10}, {10, 1}
@18 = @return(@17)
```

### 4.3.4 Option: read

```
$ /opt/rocm/bin/migraphx-driver read simple_graph.pb
```

View Output

```
Reading: simple_graph.pb
@0 = @literal{0.0136018, -0.0839988, 0.0375392, 0.0613085, -0.125795, 0.176185, 0.
→0761055, 0.0093384, -0.110057, -0.170587} -> float_type, {10}, {1}
@1 = @literal{ ... } -> float_type, {128, 10}, {10, 1}
@2 = @literal{ ... } -> float_type, {128}, {1}
@3 = @literal{ ... } -> float_type, {784, 128}, {128, 1}
@4 = @literal{-1, 784} -> int32_type, {2}, {1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@5 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@6 = identity(@3) -> float_type, {784, 128}, {128, 1}
@7 = dot[alpha=1,beta=1](@5,@6) -> float_type, {1, 128}, {128, 1}
@8 = identity(@2) -> float_type, {128}, {1}
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}
@10 = add(@7,@9) -> float_type, {1, 128}, {128, 1}
@11 = relu(@10) -> float_type, {1, 128}, {128, 1}
@12 = identity(@1) -> float_type, {128, 10}, {10, 1}
@13 = dot[alpha=1,beta=1](@11,@12) -> float_type, {1, 10}, {10, 1}
@14 = identity(@0) -> float_type, {10}, {1}
@15 = broadcast[axis=1,dims={1, 10}](@14) -> float_type, {1, 10}, {0, 1}
@16 = add(@13,@15) -> float_type, {1, 10}, {10, 1}
@17 = softmax[axis=1](@16) -> float_type, {1, 10}, {10, 1}
@18 = identity(@17) -> float_type, {1, 10}, {10, 1}
```

### 4.3.5 Option: compile (on GPU, quantized for fp16)

```
$ /opt/rocm/bin/migraphx-driver compile -gpu -fp16 simple_graph.pb
```

View Output

```
Compiling ...
Reading: simple_graph.pb
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {456}, {1},id=scratch] -> float_type,
→{456}, {1}
@2 = hip::hip_copy_literal[id=@literal:0] -> half_type, {784, 128}, {128, 1}
@3 = load[offset=256,end=1824](@1) -> half_type, {1, 28, 28}, {784, 28, 1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@4 = gpu::convert[target_type=1](x,@3) -> half_type, {1, 28, 28}, {784, 28, 1}
@5 = reshape[dims={-1, 784}](@4) -> half_type, {1, 784}, {784, 1}
@6 = load[offset=0,end=256](@1) -> half_type, {1, 128}, {128, 1}
@7 = gpu::gemm[alpha=1,beta=0](@5,@2,@6) -> half_type, {1, 128}, {128, 1}
@8 = hip::hip_copy_literal[id=@literal:2] -> half_type, {128, 10}, {10, 1}
@9 = hip::hip_copy_literal[id=@literal:1] -> half_type, {128}, {1}
@10 = hip::hip_copy_literal[id=@literal:3] -> half_type, {10}, {1}
@11 = load[offset=256,end=512](@1) -> half_type, {1, 128}, {128, 1}
@12 = broadcast[axis=1,dims={1, 128}](@9) -> half_type, {1, 128}, {0, 1}
@13 = gpu::add_relu(@7,@12,@11) -> half_type, {1, 128}, {128, 1}
```

(continues on next page)

(continued from previous page)

```

@14 = load[offset=0,end=20](@1) -> half_type, {1, 10}, {10, 1}
@15 = gpu::gemm[alpha=1,beta=0](@13,@8,@14) -> half_type, {1, 10}, {10, 1}
@16 = broadcast[axis=1,dims={1, 10}](@10) -> half_type, {1, 10}, {0, 1}
@17 = load[offset=20,end=40](@1) -> half_type, {1, 10}, {10, 1}
@18 = gpu::add(@15,@16,@17) -> half_type, {1, 10}, {10, 1}
@19 = load[offset=0,end=20](@1) -> half_type, {1, 10}, {10, 1}
@20 = gpu::softmax[axis=1](@18,@19) -> half_type, {1, 10}, {10, 1}
output = @param:output -> float_type, {1, 10}, {10, 1}
@21 = gpu::convert[target_type=2](@20,output) -> float_type, {1, 10}, {10, 1}

```

### 4.3.6 Option: verify

```
$ /opt/rocm/bin/migraphx-driver verify simple_graph.pb
```

View Output

```

Reading: simple_graph.pb
@0 = @literal{0.0136018, -0.0839988, 0.0375392, 0.0613085, -0.125795, 0.176185, 0.
→0761055, 0.0093384, -0.110057, -0.170587} -> float_type, {10}, {1}
@1 = @literal{ ... } -> float_type, {128, 10}, {10, 1}
@2 = @literal{ ... } -> float_type, {128}, {1}
@3 = @literal{ ... } -> float_type, {784, 128}, {128, 1}
@4 = @literal{-1, 784} -> int32_type, {2}, {1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@5 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@6 = identity(@3) -> float_type, {784, 128}, {128, 1}
@7 = dot[alpha=1,beta=1](@5,@6) -> float_type, {1, 128}, {128, 1}
@8 = identity(@2) -> float_type, {128}, {1}
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}
@10 = add(@7,@9) -> float_type, {1, 128}, {128, 1}
@11 = relu(@10) -> float_type, {1, 128}, {128, 1}
@12 = identity(@1) -> float_type, {128, 10}, {10, 1}
@13 = dot[alpha=1,beta=1](@11,@12) -> float_type, {1, 10}, {10, 1}
@14 = identity(@0) -> float_type, {10}, {1}
@15 = broadcast[axis=1,dims={1, 10}](@14) -> float_type, {1, 10}, {0, 1}
@16 = add(@13,@15) -> float_type, {1, 10}, {10, 1}
@17 = softmax[axis=1](@16) -> float_type, {1, 10}, {10, 1}
@18 = identity(@17) -> float_type, {1, 10}, {10, 1}

@0 = @literal{0.0136018, -0.0839988, 0.0375392, 0.0613085, -0.125795, 0.176185, 0.
→0761055, 0.0093384, -0.110057, -0.170587} -> float_type, {10}, {1}
@1 = @literal{ ... } -> float_type, {128, 10}, {10, 1}
@2 = @literal{ ... } -> float_type, {128}, {1}
@3 = @literal{ ... } -> float_type, {784, 128}, {128, 1}
@4 = @literal{-1, 784} -> int32_type, {2}, {1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@5 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@6 = identity(@3) -> float_type, {784, 128}, {128, 1}
@7 = dot[alpha=1,beta=1](@5,@6) -> float_type, {1, 128}, {128, 1}
@8 = identity(@2) -> float_type, {128}, {1}
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}

```

(continues on next page)

(continued from previous page)

```

@10 = add(@7,@9) -> float_type, {1, 128}, {128, 1}
@11 = relu(@10) -> float_type, {1, 128}, {128, 1}
@12 = identity(@1) -> float_type, {128, 10}, {10, 1}
@13 = dot[alpha=1,beta=1](@11,@12) -> float_type, {1, 10}, {10, 1}
@14 = identity(@0) -> float_type, {10}, {1}
@15 = broadcast[axis=1,dims={1, 10}](@14) -> float_type, {1, 10}, {0, 1}
@16 = add(@13,@15) -> float_type, {1, 10}, {10, 1}
@17 = softmax[axis=1](@16) -> float_type, {1, 10}, {10, 1}
@18 = identity(@17) -> float_type, {1, 10}, {10, 1}

@0 = @literal{0.0136018, -0.0839988, 0.0375392, 0.0613085, -0.125795, 0.176185, 0.
↳0761055, 0.0093384, -0.110057, -0.170587} -> float_type, {10}, {1}
@1 = @literal{ ... } -> float_type, {128, 10}, {10, 1}
@2 = @literal{ ... } -> float_type, {128}, {1}
@3 = @literal{ ... } -> float_type, {784, 128}, {128, 1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@4 = ref::reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@5 = ref::identity(@3) -> float_type, {784, 128}, {128, 1}
@6 = ref::dot[alpha=1,beta=1](@4,@5) -> float_type, {1, 128}, {128, 1}
@7 = ref::identity(@2) -> float_type, {128}, {1}
@8 = ref::broadcast[axis=1,dims={1, 128}](@7) -> float_type, {1, 128}, {0, 1}
@9 = ref::contiguous(@8) -> float_type, {1, 128}, {128, 1}
@10 = ref::add(@6,@9) -> float_type, {1, 128}, {128, 1}
@11 = ref::relu(@10) -> float_type, {1, 128}, {128, 1}
@12 = ref::identity(@1) -> float_type, {128, 10}, {10, 1}
@13 = ref::dot[alpha=1,beta=1](@11,@12) -> float_type, {1, 10}, {10, 1}
@14 = ref::identity(@0) -> float_type, {10}, {1}
@15 = ref::broadcast[axis=1,dims={1, 10}](@14) -> float_type, {1, 10}, {0, 1}
@16 = ref::contiguous(@15) -> float_type, {1, 10}, {10, 1}
@17 = ref::add(@13,@16) -> float_type, {1, 10}, {10, 1}
@18 = ref::softmax[axis=1](@17) -> float_type, {1, 10}, {10, 1}
@19 = ref::identity(@18) -> float_type, {1, 10}, {10, 1}

@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
↳{256}, {1}
@2 = hip::hip_copy_literal[id=@literal:3] -> float_type, {784, 128}, {128, 1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@3 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}
@4 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@5 = gpu::gemm[alpha=1,beta=0](@4,@2,@3) -> float_type, {1, 128}, {128, 1}
@6 = hip::hip_copy_literal[id=@literal:1] -> float_type, {128, 10}, {10, 1}
@7 = hip::hip_copy_literal[id=@literal:2] -> float_type, {128}, {1}
@8 = hip::hip_copy_literal[id=@literal:0] -> float_type, {10}, {1}
@9 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}
@10 = broadcast[axis=1,dims={1, 128}](@7) -> float_type, {1, 128}, {0, 1}
@11 = gpu::add_relu(@5,@10,@9) -> float_type, {1, 128}, {128, 1}
@12 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}
@13 = gpu::gemm[alpha=1,beta=0](@11,@6,@12) -> float_type, {1, 10}, {10, 1}
@14 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}
@15 = broadcast[axis=1,dims={1, 10}](@8) -> float_type, {1, 10}, {0, 1}
@16 = gpu::add(@13,@15,@14) -> float_type, {1, 10}, {10, 1}

```

(continues on next page)

(continued from previous page)

```
output = @param:output -> float_type, {1, 10}, {10, 1}
@17 = gpu::softmax[axis=1](@16,output) -> float_type, {1, 10}, {10, 1}
```

### 4.3.7 Option: perf

```
$ /opt/rocm/bin/migraphx-driver perf simple_graph.pb
```

View Output

```
Compiling ...
Reading: simple_graph.pb
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
->{256}, {1}
@2 = hip::hip_copy_literal[id=@literal:3] -> float_type, {784, 128}, {128, 1}
@3 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@4 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@5 = gpu::gemm[alpha=1,beta=0](@4,@2,@3) -> float_type, {1, 128}, {128, 1}
@6 = hip::hip_copy_literal[id=@literal:1] -> float_type, {128, 10}, {10, 1}
@7 = hip::hip_copy_literal[id=@literal:0] -> float_type, {10}, {1}
@8 = hip::hip_copy_literal[id=@literal:2] -> float_type, {128}, {1}
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}
@10 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}
@11 = gpu::add_relu(@5,@9,@10) -> float_type, {1, 128}, {128, 1}
@12 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}
@13 = gpu::gemm[alpha=1,beta=0](@11,@6,@12) -> float_type, {1, 10}, {10, 1}
@14 = broadcast[axis=1,dims={1, 10}](@7) -> float_type, {1, 10}, {0, 1}
@15 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}
@16 = gpu::add(@13,@14,@15) -> float_type, {1, 10}, {10, 1}
output = @param:output -> float_type, {1, 10}, {10, 1}
@17 = gpu::softmax[axis=1](@16,output) -> float_type, {1, 10}, {10, 1}

Allocating params ...
Running performance report ...
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}: 0.00057782ms, 1%
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
->{256}, {1}: 0.000295ms, 1%
@2 = hip::hip_copy_literal[id=@literal:3] -> float_type, {784, 128}, {128, 1}: 0.
->00027942ms, 1%
@3 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}: 0.000232ms, 1%
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}: 0.0003206ms, 1%
@4 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}: 0.00033842ms, 1%
@5 = gpu::gemm[alpha=1,beta=0](@4,@2,@3) -> float_type, {1, 128}, {128, 1}: 0.212592ms, 1%
->52%
@6 = hip::hip_copy_literal[id=@literal:1] -> float_type, {128, 10}, {10, 1}: 0.
->00085822ms, 1%
@7 = hip::hip_copy_literal[id=@literal:0] -> float_type, {10}, {1}: 0.000382ms, 1%
@8 = hip::hip_copy_literal[id=@literal:2] -> float_type, {128}, {1}: 0.0003486ms, 1%
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}: 0.000299ms, 1%
@10 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}: 0.000234ms, 1%
```

(continues on next page)



(continued from previous page)

```

@11 = gpu::add_relu(@5,@9,@10) -> float_type, {1, 128}, {128, 1}: 0.0416597ms, 11%
@12 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}: 0.0007548ms, 1%
@13 = gpu::gemm[alpha=1,beta=0](@11,@6,@12) -> float_type, {1, 10}, {10, 1}: 0.0733071ms,
→ 18%
@14 = broadcast[axis=1,dims={1, 10}](@7) -> float_type, {1, 10}, {0, 1}: 0.00088142ms, 1%
@15 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}: 0.000408ms, 1%
@16 = gpu::add(@13,@14,@15) -> float_type, {1, 10}, {10, 1}: 0.0410144ms, 10%
output = @param:output -> float_type, {1, 10}, {10, 1}: 0.0010222ms, 1%
@17 = gpu::softmax[axis=1](@16,output) -> float_type, {1, 10}, {10, 1}: 0.0385636ms, 10%

```

## Summary:

```

gpu::gemm: 0.285899ms, 69%
gpu::add_relu: 0.0416597ms, 11%
gpu::add: 0.0410144ms, 10%
gpu::softmax: 0.0385636ms, 10%
hip::hip_copy_literal: 0.00186824ms, 1%
load: 0.0016288ms, 1%
@param: 0.0013428ms, 1%
broadcast: 0.00118042ms, 1%
check_context::migraphx::gpu::context: 0.00057782ms, 1%
reshape: 0.00033842ms, 1%
hip::hip_allocate_memory: 0.000295ms, 1%

```

Rate: 2866.1/sec

Total time: 0.348906ms

Total instructions time: 0.414369ms

Overhead time: 0.00348144ms, -0.0654627ms

Overhead: 1%, -19%

## 4.4 Driver options

This document lists the MIGraphX driver commands along with the eligible options.

### 4.4.1 read

Loads and prints input graph.

### 4.4.2 compile

Compiles and prints input graph.



### 4.4.3 run

Loads and prints input graph.

### 4.4.4 perf

Compiles and runs input graph then prints performance report.

**--iterations, -n** [unsigned int]

Sets number of iterations to run for perf report (Default: 100)

### 4.4.5 verify

Runs reference and CPU or GPU implementations and checks outputs for consistency.

**--rms-tol** [double]

Sets tolerance for RMS error (Default: 0.001)

**--atol** [double]

Sets tolerance for elementwise absolute difference (Default: 0.001)

**--rtol** [double]

Sets tolerance for elementwise relative difference (Default: 0.001)

**-i, --per-instruction**

Verifies each instruction

**-r, --reduce**

Reduces program and verifies

**--ref-use-double**

Converts floating point values to double for the ref target

### 4.4.6 roctx

**roctx** provides marker information for each operation which allows MIGraphX to be used with **rocprof** for performance analysis. This allows you to get GPU-level kernel timing information. Here is how you can use **roctx** combined with **rocprof** for tracing:

```
/opt/rocm/bin/rocprof --hip-trace --roctx-trace --flush-rate 1ms --timestamp on -d
↪ <OUTPUT_PATH> --obj-tracking on /opt/rocm/bin/migraphx-driver roctx <ONNX_FILE>
↪ <MIGRAPHX_OPTIONS>
```

Running **rocprof** generates trace information for HIP, HCC and ROCTX in separate **.txt** files. To understand the interactions between API calls, utilize the **roctx.py** helper script.



## CONTRIBUTING TO MIGRAPHX

This document explains the internal implementation of some commonly used MIGraphX APIs. You can utilize the information provided in this document and other documents under “Contributing to MIGraphX” section to contribute to the MIGraphX API implementation. Here is how some basic operations in the MIGraphX framework are performed.

### 5.1 Performing basic operations

A program is a collection of modules, which are collections of instructions to be executed when calling `eval`. Each instruction has an associated `operation` which represents the computation to be performed by the instruction.

The following code snippets demonstrate some basic operations using MIGraphX.

#### 5.1.1 Adding literals

Here is a `add_two_literals()` function:

```
// create the program and get a pointer to the main module
migraphx::program p;
auto* mm = p.get_main_module();

// add two literals to the program
auto one = mm->add_literal(1);
auto two = mm->add_literal(2);

// make the add operation between the two literals and add it to the program
mm->add_instruction(migraphx::make_op("add"), one, two);

// compile the program on the reference device
p.compile(migraphx::ref::target{});

// evaluate the program and retrieve the result
auto result = p.eval({}).back();
std::cout << "add_two_literals: 1 + 2 = " << result << "\n";
```

In the above function, a simple `program` object is created along with a pointer to the main module of it. The program is a collection of modules which starts execution from the main module, so instructions are added to the modules rather than the program object directly. The `add_literal` function is used to add an instruction that stores the literal number 1 while returning an `instruction_ref`. The returned `instruction_ref` can be used in another instruction as an input. The same `add_literal` function is used to add the literal 2 to the program. After the literals are created, the instruction is created to add the numbers. This is done by using the `add_instruction` function with the `add operation`.

created by `make_op` and the previously created literals passed as the arguments for the instruction. You can run this [program](#) by compiling it for the reference target (CPU) and then running it with `eval`. This prints the result on the console.

To compile the program for the GPU, move the file to `test/gpu/` directory and include the given target:

```
#include <migraphx/gpu/target.hpp>
```

### 5.1.2 Adding Parameters

While the `add_two_literals()` function above demonstrates add operation on constant values 1 and 2, the following program demonstrates how to pass a parameter (`x`) to a module using `add_parameter()` function .

```
migraphx::program p; auto* mm = p.get_main_module(); migraphx::shape
s{migraphx::shape::int32_type, {1}};

// add parameter "x" with the shape s auto x = mm->add_parameter("x", s); auto two = mm->add_literal(2);

// add the "add" instruction between the "x" parameter and "two" to the module mm-
>add_instruction(migraphx::make_op("add"), x, two); p.compile(migraphx::ref::target{ });
```

In the code snippet above, an add operation is performed on a parameter of type `int32` and literal 2 followed by compilation for the CPU. To run the program, pass the parameter as a `parameter_map` while calling `eval`. To map the parameter `x` to an [argument](#) object with an `int` data type, a `parameter_map` is created as shown below:

```
// create a parameter_map object for passing a value to the "x" parameter
std::vector<int> data = {4};
migraphx::parameter_map params;
params["x"] = migraphx::argument(s, data.data());

auto result = p.eval(params).back();
std::cout << "add_parameters: 4 + 2 = " << result << "\n";
EXPECT(result.at<int>() == 6);
```

### 5.1.3 Handling Tensor Data

The above two examples demonstrate scalar operations. To describe multi-dimensional tensors, use the [shape](#) class to compute a simple convolution as shown below:

```
migraphx::program p;
auto* mm = p.get_main_module();

// create shape objects for the input tensor and weights
migraphx::shape input_shape{migraphx::shape::float_type, {2, 3, 4, 4}};
migraphx::shape weights_shape{migraphx::shape::float_type, {3, 3, 3, 3}};

// create the parameters and add the "convolution" operation to the module
auto input = mm->add_parameter("X", input_shape);
auto weights = mm->add_parameter("W", weights_shape);
mm->add_instruction(migraphx::make_op("convolution", {{padding, {1, 1}}, {stride, {2,
↪ 2}}}), input, weights);
```

Most programs take data from allocated buffers that are usually on the GPU. To pass the buffer data as an argument, create [argument](#) objects directly from the pointers to the buffers:

```
// Compile the program
p.compile(migraphx::ref::target{});

// Allocated buffers by the user
std::vector<float> a = ...;
std::vector<float> c = ...;

// Solution vector
std::vector<float> sol = ...;

// Create the arguments in a parameter_map
migraphx::parameter_map params;
params["X"] = migraphx::argument(input_shape, a.data());
params["W"] = migraphx::argument(weights_shape, c.data());

// Evaluate and confirm the result
auto result = p.eval(params).back();
std::vector<float> results_vector(64);
result.visit([&](auto output) { results_vector.assign(output.begin(), output.end()); });

EXPECT(migraphx::verify::verify_rms_range(results_vector, sol));
```

An [argument](#) can handle memory buffers from either the GPU or the CPU. When running the [program](#), buffers are allocated on the corresponding target by default. By default, the buffers are allocated on the CPU when compiling for CPU and on the GPU when compiling for GPU. To locate the buffers on the CPU even when compiling for GPU, set the option `offload_copy=true`.

## 5.1.4 Importing From ONNX

To make it convenient to use neural networks directly from other frameworks, MIGraphX ONNX parser allows you to build a [program](#) directly from an ONNX file. For usage, refer to the `parse_onnx()` function below:

```
program p = migraphx::parse_onnx("model.onnx");
p.compile(migraphx::gpu::target{});
```

## 5.2 Sample programs

You can find all the MIGraphX examples in the [Examples](#) directory.

### 5.2.1 Build MIGraphX source code

To build a sample program `ref_dev_examples.cpp`, use:

```
make -j$(nproc) test_ref_dev_examples
```

This creates an executable file `test_ref_dev_examples` in the `bin/` of the build directory.

To verify the build, use:

```
make -j$(nproc) check
```

For detailed instructions on building MIGraphX from source, refer to the [README](#) file.

## 5.3 Data types

### 5.3.1 shape

struct **shape**

#### Public Types

enum **type\_t**

*Values:*

enumerator **bool\_type**

enumerator **half\_type**

enumerator **float\_type**

enumerator **double\_type**

enumerator **uint8\_type**

enumerator **int8\_type**

enumerator **uint16\_type**

enumerator **int16\_type**

enumerator **int32\_type**

enumerator **int64\_type**

enumerator **uint32\_type**

enumerator **uint64\_type**

enumerator **fp8e4m3fnuz\_type**

enumerator **tuple\_type**

## Public Functions

**shape()**

**shape**(*type\_t* t)

**shape**(*type\_t* t, std::vector<std::size\_t> l)

**shape**(*type\_t* t, std::vector<std::size\_t> l, std::vector<std::size\_t> s)

**shape**(*type\_t* t, std::initializer\_list<std::size\_t> d)

**shape**(*type\_t* t, std::vector<*dynamic\_dimension*> dims)

**shape**(*type\_t* t, std::vector<std::size\_t> mins, std::vector<std::size\_t> maxes,  
std::vector<std::set<std::size\_t>> optimals\_list)

template<class **Range**>

inline **shape**(*type\_t* t, const *Range* &l)

template<class **Range1**, class **Range2**>

inline **shape**(*type\_t* t, const *Range1* &l, const *Range2* &s)

**shape**(const std::vector<*shape*> &subs)

*type\_t* **type**() const

const std::vector<std::size\_t> &**lens**() const

const std::vector<std::size\_t> &**strides**() const

std::size\_t **ndim**() const

The number of dimensions in the shape, either static or dynamic. Same as the number of indices required to get a data value.

std::size\_t **elements**() const

Return the number of elements in the tensor.

std::size\_t **bytes**() const

Return the number of total bytes used for storage of the tensor data; includes subshapes. For dynamic shape, returns the maximum number of bytes presuming a packed shape.

std::size\_t **type\_size**() const

Return the size of the type of the main shape. Returns 0 if there are subshapes.

const std::vector<*dynamic\_dimension*> &**dyn\_dims**() const

std::vector<std::size\_t> **min\_lens**() const

Minimum lengths for dynamic shape. *lens()* for static shape.

std::vector<std::size\_t> **max\_lens**() const

Maximum lengths for dynamic shape. *lens()* for static shape.

std::vector<std::set<std::size\_t>> **opt\_lens**() const

Optimum lengths for dynamic shape. Empty for static shape.

std::size\_t **index**(std::initializer\_list<std::size\_t> l) const

Map multiple indices to space index.

```
std::size_t index(const std::vector<std::size_t> &l) const
    Map multiple indices to space index.

template<class Iterator>
inline std::size_t index(Iterator start, Iterator last) const
    Map multiple indices from a range of iterator to a space index.

std::size_t index(std::size_t i) const
    Map element index to space index.

std::vector<std::size_t> multi(std::size_t idx) const
    Map element index to multi-dimensional index.

void multi_copy(std::size_t idx, std::size_t *start, const std::size_t *end) const
    Map element index to multi-dimensional index and put them them into location provided by pointers

bool packed() const
    Returns true if the shape is packed (number of elements and buffer size the same) with no padding

bool transposed() const
    Returns true if the shape has been transposed. That is the strides are not in descending order

bool broadcasted() const
    Returns true if the shape is broadcasting a dimension. That is, one of the strides are zero.

bool standard() const
    Returns true if the shape is in its standard format. That is, the shape is both packed and not transposed.

bool scalar() const
    Returns true if all strides are equal to 0 (scalar tensor)

bool dynamic() const
    Return true if the shape is dynamic.

bool any_of_dynamic() const
    Return true if this shape or any of the sub_shapes are dynamic.

shape normalize_standard() const

shape as_standard() const

shape with_lens(type_t t, const std::vector<std::size_t> &l) const

shape with_lens(const std::vector<std::size_t> &l) const

shape with_type(type_t t) const

shape to_dynamic() const

shape to_static(std::size_t x) const

template<class ...Visitors>
inline void visit_type(Visitors... vs) const

std::string type_string() const

const std::vector<shape> &sub_shapes() const
```



```
std::size_t element_space() const
```

Returns the number of elements in the data buffer. For a dynamic shape, returns the maximum number of elements of the data buffer and assumes it is packed. Will clip to the maximum of `size_t` if overflows for dynamic shapes.

## Public Static Functions

```
static const std::vector<type_t> &types()
```

```
static std::string name(type_t t)
```

```
static std::string cpp_type(type_t t)
```

```
static shape from_permutation(type_t t, const std::vector<std::size_t> &l, const std::vector<int64_t> &perm)
```

Creates an output shape with dimensions equal to the input lengths and strides determined by the permutation argument such that `find_permutation()` of the output shape returns the inputted permutation.

2D example: parameters: `l = [2, 3]`, `perm = [1, 0]` therefore: “original” shape = {lens = [3, 2], strides = [2, 1]} output\_shape = {lens = [2, 3], strides = [1, 2]}

3D example: parameters: `l = [2, 3, 4]`, `perm = [1, 2, 0]` therefore: “original” shape = {lens = [3, 4, 2], strides = [8, 2, 1]} output\_shape = {lens = [2, 3, 4], strides = [1, 8, 2]}

```
template<class Visitor, class TupleVisitor>
```

```
static inline void visit(type_t t, Visitor v, TupleVisitor tv)
```

```
template<class Visitor>
```

```
static inline void visit(type_t t, Visitor v)
```

```
template<class Visitor>
```

```
static inline void visit_types(Visitor v)
```

```
static type_t parse_type(const std::string &s)
```

## Friends

```
friend bool operator==(const shape &x, const shape &y)
```

```
friend bool operator!=(const shape &x, const shape &y)
```

```
friend std::ostream &operator<<(std::ostream &os, const shape &x)
```

```
template<class T>
```

```
struct as
```

## Public Types

```
using type = std::conditional_t<std::is_same<T, bool>{ }, int8_t, T>
```

## Public Functions

```
inline type max() const
```

```
inline type min() const
```

```
inline type nan() const
```

```
template<class U>  
inline type operator()(U u) const
```

```
template<class U>  
inline type *operator()(U *u) const
```

```
template<class U>  
inline const type *operator()(const U *u) const
```

```
inline type operator()() const
```

```
inline std::size_t size(std::size_t n = 1) const
```

```
inline auto is_integral() const
```

```
inline auto is_signed() const
```

```
inline auto is_unsigned() const
```

```
template<class U>  
inline type *from(U *buffer, std::size_t n = 0) const
```

```
template<class U>  
inline const type *from(const U *buffer, std::size_t n = 0) const
```

```
inline type_t type_enum() const
```

```
struct dynamic_dimension
```

## Public Functions

```
bool is_fixed() const
```

```
bool has_optimal() const
```

```
dynamic_dimension &operator+=(const std::size_t &x)
```

```
dynamic_dimension &operator--=(const std::size_t &x)
```

## Public Members

std::size\_t **min** = 0

std::size\_t **max** = 0

std::set<std::size\_t> **optimals** = { }

## Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

## Friends

friend bool **operator==**(const *dynamic\_dimension* &x, const *dynamic\_dimension* &y)

friend bool **operator!=**(const *dynamic\_dimension* &x, const *dynamic\_dimension* &y)

friend std::ostream &**operator<<**(std::ostream &os, const *dynamic\_dimension* &x)

friend bool **operator==**(const *dynamic\_dimension* &x, const std::size\_t &y)

friend bool **operator==**(const std::size\_t &x, const *dynamic\_dimension* &y)

friend bool **operator!=**(const *dynamic\_dimension* &x, const std::size\_t &y)

friend bool **operator!=**(const std::size\_t &x, const *dynamic\_dimension* &y)

friend *dynamic\_dimension* **operator+**(const *dynamic\_dimension* &x, const std::size\_t &y)

friend *dynamic\_dimension* **operator+**(const std::size\_t &x, const *dynamic\_dimension* &y)

friend *dynamic\_dimension* **operator-**(const *dynamic\_dimension* &x, const std::size\_t &y)

```
template<class T, class = void>
```

```
struct get_type
```

Subclassed by *migraphx::internal::shape::get\_type< const T >*

```
template<class T>
```

```
struct get_type<bool, T> : public std::integral_constant<type_t, bool_type>
```

```
template<class T>
```

```
struct get_type<const T> : public migraphx::internal::shape::get_type<T>
```

```
template<class T>
```

```
struct get_type<double, T> : public std::integral_constant<type_t, double_type>
```

```
template<class T>
```

```
struct get_type<float, T> : public std::integral_constant<type_t, float_type>
template<class T>
struct get_type<half, T> : public std::integral_constant<type_t, half_type>
template<class T>
struct get_type<int16_t, T> : public std::integral_constant<type_t, int16_type>
template<class T>
struct get_type<int32_t, T> : public std::integral_constant<type_t, int32_type>
template<class T>
struct get_type<int64_t, T> : public std::integral_constant<type_t, int64_type>
template<class T>
struct get_type<int8_t, T> : public std::integral_constant<type_t, int8_type>

template<class T> fp8e4m3fnuz, T > : public std::integral_constant< type_t,
fp8e4m3fnuz_type >
template<class T>
struct get_type<uint16_t, T> : public std::integral_constant<type_t, uint16_type>
template<class T>
struct get_type<uint32_t, T> : public std::integral_constant<type_t, uint32_type>
template<class T>
struct get_type<uint64_t, T> : public std::integral_constant<type_t, uint64_type>
template<class T>
struct get_type<uint8_t, T> : public std::integral_constant<type_t, uint8_type>
```

### 5.3.2 literal

```
struct literal : public migraphx::internal::raw_data<literal>
```

Represents a raw literal.

This stores the literal has a raw buffer that is owned by this class

## Public Functions

inline **literal**()

inline explicit **literal**(*shape*::type\_t shape\_type)

Empty literal with a specific shape type

template<class **U**, class **T** = deduce<*U*>, *shape*::type\_t **ShapeType** = *shape*::get\_type<*T*>{}>

inline **literal**(*U* x)

template<class **T**>

inline **literal**(const *shape* &s, const std::vector<*T*> &x)

template<class **T**>

inline **literal**(const *shape* &s, const std::initializer\_list<*T*> &x)

template<class **Iterator**>

inline **literal**(const *shape* &s, *Iterator* start, *Iterator* end)

template<class **T**, long **PrivateRequires\_\_LINE\_\_** = \_\_LINE\_\_, typename

std::enable\_if<(*PrivateRequires\_\_LINE\_\_* == \_\_LINE\_\_ && (migraphx::and\_<sizeof(*T*) == 1>{ })), int>::type = 0>

inline **literal**(const *shape* &s, *T* \*x)

inline bool **empty**() const

Whether data is available.

inline const char \***data**() const

Provides a raw pointer to the data.

inline const *shape* &**get\_shape**() const

inline std::vector<*literal*> **get\_sub\_objects**() const

inline *argument* **get\_argument**() const

Convert the data to an argument.

## 5.3.3 argument

struct **argument** : public migraphx::internal::raw\_data<*argument*>

Arguments passed to instructions.

An **argument** can represent a raw buffer of data that either be referenced from another element or it can be owned by the argument.

## Public Functions

**argument**() = default

explicit **argument**(const *shape* &s)

template<class **F**, long **PrivateRequires\_\_LINE\_\_** = \_\_LINE\_\_, typename

std::enable\_if<(*PrivateRequires\_\_LINE\_\_* == \_\_LINE\_\_ &&

(migraphx::and\_<std::is\_pointer<decltype(std::declval<*F*>())>{}>{})), int>::type = 0>

```
inline argument(shape s, F d)

template<class T>
inline argument(shape s, T *d)

template<class T>
inline argument(shape s, std::shared_ptr<T> d)

argument(shape s, std::nullptr_t)

argument(const std::vector<argument> &args)

char *data() const
    Provides a raw pointer to the data.

bool empty() const
    Whether data is available.

const shape &get_shape() const

argument reshape(const shape &s) const

argument copy() const

argument share() const
    Make copy of the argument that is always sharing the data.

std::vector<argument> get_sub_objects() const

argument element(std::size_t i) const
    Return the ith element.

template<class Iterator>
inline void fill(Iterator start, Iterator end)
```

### 5.3.4 raw\_data

```
template<class Derived>

struct raw_data : public migraphx::internal::raw_data_base
```

Provides a base class for common operations with raw buffer.

For classes that handle a raw buffer of data, this will provide common operations such as equals, printing, and visitors. To use this class the derived class needs to provide a `data()` method to retrieve a raw pointer to the data, and `get_shape` method that provides the shape of the data.

#### Public Functions

```
template<class Visitor>
inline void visit_at(Visitor v, std::size_t n = 0) const
    Visits a single data element at a certain index.
```

##### Parameters

- **v** – A function which will be called with the type of data
- **n** – The index to read from

```
template<class Visitor, class TupleVisitor>
inline void visit(Visitor v, TupleVisitor tv) const
```

```
template<class Visitor>
inline void visit(Visitor v) const
```

Visits the data.

This will call the visitor function with a *tensor\_view*<T> based on the shape of the data.

#### Parameters

**v** – A function to be called with *tensor\_view*<T>

```
inline bool single() const
```

Returns true if the raw data is only one element.

```
template<class T>
inline T at(std::size_t n = 0) const
```

Retrieves a single element of data.

#### Parameters

**n** – The index to retrieve the data from

#### Template Parameters

**T** – The type of data to be retrieved

#### Returns

The element as T

```
inline auto_cast implicit() const
```

Implicit conversion of raw data pointer.

```
template<class T>
inline tensor_view<T> get() const
```

Get a *tensor\_view* to the data.

```
template<class T>
inline T *cast() const
```

Cast the data pointer.

```
inline std::string to_string() const
```

## Friends

```
template<class Stream>
inline friend Stream &operator<<(Stream &os, const Derived &d)
```

```
struct auto_cast
```

## Public Types

```
template<class T>
using is_data_ptr = bool_c<(std::is_void<T>{} or std::is_same<char, std::remove_cv_t<T>>{}) or
std::is_same<unsigned char, std::remove_cv_t<T>>{}>>
```

```
template<class T>
using get_data_type = std::conditional_t<is_data_ptr<T>{}, float, T>
```

## Public Functions

```
template<class T>
inline operator  T()

template<class T>
inline bool matches() const

template<class T>
inline operator  T*()
```

## Public Members

```
const Derived *self
```

```
template<class T, class ...Ts>
auto migraphx::internal::visit_all(T &&x, Ts&&... xs)
```

Visits every object together.

This will visit every object, but assumes each object is the same type. This can reduce the deeply nested visit calls. Returns a function that takes the visitor callback. Calling syntax is `visit_all(xs...)([](auto... ys) {})` where `xs...` and `ys...` are the same number of parameters.

### Parameters

- **x** – A raw data object
- **xs** – Many raw data objects.

### Returns

A function to be called with the visitor

```
template<class T>
auto migraphx::internal::visit_all(const std::vector<T> &x)
```

Visits every object together.

This will visit every object, but assumes each object is the same type. This can reduce the deeply nested visit calls. Returns a function that takes the visitor callback.

### Parameters

**x** – A vector of raw data objects. Types must all be the same.

### Returns

A function to be called with the visitor



### 5.3.5 tensor\_view

```
template<class T>
struct tensor_view
```

#### Public Types

```
using value_type = T
```

```
using iterator = basic_iota_iterator<tensor_view_iterator_read<tensor_view<T>>, std::size_t>
```

```
using const_iterator = basic_iota_iterator<tensor_view_iterator_read<const tensor_view<T>>, std::size_t>
```

#### Public Functions

```
inline tensor_view()
```

```
inline tensor_view(shape s, T *d)
```

```
inline const shape &get_shape() const
```

```
inline bool empty() const
```

```
inline std::size_t size() const
```

```
inline T *data()
```

```
inline const T *data() const
```

```
template<class ...Ts, long PrivateRequires__LINE__ = __LINE__, typename
std::enable_if<(PrivateRequires__LINE__ == __LINE__ && (migraphx::and_<std::is_integral<Ts>{ }...>{ })),
int>::type = 0>
```

```
inline const T &operator() (Ts... xs) const
```

```
template<class ...Ts, long PrivateRequires__LINE__ = __LINE__, typename
std::enable_if<(PrivateRequires__LINE__ == __LINE__ && (migraphx::and_<std::is_integral<Ts>{ }...>{ })),
int>::type = 0>
```

```
inline T &operator() (Ts... xs)
```

```
template<class Iterator, long PrivateRequires__LINE__ = __LINE__, typename
std::enable_if<(PrivateRequires__LINE__ == __LINE__ && (migraphx::and_<not
std::is_integral<Iterator>{ }>{ })), int>::type = 0>
```

```
inline const T &operator() (Iterator start, Iterator last) const
```

```
template<class Iterator, long PrivateRequires__LINE__ = __LINE__, typename
std::enable_if<(PrivateRequires__LINE__ == __LINE__ && (migraphx::and_<not
std::is_integral<Iterator>{ }>{ })), int>::type = 0>
```

```
inline T &operator() (Iterator start, Iterator last)
```

```
inline T &operator[] (std::size_t i)
```

```
inline const T &operator[] (std::size_t i) const

inline T &front()

inline const T &front() const

inline T &back()

inline const T &back() const

inline iterator begin()

inline iterator end()

inline const_iterator begin() const

inline const_iterator end() const

template<class U = T>
inline std::vector<U> to_vector() const
```

### Friends

```
inline friend std::ostream &operator<<(std::ostream &os, const tensor_view<T> &x)
```

## 5.4 Operators

### 5.4.1 operation

struct **operation**

The operation interface represents an action an instruction will perform. All operation classes must be Copy-Constructible.

#### Public Functions

std::string **name**() const

A unique name identifying the operation.

void **finalize**(context &ctx)

An optional method that can be used to finalize the operator before running.

*shape* **compute\_shape**(const std::vector<*shape*> &input) const

This is used to compute the resulting shape from an operation. If an operation cannot be run with input shapes, then it should throw an exception.

*argument* **compute**(context &ctx, const *shape* &output, const std::vector<*argument*> &input) const

This performs the operation's computation.

This method can be optional when the operation is only used as a placeholder to be lowered later on.

#### Parameters

- **ctx** – This is the context created by the **target** during compilation. Implementations can use the target's **context** class rather than the **context** interface class.

- **output** – Equivalent to running `compute_shape` with each `shape` of the `argument`. For a fixed shape, the returned `argument` will have the same shape as `output`. For a dynamic shape, the returned `argument` will be a fixed shape within the bounds set in the dynamic shape `output`.
- **input** – This is the `argument` result from the previous instruction's computation.

**Returns**

Return an `argument` of the result computation. The shape of `argument` should be the same the output shape.

```
std::ptrdiff_t output_alias(const std::vector<shape> &input) const
```

An optional method to return which argument the output will alias. If there is no aliased output then -1 can be returned.

**Friends**

```
friend std::ostream &operator<<(std::ostream &os, const operation &op)
```

An optional stream operator to print the operation. When this is not implemented, it will just print the operation's name.

```
bool migraphx::internal::is_context_free(const operation &x)
```

Returns true if operation does not require a context to run compute.

```
bool migraphx::internal::has_finalize(const operation &x)
```

Returns true if the operation has a finalize method.

## 5.4.2 operators

namespace **op**

**Enums**

```
enum padding_mode_t
```

*Values:*

```
enumerator default_
```

```
enumerator same_lower
```

```
enumerator same_upper
```

```
enum class pooling_mode
```

*Values:*

```
enumerator average
```

```
enumerator max
```

enumerator **lpnorm**

enum class **rnn\_direction**

*Values:*

enumerator **forward**

enumerator **reverse**

enumerator **bidirectional**

enum class **normalize\_attribute**

**normalize\_attribute** settings: Note that default options are not included as enums.

- i. **use\_input** (default) vs. **use\_output**: Affects the rank of the attribute. **use\_input** -> **lens.size()**, **use\_output** -> **lens.size() + vec.size()**.
- ii. **use\_rank** (default) vs **use\_len**: **use\_rank** sets the max value/index of the attribute as the rank of **lens**. **use\_len** sets the max value/index as the corresponding value in **lens** at the axes index. Uses the **dynamic\_dimension.max** value for dynamic shapes. Returns the original vector (no normalization) if any of **dynamic\_dimension[axes]** are not fixed.
- iii. **clip\_min** vs. **not\_clip\_min** (default): Clip values less than the minimum to the minimum or not.
- iv. **include\_min** vs. **exclude\_min** (default): Include or exclude the minimum value/index for range checking and clipping.
- v. **clip\_max** vs. **not\_clip\_max** (default): Clip values greater than the maximum or not.
- vi. **include\_max** vs. **exclude\_max** (default): Include or exclude the maximum value/index for range checking and clipping.
- vii. **normalize\_padding**: To normalize the padding to  $2 * (\text{pad\_ndim})$  dimensions.

*Values:*

enumerator **use\_output**

enumerator **use\_len**

enumerator **clip\_max**

enumerator **clip\_min**

enumerator **include\_max**

enumerator **include\_min**

enumerator **normalize\_padding**

## Functions

```
std::ostream &operator<<(std::ostream &os, pooling_mode v)
```

```
std::ostream &operator<<(std::ostream &os, rnn_direction v)
```

```
struct abs : public migraphx::internal::op::unary<abs>
    #include <migraphx/op/abs.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct acos : public migraphx::internal::op::unary<acos>
    #include <migraphx/op/acos.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct acosh : public migraphx::internal::op::unary<acosh>
    #include <migraphx/op/acosh.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct add : public migraphx::internal::op::binary<add>
    #include <migraphx/op/add.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline std::string point_function() const
```

```
inline auto apply() const
```

```
struct allocate
```

*#include <migraphx/op/allocate.hpp>* Static allocate: No inputs: `allocate()` `this.s` attribute set to the static output shape of the buffer. `this.s` attribute can be set to a dynamic output shape; however this will allocate the maximum buffer size for that case

Dynamic allocate: One input: `allocate(output_dims)` `output_dims` are the output buffer dimensions and has a static shape. Either `this.s` or `this.buf_type` (but not both) must be set to calculate the dynamic output shape at compute time. If `this.buf_type` is set, the `compute_shape()` of `allocate` at compile time will have `dynamic_dimensions` from `{0, max_int}` with `rank = output_dims.ndim()`. If `this.s` is set then the `compute_shape()` will output `this.s`; `this.s` should be a dynamic shape.

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape* &output\_shape, const std::vector<*argument*> &args) const

### Public Members

optional<*shape*> **s**

optional<*shape*::type\_t> **buf\_type**

### Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **argmax**

*#include* <migraphx/op/argmax.hpp>

### Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize\_compute\_shape**(std::vector<*shape*> inputs) const

template<class **T**>  
inline int64\_t **calc\_argmax**(*T* &input, std::vector<std::size\_t> &indices, size\_t item\_num) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

### Public Members

int64\_t **axis** = 0

bool **select\_last\_index** = false

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct argmin
```

```
    #include <migraphx/op/argmin.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline std::string name() const
```

```
inline shape normalize_compute_shape(std::vector<shape> inputs) const
```

```
template<class T>
```

```
inline int64_t calc_argmin(T &input, std::vector<std::size_t> &indices, size_t item_num) const
```

```
inline argument compute(const shape &output_shape, std::vector<argument> args) const
```

### Public Members

```
int64_t axis = 0
```

```
bool select_last_index = false
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct as_shape
```

```
    #include <migraphx/op/as_shape.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(const std::vector<shape> &inputs) const
```

```
inline argument compute(shape output_shape, std::vector<argument> args) const
```

```
inline std::ptrdiff_t output_alias(const std::vector<shape> &) const
```

## Public Members

*shape* **s**

## Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct asin : public migraphx::internal::op::unary<asin>  
    #include <migraphx/op/asin.hpp>
```

## Public Functions

```
inline auto apply() const
```

```
struct asinh : public migraphx::internal::op::unary<asinh>  
    #include <migraphx/op/asinh.hpp>
```

## Public Functions

```
inline auto apply() const
```

```
struct atan : public migraphx::internal::op::unary<atan>  
    #include <migraphx/op/atan.hpp>
```

## Public Functions

```
inline auto apply() const
```

```
struct atanh : public migraphx::internal::op::unary<atanh>  
    #include <migraphx/op/atanh.hpp>
```

## Public Functions

```
inline auto apply() const
```

```
template<class Derived>
```

```
struct binary : public migraphx::internal::op::op_name<Derived>  
    #include <migraphx/op/binary.hpp>
```



## Public Functions

```
inline std::string point_function() const
```

```
inline std::string point_op() const
```

```
inline value base_attributes() const
```

```
inline value attributes() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

```
struct broadcast
```

*#include <migraphx/op/broadcast.hpp>* 1 input version: Broadcasts a tensor from the original shape to the broadcast\_lens by setting the stride of broadcasted dimensions to zero. axis attribute for a 1D input shape is the output dimension that stays the same. ex: broadcasting shape [1024] -> [4, 1024, 3] has axis = 1.

For higher rank input shapes, axis is an offset parameter for the broadcasting. Such that this operator would work in the opposite direction of NumPy broadcasting (left-most to rightwards element-wise comparison) ex: broadcasting shape [2, 2] -> [2, 2, 3] with axis = 0

2 input version: Broadcast the first input 1D shape into the second input shape based on the axis parameter. Handles broadcasting a 1D static shape into a higher rank dynamic shape. broadcast\_lens is not used

## Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

```
inline std::ptrdiff_t output_alias(const std::vector<shape>&) const
```

## Public Members

```
uint64_t axis = 0
```

```
std::vector<std::size_t> broadcast_lens = { }
```

## Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct capture
```

```
#include <migraphx/op/capture.hpp>
```

### Public Functions

```
inline std::string name() const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(context&, const shape&, const std::vector<argument> &args) const

inline std::ptrdiff_t output_alias(const std::vector<shape>&) const
```

### Public Members

```
std::size_t ins_index

std::function<void(std::size_t ins_index, std::vector<argument>)> f = {}
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)

struct ceil : public migraphx::internal::op::unary<ceil>
    #include <migraphx/op/ceil.hpp>
```

### Public Functions

```
inline auto apply() const

struct clip
    #include <migraphx/op/clip.hpp>
```

### Public Functions

```
inline std::string name() const

inline value attributes() const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const

struct concat
    #include <migraphx/op/concat.hpp>
```

### Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline std::vector<std::size\_t> **compute\_offsets**(const *shape* &output\_shape, const  
std::vector<*argument*> &args) const

inline *shape* **normalize\_compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

### Public Members

int64\_t **axis** = 0

### Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **contiguous**

*#include <migraphx/op/contiguous.hpp>* The contiguous operator takes a non-standard input tensor and returns the same tensor but in standard form. For example, if input tensor A which has lens = (4,5) is first transposed, i.e. lens = (5,4), this tensor's data layout remained the same during the transpose operation; only its shape lengths and strides were changed. This leaves the tensor in a non-standard form. The contiguous operator copies the underlying data such that resulting tensor is returned to a standard form.

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

inline auto **apply**() const

struct **convert** : public migraphx::internal::op::unary<*convert*>

*#include <migraphx/op/convert.hpp>*

### Public Functions

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline std::string point_op() const
```

```
inline auto apply() const
```

```
inline convert(shape::type_t t)
```

```
inline convert()
```

### Public Members

```
shape::type_t target_type = shape::half_type
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

struct **convolution**

*#include <migraphx/op/convolution.hpp>* Convolution operator. Does not support optimal dimensions for spatial dimensions. Returns empty optimals.

### Public Functions

```
inline std::string name() const
```

```
inline void check_attribute_size() const
```

```
inline value attributes() const
```

```
inline shape normalize_compute_shape(std::vector<shape> inputs) const
```

```
inline std::vector<std::size_t> calc_conv_lens(std::vector<std::size_t> x_lens, std::vector<std::size_t>  
w_lens) const
```

```
inline shape dynamic_compute_shape(shape x_shape, shape w_shape) const
```

```
inline shape static_compute_shape(shape x_shape, shape w_shape) const
```

```
inline size_t kdims() const
```

```
inline argument compute(shape output_shape, std::vector<argument> args) const
```

## Public Members

std::vector<std::size\_t> **padding** = {0, 0}

std::vector<std::size\_t> **stride** = {1, 1}

std::vector<std::size\_t> **dilation** = {1, 1}

int **group** = 1

*padding\_mode\_t* **padding\_mode** = default\_

## Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **convolution\_backwards**

*#include* <migraphx/op/convolution\_backwards.hpp>

## Public Functions

inline std::string **name**() const

inline void **check\_attribute\_size**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline std::vector<std::size\_t> **calc\_spatial\_lens**(std::vector<std::size\_t> x\_lens,  
std::vector<std::size\_t> w\_lens) const

inline *shape* **dynamic\_compute\_shape**(*shape* x\_shape, *shape* w\_shape) const

inline *shape* **static\_compute\_shape**(*shape* x\_shape, *shape* w\_shape) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

inline size\_t **kdims**() const

## Public Members

std::vector<std::size\_t> **padding** = {0, 0}

std::vector<std::size\_t> **stride** = {1, 1}

std::vector<std::size\_t> **dilation** = {1, 1}

```
padding_mode_t padding_mode = default_
```

```
int group = 1
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct cos : public migraphx::internal::op::unary<cos>  
    #include <migraphx/op/cos.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct cosh : public migraphx::internal::op::unary<cosh>  
    #include <migraphx/op/cosh.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct dequantizelinear  
    #include <migraphx/op/dequantizelinear.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(const shape &output_shape, std::vector<argument> args) const
```

```
struct dimensions_of
```

```
    #include <migraphx/op/dimensions_of.hpp> Returns the dimensions of the input argument from starting  
    axis to ending axis. Atleast end must be set to use this operator (set end to ndim for default ONNX behavior  
    of Shape operator) This should only be used for dynamic shapes as this can be simplified to a literal for  
    static shapes.
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(const std::vector<shape> &inputs) const
```

```
inline argument compute(const shape &output_shape, std::vector<argument> args) const
```

### Public Members

```
std::size_t start = 0
```

```
std::size_t end = 0
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct div : public migraphx::internal::op::binary<div>
    #include <migraphx/op/div.hpp>
```

### Public Functions

```
inline std::string point_function() const
```

```
inline auto apply() const
```

```
struct dot
    #include <migraphx/op/dot.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

```
struct elu : public migraphx::internal::op::unary<elu>
    #include <migraphx/op/elu.hpp>
```

### Public Functions

```
inline std::string point_op() const
```

```
inline auto apply() const
```

### Public Members

```
float alpha = 1
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct equal : public migraphx::internal::op::binary<equal>  
    #include <migraphx/op/equal.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline std::string point_function() const
```

```
inline auto apply() const
```

```
struct erf : public migraphx::internal::op::unary<erf>  
    #include <migraphx/op/erf.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct exp : public migraphx::internal::op::unary<exp>  
    #include <migraphx/op/exp.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct fill  
    #include <migraphx/op/fill.hpp> fill(default_value, output_buffer) Fill an output buffer with the given default_value. Note that if the default_value is a literal and the output_buffer has a static shape this operator can be replaced with a literal.
```



### Public Functions

```

inline std::string name() const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const

inline std::ptrdiff_t output_alias(const std::vector<shape>&) const

```

```
struct flatten
```

```
    #include <migraphx/op/flatten.hpp>
```

### Public Functions

```

inline value attributes() const

inline std::string name() const

inline shape normalize_compute_shape(std::vector<shape> inputs) const

inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const

inline std::ptrdiff_t output_alias(const std::vector<shape>&) const

```

### Public Members

```
int64_t axis = 1
```

### Public Static Functions

```

template<class Self, class F>
static inline auto reflect(Self &self, F f)

```

```
struct floor : public migraphx::internal::op::unary<floor>
```

```
    #include <migraphx/op/floor.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct fmod : public migraphx::internal::op::binary<fmod>
```

```
    #include <migraphx/op/fmod.hpp>
```

### Public Functions

```
inline std::string name() const  
inline value attributes() const  
inline auto apply() const
```

```
struct gather
```

```
    #include <migraphx/op/gather.hpp>
```

### Public Functions

```
inline value attributes() const  
inline std::string name() const  
inline shape normalize_compute_shape(std::vector<shape> inputs) const  
inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

### Public Members

```
int64_t axis = 0
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct gathernd
```

```
    #include <migraphx/op/gathernd.hpp>
```

### Public Functions

```
inline std::string name() const  
inline shape compute_shape(std::vector<shape> inputs) const  
inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

### Public Members

```
int batch_dims = 0
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct get_tuple_elem
    #include <migraphx/op/get_tuple_elem.hpp>
```

### Public Functions

```
inline std::string name() const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(const shape&, std::vector<argument> args) const

inline std::ptrdiff_t output_alias(const std::vector<shape>&) const
```

### Public Members

```
std::size_t index = 0
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct greater : public migraphx::internal::op::binary<greater>
    #include <migraphx/op/greater.hpp>
```

### Public Functions

```
inline std::string point_function() const

inline auto apply() const
```

```
struct gru
    #include <migraphx/op/gru.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

### Public Members

```
std::size_t hidden_size = 1
```

```
std::vector<operation> actv_funcs = {sigmoid{}, tanh{}}
```

```
rnn_direction direction = rnn_direction::forward
```

```
float clip = 0.0f
```

```
int linear_before_reset = 0
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct highest
```

```
#include <migraphx/op/reduce_op.hpp>
```

### Public Functions

```
template<class T>  
inline operator T() const
```

```
struct identity
```

```
#include <migraphx/op/identity.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(shape, std::vector<argument> args) const
```

```
inline value attributes() const
```

```
inline std::ptrdiff_t output_alias(const std::vector<shape>&) const
```

```
struct if_op
```

```
#include <migraphx/op/if_op.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(const std::vector<shape> &inputs, std::vector<module_ref> mods) const
```

```
inline argument compute(const shape&, const std::vector<argument> &args, const
                        std::vector<module_ref> &mods, const
                        std::function<std::vector<argument>(module_ref&, const
                        std::unordered_map<std::string, argument>&> &run) const
```

```
struct im2col
```

```
#include <migraphx/op/im2col.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline value attributes() const
```

```
inline shape normalize_compute_shape(std::vector<shape> inputs) const
```

### Public Members

```
std::vector<std::size_t> padding = {0, 0}
```

```
std::vector<std::size_t> stride = {1, 1}
```

```
std::vector<std::size_t> dilation = {1, 1}
```

```
padding_mode_t padding_mode = default_
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct isinf : public migraphx::internal::op::unary<isinf>
#include <migraphx/op/isinf.hpp>
```

### Public Functions

inline auto **apply**() const

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

struct **isnan** : public migraphx::internal::op::unary<*isnan*>  
#include <migraphx/op/isnan.hpp>

### Public Functions

inline auto **apply**() const

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

struct **layout** : public migraphx::internal::op::unary<*layout*>  
#include <migraphx/op/layout.hpp>

### Public Functions

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline auto **apply**() const

### Public Members

std::vector<int64\_t> **permutation**

### Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **leaky\_relu** : public migraphx::internal::op::unary<*leaky\_relu*>  
#include <migraphx/op/leaky\_relu.hpp>

### Public Functions

inline std::string **point\_op**() const

inline std::string **name**() const

inline auto **apply**() const

### Public Members

float **alpha** = 0.01

### Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **less** : public migraphx::internal::op::binary<*less*>  
*#include <migraphx/op/less.hpp>*

### Public Functions

inline std::string **point\_function**() const

inline auto **apply**() const

struct **load**  
*#include <migraphx/op/load.hpp>*

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape*&, const std::vector<*argument*> &args) const

inline lifetime **get\_lifetime**() const

inline std::ptrdiff\_t **output\_alias**(const std::vector<*shape*>&) const

## Public Members

*shape* **s**

std::size\_t **offset** = 0

## Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

## Friends

```
inline friend std::ostream &operator<<(std::ostream &os, const load &op)
```

```
struct log : public migraphx::internal::op::unary<log>  
    #include <migraphx/op/log.hpp>
```

## Public Functions

```
inline auto apply() const
```

```
struct logical_and : public migraphx::internal::op::binary<logical_and>  
    #include <migraphx/op/logical_and.hpp>
```

## Public Functions

```
inline std::string point_function() const
```

```
inline auto apply() const
```

```
struct logical_or : public migraphx::internal::op::binary<logical_or>  
    #include <migraphx/op/logical_or.hpp>
```

## Public Functions

```
inline std::string point_function() const
```

```
inline auto apply() const
```

```
struct logical_xor : public migraphx::internal::op::binary<logical_xor>  
    #include <migraphx/op/logical_xor.hpp>
```



### Public Functions

inline std::string **point\_function**() const

inline auto **apply**() const

struct **logsoftmax**

*#include <migraphx/op/logsoftmax.hpp>*

### Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize\_compute\_shape**(std::vector<*shape*> inputs) const

inline auto **output**() const

### Public Members

int64\_t **axis** = 1

### Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **loop**

*#include <migraphx/op/loop.hpp>*

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(const std::vector<*shape*> &inputs, std::vector<module\_ref> mods) const

inline *argument* **compute**(context &ctx, const *shape* &out\_shape, const std::vector<*argument*> &args,  
const std::vector<module\_ref> &mods, const  
std::function<std::vector<*argument*>(module\_ref&, const  
std::unordered\_map<std::string, *argument*>&)> &run) const

## Public Members

int64\_t **max\_iterations** = 10

## Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **ref\_loop**  
*#include <migraphx/op/loop.hpp>*

## Public Functions

template<class **T**>  
inline void **copy**(context&, const *argument* &src, *T* &dst) const

template<class **T**>  
inline void **copy**(context&, *T* src, const *argument* &dst) const

inline void **append**(const std::vector<*argument*> &iter\_state, const std::vector<*argument*>  
&concatenated\_outputs, int iter) const

inline void **set\_zero**(context&, const std::vector<*argument*> &concatenated\_outputs, int iter)  
const

inline std::unordered\_map<std::string, int> **get\_output\_params**(const module&) const

## Public Members

int64\_t **max\_iterations** = 0

struct **lowest**  
*#include <migraphx/op/reduce\_op.hpp>*

## Public Functions

template<class **T**>  
inline **operator** *T*() const

struct **lrn**  
*#include <migraphx/op/lrn.hpp>*

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

### Public Members

```
float alpha = 0.0001
```

```
float beta = 0.75
```

```
float bias = 1.0
```

```
int size = 1
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct lstm
```

```
    #include <migraphx/op/lstm.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

### Public Members

```
std::size_t hidden_size = 1
```

```
std::vector<operation> actv_funcs = {sigmoid{}, tanh{}, tanh{}}
```

```
rnn_direction direction = rnn_direction::forward
```

```
float clip = 0.0f
```

```
int input_forget = 0
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct max : public migraphx::internal::op::binary<max>  
    #include <migraphx/op/max.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline auto apply() const
```

```
struct min : public migraphx::internal::op::binary<min>  
    #include <migraphx/op/min.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline auto apply() const
```

```
struct mod : public migraphx::internal::op::binary<mod>  
    #include <migraphx/op/mod.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline value attributes() const
```

```
inline auto apply() const
```

```
struct mul : public migraphx::internal::op::binary<mul>  
    #include <migraphx/op/mul.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline std::string point_function() const
```

```
inline auto apply() const
```

```
struct multibroadcast
```

*#include <migraphx/op/multibroadcast.hpp>* Broadcast multiple dimensions between two tensors. Two versions of this operator: 1 input and 2+ inputs. One input version uses output\_lens attribute and broadcasts to it. 2+ inputs version broadcasts first input to the common shape at evaluation time.

### Public Functions

```
inline std::string name() const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const

inline std::ptrdiff_t output_alias(const std::vector<shape>&) const
```

### Public Members

```
std::vector<std::size_t> output_lens = {}

std::vector<shape::dynamic_dimension> output_dyn_dims = {}
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct multinomial
    #include <migraphx/op/multinomial.hpp>
```

### Public Functions

```
inline std::string name() const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

### Public Members

```
shape::type_t dtype = shape::type_t::int32_type
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct nearbyint : public migraphx::internal::op::unary<nearbyint>
    #include <migraphx/op/nearbyint.hpp>
```

### Public Functions

inline auto **apply**() const

```
struct neg : public migraphx::internal::op::unary<neg>
    #include <migraphx/op/neg.hpp>
```

### Public Functions

inline std::string **point\_function**() const

inline auto **apply**() const

```
struct nonmaxsuppression
    #include <migraphx/op/nonmaxsuppression.hpp>
```

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

```
template<class T>
inline box batch_box(T boxes, std::size_t box_idx) const
```

inline bool **suppress\_by\_iou**(*box* b1, *box* b2, double iou\_threshold) const

```
template<class T>
inline std::vector<std::pair<double, int64_t>> filter_boxes_by_score(T scores_start, std::size_t
                                                                    num_boxes, double
                                                                    score_threshold) const
```

```
template<class Output, class Boxes, class Scores>
inline std::size_t compute_nms(Output output, Boxes boxes, Scores scores, std::size_t
                                max_output_boxes_per_class, double iou_threshold, double
                                score_threshold) const
```

inline *argument* **compute**(const *shape* &output\_shape, std::vector<*argument*> args) const

### Public Members

bool **center\_point\_box** = false

bool **use\_dyn\_output** = false

## Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct box
```

```
    #include <migraphx/op/nonmaxsuppression.hpp>
```

## Public Functions

```
inline void sort()
```

```
inline std::array<double, 2> &operator[](std::size_t i)
```

```
inline double area() const
```

## Public Members

```
std::array<double, 2> x
```

```
std::array<double, 2> y
```

```
struct nonzero
```

```
    #include <migraphx/op/nonzero.hpp>
```

## Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(const shape &output_shape, std::vector<argument> args) const
```

```
struct one
```

```
    #include <migraphx/op/reduce_op.hpp>
```

## Public Functions

```
template<class T>
inline operator T() const
```

```
template<class Derived>
```

```
struct op_name
```

```
    #include <migraphx/op/name.hpp> Create name from class.
```

```
Subclassed by migraphx::internal::op::binary< add >, migraphx::internal::op::binary<
div >, migraphx::internal::op::binary< equal >, migraphx::internal::op::binary< fmod
>, migraphx::internal::op::binary< greater >, migraphx::internal::op::binary< less >,
```

```

migraphx::internal::op::binary< logical_and >, migraphx::internal::op::binary< logical_or >,
migraphx::internal::op::binary< logical_xor >, migraphx::internal::op::binary< max >,
migraphx::internal::op::binary< min >, migraphx::internal::op::binary< mod >,
migraphx::internal::op::binary< mul >, migraphx::internal::op::binary< pow >,
migraphx::internal::op::binary< prelu >, migraphx::internal::op::binary< sqdiff >,
migraphx::internal::op::binary< sub >, migraphx::internal::op::prefix_scan_op< prefix_scan_sum >,
migraphx::internal::op::reduce_op< reduce_max >, migraphx::internal::op::reduce_op< reduce_mean >,
migraphx::internal::op::reduce_op< reduce_min >, migraphx::internal::op::reduce_op< reduce_prod >,
migraphx::internal::op::reduce_op< reduce_sum >, migraphx::internal::op::scatter_op< scatter_add >,
migraphx::internal::op::scatter_op< scatter_max >, migraphx::internal::op::scatter_op< scatter_min >,
migraphx::internal::op::scatter_op< scatter_mul >, migraphx::internal::op::scatter_op< scatter_none >,
migraphx::internal::op::scatternd_op< scatternd_add >, migraphx::internal::op::scatternd_op< scatternd_max >,
migraphx::internal::op::scatternd_op< scatternd_min >, migraphx::internal::op::scatternd_op< scatternd_mul >,
migraphx::internal::op::scatternd_op< scatternd_none >, migraphx::internal::op::unary< abs >,
migraphx::internal::op::unary< acos >, migraphx::internal::op::unary< acosh >, migraphx::internal::op::unary< asin >,
migraphx::internal::op::unary< asinh >, migraphx::internal::op::unary< atan >,
migraphx::internal::op::unary< atanh >, migraphx::internal::op::unary< ceil >,
migraphx::internal::op::unary< convert >, migraphx::internal::op::unary< cos >,
migraphx::internal::op::unary< cosh >, migraphx::internal::op::unary< elu >,
migraphx::internal::op::unary< erf >, migraphx::internal::op::unary< exp >,
migraphx::internal::op::unary< floor >, migraphx::internal::op::unary< isinf >,
migraphx::internal::op::unary< isnan >, migraphx::internal::op::unary< layout >,
migraphx::internal::op::unary< leaky_relu >, migraphx::internal::op::unary< log >,
migraphx::internal::op::unary< nearbyint >, migraphx::internal::op::unary< neg >,
migraphx::internal::op::unary< recip >, migraphx::internal::op::unary< relu >,
migraphx::internal::op::unary< rsqrt >, migraphx::internal::op::unary< sigmoid >,
migraphx::internal::op::unary< sign >, migraphx::internal::op::unary< sin >,
migraphx::internal::op::unary< sinh >, migraphx::internal::op::unary< sqrt >,
migraphx::internal::op::unary< tan >, migraphx::internal::op::unary< tanh >,
migraphx::internal::op::unary< unary_not >, migraphx::internal::op::binary< Derived >,
migraphx::internal::op::prefix_scan_op< Derived >, migraphx::internal::op::reduce_op< Derived >,
migraphx::internal::op::scatter_op< Derived >, migraphx::internal::op::scatternd_op< Derived >,
migraphx::internal::op::unary< Derived >

```

## Public Functions

```
inline std::string name() const
```

```
struct outline
```

```
#include <migraphx/op/outline.hpp>
```

## Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(const std::vector<shape> &inputs) const
```

```
inline argument compute(const shape&, const std::vector<argument>&) const
```



## Public Members

*shape* **s**

## Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

struct **pad**

```
#include <migraphx/op/pad.hpp>
```

## Public Types

enum **pad\_op\_mode\_t**

*Values:*

enumerator **constant\_pad**

enumerator **reflect\_pad**

enumerator **edge\_pad**

## Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline std::size_t pad_ndims() const
```

```
inline bool symmetric() const
```

## Public Members

std::vector<int64\_t> **pads**

float **value** = 0.0f

*pad\_op\_mode\_t* **mode** = constant\_pad

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct pointwise
```

```
    #include <migraphx/op/pointwise.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(const std::vector<shape> &inputs, std::vector<module_ref> mods) const
```

```
inline argument compute(const shape &output_shape, const std::vector<argument> &args, const
                        std::vector<module_ref> &mods, const
                        std::function<std::vector<argument>(module_ref&, const
                        std::unordered_map<std::string, argument>&> &run) const
```

```
struct pooling
```

```
    #include <migraphx/op/pooling.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline void check_attribute_size() const
```

```
inline size_t kdims() const
```

```
inline value attributes() const
```

```
inline std::size_t dilate_dim(std::size_t dim, std::size_t dilation) const
```

```
inline std::vector<std::size_t> calc_spatial_dim_out(const std::vector<std::size_t> &input_lens,
                                                       std::size_t kdims) const
```

```
inline shape normalize_compute_shape(std::vector<shape> inputs) const
```

```
template<class Type, class Out, class In, class Op>
```

```
inline void calc_pooling(const shape &output_shape, Out &output, const In &input, const
                        std::vector<std::size_t> &kernel_dims, const std::vector<std::size_t>
                        &padding_vals, Op op) const
```

```
inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

## Public Members

```

pooling_mode mode = {pooling_mode::average}

std::vector<std::size_t> padding = {0, 0}

std::vector<std::size_t> stride = {1, 1}

std::vector<std::size_t> lengths = {1, 1}

std::vector<std::size_t> dilations = {1, 1}

bool ceil_mode = false

int lp_order = 2

padding_mode_t padding_mode = padding_mode_t::default_

bool dyn_global = false

bool count_include_pad = false

```

## Public Static Functions

```

template<class Self, class F>
static inline auto reflect(Self &self, F f)

```

```

struct avg_pool
    #include <migraphx/op/pooling.hpp>

```

## Public Functions

```

template<class T>
inline double init() const

inline double operator()(double x, double y) const

inline double final(double x, std::size_t y) const

struct lpnorm_pool
    #include <migraphx/op/pooling.hpp>

```

### Public Functions

```
lpnorm_pool() = delete  
  
inline explicit lpnorm_pool(int x)  
  
template<class T>  
inline double init() const  
  
inline double operator()(double x, double y) const  
  
inline double final(double x, std::size_t) const
```

### Public Members

```
int p = 0
```

```
struct max_pool  
    #include <migraphx/op/pooling.hpp>
```

### Public Functions

```
template<class T>  
inline T init() const  
  
inline double operator()(double x, double y) const  
  
inline double final(double x, std::size_t) const
```

```
struct pow : public migraphx::internal::op::binary<pow>  
    #include <migraphx/op/pow.hpp>
```

### Public Functions

```
inline auto apply() const  
  
template<class Derived>  
  
struct prefix_scan_op : public migraphx::internal::op::op_name<Derived>  
    #include <migraphx/op/prefix_scan_op.hpp> Parent struct for prefix scan operations. A prefix scan is  
    equivalent to the C++ std::exclusive_scan or std::inclusive_scan. Given a list of numbers, a prefix scan  
    sum op returns an equal size list of running totals of the values. Other operations besides addition can be  
    supported by their own child ops.
```

## Public Functions

```

inline value attributes() const

inline shape normalize_compute_shape(std::vector<shape> inputs) const

inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const

inline auto init() const

inline prefix_scan_op()

inline prefix_scan_op(int64_t ax)

inline prefix_scan_op(int64_t ax, bool excl)

inline prefix_scan_op(int64_t ax, bool excl, bool rev)

```

## Public Members

```

int64_t axis

bool exclusive = false

bool reverse = false

```

## Public Static Functions

```

template<class Self, class F>
static inline auto reflect(Self &self, F f)

struct prefix_scan_sum : public migraphx::internal::op::prefix_scan_op<prefix_scan_sum>
    #include <migraphx/op/prefix_scan_sum.hpp>

```

## Public Functions

```

inline prefix_scan_sum()

inline prefix_scan_sum(int64_t ax)

inline prefix_scan_sum(int64_t ax, bool excl)

inline prefix_scan_sum(int64_t ax, bool excl, bool rev)

inline auto op() const

struct prelu : public migraphx::internal::op::binary<prelu>
    #include <migraphx/op/prelu.hpp>

```

### Public Functions

inline std::string **point\_op**() const

inline auto **apply**() const

struct **quant\_convolution**

*#include <migraphx/op/quant\_convolution.hpp>*

### Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline void **check\_attribute\_size**() const

inline *shape* **normalize\_compute\_shape**(std::vector<*shape*> inputs) const

inline size\_t **kdims**() const

inline *argument* **compute**(*shape* output\_shape, std::vector<*argument*> args) const

### Public Members

std::vector<std::size\_t> **padding** = {0, 0}

std::vector<std::size\_t> **stride** = {1, 1}

std::vector<std::size\_t> **dilation** = {1, 1}

*padding\_mode\_t* **padding\_mode** = default\_

int **group** = 1

### Public Static Functions

template<class **Self**, class **F**>

static inline auto **reflect**(*Self* &self, *F* f)

struct **quant\_dot**

*#include <migraphx/op/quant\_dot.hpp>*

### Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

struct **quantizelinear**

*#include <migraphx/op/quantizelinear.hpp>*

### Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output\_shape, std::vector<*argument*> args) const

struct **random\_seed**

*#include <migraphx/op/random\_seed.hpp>* Generates a random seed for the use of random number generators. Generating the seed at runtime guarantees there will be a different random sequence on every execution. This operation has no inputs or attributes, and outputs an unsigned integer tensor with a single value.

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape* &output\_shape, const std::vector<*argument*> &) const

### Public Members

*shape::type\_t* **dtype** = *shape::type\_t::uint64\_type*

### Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **random\_uniform**

*#include <migraphx/op/random\_uniform.hpp>* *random\_uniform* populates the passed shape with random numbers, in a uniform distribution. Range for floating-point data types is (0, 1); for integer types it is [0, <max value for the type>]

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

inline std::ptrdiff\_t **output\_alias**(const std::vector<*shape*>&) const

struct **recip** : public migraphx::internal::op::unary<*recip*>

*#include <migraphx/op/recip.hpp>*

### Public Functions

inline std::string **point\_op**() const

inline auto **apply**() const

struct **reduce\_max** : public migraphx::internal::op::reduce\_op<*reduce\_max*>

*#include <migraphx/op/reduce\_max.hpp>*

### Public Functions

inline **reduce\_max**()

inline **reduce\_max**(std::vector<int64\_t> ax)

inline auto **op**() const

inline auto **init**() const

struct **reduce\_mean** : public migraphx::internal::op::reduce\_op<*reduce\_mean*>

*#include <migraphx/op/reduce\_mean.hpp>*

### Public Functions

inline **reduce\_mean**()

inline **reduce\_mean**(std::vector<int64\_t> ax)

inline auto **op**() const

inline auto **output**(const *shape* &s) const

struct **reduce\_min** : public migraphx::internal::op::reduce\_op<*reduce\_min*>

*#include <migraphx/op/reduce\_min.hpp>*



## Public Functions

inline **reduce\_min**()

inline **reduce\_min**(std::vector<int64\_t> ax)

inline auto **op**() const

inline auto **init**() const

template<class **Derived**>

struct **reduce\_op** : public migraphx::internal::op::op\_name<*Derived*>

*#include <migraphx/op/reduce\_op.hpp>*

## Public Functions

inline value **attributes**() const

inline std::vector<int64\_t> **tune\_axes**(std::size\_t n\_dim) const

inline *shape* **normalize\_compute\_shape**(std::vector<*shape*> inputs) const

returns a shape in which the axis or axes named for reduction by this op are set, to size 1.

### Parameters

**inputs** – list of input shapes

### Returns

shape

template<class **T**>

inline void **tune\_dims**(const std::vector<int64\_t> &tuned\_axes, const std::vector<*T*> &in\_lens,  
std::vector<*T*> &out\_lens) const

template<class **T**>

inline void **reduce**(tensor\_view<*T*> &input, *shape* &batch\_shape, std::vector<int64\_t> &tuned\_axes,  
std::vector<std::size\_t> &out\_idx, tensor\_view<*T*> &output) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

inline auto **init**() const

inline auto **input**() const

inline auto **output**(const *shape*&) const

inline **reduce\_op**()

inline **reduce\_op**(std::vector<int64\_t> ax)

## Public Members

```
std::vector<std::int64_t> axes = { }
```

## Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct reduce_prod : public migraphx::internal::op::reduce_op<reduce_prod>  
    #include <migraphx/op/reduce_prod.hpp>
```

## Public Functions

```
inline reduce_prod()  
  
inline reduce_prod(std::vector<int64_t> ax)  
  
inline auto op() const  
  
inline auto init() const
```

```
struct reduce_sum : public migraphx::internal::op::reduce_op<reduce_sum>  
    #include <migraphx/op/reduce_sum.hpp>
```

## Public Functions

```
inline reduce_sum()  
  
inline reduce_sum(std::vector<int64_t> ax)  
  
inline auto op() const
```

```
struct relu : public migraphx::internal::op::unary<relu>  
    #include <migraphx/op/relu.hpp>
```

## Public Functions

```
inline std::string point_op() const  
  
inline auto apply() const
```

```
struct reshape
```

```
#include <migraphx/op/reshape.hpp> 1 input version: reshape(input_data) this.dims = output_dims Makes  
a copy of input_data to the output shape.
```

```
2 input version: reshape(input_data, output_buffer) this.dims = unset Copies input_data to output_buffer;  
output_buffer already has the output shape. This version will not fail gracefully if the input shape and  
output_buffer shape are incompatible. There's a throw that will catch when the number of elements do not
```

match at runtime. This version should only be used for dynamic reshapes (output dimensions only known at runtime). If output\_buffer has a static shape during compile/parse, you can use the 1 input version.

### Public Functions

```
inline std::string name() const

inline shape dyn_compute_shape(shape s0) const

inline shape static_compute_shape(std::vector<shape> inputs, std::size_t n_neg_dims) const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

### Public Members

```
std::vector<int64_t> dims
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct reshape_lazy
    #include <migraphx/op/reshape_lazy.hpp>
```

### Public Functions

```
inline value attributes() const

inline std::string name() const

inline shape dyn_compute_shape(shape s0) const

inline shape static_compute_shape(std::vector<shape> inputs, std::size_t n_neg_dims) const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const

inline std::ptrdiff_t output_alias(const std::vector<shape> &) const
```

## Public Members

```
std::vector<int64_t> dims
```

## Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
template<class Iterator>
static inline auto compute_end_dim(Iterator start, Iterator last, std::size_t dim)
```

```
template<class OptionalPair>
static inline OptionalPair try_merge_pairs(OptionalPair p2, OptionalPair p1)
```

```
template<class DimIterator, class StrideIterator>
static inline optional<std::size_t> merge_strides(DimIterator dim_start, DimIterator dim_last,
                                                    StrideIterator stride_start, StrideIterator stride_last)
```

```
template<class DimIterator, class StrideIterator>
static inline auto can_strides_merge(DimIterator dim_start, DimIterator dim_last, StrideIterator
                                     stride_start, StrideIterator stride_last)
```

```
static inline optional<shape> reshape_lazy_dims(const shape &input, const std::vector<std::size_t>
&rdims)
```

```
struct reverse
    #include <migraphx/op/reverse.hpp>
```

## Public Functions

```
inline std::string name() const
```

inline value **attributes()** const

```
inline shape normalize_compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(const shape &s, std::vector<argument> args) const
```

## Public Members

```
std::vector<int64_t> axes
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct rnn
```

```
    #include <migraphx/op/rnn.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

### Public Members

```
std::size_t hidden_size = 1
```

```
std::vector<operation> actv_funcs = {tanh{}, tanh{}}
```

```
rnn_direction direction = rnn_direction::forward
```

```
float clip = 0.0f
```

### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct rnn_last_cell_output
```

```
    #include <migraphx/op/rnn_last_cell_output.hpp>
```

### Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
struct rnn_last_hs_output
```

```
    #include <migraphx/op/rnn_last_hs_output.hpp>
```

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

struct **rnn\_var\_sl\_last\_output**

*#include <migraphx/op/rnn\_var\_sl\_last\_output.hpp>*

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

### Public Members

*rnn\_direction* **direction** = *rnn\_direction::forward*

### Public Static Functions

template<class **Self**, class **F**>

static inline auto **reflect**(*Self* &self, *F* f)

struct **rnn\_var\_sl\_shift\_output**

*#include <migraphx/op/rnn\_variable\_seq\_lens.hpp>*

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output\_shape, std::vector<*argument*> args) const

### Public Members

std::string **output\_name** = "hidden\_states"

*rnn\_direction* **direction** = *rnn\_direction::forward*

## Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct rnn_var_sl_shift_sequence
    #include <migraphx/op/rnn_variable_seq_lens.hpp>
```

## Public Functions

```
inline std::string name() const

inline shape compute_shape(std::vector<shape> inputs) const

inline argument compute(const shape &output_shape, std::vector<argument> args) const
```

```
struct roialign
    #include <migraphx/op/roialign.hpp>
```

## Public Functions

```
inline std::string name() const

inline shape compute_shape(std::vector<shape> inputs) const

inline auto calc_pos_weight(const std::array<std::size_t, 2> &dims, const shape &comp_s, const
                             std::array<float, 2> &roi_start, const std::array<float, 2> &bin_size,
                             const std::array<std::size_t, 2> &bin_grid_size) const

template<class T, class Op>
inline std::tuple<double, int64_t> calc_pooling(const T &data, const std::array<std::size_t, 2>
                                                  &bin_grid_size, const std::vector<pos_weight>
                                                  &pos_weights, int64_t index, Op op) const

inline argument compute(const shape &output_shape, std::vector<argument> args) const
```

## Public Members

```
std::string coord_trans_mode = "half_pixel"

pooling_mode mode = {pooling_mode::average}

int64_t output_height = 1

int64_t output_width = 1

int64_t sampling_ratio = 0

float spatial_scale = 1.0f
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct avg_pool  
    #include <migraphx/op/roialign.hpp>
```

### Public Functions

```
inline double init()  
  
inline double operator()(double x, double y)  
  
inline double final(double x, std::size_t y)
```

```
struct max_pool  
    #include <migraphx/op/roialign.hpp>
```

### Public Functions

```
inline double init()  
  
inline double operator()(double x, double y)  
  
inline double final(double x, std::size_t)
```

```
struct pos_weight  
    #include <migraphx/op/roialign.hpp>
```

### Public Members

```
std::array<std::size_t, 4> pos = {0, 0, 0, 0}  
  
std::array<float, 4> w = {0.0f, 0.0f, 0.0f, 0.0f}
```

```
struct rsqrt : public migraphx::internal::op::unary<rsqrt>  
    #include <migraphx/op/rsqrt.hpp>
```



## Public Functions

```
inline auto apply() const
```

```
struct run_on_target
```

```
#include <migraphx/op/run_on_target.hpp>
```

## Public Functions

```
inline std::string name() const
```

```
inline migraphx::shape compute_shape(const std::vector<migraphx::shape> &inputs,  
std::vector<migraphx::module_ref> mods) const
```

```
inline migraphx::argument compute(const migraphx::shape&, const std::vector<migraphx::argument>  
&args, const std::vector<migraphx::module_ref> &mods, const  
std::function<std::vector<migraphx::argument>(migraphx::module_ref&,  
const std::unordered_map<std::string, migraphx::argument>&)>  
&run) const
```

## Public Members

```
std::size_t target_id = 0
```

## Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct scalar
```

```
#include <migraphx/op/scalar.hpp>
```

## Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(shape output_shape, std::vector<argument> args) const
```

```
inline std::ptrdiff_t output_alias(const std::vector<shape>&) const
```

## Public Members

`std::vector<std::size_t> scalar_bcast_lens`

## Public Static Functions

`template<class Self, class F>  
static inline auto reflect(Self &self, F f)`

`struct scatter_add : public migraphx::internal::op::scatter_op<scatter_add>  
#include <migraphx/op/scatter_add.hpp>`

## Public Functions

`inline auto reduction() const`

`struct scatter_max : public migraphx::internal::op::scatter_op<scatter_max>  
#include <migraphx/op/scatter_max.hpp>`

## Public Functions

`inline auto reduction() const`

`struct scatter_min : public migraphx::internal::op::scatter_op<scatter_min>  
#include <migraphx/op/scatter_min.hpp>`

## Public Functions

`inline auto reduction() const`

`struct scatter_mul : public migraphx::internal::op::scatter_op<scatter_mul>  
#include <migraphx/op/scatter_mul.hpp>`

## Public Functions

`inline auto reduction() const`

`struct scatter_none : public migraphx::internal::op::scatter_op<scatter_none>  
#include <migraphx/op/scatter_none.hpp>`

### Public Functions

```

inline auto reduction() const

template<typename Derived>

struct scatter_op : public migraphx::internal::op::op_name<Derived>
    #include <migraphx/op/scatter_op.hpp>

```

### Public Functions

```

inline value attributes() const

inline shape normalize_compute_shape(std::vector<shape> inputs) const

inline argument compute(const shape &output_shape, std::vector<argument> args) const

inline const Derived &derived() const

```

### Public Members

```

int64_t axis = 0

```

### Public Static Functions

```

template<class Self, class F>
static inline auto reflect(Self &self, F f)

struct scatternd_add : public migraphx::internal::op::scatternd_op<scatternd_add>
    #include <migraphx/op/scatternd_add.hpp>

```

### Public Functions

```

inline scatternd_add()

inline auto reduction() const

struct scatternd_max : public migraphx::internal::op::scatternd_op<scatternd_max>
    #include <migraphx/op/scatternd_max.hpp>

```

### Public Functions

```

inline scatternd_max()

inline auto reduction() const

struct scatternd_min : public migraphx::internal::op::scatternd_op<scatternd_min>
    #include <migraphx/op/scatternd_min.hpp>

```

### Public Functions

inline **scatternd\_min**()

inline auto **reduction**() const

```
struct scatternd_mul : public migraphx::internal::op::scatternd_op<scatternd_mul>
    #include <migraphx/op/scatternd_mul.hpp>
```

### Public Functions

inline **scatternd\_mul**()

inline auto **reduction**() const

```
struct scatternd_none : public migraphx::internal::op::scatternd_op<scatternd_none>
    #include <migraphx/op/scatternd_none.hpp>
```

### Public Functions

inline **scatternd\_none**()

inline auto **reduction**() const

template<class **Derived**>

struct **scatternd\_op** : public migraphx::internal::op\_name<*Derived*>

*#include <migraphx/op/scatternd\_op.hpp>* N-dimensional Scatter operations. This struct is parent class to ops which differ in what formula is used to reduce (combine old and new values of) the scattered value. It was originally based on Onnx ScatterND operation (see <https://github.com/onnx/onnx/blob/main/docs/Operators.md#ScatterND>) and is also similar to Numpy `numpy.add.at()`.

#### Template Parameters

**Derived** – a template parameter in the CRTP inheritance idiom, represents one of the child operations.

### Public Functions

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

Validate input shapes and return the correct output shape. For Scatter ops, the output is the same shape as the data tensor (first input), but cast to a standard shape.

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

inline auto **init**() const

inline **scatternd\_op**()

struct **select\_module**

*#include <migraphx/op/select\_module.hpp>*

## Public Functions

```

inline std::string name() const

inline shape compute_shape(const std::vector<shape> &inputs, const std::vector<module_ref>&) const

inline std::vector<std::string> get_input_parameter_names(module_ref mod) const

inline std::vector<std::string> get_output_parameter_names(module_ref mod) const

inline argument compute(const shape&, const std::vector<argument> &args, const
                        std::vector<module_ref> &submodule_list, const
                        std::function<std::vector<argument>(module_ref&, const
                        std::unordered_map<std::string, argument>&)> &run) const

inline std::ptrdiff_t output_alias(const std::vector<shape> &shapes) const

```

## Public Members

*shape* **output\_dyn\_shapes**

## Public Static Functions

```

template<class Self, class F>
static inline auto reflect(Self &self, F f)

```

```

struct sigmoid : public migraphx::internal::op::unary<sigmoid>
    #include <migraphx/op/sigmoid.hpp>

```

## Public Functions

```

inline std::string point_op() const

inline auto apply() const

```

```

struct sign : public migraphx::internal::op::unary<sign>
    #include <migraphx/op/sign.hpp>

```

## Public Functions

```

inline std::string point_op() const

inline auto apply() const

```

```

struct sin : public migraphx::internal::op::unary<sin>
    #include <migraphx/op/sin.hpp>

```

## Public Functions

inline auto **apply**() const

```
struct sinh : public migraphx::internal::op::unary<sinh>
#include <migraphx/op/sinh.hpp>
```

## Public Functions

inline auto **apply**() const

struct **slice**

*#include <migraphx/op/slice.hpp>* Slice operator that accepts variable axes, starts and ends. All of **starts**, **ends**, and **axes** must be supplied by either their attribute or an input (but not both).

Valid calls: slice(input); axes, starts, ends set slice(input, starts); axes, ends set slice(input, ends); starts, axes set slice(input, axes); starts, ends set slice(input, starts, ends); axes set slice(input, starts, axes); ends set slice(input, ends, axes); starts set slice(input, start, ends, axes); none set

Attributes: axes: constant axes to slice over (optional) starts: constant slice starting indices (optional) ends: constant slice ending indices (optional)

Parameters: data: the input tensor to slice (dynamic or static shape) input\_starts: starting indices of slice (optional, static shape) input\_ends: ending indices of slice (optional, static shape) input\_axes: axes to slice over (optional, static shape)

## Public Functions

inline value **attributes**() const

Ensure that attribute axes is within limits. Will attempt to normalize starts and ends; but will use the dynamic\_dimension.max values for dynamic shapes. This makes it so you have to renormalize for non-fixed dynamic\_dimensions.

inline std::string **name**() const

template<class **A**, class **B**>

inline std::vector<std::size\_t> **lens\_calc**(const std::vector<std::size\_t> &lengths, *A* in\_starts, *A* in\_ends, *B* in\_axes) const

Computes the slice output shape dimensions for given starts, ends, and axes. Templated to also handle tensor views. Possibly different type between [in\_starts, in\_ends] and [in\_axes] if in\_axes is this object's axes attribute. Assumes in\_starts and in\_ends are normalized; in\_axes are valid.

inline std::array<bool, 3> **get\_set\_attributes**() const

Get the attributes that are non-empty.

inline *shape* **compute\_two\_or\_more**(std::vector<*shape*> inputs) const

Helper function for *normalize\_compute\_shape*()

inline *shape* **normalize\_compute\_shape**(std::vector<*shape*> inputs) const

inline auto **compute\_offset**(const *shape* &s) const

Calculates the starting offset for the sliced tensor. Used in compute when only data input and all other information are in the attributes.

**Parameters****s** – static input shapetemplate<class **T**>inline auto **compute\_offset**(const *shape* &s, const *T* &input\_starts, const *T* &ax\_vec) const

Calculates the starting offset for the sliced tensor (for aliasing). Used for 2-4 inputs to `slice`.

**Parameters**

- **s** – static input shape
- **input\_starts** – starting indices of slice
- **ax\_vec** – axes to slice on

```
inline std::unordered_map<std::string, std::vector<int64_t>> normalize_starts_ends_axes(shape
                                                                 in-
                                                                 put_shape,
                                                                 const
                                                                 op-
                                                                 tional<std::vector<int64_t>>
                                                                 &in-
                                                                 put_starts,
                                                                 const
                                                                 op-
                                                                 tional<std::vector<int64_t>>
                                                                 &in-
                                                                 put_ends,
                                                                 const
                                                                 op-
                                                                 tional<std::vector<int64_t>>
                                                                 &in-
                                                                 put_axes)
                                                                 const
```

If given, normalize the inputs. Otherwise get from operator attributes. Return the values in a map.

Parameters input\_shape: static shape of the input input\_starts: optional input\_ends: optional input\_ends: optional

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) constinline std::ptrdiff\_t **output\_alias**(const std::vector<*shape*>&) const**Public Members**std::vector<int64\_t> **axes** = { }std::vector<int64\_t> **starts** = { }std::vector<int64\_t> **ends** = { }

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

### Public Static Attributes

```
static constexpr std::array<bool, 3> all_set = {true, true, true}
```

Named arrays for the set attribute possibilities.

```
static constexpr std::array<bool, 3> ends_axes = {false, true, true}
```

```
static constexpr std::array<bool, 3> starts_axes = {true, false, true}
```

```
static constexpr std::array<bool, 3> starts_ends = {true, true, false}
```

```
static constexpr std::array<bool, 3> axes_only = {false, false, true}
```

```
static constexpr std::array<bool, 3> ends_only = {false, true, false}
```

```
static constexpr std::array<bool, 3> starts_only = {true, false, false}
```

```
static constexpr std::array<bool, 3> none_set = {false, false, false}
```

```
struct softmax
```

```
    #include <migraphx/op/softmax.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline std::string name() const
```

```
inline shape normalize_compute_shape(std::vector<shape> inputs) const
```

```
inline auto output() const
```

### Public Members

```
int64_t axis = 1
```



### Public Static Functions

```
template<class Self, class F>
static inline auto reflect(Self &self, F f)
```

```
struct sqdiff : public migraphx::internal::op::binary<sqdiff>
    #include <migraphx/op/sqdiff.hpp>
```

### Public Functions

```
inline std::string point_op() const
```

```
inline auto apply() const
```

```
struct sqr : public migraphx::internal::op::unary<sqr>
    #include <migraphx/op/sqr.hpp>
```

### Public Functions

```
inline auto apply() const
```

```
struct squeeze
    #include <migraphx/op/squeeze.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline std::string name() const
```

```
inline shape normalize_compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

```
inline std::ptrdiff_t output_alias(const std::vector<shape> &) const
```

### Public Members

```
std::vector<int64_t> axes
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct step
```

```
    #include <migraphx/op/step.hpp>
```

### Public Functions

```
inline value attributes() const
```

```
inline std::string name() const
```

```
inline shape normalize_compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(shape output_shape, std::vector<argument> args) const
```

```
inline std::ptrdiff_t output_alias(const std::vector<shape>&) const
```

### Public Members

```
std::vector<int64_t> axes
```

```
std::vector<int64_t> steps
```

### Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct sub : public migraphx::internal::op::binary<sub>
```

```
    #include <migraphx/op/sub.hpp>
```

### Public Functions

```
inline std::string point_function() const
```

```
inline auto apply() const
```

```
struct tan : public migraphx::internal::op::unary<tan>
```

```
    #include <migraphx/op/tan.hpp>
```

### Public Functions

inline auto **apply**() const

```
struct tanh : public migraphx::internal::op::unary<tanh>
    #include <migraphx/op/tanh.hpp>
```

### Public Functions

inline auto **apply**() const

```
struct topk
    #include <migraphx/op/topk.hpp>
```

### Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize\_compute\_shape**(std::vector<*shape*> inputs) const

template<class **T**, class **Compare**>

inline *heap\_vector*<*T*, *Compare*> **make\_heap**(std::vector<*T*> val, *Compare* compare) const

inline *argument* **compute**(const *shape* &output\_shape, std::vector<*argument*> args) const

### Public Members

int64\_t **k** = 1

int64\_t **axis** = 0

bool **largest** = true

### Public Static Functions

template<class **Self**, class **F**>

static inline auto **reflect**(*Self* &self, *F* f)

template<class **T**, class **Compare**>

struct **heap\_vector**

*#include <migraphx/op/topk.hpp>*

### Public Functions

inline **heap\_vector**(const std::vector<*T*> &val, *Compare* comp)

inline void **try\_push**(*T* val)

inline std::vector<*T*> **sort**()

### Public Members

std::vector<*T*> **data**

*Compare* **compare**

struct **transpose**

*#include* <migraphx/op/transpose.hpp>

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

inline std::ptrdiff\_t **output\_alias**(const std::vector<*shape*>&) const

### Public Members

std::vector<int64\_t> **dims**

### Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

template<class **Derived**>

struct **unary** : public migraphx::internal::op::op\_name<*Derived*>

*#include* <migraphx/op/unary.hpp>

### Public Functions

inline std::string **point\_function**() const

inline std::string **point\_op**() const

inline value **base\_attributes**() const

inline value **attributes**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

struct **unary\_not** : public migraphx::internal::op::unary<*unary\_not*>  
#include <migraphx/op/unary\_not.hpp>

### Public Functions

inline std::string **point\_function**() const

inline auto **apply**() const

inline std::string **name**() const

struct **undefined**  
#include <migraphx/op/undefined.hpp>

### Public Functions

inline std::string **name**() const

inline *shape* **compute\_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape*&, const std::vector<*argument*>&) const

struct **unique**  
#include <migraphx/op/unique.hpp>

### Public Functions

template<class T>  
inline auto **make\_idx\_less\_fn**(const T &data, size\_t chunk\_sz) const

template<class T>  
inline auto **sorted\_uniq\_indices**(const T &input\_data, size\_t chunk\_sz) const

template<class T>  
inline auto **unsorted\_uniq\_indices**(const T &input\_data, size\_t chunk\_sz) const

inline std::string **name**() const

```
inline shape compute_shape(std::vector<shape> inputs) const
```

```
inline argument compute(const dyn_output &dyn_out, std::vector<argument> args) const
```

## Public Members

```
std::optional<int64_t> axis
```

```
bool sorted = true
```

## Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

```
struct unknown  
    #include <migraphx/op/unknown.hpp>
```

## Public Functions

```
inline std::string name() const
```

```
inline shape compute_shape(std::vector<shape> input) const
```

## Public Members

```
std::string op
```

## Public Static Functions

```
template<class Self, class F>  
static inline auto reflect(Self &self, F f)
```

## Friends

```
inline friend std::ostream &operator<<(std::ostream &os, const unknown &x)
```

```
struct unsqueeze  
    #include <migraphx/op/unsqueeze.hpp> Adds dimensions to a tensor based on the axes attribute. axes  
    are based on the number of output shape dimensions and should not contain duplicates. steps are for  
    modifying dimensions added to the middle of the original shape. Each step must be a factor of the original  
    dimension. ex: unsqueeze(shape = [3, 4, 10], axes = [2, 4, 5], steps = [2]) -> shape = [3, 4, 2, 5, 1, 1]  
    Dynamic shape version does not handle steps.
```

### Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize\_compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

inline std::ptrdiff\_t **output\_alias**(const std::vector<*shape*>&) const

### Public Members

std::vector<int64\_t> **axes**

std::vector<int64\_t> **steps**

### Public Static Functions

template<class **Self**, class **F**>  
static inline auto **reflect**(*Self* &self, *F* f)

struct **where**

*#include <migraphx/op/where.hpp>*

### Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline *shape* **compute\_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn\_output &dyn\_out, std::vector<*argument*> args) const

struct **zero**

*#include <migraphx/op/reduce\_op.hpp>*

### Public Functions

template<class **T**>  
inline **operator** *T*() const

## 5.5 Program

### 5.5.1 instruction

struct **instruction**

#### Public Functions

inline **instruction**()

**instruction**(*operation* o, *shape* r, std::vector<*instruction\_ref*> args)

**instruction**(*operation* o, *shape* r, std::vector<*instruction\_ref*> args, std::vector<module\_ref> modules)

**instruction**(*literal* l)

void **replace**(*operation* o)

void **recompute\_shape**()

void **clear\_arguments**()

bool **valid**(*instruction\_ref* start, bool check\_order = false) const

bool **valid**() const

*shape* **get\_shape**() const

const *literal* &**get\_literal**() const

const *operation* &**get\_operator**() const

std::string **name**() const

const std::vector<*instruction\_ref*> &**inputs**() const

const std::vector<module\_ref> &**module\_inputs**() const

const std::vector<*instruction\_ref*> &**outputs**() const

Where this instruction is used as an input to another instruction.

void **add\_output**(*instruction\_ref* ins)

template<class T>

inline void **remove\_output**(const T &ins)

bool **can\_eval**() const

bool **is\_undefined**() const

*argument* **eval**(bool check\_eval = true) const

void **finalize**(context &ctx)

void **set\_normalized**(bool value = true)



```

bool is_normalized() const

bool need_normalization() const

operation normalized_operator() const

std::size_t get_target_id() const

void set_target_id(std::size_t tid)

void debug_print() const

```

### Public Static Functions

```

static void replace_refs(instruction_ref ins, const std::unordered_map<instruction_ref, instruction_ref>
                        &map_insts, const std::unordered_map<module_ref, module_ref> &map_mods)

static void backreference(instruction_ref ref)

static void replace_argument(instruction_ref ins, instruction_ref old, instruction_ref new_ins)

static void replace_mod_argument(instruction_ref ins, module_ref old, module_ref new_mod)

static void replace(instruction_ref ins, operation o, const shape &r, std::vector<instruction_ref> args)

static void replace(instruction_ref ins, operation o, const shape &r, std::vector<instruction_ref> args,
                    std::vector<module_ref> module_args)

static instruction_ref get_output_alias(instruction_ref ins, bool shallow = false)

static void print(std::ostream &os, instruction_ref ins, const std::unordered_map<instruction_ref,
                    std::string> &names)

```

### Friends

```

friend bool operator==(const instruction &i, instruction_ref ref)

friend bool operator==(const instruction &x, const instruction &y)

friend bool operator!=(const instruction &x, const instruction &y)

friend bool operator==(instruction_ref ref, const instruction &i)

friend bool operator!=(const instruction &i, instruction_ref ref)

friend bool operator!=(instruction_ref ref, const instruction &i)

```

### 5.5.2 instruction\_ref

type `migraphx::internal::instruction_ref`

References an instruction in the program.

### 5.5.3 program

struct **program**

Stores the instruction stream.

#### Public Functions

**program**()

**program**(*program*&&) noexcept

**program**(const *program*&)

*program* &**operator**=(*program*)

**~program**() noexcept

std::vector<std::string> **get\_parameter\_names**() const

*shape* **get\_parameter\_shape**(std::string name) const

*instruction\_ref* **get\_parameter**(std::string name) const

std::unordered\_map<std::string, *shape*> **get\_parameter\_shapes**() const

std::vector<*argument*> **eval**(parameter\_map params, execution\_environment exec\_env =  
execution\_environment{ }) const

void **finish**() const

std::size\_t **size**() const

std::vector<*shape*> **get\_output\_shapes**() const

context &**get\_context**() const

*instruction\_ref* **validate**() const

target\_assignments **get\_target\_assignments**(const std::vector<*target*> &targets, assignment\_options  
options = assignment\_options{ })

void **compile**(const *target* &t, compile\_options options = compile\_options{ })

void **compile**(const std::vector<*target*> &targets, std::vector<compile\_options> compile\_opts = { })

bool **is\_compiled**() const

void **finalize**()

void **perf\_report**(std::ostream &os, std::size\_t n, parameter\_map params, std::size\_t batch = 1) const

```

void mark(const parameter_map &params, marker &&m)

value to_value() const

void from_value(const value &v)

void debug_print() const

void debug_print(instruction_ref ins) const

void print(std::unordered_map<instruction_ref, std::string> &names, const
           std::function<void(instruction_ref, std::unordered_map<instruction_ref, std::string>)>
           &print_func) const

void print(const std::function<void(instruction_ref ins, std::unordered_map<instruction_ref, std::string>)>
           &print_func) const

void print_graph(std::ostream &os, bool brief = false) const

void print_py(std::ostream &os) const

void print_cpp(std::ostream &os) const

void dry_run(parameter_map params) const

void annotate(std::ostream &os, const std::function<void(instruction_ref)> &a) const

program &sort()

module *create_module(const std::string &name)

module *create_module(const std::string &name, module m)

module *get_module(const std::string &name)

const module *get_module(const std::string &name) const

module *get_main_module()

const module *get_main_module() const

std::vector<const module*> get_modules() const

std::vector<module*> get_modules()

std::unordered_multimap<module_ref, module_ref> get_module_tree()

void remove_module(const std::string &name)

void remove_unused_modules()

```

### Friends

friend std::ostream &**operator**<<(std::ostream &os, const *program* &p)

friend bool **operator**==(const *program* &x, const *program* &y)

inline friend bool **operator**!=(const *program* &x, const *program* &y)

## 5.5.4 parse\_onnx

*program* migraphx::internal::**parse\_onnx**(const std::string &name, const *onnx\_options*& = *onnx\_options*{})

Create a program from an onnx file.

## 5.5.5 parse\_tf

*program* migraphx::internal::**parse\_tf**(const std::string &name, const *tf\_options* &options = *tf\_options*{})

Create a program from a tf pb file (default is nhwc format)

## 5.5.6 onnx\_options

struct **onnx\_options**

struct to pass in onnx options to parser

## 5.5.7 tf\_options

struct **tf\_options**

struct to pass in tf options to parser

# 5.6 Targets

## 5.6.1 target

struct **target**

An interface for a compilation target.

### Public Functions

std::string **name**() const

A unique name used to identify the target.

`std::vector<pass> get_passes(context &ctx, const compile_options &options) const`

The transformation pass to be run during compilation.

#### Parameters

- **ctx** – This is the target-dependent context that is created by `get_context`
- **options** – Compiling options passed in by the user

#### Returns

The passes to be ran

`context get_context()` const

Construct a context for the target.

#### Returns

The context to be used during compilation and execution.

`supported_segments target_is_supported(T&, const_module_ref mod, support_metric metric) const`

Get the ranges of instructions that are supported on a target.

#### Parameters

- **module** – Module to check for supported instructions
- **metric** – Used to define how the quality of the support should be measured

#### Returns

the supported segments of the graph

`argument copy_to(const argument &arg) const`

copy an argument to the current target.

#### Parameters

**arg** – Input argument to be copied to the target

#### Returns

Argument in the target.

`argument copy_from(const argument &arg) const`

copy an argument from the current target.

#### Parameters

**arg** – Input argument to be copied from the target

#### Returns

Argument in the host.

`argument allocate(const shape &s) const`

Allocate an argument based on the input shape.

#### Parameters

**s** – Shape of the argument to be allocated in the target

#### Returns

Allocated argument in the target.

## 5.6.2 gpu::target

struct **target**

### Public Functions

std::string **name**() const

std::vector<*pass*> **get\_passes**(migraphx::context &gctx, const compile\_options &options) const

migraphx::context **get\_context**() const

*argument* **copy\_to**(const *argument* &arg) const

*argument* **copy\_from**(const *argument* &arg) const

*argument* **allocate**(const *shape* &s) const

## 5.6.3 cpu::target

struct **target**

### Public Functions

std::string **name**() const

std::vector<*pass*> **get\_passes**(migraphx::context &gctx, const compile\_options&) const

inline migraphx::context **get\_context**() const

inline *argument* **copy\_to**(const *argument* &arg) const

inline *argument* **copy\_from**(const *argument* &arg) const

*argument* **allocate**(const *shape* &s) const

## 5.7 Quantization

### 5.7.1 quantize\_fp16

```
void migraphx::internal::quantize_fp16(program &prog, const std::vector<std::string> &ins_names = {"all"})
```

### 5.7.2 quantize\_int8

```
void migraphx::internal::quantize_int8(program &prog, const target &t, const std::vector<parameter_map>
                                     &calibration, const std::unordered_set<std::string> &ins_names =
                                     {"dot", "convolution"})
```

## 5.8 Passes

### 5.8.1 pass

struct **pass**

An interface for applying a transformation to the instructions in a **program**

#### Public Functions

std::string **name**() const

A unique name used to identify the pass.

void **apply**(module\_pass\_manager &mpm) const

Run the pass on the module.

void **apply**(module &m) const

void **apply**(*program* &p) const

Run the pass on the program.

### 5.8.2 dead\_code\_elimination

struct **dead\_code\_elimination**

Remove instructions where the output is not used.

#### Public Functions

inline std::string **name**() const

void **apply**(module &m) const

void **apply**(*program* &p) const

### 5.8.3 eliminate\_common\_subexpression

struct **eliminate\_common\_subexpression**

Remove identical instructions.

#### Public Functions

inline std::string **name**() const

void **apply**(module &m) const

### 5.8.4 eliminate\_concat

struct **eliminate\_concat**

Remove concat operators by having each operator can write to different chunk of memory.

#### Public Functions

inline std::string **name**() const

void **apply**(module &m) const

#### Public Members

concat\_optimization **concat\_opt**

### 5.8.5 eliminate\_contiguous

struct **eliminate\_contiguous**

Remove contiguous instructions by checking if the operator can use non-standard shapes.

#### Public Functions

inline std::string **name**() const

void **apply**(module &m) const



## Public Members

std::string **op\_name**

### 5.8.6 eliminate\_identity

struct **eliminate\_identity**

Remove identity instructions. Currently when used as the last pass, it will preserve the semantics of previous program state, therefore dead code elimination should not be used afterwards.

## Public Functions

inline std::string **name**() const

void **apply**(module &m) const

### 5.8.7 eliminate\_pad

struct **eliminate\_pad**

Remove pads if they can be written as an attribute to another op (im2col, convolution, pooling)

## Public Functions

inline std::string **name**() const

void **apply**(module &m) const

### 5.8.8 propagate\_constant

struct **propagate\_constant**

Replace instructions which take all literals with a literal of the computation.

## Public Functions

inline std::string **name**() const

void **apply**(module &m) const

### Public Members

`std::unordered_set<std::string> skip_ops = {}`

## 5.8.9 `rewrite_rnn`

struct **rewrite\_rnn**

Rewrite rnn to gemm and add.

### Public Functions

`inline std::string name() const`

`void apply(module &m) const`

## 5.8.10 `schedule`

struct **schedule**

Schedule instructions for concurrent execution

### Public Functions

`inline std::string name() const`

`void apply(module &m) const`

### Public Members

`schedule_model model = {}`

`bool enable = true`

## 5.8.11 `simplify_algebra`

struct **simplify\_algebra**

Simplify many algebraic instructions to more efficient versions.

### Public Functions

```
inline std::string name() const
void apply(module &m) const
```

## 5.8.12 simplify\_rescales

```
struct simplify_rescales
    Eliminate redundant rescales.
```

### Public Functions

```
inline std::string name() const
void apply(module &m) const
```

### Public Members

```
size_t depth = 4
```

## 5.9 Matchers

### 5.9.1 Introduction

The matchers provide a way to compose several predicates together. A matcher such as `m(m1, m2)` first checks a match for `m` followed by a match for `m1` and `m2` subsequently.

The most commonly used matcher is the `name` matcher. It matches the instruction with the operator equal to the name specified:

```
auto match_sum = name("sum");
```

The above matcher finds sum operators. To find sum operators with the output `standard_shape`, use:

```
auto match_sum = name("sum")(standard_shape());
```

### 5.9.2 Arguments

To match arguments in the instructions, match each argument using the `arg` matcher:

```
auto match_sum = name("sum")(arg(0)(name("@literal")), arg(1)(name("@literal")));
```

The above matcher matches a sum operator with two arguments that are literals. Note that the `args` matcher eliminates the need to write `arg(0)` and `arg(1)` everytime:

```
auto match_sum = name("sum")(args(name("@literal"), name("@literal")));
```

### 5.9.3 Binding

To reference other instructions encountered while traversing through the instructions, use `.bind`:

```
auto match_sum = name("sum")(args(  
    name("@literal").bind("one"),  
    name("@literal").bind("two")  
)).bind("sum");
```

This associates the instruction to a name that can be read from the `matcher_result` when it matches.

### 5.9.4 Finding matches

To use the matchers to find instructions, write a callback object that contains the matcher and an `apply` function that takes the `matcher_result` when the match is found:

```
struct match_find_sum  
{  
    auto matcher() const { return name("sum"); }  
  
    void apply(program& p, matcher_result r) const  
    {  
        // Do something with the result  
    }  
};  
  
find_matches(prog, match_find_sum{});
```

### 5.9.5 Creating matchers

The macros `MIGRAPH_BASIC_MATCHER` and `MIGRAPH_PRED_MATCHER` help in the creation of the matchers. Here is how you can create a matcher for shapes that are broadcasted:

```
MIGRAPH_PRED_MATCHER(broadcasted_shape, instruction_ref ins)  
{  
    return ins->get_shape().broadcasted();  
}
```

For parameters to the predicate, use `make_basic_pred_matcher` to create the matcher. Here is how you can create a matcher to check the number of dimensions of the shape:

```
inline auto number_of_dims(std::size_t n)  
{  
    return make_basic_pred_matcher( [=](instruction_ref ins) {  
        return ins->get_shape().lens().size() == n;  
    });  
}
```

## 5.10 Tools

### 5.10.1 roctx.py

You can use the `roctx` command with `rocprof` binary to get marker timing information for each MIGraphX operator. To process timing information, use `roctx.py` helper script.

```
Usage: roctx.py [-h] [--json-path json_path] [--out out]
[--study-name study-name] [--repeat repeat] [--parse]
[--run run] [--debug]
```

The `roctx.py` helper script provides two main functionalities: `run` and `parse`.

#### **--run**

Runs `migraphx-driver roctx` command with the given `migraphx-driver` knobs followed by the parsing of the result which provides GPU kernel timing information. You can pass the MIGraphX knobs via a string to `-run` knob. See the `_roctx-examples` for usage.

#### **--parse**

Parses JSON file in the given `--json-path` and provides GPU kernel timing information.

#### **--out**

Output folder

#### **--study-name**

Optional. Allows user to name a study for easy interpretation. Defaults to timestamp.

#### **--repeat**

Number of iterations. Sets to **2** by default.

#### **--debug**

Provides additional debug information related to data. Use for debugging purposes only.

#### **Examples:**

##### **Running inference with rocTX for a given ONNX file:**

```
python roctx.py --run '--onnx --gpu fcn-resnet50-11.onnx' --out output_folder --repeat 5
```

Example output:

## \*\*\* RESULTS \*\*\*

	SUM_avg	MIN_avg	MAX_avg	COUNT
Marker start: gpu::convolution	1625	8	103	31
Marker start: gpu::conv_bias_relu	1212	8	450	18
Marker start: gpu::add_relu	155	1	11	23
Marker start: gpu::conv_bias	112	12	43	4
Marker start: load	110	0	2	160
Marker start: gpu::triadd_relu	90	7	11	10
Marker start: hip::hip_copy_literal	77	0	2	110
Marker start: broadcast	41	0	2	53
Marker start: gpu::concat	39	6	7	6
Marker start: gpu::mul_add	27	2	8	7
Marker start: gpu::sub	22	2	5	8
Marker start: slice	12	0	1	16
Marker start: gpu::pooling	7	7	7	1
Marker start: step	3	3	3	1
Marker start: multibroadcast	2	1	1	2
Marker start: @param	2	0	1	3
Marker start: hip::hip_allocate_memory	1	1	1	1
Marker start: check_context::migraphx::version_...	0	0	0	0

AVG TOTAL TIME: 3544 us

OUTPUT CSV FILE: output2021\_11\_04-03:03:02\_AM.csv  
 KERNEL TIMING DETAILS: roctx\_kernel\_timing\_details.txt  
 ALL DATA FROM ALL RUNS: roctx\_runs\_dataframe.csv

Hotspot kernel timing information:

**MOST TIME CONSUMING KERNELS IN EACH ITERATION (EXPECTED TO BE SAME KERNEL):**

KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1	448
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1	450
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1	449
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1	451
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1	456

The output provides SUM, MIN, MAX and COUNT information for each kernel executed for a given model. It also provides the average total time. The following three files are provided for reference:

- OUTPUT CSV FILE: Provides a summary of the run which includes utilized MIGraphX knobs and related kernel timing information.
- KERNEL TIMING DETAILS: Provides the hotspot kernel timing information.
- ALL DATA FROM ALL RUNS: Provides all output data related to all iterations executed during a run.

**Parsing an existing JSON file:**

```
python roctx.py --parse --json-path ../trace.json
```

## 5.11 Environment Variables

### 5.11.1 For parsing

#### **MIGRAPHX\_TRACE\_ONNX\_PARSER**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints debugging traces for the ONNX parser. Prints: Initializers (if used), ONNX node operators, added MIGraphX instructions.

#### **MIGRAPHX\_DISABLE\_FP16\_INSTANCENORM\_CONVERT**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Disables the conversion from fp16 to fp32 for the InstanceNormalization ONNX operator that MIGX does as a workaround for accuracy issues with *reduce\_mean/variance*. See `parse_instancenorm.cpp` for more details.

### 5.11.2 Matchers

#### **MIGRAPHX\_TRACE\_MATCHES**

Set to “1” to print the matcher that matches an instruction and the matched instruction. Set to “2” and use the `MIGRAPHX_TRACE_MATCHES_FOR` flag to filter out results.

#### **MIGRAPHX\_TRACE\_MATCHES\_FOR**

Set to the name of any matcher to print the traces for that matcher only.

#### **MIGRAPHX\_VALIDATE\_MATCHES**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Validates the module after finding the matches (runs `module.validate()`).

### 5.11.3 Program Execution

#### **MIGRAPHX\_TRACE\_EVAL**

Set to “1”, “2”, or “3” to use. “1” prints the instruction run and the time taken. “2” prints everything in “1” and a snippet of the output argument and some output statistics (e.g. min, max, mean). “3” prints everything in “1” and all output buffers.

### 5.11.4 Program Verification

#### **MIGRAPHX\_VERIFY\_ENABLE\_ALLCLOSE**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Uses `allclose` with the given `atol` and `rtol` for verifying ranges with `driver verify` or the tests that use `migraphx/verify.hpp`.

### 5.11.5 Pass debugging or Pass controls

#### **MIGRAPHX\_TRACE\_ELIMINATE\_CONTIGUOUS**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Debug print the instructions that have input contiguous instructions removed.

#### **MIGRAPHX\_DISABLE\_POINTWISE\_FUSION**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Disables the `fuse_pointwise` compile pass.

#### **MIGRAPHX\_DEBUG\_MEMORY\_COLORING**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints debug statements for the `memory_coloring` pass.

#### **MIGRAPHX\_TRACE\_SCHEDULE**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints debug statements for the `schedule` pass.

#### **MIGRAPHX\_TRACE\_PROPAGATE\_CONSTANT**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Traces instructions replaced with a constant.

#### **MIGRAPHX\_8BITS\_QUANTIZATION\_PARAMS**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints the quantization parameters in the main module only.

#### **MIGRAPHX\_DISABLE\_DNNL\_POST\_OPS\_WORKAROUND**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Disables the DNNL post ops workaround.

#### **MIGRAPHX\_DISABLE\_MIOOPEN\_FUSION**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Disables MIOpen fusions.

#### **MIGRAPHX\_DISABLE\_SCHEDULE\_PASS**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Disables the `schedule` pass.

#### **MIGRAPHX\_DISABLE\_REDUCE\_FUSION**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Disables the `fuse_reduce` pass.

#### **MIGRAPHX\_ENABLE\_NHWC**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Enables the `layout_nhwc` pass.

#### **MIGRAPHX\_ENABLE\_CK**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Enables use of the Composable Kernels library. Use it in conjunction with `MIGRAPHX_DISABLE_MLIR=1`.

#### **MIGRAPHX\_DISABLE\_MLIR\***

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Disables use of the rocMLIR library.

#### **MIGRAPHX\_ENABLE\_EXTRA\_MLIR**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Enables additional opportunities to use MLIR for chances of an improved performance.

#### **MIGRAPHX\_COPY\_LITERALS**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Uses `hip_copy_to_gpu` with a new `literal` instruction rather than using `hip_copy_literal{}`.



### 5.11.6 Compilation traces

#### **MIGRAPHX\_TRACE\_FINALIZE**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Debug print instructions during the `module.finalize()` step.

#### **MIGRAPHX\_TRACE\_COMPILE**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints trace information for the graph compilation process.

#### **MIGRAPHX\_TRACE\_PASSES**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints the compile pass and the program after the pass.

#### **MIGRAPHX\_TIME\_PASSES**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Times the compile passes.

### 5.11.7 GPU kernels JIT compilation debugging

These environment variables are applicable for both `hiprtc` and `hipclang`.

#### **MIGRAPHX\_TRACE\_CMD\_EXECUTE**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints commands executed by the MIGraphX process.

#### **MIGRAPHX\_TRACE\_HIPRTC**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints HIPRTC options and C++ file executed.

#### **MIGRAPHX\_DEBUG\_SAVE\_TEMP\_DIR**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prevents deletion of the created temporary directories.

#### **MIGRAPHX\_GPU\_DEBUG**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Internally, this adds the option `-DMIGRAPHX_DEBUG` when compiling GPU kernels. It enables assertions and capture of source locations for the errors.

#### **MIGRAPHX\_GPU\_DEBUG\_SYM**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Adds the option `-g` when compiling HIPRTC.

#### **MIGRAPHX\_GPU\_DUMP\_SRC**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Dumps the compiled HIPRTC source files.

#### **MIGRAPHX\_GPU\_DUMP\_ASM**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Dumps the hip-clang assembly.

#### **MIGRAPHX\_GPU\_OPTIMIZE**

Set the optimization mode for GPU compile (`-O` option). Defaults to `-O3`.

#### **MIGRAPHX\_GPU\_COMPILE\_PARALLEL**

Set to the number of threads to use. Compiles GPU code in parallel with the given number of threads.

#### **MIGRAPHX\_TRACE\_NARY**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints the nary device functions used.

**MIGRAPHX\_ENABLE\_HIPRTC\_WORKAROUNDS**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Enables HIPRTC workarounds for bugs in HIPRTC.

**MIGRAPHX\_USE\_FAST\_SOFTMAX**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Uses fast softmax optimization.

**MIGRAPHX\_ENABLE\_NULL\_STREAM**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Allows using null stream for miopen and hipStream.

**MIGRAPHX\_NSTREAMS**

Set to the number of streams to use. Defaults to 1.

**MIGRAPHX\_TRACE\_BENCHMARKING**

Set to “1” to print benchmarking trace. Set to “2” to print detailed benchmarking trace.

### 5.11.8 MLIR vars

**MIGRAPHX\_TRACE\_MLIR**

Set to “1” to trace MLIR and print any failures. Set to “2” to additionally print all MLIR operations.

**MIGRAPHX\_MLIR\_USE\_SPECIFIC\_OPS**

Set to the MLIR operations you want to always use regardless of the GPU architecture. Accepts a list of operators separated by commas (e.g. “fused”, “convolution”, “dot”).

**MIGRAPHX\_MLIR\_TUNING\_DB**

Set to the path of the MLIR tuning database to load.

**MIGRAPHX\_MLIR\_TUNING\_CFG**

Set to the path of the tuning configuration. Appends to tuning cfg file that could be used with rocMLIR tuning scripts.

**MIGRAPHX\_MLIR\_TUNE\_EXHAUSTIVE**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Performs exhaustive tuning for MLIR.

**MIGRAPHX\_MLIR\_TUNE\_LIMIT**

Set to an integer greater than 1. Limits the number of solutions available to MLIR for tuning.

### 5.11.9 CK vars

**MIGRAPHX\_LOG\_CK\_GEMM**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints composable kernels GEMM traces.

**MIGRAPHX\_CK\_DEBUG**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Mandatorily adds -DMIGRAPHX\_CK\_CHECK=1 for compiling composable kernel operators.

**MIGRAPHX\_TUNE\_CK**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Performs tuning for composable kernels.

### 5.11.10 Testing

#### **MIGRAPHX\_TRACE\_TEST\_COMPILE**

Set to the target whose compilation you want to trace (e.g. “gpu”, “cpu”). Prints the compile trace for verify tests on the given target. Don’t use this flag in conjunction with MIGRAPHX\_TRACE\_COMPILE.

#### **MIGRAPHX\_TRACE\_TEST**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Prints the reference and target programs even if the verify tests pass.

#### **MIGRAPHX\_DUMP\_TEST**

Set to “1”, “enable”, “enabled”, “yes”, or “true” to use. Dumps verify tests to .mxr files.



## C++ API REFERENCE

### 6.1 shape

enum **migraphx\_shape\_datatype\_t**

An enum to represent the different data type inputs.

*Values:*

enumerator **migraphx\_shape\_tuple\_type**

enumerator **migraphx\_shape\_bool\_type**

enumerator **migraphx\_shape\_half\_type**

enumerator **migraphx\_shape\_float\_type**

enumerator **migraphx\_shape\_double\_type**

enumerator **migraphx\_shape\_uint8\_type**

enumerator **migraphx\_shape\_int8\_type**

enumerator **migraphx\_shape\_uint16\_type**

enumerator **migraphx\_shape\_int16\_type**

enumerator **migraphx\_shape\_int32\_type**

enumerator **migraphx\_shape\_int64\_type**

enumerator **migraphx\_shape\_uint32\_type**

enumerator **migraphx\_shape\_uint64\_type**

enumerator **migraphx\_shape\_fp8e4m3fnuz\_type**

template<class **Lens**, class **Strides**>

struct **shape** : public `migraphx::handle_base<>`

Describe shape of tensor.

A shape consists of a data type, lengths of multi-dimension tensor, and strides

## Public Types

using **shape\_type** = *shape*

using **index\_array** = typename *Lens*::base\_array

## Public Functions

inline **shape**()

inline **shape**(const `migraphx_shape` \*p)

template<class **HandleType**, class **Lifetime**, class = typename  
std::enable\_if<std::is\_convertible<*HandleType*\*, `handle_type`\*>{}>::type>

inline **shape**(*HandleType* \*p, *Lifetime* lifetime)

inline **shape**(*migraphx\_shape\_datatype\_t* type)

Construct a scalar shape.

inline **shape**(*migraphx\_shape\_datatype\_t* type, std::vector<size\_t> lengths)

Construct a shape with its type and lengths. The strides are automatically computed assuming a packed layout.

inline **shape**(*migraphx\_shape\_datatype\_t* t, std::initializer\_list<std::size\_t> d)

inline **shape**(*migraphx\_shape\_datatype\_t* type, std::vector<size\_t> lengths, std::vector<size\_t> pstrides)

inline **shape**(*migraphx\_shape\_datatype\_t* type, const dynamic\_dimensions &dyn\_dims)

inline std::vector<size\_t> **lengths**() const

inline std::vector<size\_t> **strides**() const

inline dynamic\_dimensions **dyn\_dims**() const

Get the dynamic dimensions of the shape.

inline *migraphx\_shape\_datatype\_t* **type**() const

inline size\_t **elements**() const

inline size\_t **bytes**() const

inline bool **standard**() const

```

inline bool dynamic() const
    Is the shape dynamic.
inline size_t index(size_t i) const
constexpr shape() = default
inline constexpr shape(Lens l, Strides s)
inline constexpr auto elements() const
inline constexpr auto element_space() const
inline constexpr auto packed() const
inline constexpr auto broadcasted() const
inline constexpr auto transposed() const
inline constexpr auto skips() const
inline constexpr auto standard() const
inline constexpr index_int index(index_array x) const
inline constexpr index_int index(index_int i) const
inline constexpr index_int compute_index(index_int i) const
inline constexpr index_array multi(index_int idx) const
    Convert single index into a multi-index.
inline constexpr index_int single(index_array idx) const
    Convert multi-index into a single index.
inline constexpr shape get_shape() const

```

## Public Members

```
Lens lens = {}
```

```
Strides strides = {}
```

## Friends

```

inline friend bool operator==(const shape &px, const shape &py)
inline friend bool operator!=(const shape &px, const shape &py)
template<class Stream>
inline friend constexpr const Stream &operator<<(const Stream &ss, const shape &s)

```

## 6.2 argument

struct **argument** : public migraphx::handle\_base<>

Arguments to be passed to an migraphx arguments.

An argument represents a raw buffer of data with a shape.

### Public Functions

inline **argument**()

template<class **HandleType**, class **Lifetime**, class = typename  
std::enable\_if<std::is\_convertible<*HandleType*\*, handle\_type\*>{}>::type>  
inline **argument**(*HandleType* \*p, *Lifetime* lifetime)

inline **argument**(const migraphx\_argument \*p)

inline **argument**(*shape* pshape)

inline **argument**(*shape* pshape, void \*pbuffer)

inline *shape* **get\_shape**() const

inline char \***data**() const

template<typename T>  
inline std::vector<*T*> **as\_vector**() const

### Public Static Functions

static inline *argument* **generate**(*shape* ps, size\_t pseed = 0)

Generate an argument using random data.

### Friends

inline friend bool **operator==**(const *argument* &px, const *argument* &py)

inline friend bool **operator!=**(const *argument* &px, const *argument* &py)

## 6.3 target

struct **target** : public migraphx::handle\_base<>

A target for compilation.



## Public Functions

inline **target**()

```
template<class HandleType, class Lifetime, class = typename
std::enable_if<std::is_convertible<HandleType*, handle_type*>{ }>::type>
inline target(HandleType *p, Lifetime lifetime)
```

inline **target**(const char \*name)

Construct a target from its name.

## 6.4 program

```
struct program_parameter_shapes : public migraphx::handle_base<>
```

### Public Functions

inline **program\_parameter\_shapes**()

```
template<class HandleType, class Lifetime, class = typename
std::enable_if<std::is_convertible<HandleType*, handle_type*>{ }>::type>
inline program_parameter_shapes(HandleType *p, Lifetime lifetime)
```

inline size\_t **size**() const

inline *shape* **operator[]**(const char \*pname) const

inline std::vector<const char\*> **names**() const

```
struct program_parameters : public migraphx::handle_base<>
```

A class to construct the inputs parameters for a program.

### Public Functions

```
template<class HandleType, class Lifetime, class = typename
std::enable_if<std::is_convertible<HandleType*, handle_type*>{ }>::type>
inline program_parameters(HandleType *p, Lifetime lifetime)
```

inline **program\_parameters**(migraphx\_program\_parameters \*p)

inline **program\_parameters**()

inline **program\_parameters**(std::initializer\_list<std::pair<std::string, *argument*>> l)

Construct the parameters from initializer\_list.

inline void **add**(const char \*pname, const *argument* &pargument) const

Add a new parameter.

```
struct migraphx_compile_options
```

## Public Functions

```
template<class ...Ts>
inline migraphx_compile_options(Ts&&... xs)
```

## Public Members

migraphx::compile\_options **object**

```
struct program : public migraphx::handle_base<>
```

A program represents the all computation graphs to be compiled and executed.

## Public Functions

```
inline program()
```

```
template<class HandleType, class Lifetime, class = typename
std::enable_if<std::is_convertible<HandleType*, handle_type*>{ }>::type>
inline program(HandleType *p, Lifetime lifetime)
```

```
inline void compile(const target &ptarget, const compile_options &poptions) const
```

Compile the program for a specific target to be ran on.

```
inline void compile(const target &ptarget) const
```

Compile the program for a specific target to be ran on.

```
inline program_parameter_shapes get_parameter_shapes() const
```

Return the shapes for the input parameters.

```
inline shapes get_output_shapes() const
```

Get the shapes of all the outputs returned by this program.

```
inline arguments eval(const program_parameters &pparams) const
```

Run the program using the inputs passed in.

```
template<class Stream>
```

```
inline arguments run_async(const program_parameters &pparams, Stream *s) const
```

Overloaded to allow for execution\_environment input.

```
inline void print() const
```

```
inline program sort()
```

```
inline module get_main_module()
```

```
inline context experimental_get_context()
```

```
inline module create_module(const std::string &name)
```

## Friends

```
inline friend bool operator==(const program &px, const program &py)
```

```
inline friend bool operator!=(const program &px, const program &py)
```

## 6.5 quantize

```
struct quantize_op_names : public migraphx::handle_base<>
```

### Public Functions

```
inline quantize_op_names()
```

```
template<class HandleType, class Lifetime, class = typename  
std::enable_if<std::is_convertible<HandleType*, handle_type*>{ }>::type>  
inline quantize_op_names(HandleType *p, Lifetime lifetime)
```

```
inline void add(const std::string &name)
```

```
inline void migraphx:::quantize_fp16(const program &prog)
```

Quantize program to use fp16.

```
inline void migraphx:::quantize_fp16(const program &prog, const quantize_op_names &names)
```

Quantize program to use fp16.

```
struct quantize_int8_options : public migraphx::handle_base<>
```

Options to be passed when quantizing for int8.

### Public Functions

```
inline quantize_int8_options()
```

```
template<class HandleType, class Lifetime, class = typename  
std::enable_if<std::is_convertible<HandleType*, handle_type*>{ }>::type>  
inline quantize_int8_options(HandleType *p, Lifetime lifetime)
```

```
inline void add_op_name(const std::string &name)
```

Add an operator that should be quantized.

```
inline void add_calibration_data(const program_parameters &pp)
```

Add calibration data to be used for quantizing.

## Public Members

`std::vector<parameter_map> calibration = {}`

`std::unordered_set<std::string> op_names = {}`

## 6.6 parse\_onnx

struct **onnx\_options** : public migraphx::handle\_base<>

Options for parsing onnx options.

### Public Functions

inline **onnx\_options**()

template<class **HandleType**, class **Lifetime**, class = typename  
std::enable\_if<std::is\_convertible<*HandleType*\*, handle\_type\*>{}>::type>  
inline **onnx\_options**(*HandleType* \*p, *Lifetime* lifetime)

inline void **set\_input\_parameter\_shape**(const std::string &name, std::vector<std::size\_t> dim)

Make onnx parser treat an inputs with a certain dimensions.

inline void **set\_dyn\_input\_parameter\_shape**(const std::string &name, const dynamic\_dimensions  
&dyn\_dims)

inline void **set\_default\_dim\_value**(unsigned int value)

When there is a dimension parameter, then use this default value.

inline void **set\_default\_dyn\_dim\_value**(const dynamic\_dimension &dd)

inline void **set\_default\_loop\_iterations**(int64\_t value)

Set default max iteration number for the loop operator.

inline void **set\_limit\_loop\_iterations**(int64\_t value)

Set max iteration limit for the loop operator.

inline *program* migraphx:: **parse\_onnx**(const char \*filename)

Parse an onnx file into a migraphx program.

inline *program* migraphx:: **parse\_onnx**(const char \*filename, const migraphx::*onnx\_options* &options)

Parse an onnx file into a migraphx program.

inline *program* migraphx:: **parse\_onnx\_buffer**(const std::string &buffer)

Parse a buffer of memory as an onnx file.

inline *program* migraphx:: **parse\_onnx\_buffer**(const std::string &buffer, const migraphx::*onnx\_options*  
&options)

Parse a buffer of memory as an onnx file.

inline *program* migraphx:: **parse\_onnx\_buffer**(const void \*data, size\_t size)

Parse a buffer of memory as an onnx file.

```
inline program migraphx::parse_onnx_buffer(const void *data, size_t size, const migraphx::onnx_options
                                         &options)
```

Parse a buffer of memory as an onnx file.

## 6.7 load

```
struct file_options : public migraphx::handle_base<>
```

### Public Functions

```
template<class HandleType, class Lifetime, class = typename
std::enable_if<std::is_convertible<HandleType*, handle_type*>{ }>::type>
inline file_options(HandleType *p, Lifetime lifetime)
```

```
inline file_options()
```

```
inline void set_file_format(const char *format)
```

```
inline program migraphx::load(const char *filename)
```

Load a saved migraphx program from a file.

```
inline program migraphx::load(const char *filename, const file_options &options)
```

Load a saved migraphx program from a file.

## 6.8 save

```
inline void migraphx::save(const program &p, const char *filename)
```

Save a program to a file.

```
inline void migraphx::save(const program &p, const char *filename, const file_options &options)
```

Save a program to a file.



## PYTHON API REFERENCE

### 7.1 shape

**class** `migraphx.shape`(*type*, *lens*, *strides*=None, *dyn\_dims*)

Describes the shape of a tensor. This includes size, layout, and data type. Use `dyn_dims` for a dynamic shape.

`migraphx.type`()

An integer that represents the type.

**Return type**

`int`

`migraphx.lens`()

A list of the lengths of the shape.

**Return type**

`list[int]`

`migraphx.strides`()

A list of the strides of the shape.

**Return type**

`list[int]`

`migraphx.elements`()

The number of elements in the shape.

**Return type**

`int`

`migraphx.dyn_dims`()

The dynamic dimensions of the shape.

**Return type**

`list[dynamic_dimension]`

`migraphx.bytes`()

The number of bytes the shape uses.

**Return type**

`int`

`migraphx.type_size`()

The number of bytes one element uses

**Return type**`int``migraphx.ndim()`

The number of dimensions for the shape.

**Return type**`int``migraphx.packed()`

Returns true if the shape is packed.

**Return type**`bool``migraphx.transposed()`

Returns true if the shape is transposed.

**Return type**`bool``migraphx.broadcasted()`

Returns true if the shape is broadcasted.

**Return type**`bool``migraphx.dynamic()`

Returns true if the shape is dynamic.

**Return type**`bool``migraphx.standard()`

Returns true if the shape is a standard shape. That is, the shape is both packed and not transposed.

**Return type**`bool``migraphx.scalar()`

Returns true if all strides are equal to 0 (scalar tensor).

**Return type**`bool`

## 7.2 dynamic\_dimension

```
class migraphx.dynamic_dimension(min, max, optimals)
```

Constructs a *dynamic\_dimension* from a minimum, a maximum, and optionally a set of optimals.

```
migraphx.is_fixed()
```

Returns true if the *dynamic\_dimension* is fixed.

:rtype : int



## 7.3 argument

**class** `migraphx.argument(data)`

Constructs an argument from a python buffer. This can include numpy arrays.

`migraphx.data_ptr()`

Returns the address to the underlying argument data.

**Return type**

`int`

`migraphx.get_shape()`

Returns the shape of the argument.

**Return type**

`shape`

`migraphx.tolist()`

Converts the elements of the argument to a python list.

**Return type**

`list`

`migraphx.generate_argument(s, seed=0)`

Generates an argument with random data.

**Parameters**

- **s** (`shape`) – Shape of argument to generate.
- **seed** (`int`) – The seed used for random number generation.

**Return type**

`argument`

`migraphx.fill_argument(s, value)`

Fills argument of shape *s* with the given value.

**Parameters**

- **s** (`shape`) – Shape of argument to fill.
- **value** (`int`) – Value to fill in the argument.

**Return type**

`argument`

`migraphx.create_argument(s, values)`

Creates an argument of shape *s* with a set of values.

**Parameters**

- **s** (`shape`) – Shape of argument to create.
- **values** (`list`) – Values to put in the argument. Must be the same number of elements as the shape.

**Return type**

`argument`

`migraphx.argument_from_pointer(shape, address)`

Creates argument from data stored in given address without copy.

**Parameters**

- **shape** (*shape*) – Shape of the data stored in address.
- **address** (*long*) – Memory address of data from another source

**Return type**

*argument*

## 7.4 target

`class migraphx.target`

This represents the compilation target.

`migraphx.get_target(name)`

Constructs the target.

**Parameters**

**name** (*str*) – The name of the target to construct. This can either be ‘gpu’ or ‘ref’.

**Return type**

*target*

## 7.5 module

`migraphx.print()`

Prints the contents of the module as list of instructions.

`migraphx.add_instruction(op, args, mod_args=[])`

Adds instruction into the module.

**Parameters**

- **op** (*operation*) – ‘migraphx.op’ to be added as instruction.
- **args** (*list[instruction]*) – list of inputs to the op.
- **mod\_args** (*list[module]*) – optional list of module arguments to the operator.

:rtype instruction

`migraphx.add_literal(data)`

Adds constant or literal data of provided shape into the module from python buffer which includes numpy array.

**Parameters**

**data** (*py: :buffer*) – Python buffer or numpy array

:rtype instruction

`migraphx.add_parameter(name, shape)`

Adds a parameter to the module with the provided name and shape.

**Parameters**

- **name** (*str*) – name of the parameter.

- **shape** (*shape*) – shape of the parameter.

:rtype instruction

`migraphx.add_return(args)`

Adds a return instruction into the module.

#### Parameters

**args** (*list[instruction]*) – instruction arguments which need to be returned from the module.

:rtype instruction

## 7.6 program

**class** `migraphx.program`

Represents the computation graph to be compiled and run.

`migraphx.clone()`

Makes a copy of the program.

#### Return type

*program*

`migraphx.get_parameter_names()`

Gets all the input argument's or parameter's names to the program as a list.

:rtype list[str]

`migraphx.get_parameter_shapes()`

Gets the shapes of all the input parameters in the program.

#### Return type

dict[str, *shape*]

`migraphx.get_output_shapes()`

Gets the shapes of the final outputs of the program.

#### Return type

list[*shape*]

`migraphx.compile(t, offload_copy=True, fast_math=True, exhaustive_tune=False)`

Compiles the program for the target and optimizes it.

#### Parameters

- **t** (*target*) – Compilation target for the program.
- **offload\_copy** (*bool*) – For targets with offloaded memory (such as the gpu), this will insert instructions during compilation to copy the input parameters to the offloaded memory and to copy the final result from the offloaded memory back to main memory.
- **fast\_math** (*bool*) – Optimize math functions to use faster approximate versions. There may be slight accuracy degradation when enabled.
- **exhaustive\_tune** – Flag to enable exhaustive search to find the fastest version of generated kernels for selected backend.

`migraphx.get_main_module()`

Gets main module of the program.

:rtype module

`migraphx.create_module(name)`

Creates and adds a module with the provided name into the program.

:param str name : name of the new module. :rtype module

`migraphx.run(params)`

Runs the program.

**Parameters**

**params** (`dict[str, argument]`) – Map of the input parameters to be used when running the program.

**Returns**

The result of the last instruction.

**Return type**

`list[argument]`

`migraphx.sort()`

Sorts the modules of the program for the instructions to appear in topologically sorted order.

`migraphx.quantize_fp16(prog, ins_names=['all'])`

Quantizes the program to use fp16.

**Parameters**

- **prog** (`program`) – Program to quantize.
- **ins\_names** (`list[str]`) – List of instructions to quantize.

`migraphx.quantize_int8(prog, t, calibration=[], ins_names=['dot', 'convolution'])`

Quantizes the program to use int8.

**Parameters**

- **prog** (`program`) – Program to quantize.
- **t** (`target`) – Target to be used to run the calibration data.
- **calibration** (`list[dict[str, argument]]`) – Calibration data used to decide the parameters to the int8 optimization.
- **ins\_names** (`list[str]`) – List of instructions to quantize.

## 7.7 op

## 7.8 parse\_onnx

`migraphx.parse_onnx(filename, default_dim_value=1, map_input_dims={}, skip_unknown_operators=false, print_program_on_error=false, max_loop_iterations=10, limit_max_iterations=65535)`

Loads and parses an ONNX file.

**Parameters**

- **filename** (`str`) – Path to file.

- **default\_dim\_value** (*str*) – default dimension to use (if not specified in onnx file).
- **default\_dyn\_dim\_value** (*dynamic\_dimension*) – default dynamic\_dimension value to use.
- **map\_input\_dims** (*str*) – Explicitly specify the dims of an input.
- **map\_dyn\_input\_dims** (*list[dynamic\_dimension]*) – Explicitly specify the dynamic\_dimensions of an input.
- **skip\_unknown\_operators** (*str*) – Continue parsing onnx file if an unknown operator is found.
- **print\_program\_on\_error** (*str*) – Print program if an error occurs.
- **max\_loop\_iterations** (*int*) – Maximum iteration number for the loop operator if trip count is not set.
- **limit\_max\_iterations** (*int*) – Maximum iteration limit for the loop operator.

**Return type**

*program*

## 7.9 parse\_tf

`migraphx.parse_tf(filename, is_nhwc=True, batch_size=1, map_input_dims=dict(), output_names=[])`

Loads and parses a tensorflow protobuf file.

**Parameters**

- **filename** (*str*) – Path to file.
- **is\_nhwc** (*bool*) – Use nhwc as default format.
- **batch\_size** (*str*) – default batch size to use (if not specified in protobuf).
- **map\_input\_dims** (*dict[str, list[int]]*) – Optional arg to explicitly specify dimensions of the inputs.
- **output\_names** (*list[str]*) – Optional argument specify names of the output nodes.

**Return type**

*program*

## 7.10 load

`migraphx.load(filename, format='msgpack')`

Loads a MIGraphX program.

**Parameters**

- **filename** (*str*) – Path to file.
- **format** (*str*) – Format of file. Valid options are msgpack or json.

**Return type**

*program*

## 7.11 save

`migraphx.save(p, filename, format='msgpack')`

Saves a MIGraphX program.

### Parameters

- **p** (`program`) – Program to save.
- **filename** (`str`) – Path to file.
- **format** (`str`) – Format of file. Valid options are msgpack or json.

## **LICENSE**

### The MIT License (MIT)

Copyright © 2015-2024 Advanced Micro Devices, Inc. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.





## PYTHON MODULE INDEX

### m

migraphx, [129](#)



## Symbols

- atol
  - migraphx-driver-verify command line option, 21
- debug
  - command line option, 113
- iterations
  - migraphx-driver-perf command line option, 21
- out
  - command line option, 113
- parse
  - command line option, 113
- per-instruction
  - migraphx-driver-verify command line option, 21
- reduce
  - migraphx-driver-verify command line option, 21
- ref-use-double
  - migraphx-driver-verify command line option, 21
- repeat
  - command line option, 113
- rms-tol
  - migraphx-driver-verify command line option, 21
- rtol
  - migraphx-driver-verify command line option, 21
- run
  - command line option, 113
- study-name
  - command line option, 113
- i
  - migraphx-driver-verify command line option, 21
- n
  - migraphx-driver-perf command line option, 21
- r
  - migraphx-driver-verify command line

option, 21

## A

- add\_instruction() *(in module migraphx)*, 134
- add\_literal() *(in module migraphx)*, 134
- add\_parameter() *(in module migraphx)*, 134
- add\_return() *(in module migraphx)*, 135
- argument *(class in migraphx)*, 133
- argument\_from\_pointer() *(in module migraphx)*, 133

## B

- broadcasted() *(in module migraphx)*, 132
- bytes() *(in module migraphx)*, 131

## C

- clone() *(in module migraphx)*, 135
- command line option
  - debug, 113
  - out, 113
  - parse, 113
  - repeat, 113
  - run, 113
  - study-name, 113
- compile() *(in module migraphx)*, 135
- create\_argument() *(in module migraphx)*, 133
- create\_module() *(in module migraphx)*, 136

## D

- data\_ptr() *(in module migraphx)*, 133
- dyn\_dims() *(in module migraphx)*, 131
- dynamic() *(in module migraphx)*, 132
- dynamic\_dimension *(class in migraphx)*, 132

## E

- elements() *(in module migraphx)*, 131
- environment variable
  - MIGRAPHX\_8BITS\_QUANTIZATION\_PARAMS, 116
  - MIGRAPHX\_CK\_DEBUG, 118
  - MIGRAPHX\_COPY\_LITERALS, 116
  - MIGRAPHX\_DEBUG\_MEMORY\_COLORING, 116
  - MIGRAPHX\_DEBUG\_SAVE\_TEMP\_DIR, 117

MIGRAPHX\_DISABLE\_DNNL\_POST\_OPS\_WORKAROUND, **G**

116

MIGRAPHX\_DISABLE\_FP16\_INSTANCENORM\_CONVERT, **G**

115

MIGRAPHX\_DISABLE\_MIOOPEN\_FUSION, 116

MIGRAPHX\_DISABLE\_MLIR\*, 116

MIGRAPHX\_DISABLE\_POINTWISE\_FUSION, 116

MIGRAPHX\_DISABLE\_REDUCE\_FUSION, 116

MIGRAPHX\_DISABLE\_SCHEDULE\_PASS, 116

MIGRAPHX\_DUMP\_TEST, 119

MIGRAPHX\_ENABLE\_CK, 116

MIGRAPHX\_ENABLE\_EXTRA\_MLIR, 116

MIGRAPHX\_ENABLE\_HIPRTC\_WORKAROUNDS, 117

MIGRAPHX\_ENABLE\_NHWC, 116

MIGRAPHX\_ENABLE\_NULL\_STREAM, 118

MIGRAPHX\_GPU\_COMPILE\_PARALLEL, 117

MIGRAPHX\_GPU\_DEBUG, 117

MIGRAPHX\_GPU\_DEBUG\_SYM, 117

MIGRAPHX\_GPU\_DUMP\_ASM, 117

MIGRAPHX\_GPU\_DUMP\_SRC, 117

MIGRAPHX\_GPU\_OPTIMIZE, 117

MIGRAPHX\_LOG\_CK\_GEMM, 118

MIGRAPHX\_MLIR\_TUNE\_EXHAUSTIVE, 118

MIGRAPHX\_MLIR\_TUNE\_LIMIT, 118

MIGRAPHX\_MLIR\_TUNING\_CFG, 118

MIGRAPHX\_MLIR\_TUNING\_DB, 118

MIGRAPHX\_MLIR\_USE\_SPECIFIC\_OPS, 118

MIGRAPHX\_NSTREAMS, 118

MIGRAPHX\_TIME\_PASSES, 117

MIGRAPHX\_TRACE\_BENCHMARKING, 118

MIGRAPHX\_TRACE\_CMD\_EXECUTE, 117

MIGRAPHX\_TRACE\_COMPILE, 117

MIGRAPHX\_TRACE\_ELIMINATE\_CONTIGUOUS, 116

MIGRAPHX\_TRACE\_EVAL, 115

MIGRAPHX\_TRACE\_FINALIZE, 117

MIGRAPHX\_TRACE\_HIPRTC, 117

MIGRAPHX\_TRACE\_MATCHES, 115

MIGRAPHX\_TRACE\_MATCHES\_FOR, 115

MIGRAPHX\_TRACE\_MLIR, 118

MIGRAPHX\_TRACE\_NARY, 117

MIGRAPHX\_TRACE\_ONNX\_PARSER, 115

MIGRAPHX\_TRACE\_PASSES, 117

MIGRAPHX\_TRACE\_PROPAGATE\_CONSTANT, 116

MIGRAPHX\_TRACE\_SCHEDULE, 116

MIGRAPHX\_TRACE\_TEST, 119

MIGRAPHX\_TRACE\_TEST\_COMPILE, 119

MIGRAPHX\_TUNE\_CK, 118

MIGRAPHX\_USE\_FAST\_SOFTMAX, 118

MIGRAPHX\_VALIDATE\_MATCHES, 115

MIGRAPHX\_VERIFY\_ENABLE\_ALLCLOSE, 115

**F**

fill\_argument() (in module migraphx), 133

generate\_argument() (in module migraphx), 133

get\_main\_module() (in module migraphx), 135

get\_output\_shapes() (in module migraphx), 135

get\_parameter\_names() (in module migraphx), 135

get\_parameter\_shapes() (in module migraphx), 135

get\_shape() (in module migraphx), 133

get\_target() (in module migraphx), 134

**I**

is\_fixed() (in module migraphx), 132

**L**

lens() (in module migraphx), 131

load() (in module migraphx), 137

**M**

migraphx

module, 129

migraphx::argument (C++ struct), 124

migraphx::argument::argument (C++ function), 124

migraphx::argument::as\_vector (C++ function), 124

migraphx::argument::data (C++ function), 124

migraphx::argument::generate (C++ function), 124

migraphx::argument::get\_shape (C++ function), 124

migraphx::argument::operator!= (C++ function), 124

migraphx::argument::operator== (C++ function), 124

migraphx::file\_options (C++ struct), 129

migraphx::file\_options::file\_options (C++ function), 129

migraphx::file\_options::set\_file\_format (C++ function), 129

migraphx::internal::argument (C++ struct), 33

migraphx::internal::argument::argument (C++ function), 33, 34

migraphx::internal::argument::copy (C++ function), 34

migraphx::internal::argument::data (C++ function), 34

migraphx::internal::argument::element (C++ function), 34

migraphx::internal::argument::empty (C++ function), 34

migraphx::internal::argument::fill (C++ function), 34

migraphx::internal::argument::get\_shape (C++ function), 34

migraphx::internal::argument::get\_sub\_objects (C++ function), 34

migraphx::internal::argument::reshape (C++ function), 34	migraphx::internal::eliminate_pad::apply (C++ function), 109
migraphx::internal::argument::share (C++ function), 34	migraphx::internal::eliminate_pad::name (C++ function), 109
migraphx::internal::cpu::target (C++ struct), 106	migraphx::internal::gpu::target (C++ struct), 106
migraphx::internal::cpu::target::allocate (C++ function), 106	migraphx::internal::gpu::target::allocate (C++ function), 106
migraphx::internal::cpu::target::copy_from (C++ function), 106	migraphx::internal::gpu::target::copy_from (C++ function), 106
migraphx::internal::cpu::target::copy_to (C++ function), 106	migraphx::internal::gpu::target::copy_to (C++ function), 106
migraphx::internal::cpu::target::get_context (C++ function), 106	migraphx::internal::gpu::target::get_context (C++ function), 106
migraphx::internal::cpu::target::get_passes (C++ function), 106	migraphx::internal::gpu::target::get_passes (C++ function), 106
migraphx::internal::cpu::target::name (C++ function), 106	migraphx::internal::gpu::target::name (C++ function), 106
migraphx::internal::dead_code_elimination (C++ struct), 107	migraphx::internal::has_finalize (C++ function), 39
migraphx::internal::dead_code_elimination::apply (C++ function), 107	migraphx::internal::instruction (C++ struct), 100
migraphx::internal::dead_code_elimination::name (C++ function), 107	migraphx::internal::instruction::add_output (C++ function), 100
migraphx::internal::eliminate_common_subexpression (C++ struct), 108	migraphx::internal::instruction::backreference (C++ function), 101
migraphx::internal::eliminate_common_subexpression::apply (C++ function), 108	migraphx::internal::instruction::can_eval (C++ function), 100
migraphx::internal::eliminate_common_subexpression::name (C++ function), 108	migraphx::internal::instruction::clear_arguments (C++ function), 100
migraphx::internal::eliminate_concat (C++ struct), 108	migraphx::internal::instruction::debug_print (C++ function), 101
migraphx::internal::eliminate_concat::apply (C++ function), 108	migraphx::internal::instruction::eval (C++ function), 100
migraphx::internal::eliminate_concat::concat_op (C++ member), 108	migraphx::internal::instruction::finalize (C++ function), 100
migraphx::internal::eliminate_concat::name (C++ function), 108	migraphx::internal::instruction::get_literal (C++ function), 100
migraphx::internal::eliminate_contiguous (C++ struct), 108	migraphx::internal::instruction::get_operator (C++ function), 100
migraphx::internal::eliminate_contiguous::apply (C++ function), 108	migraphx::internal::instruction::get_output_alias (C++ function), 101
migraphx::internal::eliminate_contiguous::name (C++ function), 108	migraphx::internal::instruction::get_shape (C++ function), 100
migraphx::internal::eliminate_contiguous::op_name (C++ member), 109	migraphx::internal::instruction::get_target_id (C++ function), 101
migraphx::internal::eliminate_identity (C++ struct), 109	migraphx::internal::instruction::inputs (C++ function), 100
migraphx::internal::eliminate_identity::apply (C++ function), 109	migraphx::internal::instruction::instruction (C++ function), 100
migraphx::internal::eliminate_identity::name (C++ function), 109	migraphx::internal::instruction::is_normalized (C++ function), 100
migraphx::internal::eliminate_pad (C++ struct), 109	migraphx::internal::instruction::is_undefined (C++ function), 100

```

migraphx::internal::instruction::module_inputs      104
    (C++ function), 100
migraphx::internal::instruction::name (C++          migraphx::internal::parse_tf (C++ function), 104
    function), 100                               migraphx::internal::program (C++ struct), 102
migraphx::internal::instruction::need_normalization  function), 102
    (C++ function), 101                               migraphx::internal::program::~~program (C++
migraphx::internal::instruction::normalized_operator function), 103
    (C++ function), 101                               migraphx::internal::program::annotate (C++
migraphx::internal::instruction::operator!=          function), 102
    (C++ function), 101                               migraphx::internal::program::compile (C++
migraphx::internal::instruction::operator==          function), 103
    (C++ function), 101                               migraphx::internal::program::create_module
migraphx::internal::instruction::outputs             (C++ function), 103
    (C++ function), 100                               migraphx::internal::program::debug_print
migraphx::internal::instruction::print (C++          (C++ function), 103
    function), 101                               migraphx::internal::program::dry_run (C++
migraphx::internal::instruction::recompute_shape     function), 103
    (C++ function), 100                               migraphx::internal::program::eval (C++ func-
migraphx::internal::instruction::remove_output       tion), 102
    (C++ function), 100                               migraphx::internal::program::finalize (C++
migraphx::internal::instruction::replace             function), 102
    (C++ function), 100, 101                         migraphx::internal::program::finish (C++
migraphx::internal::instruction::replace_argument    function), 102
    (C++ function), 101                               migraphx::internal::program::from_value
migraphx::internal::instruction::replace_mod_argu-   (C++ function), 103
    ment (C++ function), 101                         migraphx::internal::program::get_context
migraphx::internal::instruction::replace_mod_argu-   (C++ function), 102
    ment (C++ function), 101                         migraphx::internal::program::get_main_module
migraphx::internal::instruction::replace_refs        (C++ function), 103
    (C++ function), 101                               migraphx::internal::program::get_module
migraphx::internal::instruction::set_normalized       (C++ function), 103
    (C++ function), 100                               migraphx::internal::program::get_module_tree
migraphx::internal::instruction::set_target_id       (C++ function), 103
    (C++ function), 101                               migraphx::internal::program::get_modules
migraphx::internal::instruction::valid (C++          (C++ function), 103
    function), 100                               migraphx::internal::program::get_output_shapes
migraphx::internal::instruction_ref (C++             (C++ function), 102
    type), 102                                       migraphx::internal::program::get_parameter
migraphx::internal::is_context_free (C++            (C++ function), 102
    function), 39                                       migraphx::internal::program::get_parameter_names
migraphx::internal::onnx_options (C++ struct),      (C++ function), 102
    104                                               migraphx::internal::program::get_parameter_shape
migraphx::internal::operation (C++ struct), 38      (C++ function), 102
migraphx::internal::operation::compute (C++         migraphx::internal::program::get_parameter_shapes
    function), 38                                       (C++ function), 102
migraphx::internal::operation::compute_shape        migraphx::internal::program::get_target_assignments
    (C++ function), 38                                       (C++ function), 102
migraphx::internal::operation::finalize             migraphx::internal::program::is_compiled
    (C++ function), 38                                       (C++ function), 102
migraphx::internal::operation::name (C++            migraphx::internal::program::mark (C++ func-
    function), 38                                       tion), 102
migraphx::internal::operation::operator<<          migraphx::internal::program::operator!=
    (C++ function), 39                                       (C++ function), 104
migraphx::internal::operation::output_alias         migraphx::internal::program::operator= (C++
    (C++ function), 39                                       function), 102
migraphx::internal::parse_onnx (C++ function),     migraphx::internal::program::operator==

```

(C++ function), 104  
 migraphx::internal::program::operator<<  
 (C++ function), 104  
 migraphx::internal::program::perf\_report  
 (C++ function), 102  
 migraphx::internal::program::print (C++ func-  
 tion), 103  
 migraphx::internal::program::print\_cpp (C++  
 function), 103  
 migraphx::internal::program::print\_graph  
 (C++ function), 103  
 migraphx::internal::program::print\_py (C++  
 function), 103  
 migraphx::internal::program::program (C++  
 function), 102  
 migraphx::internal::program::remove\_module  
 (C++ function), 103  
 migraphx::internal::program::remove\_unused\_modules  
 (C++ function), 103  
 migraphx::internal::program::size (C++ func-  
 tion), 102  
 migraphx::internal::program::sort (C++ func-  
 tion), 103  
 migraphx::internal::program::to\_value (C++  
 function), 103  
 migraphx::internal::program::validate (C++  
 function), 102  
 migraphx::internal::quantize\_fp16 (C++ func-  
 tion), 106  
 migraphx::internal::quantize\_int8 (C++ func-  
 tion), 107  
 migraphx::internal::shape (C++ struct), 26  
 migraphx::internal::shape::any\_of\_dynamic  
 (C++ function), 28  
 migraphx::internal::shape::as (C++ struct), 29  
 migraphx::internal::shape::as::from (C++  
 function), 30  
 migraphx::internal::shape::as::is\_integral  
 (C++ function), 30  
 migraphx::internal::shape::as::is\_signed  
 (C++ function), 30  
 migraphx::internal::shape::as::is\_unsigned  
 (C++ function), 30  
 migraphx::internal::shape::as::max (C++ func-  
 tion), 30  
 migraphx::internal::shape::as::min (C++ func-  
 tion), 30  
 migraphx::internal::shape::as::nan (C++ func-  
 tion), 30  
 migraphx::internal::shape::as::operator()  
 (C++ function), 30  
 migraphx::internal::shape::as::size (C++  
 function), 30  
 migraphx::internal::shape::as::type (C++  
 type), 30  
 migraphx::internal::shape::as::type\_enum  
 (C++ function), 30  
 migraphx::internal::shape::as\_standard (C++  
 function), 28  
 migraphx::internal::shape::broadcasted (C++  
 function), 28  
 migraphx::internal::shape::bytes (C++ func-  
 tion), 27  
 migraphx::internal::shape::cpp\_type (C++  
 function), 29  
 migraphx::internal::shape::dyn\_dims (C++  
 function), 27  
 migraphx::internal::shape::dynamic (C++ func-  
 tion), 28  
 migraphx::internal::shape::dynamic\_dimension  
 (C++ struct), 30  
 migraphx::internal::shape::dynamic\_dimension::has\_optimal  
 (C++ function), 30  
 migraphx::internal::shape::dynamic\_dimension::is\_fixed  
 (C++ function), 30  
 migraphx::internal::shape::dynamic\_dimension::max  
 (C++ member), 31  
 migraphx::internal::shape::dynamic\_dimension::min  
 (C++ member), 31  
 migraphx::internal::shape::dynamic\_dimension::operator!=  
 (C++ function), 31  
 migraphx::internal::shape::dynamic\_dimension::operator+  
 (C++ function), 31  
 migraphx::internal::shape::dynamic\_dimension::operator+=  
 (C++ function), 30  
 migraphx::internal::shape::dynamic\_dimension::operator==  
 (C++ function), 31  
 migraphx::internal::shape::dynamic\_dimension::operator-  
 (C++ function), 31  
 migraphx::internal::shape::dynamic\_dimension::operator-=  
 (C++ function), 30  
 migraphx::internal::shape::dynamic\_dimension::operator<<  
 (C++ function), 31  
 migraphx::internal::shape::dynamic\_dimension::optimals  
 (C++ member), 31  
 migraphx::internal::shape::dynamic\_dimension::reflect  
 (C++ function), 31  
 migraphx::internal::shape::element\_space  
 (C++ function), 28  
 migraphx::internal::shape::elements (C++  
 function), 27  
 migraphx::internal::shape::from\_permutation  
 (C++ function), 29  
 migraphx::internal::shape::get\_type (C++  
 struct), 31  
 migraphx::internal::shape::get\_type<bool,  
 T> (C++ struct), 31  
 migraphx::internal::shape::get\_type<const

`T> (C++ struct), 31`  
`migraphx::internal::shape::get_type<double, T> (C++ struct), 31`  
`migraphx::internal::shape::get_type<float, T> (C++ struct), 31`  
`migraphx::internal::shape::get_type<half, T> (C++ struct), 32`  
`migraphx::internal::shape::get_type<int16_t, T> (C++ struct), 32`  
`migraphx::internal::shape::get_type<int32_t, T> (C++ struct), 32`  
`migraphx::internal::shape::get_type<int64_t, T> (C++ struct), 32`  
`migraphx::internal::shape::get_type<int8_t, T> (C++ struct), 32`  
`migraphx::internal::shape::get_type<uint16_t, T> (C++ struct), 32`  
`migraphx::internal::shape::get_type<uint32_t, T> (C++ struct), 32`  
`migraphx::internal::shape::get_type<uint64_t, T> (C++ struct), 32`  
`migraphx::internal::shape::get_type<uint8_t, T> (C++ struct), 32`  
`migraphx::internal::shape::index (C++ function), 27, 28`  
`migraphx::internal::shape::lens (C++ function), 27`  
`migraphx::internal::shape::max_lens (C++ function), 27`  
`migraphx::internal::shape::min_lens (C++ function), 27`  
`migraphx::internal::shape::multi (C++ function), 28`  
`migraphx::internal::shape::multi_copy (C++ function), 28`  
`migraphx::internal::shape::name (C++ function), 29`  
`migraphx::internal::shape::ndim (C++ function), 27`  
`migraphx::internal::shape::normalize_standard (C++ function), 28`  
`migraphx::internal::shape::operator!= (C++ function), 29`  
`migraphx::internal::shape::operator== (C++ function), 29`  
`migraphx::internal::shape::operator<< (C++ function), 29`  
`migraphx::internal::shape::opt_lens (C++ function), 27`  
`migraphx::internal::shape::packed (C++ function), 28`  
`migraphx::internal::shape::parse_type (C++ function), 29`  
`migraphx::internal::shape::scalar (C++ function), 28`  
`migraphx::internal::shape::shape (C++ function), 27`  
`migraphx::internal::shape::standard (C++ function), 28`  
`migraphx::internal::shape::strides (C++ function), 27`  
`migraphx::internal::shape::sub_shapes (C++ function), 28`  
`migraphx::internal::shape::to_dynamic (C++ function), 28`  
`migraphx::internal::shape::to_static (C++ function), 28`  
`migraphx::internal::shape::transposed (C++ function), 28`  
`migraphx::internal::shape::type (C++ function), 27`  
`migraphx::internal::shape::type_size (C++ function), 27`  
`migraphx::internal::shape::type_string (C++ function), 28`  
`migraphx::internal::shape::type_t (C++ enum), 26`  
`migraphx::internal::shape::type_t::bool_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::double_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::float_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::fp8e4m3fnuz_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::half_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::int16_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::int32_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::int64_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::int8_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::tuple_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::uint16_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::uint32_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::uint64_type (C++ enumerator), 26`  
`migraphx::internal::shape::type_t::uint8_type (C++ enumerator), 26`  
`migraphx::internal::shape::types (C++ function), 29`  
`migraphx::internal::shape::visit (C++ function), 28`



tion), 29

migraphx::internal::shape::visit\_type (C++ function), 28

migraphx::internal::shape::visit\_types (C++ function), 29

migraphx::internal::shape::with\_lens (C++ function), 28

migraphx::internal::shape::with\_type (C++ function), 28

migraphx::internal::target (C++ struct), 104

migraphx::internal::target::allocate (C++ function), 105

migraphx::internal::target::copy\_from (C++ function), 105

migraphx::internal::target::copy\_to (C++ function), 105

migraphx::internal::target::get\_context (C++ function), 105

migraphx::internal::target::get\_passes (C++ function), 104

migraphx::internal::target::name (C++ function), 104

migraphx::internal::target::target\_is\_supported (C++ function), 105

migraphx::internal::tensor\_view (C++ struct), 37

migraphx::internal::tensor\_view::back (C++ function), 38

migraphx::internal::tensor\_view::begin (C++ function), 38

migraphx::internal::tensor\_view::const\_iterator (C++ type), 37

migraphx::internal::tensor\_view::data (C++ function), 37

migraphx::internal::tensor\_view::empty (C++ function), 37

migraphx::internal::tensor\_view::end (C++ function), 38

migraphx::internal::tensor\_view::front (C++ function), 38

migraphx::internal::tensor\_view::get\_shape (C++ function), 37

migraphx::internal::tensor\_view::iterator (C++ type), 37

migraphx::internal::tensor\_view::operator() (C++ function), 37

migraphx::internal::tensor\_view::operator<< (C++ function), 38

migraphx::internal::tensor\_view::operator[] (C++ function), 37

migraphx::internal::tensor\_view::size (C++ function), 37

migraphx::internal::tensor\_view::tensor\_view (C++ function), 37

migraphx::internal::tensor\_view::to\_vector (C++ function), 38

migraphx::internal::tensor\_view::value\_type (C++ type), 37

migraphx::internal::tf\_options (C++ struct), 104

migraphx::internal::visit\_all (C++ function), 36

migraphx::literal (C++ struct), 32

migraphx::literal::data (C++ function), 33

migraphx::literal::empty (C++ function), 33

migraphx::literal::get\_argument (C++ function), 33

migraphx::literal::get\_shape (C++ function), 33

migraphx::literal::get\_sub\_objects (C++ function), 33

migraphx::literal::literal (C++ function), 33

migraphx::load (C++ function), 129

migraphx::onnx\_options (C++ struct), 128

migraphx::onnx\_options::onnx\_options (C++ function), 128

migraphx::onnx\_options::set\_default\_dim\_value (C++ function), 128

migraphx::onnx\_options::set\_default\_dyn\_dim\_value (C++ function), 128

migraphx::onnx\_options::set\_default\_loop\_iterations (C++ function), 128

migraphx::onnx\_options::set\_dyn\_input\_parameter\_shape (C++ function), 128

migraphx::onnx\_options::set\_input\_parameter\_shape (C++ function), 128

migraphx::onnx\_options::set\_limit\_loop\_iterations (C++ function), 128

migraphx::op (C++ type), 39

migraphx::op::abs (C++ struct), 41

migraphx::op::abs::apply (C++ function), 41

migraphx::op::acos (C++ struct), 41

migraphx::op::acos::apply (C++ function), 41

migraphx::op::acosh (C++ struct), 41

migraphx::op::acosh::apply (C++ function), 41

migraphx::op::add (C++ struct), 41

migraphx::op::add::apply (C++ function), 41

migraphx::op::add::attributes (C++ function), 41

migraphx::op::add::point\_function (C++ function), 41

migraphx::op::allocate (C++ struct), 41

migraphx::op::allocate::buf\_type (C++ member), 42

migraphx::op::allocate::compute (C++ function), 42

migraphx::op::allocate::compute\_shape (C++ function), 42

migraphx::op::allocate::name (C++ function), 42

migraphx::op::allocate::reflect (C++ function), 42

migraphx::op::allocate::s (C++ member), 42  
 migraphx::op::argmax (C++ struct), 42  
 migraphx::op::argmax::attributes (C++ function), 42  
 migraphx::op::argmax::axis (C++ member), 42  
 migraphx::op::argmax::calc\_argmax (C++ function), 42  
 migraphx::op::argmax::compute (C++ function), 42  
 migraphx::op::argmax::name (C++ function), 42  
 migraphx::op::argmax::normalize\_compute\_shape (C++ function), 42  
 migraphx::op::argmax::reflect (C++ function), 43  
 migraphx::op::argmax::select\_last\_index (C++ member), 42  
 migraphx::op::argmin (C++ struct), 43  
 migraphx::op::argmin::attributes (C++ function), 43  
 migraphx::op::argmin::axis (C++ member), 43  
 migraphx::op::argmin::calc\_argmin (C++ function), 43  
 migraphx::op::argmin::compute (C++ function), 43  
 migraphx::op::argmin::name (C++ function), 43  
 migraphx::op::argmin::normalize\_compute\_shape (C++ function), 43  
 migraphx::op::argmin::reflect (C++ function), 43  
 migraphx::op::argmin::select\_last\_index (C++ member), 43  
 migraphx::op::as\_shape (C++ struct), 43  
 migraphx::op::as\_shape::compute (C++ function), 43  
 migraphx::op::as\_shape::compute\_shape (C++ function), 43  
 migraphx::op::as\_shape::name (C++ function), 43  
 migraphx::op::as\_shape::output\_alias (C++ function), 43  
 migraphx::op::as\_shape::reflect (C++ function), 44  
 migraphx::op::as\_shape::s (C++ member), 44  
 migraphx::op::asin (C++ struct), 44  
 migraphx::op::asin::apply (C++ function), 44  
 migraphx::op::asinh (C++ struct), 44  
 migraphx::op::asinh::apply (C++ function), 44  
 migraphx::op::atan (C++ struct), 44  
 migraphx::op::atan::apply (C++ function), 44  
 migraphx::op::atanh (C++ struct), 44  
 migraphx::op::atanh::apply (C++ function), 44  
 migraphx::op::binary (C++ struct), 44  
 migraphx::op::binary::attributes (C++ function), 45  
 migraphx::op::binary::base\_attributes (C++ function), 45  
 migraphx::op::binary::compute (C++ function), 45  
 migraphx::op::binary::compute\_shape (C++ function), 45  
 migraphx::op::binary::point\_function (C++ function), 45  
 migraphx::op::binary::point\_op (C++ function), 45  
 migraphx::op::broadcast (C++ struct), 45  
 migraphx::op::broadcast::axis (C++ member), 45  
 migraphx::op::broadcast::broadcast\_lens (C++ member), 45  
 migraphx::op::broadcast::compute (C++ function), 45  
 migraphx::op::broadcast::compute\_shape (C++ function), 45  
 migraphx::op::broadcast::name (C++ function), 45  
 migraphx::op::broadcast::output\_alias (C++ function), 45  
 migraphx::op::broadcast::reflect (C++ function), 45  
 migraphx::op::capture (C++ struct), 45  
 migraphx::op::capture::compute (C++ function), 46  
 migraphx::op::capture::compute\_shape (C++ function), 46  
 migraphx::op::capture::f (C++ member), 46  
 migraphx::op::capture::ins\_index (C++ member), 46  
 migraphx::op::capture::name (C++ function), 46  
 migraphx::op::capture::output\_alias (C++ function), 46  
 migraphx::op::capture::reflect (C++ function), 46  
 migraphx::op::ceil (C++ struct), 46  
 migraphx::op::ceil::apply (C++ function), 46  
 migraphx::op::clip (C++ struct), 46  
 migraphx::op::clip::attributes (C++ function), 46  
 migraphx::op::clip::compute (C++ function), 46  
 migraphx::op::clip::compute\_shape (C++ function), 46  
 migraphx::op::clip::name (C++ function), 46  
 migraphx::op::concat (C++ struct), 46  
 migraphx::op::concat::attributes (C++ function), 47  
 migraphx::op::concat::axis (C++ member), 47  
 migraphx::op::concat::compute (C++ function), 47  
 migraphx::op::concat::compute\_offsets (C++ function), 47  
 migraphx::op::concat::name (C++ function), 47  
 migraphx::op::concat::normalize\_compute\_shape (C++ function), 47  
 migraphx::op::concat::reflect (C++ function), 47  
 migraphx::op::contiguous (C++ struct), 47  
 migraphx::op::contiguous::apply (C++ function), 47  
 migraphx::op::contiguous::compute (C++ function), 47

tion), 47  
 migraphx::op::contiguous::compute\_shape  
 (C++ function), 47  
 migraphx::op::contiguous::name (C++ function),  
 47  
 migraphx::op::convert (C++ struct), 47  
 migraphx::op::convert::apply (C++ function), 48  
 migraphx::op::convert::compute\_shape (C++  
 function), 48  
 migraphx::op::convert::convert (C++ function),  
 48  
 migraphx::op::convert::point\_op (C++ function),  
 48  
 migraphx::op::convert::reflect (C++ function),  
 48  
 migraphx::op::convert::target\_type (C++ mem-  
 ber), 48  
 migraphx::op::convolution (C++ struct), 48  
 migraphx::op::convolution::attributes (C++  
 function), 48  
 migraphx::op::convolution::calc\_conv\_lens  
 (C++ function), 48  
 migraphx::op::convolution::check\_attribute\_size  
 (C++ function), 48  
 migraphx::op::convolution::compute (C++ func-  
 tion), 48  
 migraphx::op::convolution::dilation (C++  
 member), 49  
 migraphx::op::convolution::dynamic\_compute\_shape  
 (C++ function), 48  
 migraphx::op::convolution::group (C++ mem-  
 ber), 49  
 migraphx::op::convolution::kdims (C++ func-  
 tion), 48  
 migraphx::op::convolution::name (C++ function),  
 48  
 migraphx::op::convolution::normalize\_compute\_shape  
 (C++ function), 48  
 migraphx::op::convolution::padding (C++ mem-  
 ber), 49  
 migraphx::op::convolution::padding\_mode  
 (C++ member), 49  
 migraphx::op::convolution::reflect (C++ func-  
 tion), 49  
 migraphx::op::convolution::static\_compute\_shape  
 (C++ function), 48  
 migraphx::op::convolution::stride (C++ mem-  
 ber), 49  
 migraphx::op::convolution\_backwards (C++  
 struct), 49  
 migraphx::op::convolution\_backwards::calc\_spatial\_dims  
 (C++ function), 49  
 migraphx::op::convolution\_backwards::check\_attribute\_size  
 (C++ function), 49  
 migraphx::op::convolution\_backwards::compute  
 (C++ function), 49  
 migraphx::op::convolution\_backwards::compute\_shape  
 (C++ function), 49  
 migraphx::op::convolution\_backwards::dilation  
 (C++ member), 49  
 migraphx::op::convolution\_backwards::dynamic\_compute\_shape  
 (C++ function), 49  
 migraphx::op::convolution\_backwards::group  
 (C++ member), 50  
 migraphx::op::convolution\_backwards::kdims  
 (C++ function), 49  
 migraphx::op::convolution\_backwards::name  
 (C++ function), 49  
 migraphx::op::convolution\_backwards::padding  
 (C++ member), 49  
 migraphx::op::convolution\_backwards::padding\_mode  
 (C++ member), 49  
 migraphx::op::convolution\_backwards::reflect  
 (C++ function), 50  
 migraphx::op::convolution\_backwards::static\_compute\_shape  
 (C++ function), 49  
 migraphx::op::convolution\_backwards::stride  
 (C++ member), 49  
 migraphx::op::cos (C++ struct), 50  
 migraphx::op::cos::apply (C++ function), 50  
 migraphx::op::cosh (C++ struct), 50  
 migraphx::op::cosh::apply (C++ function), 50  
 migraphx::op::dequantizelinear (C++ struct), 50  
 migraphx::op::dequantizelinear::attributes  
 (C++ function), 50  
 migraphx::op::dequantizelinear::compute  
 (C++ function), 50  
 migraphx::op::dequantizelinear::compute\_shape  
 (C++ function), 50  
 migraphx::op::dequantizelinear::name (C++  
 function), 50  
 migraphx::op::dimensions\_of (C++ struct), 50  
 migraphx::op::dimensions\_of::compute (C++  
 function), 51  
 migraphx::op::dimensions\_of::compute\_shape  
 (C++ function), 51  
 migraphx::op::dimensions\_of::end (C++ mem-  
 ber), 51  
 migraphx::op::dimensions\_of::name (C++ func-  
 tion), 51  
 migraphx::op::dimensions\_of::reflect (C++  
 function), 51  
 migraphx::op::dimensions\_of::start (C++ mem-  
 ber), 51  
 migraphx::op::div (C++ struct), 51  
 migraphx::op::div::apply (C++ function), 51  
 migraphx::op::div::point\_function (C++ func-  
 tion), 51

```

migraphx::op::dot (C++ struct), 51
migraphx::op::dot::compute (C++ function), 51
migraphx::op::dot::compute_shape (C++ function), 51
migraphx::op::dot::name (C++ function), 51
migraphx::op::elu (C++ struct), 51
migraphx::op::elu::alpha (C++ member), 52
migraphx::op::elu::apply (C++ function), 52
migraphx::op::elu::point_op (C++ function), 52
migraphx::op::elu::reflect (C++ function), 52
migraphx::op::equal (C++ struct), 52
migraphx::op::equal::apply (C++ function), 52
migraphx::op::equal::attributes (C++ function), 52
migraphx::op::equal::point_function (C++ function), 52
migraphx::op::erf (C++ struct), 52
migraphx::op::erf::apply (C++ function), 52
migraphx::op::exp (C++ struct), 52
migraphx::op::exp::apply (C++ function), 52
migraphx::op::fill (C++ struct), 52
migraphx::op::fill::compute (C++ function), 53
migraphx::op::fill::compute_shape (C++ function), 53
migraphx::op::fill::name (C++ function), 53
migraphx::op::fill::output_alias (C++ function), 53
migraphx::op::flatten (C++ struct), 53
migraphx::op::flatten::attributes (C++ function), 53
migraphx::op::flatten::axis (C++ member), 53
migraphx::op::flatten::compute (C++ function), 53
migraphx::op::flatten::name (C++ function), 53
migraphx::op::flatten::normalize_compute_shape (C++ function), 53
migraphx::op::flatten::output_alias (C++ function), 53
migraphx::op::flatten::reflect (C++ function), 53
migraphx::op::floor (C++ struct), 53
migraphx::op::floor::apply (C++ function), 53
migraphx::op::fmod (C++ struct), 53
migraphx::op::fmod::apply (C++ function), 54
migraphx::op::fmod::attributes (C++ function), 54
migraphx::op::fmod::name (C++ function), 54
migraphx::op::gather (C++ struct), 54
migraphx::op::gather::attributes (C++ function), 54
migraphx::op::gather::axis (C++ member), 54
migraphx::op::gather::compute (C++ function), 54
migraphx::op::gather::name (C++ function), 54
migraphx::op::gather::normalize_compute_shape (C++ function), 54
migraphx::op::gather::reflect (C++ function), 54
migraphx::op::gathernd (C++ struct), 54
migraphx::op::gathernd::batch_dims (C++ member), 55
migraphx::op::gathernd::compute (C++ function), 54
migraphx::op::gathernd::compute_shape (C++ function), 54
migraphx::op::gathernd::name (C++ function), 54
migraphx::op::gathernd::reflect (C++ function), 55
migraphx::op::get_tuple_elem (C++ struct), 55
migraphx::op::get_tuple_elem::compute (C++ function), 55
migraphx::op::get_tuple_elem::compute_shape (C++ function), 55
migraphx::op::get_tuple_elem::index (C++ member), 55
migraphx::op::get_tuple_elem::name (C++ function), 55
migraphx::op::get_tuple_elem::output_alias (C++ function), 55
migraphx::op::get_tuple_elem::reflect (C++ function), 55
migraphx::op::greater (C++ struct), 55
migraphx::op::greater::apply (C++ function), 55
migraphx::op::greater::point_function (C++ function), 55
migraphx::op::gru (C++ struct), 55
migraphx::op::gru::actv_funcs (C++ member), 56
migraphx::op::gru::clip (C++ member), 56
migraphx::op::gru::compute_shape (C++ function), 56
migraphx::op::gru::direction (C++ member), 56
migraphx::op::gru::hidden_size (C++ member), 56
migraphx::op::gru::linear_before_reset (C++ member), 56
migraphx::op::gru::name (C++ function), 56
migraphx::op::gru::reflect (C++ function), 56
migraphx::op::highest (C++ struct), 56
migraphx::op::highest::operator T (C++ function), 56
migraphx::op::identity (C++ struct), 56
migraphx::op::identity::attributes (C++ function), 56
migraphx::op::identity::compute (C++ function), 56
migraphx::op::identity::compute_shape (C++ function), 56
migraphx::op::identity::name (C++ function), 56
migraphx::op::identity::output_alias (C++

```

*function*), 56  
 migraphx::op::if\_op (C++ struct), 56  
 migraphx::op::if\_op::compute (C++ function), 57  
 migraphx::op::if\_op::compute\_shape (C++ function), 57  
 migraphx::op::if\_op::name (C++ function), 57  
 migraphx::op::im2col (C++ struct), 57  
 migraphx::op::im2col::attributes (C++ function), 57  
 migraphx::op::im2col::dilation (C++ member), 57  
 migraphx::op::im2col::name (C++ function), 57  
 migraphx::op::im2col::normalize\_compute\_shape (C++ function), 57  
 migraphx::op::im2col::padding (C++ member), 57  
 migraphx::op::im2col::padding\_mode (C++ member), 57  
 migraphx::op::im2col::reflect (C++ function), 57  
 migraphx::op::im2col::stride (C++ member), 57  
 migraphx::op::isinf (C++ struct), 57  
 migraphx::op::isinf::apply (C++ function), 58  
 migraphx::op::isinf::compute\_shape (C++ function), 58  
 migraphx::op::isinf::name (C++ function), 58  
 migraphx::op::isnan (C++ struct), 58  
 migraphx::op::isnan::apply (C++ function), 58  
 migraphx::op::isnan::compute\_shape (C++ function), 58  
 migraphx::op::isnan::name (C++ function), 58  
 migraphx::op::layout (C++ struct), 58  
 migraphx::op::layout::apply (C++ function), 58  
 migraphx::op::layout::compute\_shape (C++ function), 58  
 migraphx::op::layout::permutation (C++ member), 58  
 migraphx::op::layout::reflect (C++ function), 58  
 migraphx::op::leaky\_relu (C++ struct), 58  
 migraphx::op::leaky\_relu::alpha (C++ member), 59  
 migraphx::op::leaky\_relu::apply (C++ function), 59  
 migraphx::op::leaky\_relu::name (C++ function), 59  
 migraphx::op::leaky\_relu::point\_op (C++ function), 59  
 migraphx::op::leaky\_relu::reflect (C++ function), 59  
 migraphx::op::less (C++ struct), 59  
 migraphx::op::less::apply (C++ function), 59  
 migraphx::op::less::point\_function (C++ function), 59  
 migraphx::op::load (C++ struct), 59  
 migraphx::op::load::compute (C++ function), 59  
 migraphx::op::load::compute\_shape (C++ function), 59  
 migraphx::op::load::get\_lifetime (C++ function), 59  
 migraphx::op::load::name (C++ function), 59  
 migraphx::op::load::offset (C++ member), 60  
 migraphx::op::load::operator<< (C++ function), 60  
 migraphx::op::load::output\_alias (C++ function), 59  
 migraphx::op::load::reflect (C++ function), 60  
 migraphx::op::load::s (C++ member), 60  
 migraphx::op::log (C++ struct), 60  
 migraphx::op::log::apply (C++ function), 60  
 migraphx::op::logical\_and (C++ struct), 60  
 migraphx::op::logical\_and::apply (C++ function), 60  
 migraphx::op::logical\_and::point\_function (C++ function), 60  
 migraphx::op::logical\_or (C++ struct), 60  
 migraphx::op::logical\_or::apply (C++ function), 60  
 migraphx::op::logical\_or::point\_function (C++ function), 60  
 migraphx::op::logical\_xor (C++ struct), 60  
 migraphx::op::logical\_xor::apply (C++ function), 61  
 migraphx::op::logical\_xor::point\_function (C++ function), 61  
 migraphx::op::logsoftmax (C++ struct), 61  
 migraphx::op::logsoftmax::attributes (C++ function), 61  
 migraphx::op::logsoftmax::axis (C++ member), 61  
 migraphx::op::logsoftmax::name (C++ function), 61  
 migraphx::op::logsoftmax::normalize\_compute\_shape (C++ function), 61  
 migraphx::op::logsoftmax::output (C++ function), 61  
 migraphx::op::logsoftmax::reflect (C++ function), 61  
 migraphx::op::loop (C++ struct), 61  
 migraphx::op::loop::compute (C++ function), 61  
 migraphx::op::loop::compute\_shape (C++ function), 61  
 migraphx::op::loop::max\_iterations (C++ member), 62  
 migraphx::op::loop::name (C++ function), 61  
 migraphx::op::loop::ref\_loop (C++ struct), 62  
 migraphx::op::loop::ref\_loop::append (C++ function), 62  
 migraphx::op::loop::ref\_loop::copy (C++ function), 62



```

migraphx::op::loop::ref_loop::get_output_params (C++ function), 62
migraphx::op::loop::ref_loop::max_iterations (C++ member), 62
migraphx::op::loop::ref_loop::set_zero (C++ function), 62
migraphx::op::loop::reflect (C++ function), 62
migraphx::op::lowest (C++ struct), 62
migraphx::op::lowest::operator T (C++ function), 62
migraphx::op::lrn (C++ struct), 62
migraphx::op::lrn::alpha (C++ member), 63
migraphx::op::lrn::beta (C++ member), 63
migraphx::op::lrn::bias (C++ member), 63
migraphx::op::lrn::compute_shape (C++ function), 63
migraphx::op::lrn::name (C++ function), 63
migraphx::op::lrn::reflect (C++ function), 63
migraphx::op::lrn::size (C++ member), 63
migraphx::op::lstm (C++ struct), 63
migraphx::op::lstm::actv_funcs (C++ member), 63
migraphx::op::lstm::clip (C++ member), 63
migraphx::op::lstm::compute_shape (C++ function), 63
migraphx::op::lstm::direction (C++ member), 63
migraphx::op::lstm::hidden_size (C++ member), 63
migraphx::op::lstm::input_forget (C++ member), 63
migraphx::op::lstm::name (C++ function), 63
migraphx::op::lstm::reflect (C++ function), 64
migraphx::op::max (C++ struct), 64
migraphx::op::max::apply (C++ function), 64
migraphx::op::max::attributes (C++ function), 64
migraphx::op::min (C++ struct), 64
migraphx::op::min::apply (C++ function), 64
migraphx::op::min::attributes (C++ function), 64
migraphx::op::mod (C++ struct), 64
migraphx::op::mod::apply (C++ function), 64
migraphx::op::mod::attributes (C++ function), 64
migraphx::op::mod::name (C++ function), 64
migraphx::op::mul (C++ struct), 64
migraphx::op::mul::apply (C++ function), 64
migraphx::op::mul::attributes (C++ function), 64
migraphx::op::mul::point_function (C++ function), 64
migraphx::op::multibroadcast (C++ struct), 64
migraphx::op::multibroadcast::compute (C++ function), 65
migraphx::op::multibroadcast::compute_shape (C++ function), 65
migraphx::op::multibroadcast::name (C++ function), 65
migraphx::op::multibroadcast::output_alias (C++ function), 65
migraphx::op::multibroadcast::output_dyn_dims (C++ member), 65
migraphx::op::multibroadcast::output_lens (C++ member), 65
migraphx::op::multibroadcast::reflect (C++ function), 65
migraphx::op::multinomial (C++ struct), 65
migraphx::op::multinomial::compute (C++ function), 65
migraphx::op::multinomial::compute_shape (C++ function), 65
migraphx::op::multinomial::dtype (C++ member), 65
migraphx::op::multinomial::name (C++ function), 65
migraphx::op::multinomial::reflect (C++ function), 65
migraphx::op::nearbyint (C++ struct), 65
migraphx::op::nearbyint::apply (C++ function), 66
migraphx::op::neg (C++ struct), 66
migraphx::op::neg::apply (C++ function), 66
migraphx::op::neg::point_function (C++ function), 66
migraphx::op::nonmaxsuppression (C++ struct), 66
migraphx::op::nonmaxsuppression::batch_box (C++ function), 66
migraphx::op::nonmaxsuppression::box (C++ struct), 67
migraphx::op::nonmaxsuppression::box::area (C++ function), 67
migraphx::op::nonmaxsuppression::box::operator[] (C++ function), 67
migraphx::op::nonmaxsuppression::box::sort (C++ function), 67
migraphx::op::nonmaxsuppression::box::x (C++ member), 67
migraphx::op::nonmaxsuppression::box::y (C++ member), 67
migraphx::op::nonmaxsuppression::center_point_box (C++ member), 66
migraphx::op::nonmaxsuppression::compute (C++ function), 66
migraphx::op::nonmaxsuppression::compute_nms (C++ function), 66
migraphx::op::nonmaxsuppression::compute_shape (C++ function), 66
migraphx::op::nonmaxsuppression::filter_boxes_by_score (C++ function), 66
migraphx::op::nonmaxsuppression::name (C++ function), 66

```

---

```

migraphx::op::nonmaxsuppression::reflect      (C++ function), 67
migraphx::op::nonmaxsuppression::suppress_by_image (C++ function), 66
migraphx::op::nonmaxsuppression::use_dyn_output (C++ member), 66
migraphx::op::nonzero (C++ struct), 67
migraphx::op::nonzero::compute (C++ function), 67
migraphx::op::nonzero::compute_shape (C++ function), 67
migraphx::op::nonzero::name (C++ function), 67
migraphx::op::normalize_attribute (C++ enum), 40
migraphx::op::normalize_attribute::clip_max (C++ enumerator), 40
migraphx::op::normalize_attribute::clip_min (C++ enumerator), 40
migraphx::op::normalize_attribute::include_max (C++ enumerator), 40
migraphx::op::normalize_attribute::include_min (C++ enumerator), 40
migraphx::op::normalize_attribute::normalize_padding (C++ enumerator), 40
migraphx::op::normalize_attribute::use_len (C++ enumerator), 40
migraphx::op::normalize_attribute::use_output (C++ enumerator), 40
migraphx::op::one (C++ struct), 67
migraphx::op::one::operator T (C++ function), 67
migraphx::op::op_name (C++ struct), 67
migraphx::op::op_name::name (C++ function), 68
migraphx::op::operator<< (C++ function), 41
migraphx::op::outline (C++ struct), 68
migraphx::op::outline::compute (C++ function), 68
migraphx::op::outline::compute_shape (C++ function), 68
migraphx::op::outline::name (C++ function), 68
migraphx::op::outline::reflect (C++ function), 69
migraphx::op::outline::s (C++ member), 69
migraphx::op::pad (C++ struct), 69
migraphx::op::pad::compute_shape (C++ function), 69
migraphx::op::pad::mode (C++ member), 69
migraphx::op::pad::name (C++ function), 69
migraphx::op::pad::pad_ndims (C++ function), 69
migraphx::op::pad::pad_op_mode_t (C++ enum), 69
migraphx::op::pad::pad_op_mode_t::constant_pad (C++ enumerator), 69
migraphx::op::pad::pad_op_mode_t::edge_pad (C++ enumerator), 69
migraphx::op::pad::pad_op_mode_t::reflect_pad (C++ enumerator), 69
migraphx::op::pad::pads (C++ member), 69
migraphx::op::pad::reflect (C++ function), 70
migraphx::op::pad::symmetric (C++ function), 69
migraphx::op::pad::value (C++ member), 69
migraphx::op::padding_mode_t (C++ enum), 39
migraphx::op::padding_mode_t::default_ (C++ enumerator), 39
migraphx::op::padding_mode_t::same_lower (C++ enumerator), 39
migraphx::op::padding_mode_t::same_upper (C++ enumerator), 39
migraphx::op::pointwise (C++ struct), 70
migraphx::op::pointwise::compute (C++ function), 70
migraphx::op::pointwise::compute_shape (C++ function), 70
migraphx::op::pointwise::name (C++ function), 70
migraphx::op::pooling (C++ struct), 70
migraphx::op::pooling::attributes (C++ function), 70
migraphx::op::pooling::avg_pool (C++ struct), 71
migraphx::op::pooling::avg_pool::final (C++ function), 71
migraphx::op::pooling::avg_pool::init (C++ function), 71
migraphx::op::pooling::avg_pool::operator() (C++ function), 71
migraphx::op::pooling::calc_pooling (C++ function), 70
migraphx::op::pooling::calc_spatial_dim_out (C++ function), 70
migraphx::op::pooling::ceil_mode (C++ member), 71
migraphx::op::pooling::check_attribute_size (C++ function), 70
migraphx::op::pooling::compute (C++ function), 70
migraphx::op::pooling::count_include_pad (C++ member), 71
migraphx::op::pooling::dilate_dim (C++ function), 70
migraphx::op::pooling::dilations (C++ member), 71
migraphx::op::pooling::dyn_global (C++ member), 71
migraphx::op::pooling::kdims (C++ function), 70
migraphx::op::pooling::lengths (C++ member), 71
migraphx::op::pooling::lp_order (C++ member), 71
migraphx::op::pooling::lpnorm_pool (C++

```

```

    struct), 71
migraphx::op::pooling::lpnorm_pool::final
    (C++ function), 72
migraphx::op::pooling::lpnorm_pool::init
    (C++ function), 72
migraphx::op::pooling::lpnorm_pool::lpnorm_pool
    (C++ function), 72
migraphx::op::pooling::lpnorm_pool::operator()
    (C++ function), 72
migraphx::op::pooling::lpnorm_pool::p (C++
    member), 72
migraphx::op::pooling::max_pool (C++ struct),
    72
migraphx::op::pooling::max_pool::final (C++
    function), 72
migraphx::op::pooling::max_pool::init (C++
    function), 72
migraphx::op::pooling::max_pool::operator()
    (C++ function), 72
migraphx::op::pooling::mode (C++ member), 71
migraphx::op::pooling::name (C++ function), 70
migraphx::op::pooling::normalize_compute_shape
    (C++ function), 70
migraphx::op::pooling::padding (C++ member),
    71
migraphx::op::pooling::padding_mode (C++
    member), 71
migraphx::op::pooling::reflect (C++ function),
    71
migraphx::op::pooling::stride (C++ member), 71
migraphx::op::pooling_mode (C++ enum), 39
migraphx::op::pooling_mode::average (C++ enu-
    merator), 39
migraphx::op::pooling_mode::lpnorm (C++ enu-
    merator), 39
migraphx::op::pooling_mode::max (C++ enumera-
    tor), 39
migraphx::op::pow (C++ struct), 72
migraphx::op::pow::apply (C++ function), 72
migraphx::op::prefix_scan_op (C++ struct), 72
migraphx::op::prefix_scan_op::attributes
    (C++ function), 73
migraphx::op::prefix_scan_op::axis (C++ mem-
    ber), 73
migraphx::op::prefix_scan_op::compute (C++
    function), 73
migraphx::op::prefix_scan_op::exclusive
    (C++ member), 73
migraphx::op::prefix_scan_op::init (C++ func-
    tion), 73
migraphx::op::prefix_scan_op::normalize_compute_shape (C++ function), 73
migraphx::op::prefix_scan_op::prefix_scan_op
    (C++ function), 73
migraphx::op::prefix_scan_op::reflect (C++
    function), 73
migraphx::op::prefix_scan_op::reverse (C++
    member), 73
migraphx::op::prefix_scan_sum (C++ struct), 73
migraphx::op::prefix_scan_sum::op (C++ func-
    tion), 73
migraphx::op::prefix_scan_sum::prefix_scan_sum
    (C++ function), 73
migraphx::op::prelu (C++ struct), 73
migraphx::op::prelu::apply (C++ function), 74
migraphx::op::prelu::point_op (C++ function), 74
migraphx::op::quant_convolution (C++ struct),
    74
migraphx::op::quant_convolution::attributes
    (C++ function), 74
migraphx::op::quant_convolution::check_attribute_size
    (C++ function), 74
migraphx::op::quant_convolution::compute
    (C++ function), 74
migraphx::op::quant_convolution::dilation
    (C++ member), 74
migraphx::op::quant_convolution::group (C++
    member), 74
migraphx::op::quant_convolution::kdims (C++
    function), 74
migraphx::op::quant_convolution::name (C++
    function), 74
migraphx::op::quant_convolution::normalize_compute_shape
    (C++ function), 74
migraphx::op::quant_convolution::padding
    (C++ member), 74
migraphx::op::quant_convolution::padding_mode
    (C++ member), 74
migraphx::op::quant_convolution::reflect
    (C++ function), 74
migraphx::op::quant_convolution::stride
    (C++ member), 74
migraphx::op::quant_dot (C++ struct), 74
migraphx::op::quant_dot::attributes (C++
    function), 75
migraphx::op::quant_dot::compute_shape (C++
    function), 75
migraphx::op::quant_dot::name (C++ function), 75
migraphx::op::quantizelinear (C++ struct), 75
migraphx::op::quantizelinear::attributes
    (C++ function), 75
migraphx::op::quantizelinear::compute (C++
    function), 75
migraphx::op::quantizelinear::compute_shape
    (C++ function), 75
migraphx::op::quantizelinear::name (C++ func-
    tion), 75
migraphx::op::random_seed (C++ struct), 75

```



[migraphx::op::random\\_seed::compute \(C++ function\), 75](#)  
[migraphx::op::random\\_seed::compute\\_shape \(C++ function\), 75](#)  
[migraphx::op::random\\_seed::dtype \(C++ member\), 75](#)  
[migraphx::op::random\\_seed::name \(C++ function\), 75](#)  
[migraphx::op::random\\_seed::reflect \(C++ function\), 75](#)  
[migraphx::op::random\\_uniform \(C++ struct\), 75](#)  
[migraphx::op::random\\_uniform::compute \(C++ function\), 76](#)  
[migraphx::op::random\\_uniform::compute\\_shape \(C++ function\), 76](#)  
[migraphx::op::random\\_uniform::name \(C++ function\), 76](#)  
[migraphx::op::random\\_uniform::output\\_alias \(C++ function\), 76](#)  
[migraphx::op::recip \(C++ struct\), 76](#)  
[migraphx::op::recip::apply \(C++ function\), 76](#)  
[migraphx::op::recip::point\\_op \(C++ function\), 76](#)  
[migraphx::op::reduce\\_max \(C++ struct\), 76](#)  
[migraphx::op::reduce\\_max::init \(C++ function\), 76](#)  
[migraphx::op::reduce\\_max::op \(C++ function\), 76](#)  
[migraphx::op::reduce\\_max::reduce\\_max \(C++ function\), 76](#)  
[migraphx::op::reduce\\_mean \(C++ struct\), 76](#)  
[migraphx::op::reduce\\_mean::op \(C++ function\), 76](#)  
[migraphx::op::reduce\\_mean::output \(C++ function\), 76](#)  
[migraphx::op::reduce\\_mean::reduce\\_mean \(C++ function\), 76](#)  
[migraphx::op::reduce\\_min \(C++ struct\), 76](#)  
[migraphx::op::reduce\\_min::init \(C++ function\), 77](#)  
[migraphx::op::reduce\\_min::op \(C++ function\), 77](#)  
[migraphx::op::reduce\\_min::reduce\\_min \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op \(C++ struct\), 77](#)  
[migraphx::op::reduce\\_op::attributes \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::axes \(C++ member\), 78](#)  
[migraphx::op::reduce\\_op::compute \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::init \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::input \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::normalize\\_compute\\_shape \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::output \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::reduce \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::reduce\\_op \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::reflect \(C++ function\), 78](#)  
[migraphx::op::reduce\\_op::tune\\_axes \(C++ function\), 77](#)  
[migraphx::op::reduce\\_op::tune\\_dims \(C++ function\), 77](#)  
[migraphx::op::reduce\\_prod \(C++ struct\), 78](#)  
[migraphx::op::reduce\\_prod::init \(C++ function\), 78](#)  
[migraphx::op::reduce\\_prod::op \(C++ function\), 78](#)  
[migraphx::op::reduce\\_prod::reduce\\_prod \(C++ function\), 78](#)  
[migraphx::op::reduce\\_sum \(C++ struct\), 78](#)  
[migraphx::op::reduce\\_sum::op \(C++ function\), 78](#)  
[migraphx::op::reduce\\_sum::reduce\\_sum \(C++ function\), 78](#)  
[migraphx::op::relu \(C++ struct\), 78](#)  
[migraphx::op::relu::apply \(C++ function\), 78](#)  
[migraphx::op::relu::point\\_op \(C++ function\), 78](#)  
[migraphx::op::reshape \(C++ struct\), 78](#)  
[migraphx::op::reshape::compute \(C++ function\), 79](#)  
[migraphx::op::reshape::compute\\_shape \(C++ function\), 79](#)  
[migraphx::op::reshape::dims \(C++ member\), 79](#)  
[migraphx::op::reshape::dyn\\_compute\\_shape \(C++ function\), 79](#)  
[migraphx::op::reshape::name \(C++ function\), 79](#)  
[migraphx::op::reshape::reflect \(C++ function\), 79](#)  
[migraphx::op::reshape::static\\_compute\\_shape \(C++ function\), 79](#)  
[migraphx::op::reshape\\_lazy \(C++ struct\), 79](#)  
[migraphx::op::reshape\\_lazy::attributes \(C++ function\), 79](#)  
[migraphx::op::reshape\\_lazy::can\\_strides\\_merge \(C++ function\), 80](#)  
[migraphx::op::reshape\\_lazy::compute \(C++ function\), 79](#)  
[migraphx::op::reshape\\_lazy::compute\\_end\\_dim \(C++ function\), 80](#)  
[migraphx::op::reshape\\_lazy::compute\\_shape \(C++ function\), 79](#)  
[migraphx::op::reshape\\_lazy::dims \(C++ member\), 80](#)  
[migraphx::op::reshape\\_lazy::dyn\\_compute\\_shape \(C++ function\), 79](#)  
[migraphx::op::reshape\\_lazy::merge\\_strides \(C++ function\), 80](#)  
[migraphx::op::reshape\\_lazy::name \(C++ function\), 79](#)

```

migraphx::op::reshape_lazy::output_alias      (C++ function), 79
migraphx::op::reshape_lazy::reflect            (C++ function), 80
migraphx::op::reshape_lazy::reshape_lazy_dims (C++ function), 80
migraphx::op::reshape_lazy::static_compute_shape (C++ function), 79
migraphx::op::reshape_lazy::try_merge_pairs    (C++ function), 80
migraphx::op::reverse (C++ struct), 80
migraphx::op::reverse::attributes (C++ function), 80
migraphx::op::reverse::axes (C++ member), 80
migraphx::op::reverse::compute (C++ function), 80
migraphx::op::reverse::name (C++ function), 80
migraphx::op::reverse::normalize_compute_shape (C++ function), 80
migraphx::op::reverse::reflect (C++ function), 81
migraphx::op::rnn (C++ struct), 81
migraphx::op::rnn::actv_funcs (C++ member), 81
migraphx::op::rnn::clip (C++ member), 81
migraphx::op::rnn::compute_shape (C++ function), 81
migraphx::op::rnn::direction (C++ member), 81
migraphx::op::rnn::hidden_size (C++ member), 81
migraphx::op::rnn::name (C++ function), 81
migraphx::op::rnn::reflect (C++ function), 81
migraphx::op::rnn_direction (C++ enum), 40
migraphx::op::rnn_direction::bidirectional (C++ enumerator), 40
migraphx::op::rnn_direction::forward (C++ enumerator), 40
migraphx::op::rnn_direction::reverse (C++ enumerator), 40
migraphx::op::rnn_last_cell_output (C++ struct), 81
migraphx::op::rnn_last_cell_output::compute_shape (C++ function), 81
migraphx::op::rnn_last_cell_output::name (C++ function), 81
migraphx::op::rnn_last_hs_output (C++ struct), 81
migraphx::op::rnn_last_hs_output::compute_shape (C++ function), 82
migraphx::op::rnn_last_hs_output::name (C++ function), 82
migraphx::op::rnn_var_sl_last_output (C++ struct), 82
migraphx::op::rnn_var_sl_last_output::compute_shape (C++ function), 82
migraphx::op::rnn_var_sl_last_output::direction (C++ member), 82
migraphx::op::rnn_var_sl_last_output::name (C++ function), 82
migraphx::op::rnn_var_sl_last_output::reflect (C++ function), 82
migraphx::op::rnn_var_sl_shift_output (C++ struct), 82
migraphx::op::rnn_var_sl_shift_output::compute (C++ function), 82
migraphx::op::rnn_var_sl_shift_output::compute_shape (C++ function), 82
migraphx::op::rnn_var_sl_shift_output::direction (C++ member), 82
migraphx::op::rnn_var_sl_shift_output::name (C++ function), 82
migraphx::op::rnn_var_sl_shift_output::output_name (C++ member), 82
migraphx::op::rnn_var_sl_shift_output::reflect (C++ function), 83
migraphx::op::rnn_var_sl_shift_sequence (C++ struct), 83
migraphx::op::rnn_var_sl_shift_sequence::compute (C++ function), 83
migraphx::op::rnn_var_sl_shift_sequence::compute_shape (C++ function), 83
migraphx::op::rnn_var_sl_shift_sequence::name (C++ function), 83
migraphx::op::roialign (C++ struct), 83
migraphx::op::roialign::avg_pool (C++ struct), 84
migraphx::op::roialign::avg_pool::final (C++ function), 84
migraphx::op::roialign::avg_pool::init (C++ function), 84
migraphx::op::roialign::avg_pool::operator() (C++ function), 84
migraphx::op::roialign::calc_pooling (C++ function), 83
migraphx::op::roialign::calc_pos_weight (C++ function), 83
migraphx::op::roialign::compute (C++ function), 83
migraphx::op::roialign::compute_shape (C++ function), 83
migraphx::op::roialign::coord_trans_mode (C++ member), 83
migraphx::op::roialign::max_pool (C++ struct), 84
migraphx::op::roialign::max_pool::final (C++ function), 84
migraphx::op::roialign::max_pool::init (C++ function), 84
migraphx::op::roialign::max_pool::operator()

```

(C++ function), 84  
 migraphx::op::roialign::mode (C++ member), 83  
 migraphx::op::roialign::name (C++ function), 83  
 migraphx::op::roialign::output\_height (C++ member), 83  
 migraphx::op::roialign::output\_width (C++ member), 83  
 migraphx::op::roialign::pos\_weight (C++ struct), 84  
 migraphx::op::roialign::pos\_weight::pos (C++ member), 84  
 migraphx::op::roialign::pos\_weight::w (C++ member), 84  
 migraphx::op::roialign::reflect (C++ function), 84  
 migraphx::op::roialign::sampling\_ratio (C++ member), 83  
 migraphx::op::roialign::spatial\_scale (C++ member), 83  
 migraphx::op::rsqrt (C++ struct), 84  
 migraphx::op::rsqrt::apply (C++ function), 85  
 migraphx::op::run\_on\_target (C++ struct), 85  
 migraphx::op::run\_on\_target::compute (C++ function), 85  
 migraphx::op::run\_on\_target::compute\_shape (C++ function), 85  
 migraphx::op::run\_on\_target::name (C++ function), 85  
 migraphx::op::run\_on\_target::reflect (C++ function), 85  
 migraphx::op::run\_on\_target::target\_id (C++ member), 85  
 migraphx::op::scalar (C++ struct), 85  
 migraphx::op::scalar::compute (C++ function), 85  
 migraphx::op::scalar::compute\_shape (C++ function), 85  
 migraphx::op::scalar::name (C++ function), 85  
 migraphx::op::scalar::output\_alias (C++ function), 85  
 migraphx::op::scalar::reflect (C++ function), 86  
 migraphx::op::scalar::scalar\_bcast\_lens (C++ member), 86  
 migraphx::op::scatter\_add (C++ struct), 86  
 migraphx::op::scatter\_add::reduction (C++ function), 86  
 migraphx::op::scatter\_max (C++ struct), 86  
 migraphx::op::scatter\_max::reduction (C++ function), 86  
 migraphx::op::scatter\_min (C++ struct), 86  
 migraphx::op::scatter\_min::reduction (C++ function), 86  
 migraphx::op::scatter\_mul (C++ struct), 86  
 migraphx::op::scatter\_mul::reduction (C++ function), 86  
 migraphx::op::scatter\_none (C++ struct), 86  
 migraphx::op::scatter\_none::reduction (C++ function), 87  
 migraphx::op::scatter\_op (C++ struct), 87  
 migraphx::op::scatter\_op::attributes (C++ function), 87  
 migraphx::op::scatter\_op::axis (C++ member), 87  
 migraphx::op::scatter\_op::compute (C++ function), 87  
 migraphx::op::scatter\_op::derived (C++ function), 87  
 migraphx::op::scatter\_op::normalize\_compute\_shape (C++ function), 87  
 migraphx::op::scatter\_op::reflect (C++ function), 87  
 migraphx::op::scatternd\_add (C++ struct), 87  
 migraphx::op::scatternd\_add::reduction (C++ function), 87  
 migraphx::op::scatternd\_add::scatternd\_add (C++ function), 87  
 migraphx::op::scatternd\_max (C++ struct), 87  
 migraphx::op::scatternd\_max::reduction (C++ function), 87  
 migraphx::op::scatternd\_max::scatternd\_max (C++ function), 87  
 migraphx::op::scatternd\_min (C++ struct), 87  
 migraphx::op::scatternd\_min::reduction (C++ function), 88  
 migraphx::op::scatternd\_min::scatternd\_min (C++ function), 88  
 migraphx::op::scatternd\_mul (C++ struct), 88  
 migraphx::op::scatternd\_mul::reduction (C++ function), 88  
 migraphx::op::scatternd\_mul::scatternd\_mul (C++ function), 88  
 migraphx::op::scatternd\_none (C++ struct), 88  
 migraphx::op::scatternd\_none::reduction (C++ function), 88  
 migraphx::op::scatternd\_none::scatternd\_none (C++ function), 88  
 migraphx::op::scatternd\_op (C++ struct), 88  
 migraphx::op::scatternd\_op::compute (C++ function), 88  
 migraphx::op::scatternd\_op::compute\_shape (C++ function), 88  
 migraphx::op::scatternd\_op::init (C++ function), 88  
 migraphx::op::scatternd\_op::scatternd\_op (C++ function), 88  
 migraphx::op::select\_module (C++ struct), 88  
 migraphx::op::select\_module::compute (C++ function), 89  
 migraphx::op::select\_module::compute\_shape

(C++ function), 89  
 migraphx::op::select\_module::get\_input\_parameters (C++ function), 89  
 migraphx::op::select\_module::get\_output\_parameters (C++ function), 89  
 migraphx::op::select\_module::name (C++ function), 89  
 migraphx::op::select\_module::output\_alias (C++ function), 89  
 migraphx::op::select\_module::output\_dyn\_shapes (C++ member), 89  
 migraphx::op::select\_module::reflect (C++ function), 89  
 migraphx::op::sigmoid (C++ struct), 89  
 migraphx::op::sigmoid::apply (C++ function), 89  
 migraphx::op::sigmoid::point\_op (C++ function), 89  
 migraphx::op::sign (C++ struct), 89  
 migraphx::op::sign::apply (C++ function), 89  
 migraphx::op::sign::point\_op (C++ function), 89  
 migraphx::op::sin (C++ struct), 89  
 migraphx::op::sin::apply (C++ function), 90  
 migraphx::op::sinh (C++ struct), 90  
 migraphx::op::sinh::apply (C++ function), 90  
 migraphx::op::slice (C++ struct), 90  
 migraphx::op::slice::all\_set (C++ member), 92  
 migraphx::op::slice::attributes (C++ function), 90  
 migraphx::op::slice::axes (C++ member), 91  
 migraphx::op::slice::axes\_only (C++ member), 92  
 migraphx::op::slice::compute (C++ function), 91  
 migraphx::op::slice::compute\_offset (C++ function), 90, 91  
 migraphx::op::slice::compute\_two\_or\_more (C++ function), 90  
 migraphx::op::slice::ends (C++ member), 91  
 migraphx::op::slice::ends\_axes (C++ member), 92  
 migraphx::op::slice::ends\_only (C++ member), 92  
 migraphx::op::slice::get\_set\_attributes (C++ function), 90  
 migraphx::op::slice::lens\_calc (C++ function), 90  
 migraphx::op::slice::name (C++ function), 90  
 migraphx::op::slice::none\_set (C++ member), 92  
 migraphx::op::slice::normalize\_compute\_shape (C++ function), 90  
 migraphx::op::slice::normalize\_starts\_ends\_axes (C++ function), 91  
 migraphx::op::slice::output\_alias (C++ function), 91  
 migraphx::op::slice::reflect (C++ function), 92  
 migraphx::op::slice::starts (C++ member), 91  
 migraphx::op::slice::starts\_axes (C++ member), 92  
 migraphx::op::slice::starts\_ends (C++ member), 92  
 migraphx::op::slice::starts\_only (C++ member), 92  
 migraphx::op::softmax (C++ struct), 92  
 migraphx::op::softmax::attributes (C++ function), 92  
 migraphx::op::softmax::axis (C++ member), 92  
 migraphx::op::softmax::name (C++ function), 92  
 migraphx::op::softmax::normalize\_compute\_shape (C++ function), 92  
 migraphx::op::softmax::output (C++ function), 92  
 migraphx::op::softmax::reflect (C++ function), 93  
 migraphx::op::sqdiff (C++ struct), 93  
 migraphx::op::sqdiff::apply (C++ function), 93  
 migraphx::op::sqdiff::point\_op (C++ function), 93  
 migraphx::op::sqrt (C++ struct), 93  
 migraphx::op::sqrt::apply (C++ function), 93  
 migraphx::op::squeeze (C++ struct), 93  
 migraphx::op::squeeze::attributes (C++ function), 93  
 migraphx::op::squeeze::axes (C++ member), 93  
 migraphx::op::squeeze::compute (C++ function), 93  
 migraphx::op::squeeze::name (C++ function), 93  
 migraphx::op::squeeze::normalize\_compute\_shape (C++ function), 93  
 migraphx::op::squeeze::output\_alias (C++ function), 93  
 migraphx::op::squeeze::reflect (C++ function), 94  
 migraphx::op::step (C++ struct), 94  
 migraphx::op::step::attributes (C++ function), 94  
 migraphx::op::step::axes (C++ member), 94  
 migraphx::op::step::compute (C++ function), 94  
 migraphx::op::step::name (C++ function), 94  
 migraphx::op::step::normalize\_compute\_shape (C++ function), 94  
 migraphx::op::step::output\_alias (C++ function), 94  
 migraphx::op::step::reflect (C++ function), 94  
 migraphx::op::step::steps (C++ member), 94  
 migraphx::op::sub (C++ struct), 94  
 migraphx::op::sub::apply (C++ function), 94  
 migraphx::op::sub::point\_function (C++ function), 94  
 migraphx::op::tan (C++ struct), 94  
 migraphx::op::tan::apply (C++ function), 95

migraphx::op::tanh (C++ struct), 95  
 migraphx::op::tanh::apply (C++ function), 95  
 migraphx::op::topk (C++ struct), 95  
 migraphx::op::topk::attributes (C++ function), 95  
 migraphx::op::topk::axis (C++ member), 95  
 migraphx::op::topk::compute (C++ function), 95  
 migraphx::op::topk::heap\_vector (C++ struct), 95  
 migraphx::op::topk::heap\_vector::compare (C++ member), 96  
 migraphx::op::topk::heap\_vector::data (C++ member), 96  
 migraphx::op::topk::heap\_vector::heap\_vector (C++ function), 96  
 migraphx::op::topk::heap\_vector::sort (C++ function), 96  
 migraphx::op::topk::heap\_vector::try\_push (C++ function), 96  
 migraphx::op::topk::k (C++ member), 95  
 migraphx::op::topk::largest (C++ member), 95  
 migraphx::op::topk::make\_heap (C++ function), 95  
 migraphx::op::topk::name (C++ function), 95  
 migraphx::op::topk::normalize\_compute\_shape (C++ function), 95  
 migraphx::op::topk::reflect (C++ function), 95  
 migraphx::op::transpose (C++ struct), 96  
 migraphx::op::transpose::compute (C++ function), 96  
 migraphx::op::transpose::compute\_shape (C++ function), 96  
 migraphx::op::transpose::dims (C++ member), 96  
 migraphx::op::transpose::name (C++ function), 96  
 migraphx::op::transpose::output\_alias (C++ function), 96  
 migraphx::op::transpose::reflect (C++ function), 96  
 migraphx::op::unary (C++ struct), 96  
 migraphx::op::unary::attributes (C++ function), 97  
 migraphx::op::unary::base\_attributes (C++ function), 97  
 migraphx::op::unary::compute (C++ function), 97  
 migraphx::op::unary::compute\_shape (C++ function), 97  
 migraphx::op::unary::point\_function (C++ function), 97  
 migraphx::op::unary::point\_op (C++ function), 97  
 migraphx::op::unary\_not (C++ struct), 97  
 migraphx::op::unary\_not::apply (C++ function), 97  
 migraphx::op::unary\_not::name (C++ function), 97  
 migraphx::op::unary\_not::point\_function (C++ function), 97  
 migraphx::op::undefined (C++ struct), 97  
 migraphx::op::undefined::compute (C++ function), 97  
 migraphx::op::undefined::compute\_shape (C++ function), 97  
 migraphx::op::undefined::name (C++ function), 97  
 migraphx::op::unique (C++ struct), 97  
 migraphx::op::unique::axis (C++ member), 98  
 migraphx::op::unique::compute (C++ function), 98  
 migraphx::op::unique::compute\_shape (C++ function), 97  
 migraphx::op::unique::make\_idx\_less\_fn (C++ function), 97  
 migraphx::op::unique::name (C++ function), 97  
 migraphx::op::unique::reflect (C++ function), 98  
 migraphx::op::unique::sorted (C++ member), 98  
 migraphx::op::unique::sorted\_uniq\_indices (C++ function), 97  
 migraphx::op::unique::unsorted\_uniq\_indices (C++ function), 97  
 migraphx::op::unknown (C++ struct), 98  
 migraphx::op::unknown::compute\_shape (C++ function), 98  
 migraphx::op::unknown::name (C++ function), 98  
 migraphx::op::unknown::op (C++ member), 98  
 migraphx::op::unknown::operator<< (C++ function), 98  
 migraphx::op::unknown::reflect (C++ function), 98  
 migraphx::op::unsqueeze (C++ struct), 98  
 migraphx::op::unsqueeze::attributes (C++ function), 99  
 migraphx::op::unsqueeze::axes (C++ member), 99  
 migraphx::op::unsqueeze::compute (C++ function), 99  
 migraphx::op::unsqueeze::name (C++ function), 99  
 migraphx::op::unsqueeze::normalize\_compute\_shape (C++ function), 99  
 migraphx::op::unsqueeze::output\_alias (C++ function), 99  
 migraphx::op::unsqueeze::reflect (C++ function), 99  
 migraphx::op::unsqueeze::steps (C++ member), 99  
 migraphx::op::where (C++ struct), 99  
 migraphx::op::where::attributes (C++ function), 99  
 migraphx::op::where::compute (C++ function), 99  
 migraphx::op::where::compute\_shape (C++ function), 99  
 migraphx::op::where::name (C++ function), 99  
 migraphx::op::zero (C++ struct), 99  
 migraphx::op::zero::operator T (C++ function), 99



```

migraphx::parse_onnx (C++ function), 128
migraphx::parse_onnx_buffer (C++ function), 128
migraphx::pass (C++ struct), 107
migraphx::pass::apply (C++ function), 107
migraphx::pass::name (C++ function), 107
migraphx::program (C++ struct), 126
migraphx::program::compile (C++ function), 126
migraphx::program::create_module (C++ function), 126
migraphx::program::eval (C++ function), 126
migraphx::program::experimental_get_context (C++ function), 126
migraphx::program::get_main_module (C++ function), 126
migraphx::program::get_output_shapes (C++ function), 126
migraphx::program::get_parameter_shapes (C++ function), 126
migraphx::program::operator!= (C++ function), 127
migraphx::program::operator== (C++ function), 127
migraphx::program::print (C++ function), 126
migraphx::program::program (C++ function), 126
migraphx::program::run_async (C++ function), 126
migraphx::program::sort (C++ function), 126
migraphx::program_parameter_shapes (C++ struct), 125
migraphx::program_parameter_shapes::names (C++ function), 125
migraphx::program_parameter_shapes::operator[] (C++ function), 125
migraphx::program_parameter_shapes::program_parameter_shapes (C++ function), 125
migraphx::program_parameter_shapes::size (C++ function), 125
migraphx::program_parameters (C++ struct), 125
migraphx::program_parameters::add (C++ function), 125
migraphx::program_parameters::program_parameters (C++ function), 125
migraphx::propagate_constant (C++ struct), 109
migraphx::propagate_constant::apply (C++ function), 109
migraphx::propagate_constant::name (C++ function), 109
migraphx::propagate_constant::skip_ops (C++ member), 110
migraphx::quantize_fp16 (C++ function), 127
migraphx::quantize_int8_options (C++ struct), 127
migraphx::quantize_int8_options::add_calibration (C++ function), 127
migraphx::quantize_int8_options::add_op_name (C++ function), 127
migraphx::quantize_int8_options::calibration (C++ member), 128
migraphx::quantize_int8_options::op_names (C++ member), 128
migraphx::quantize_int8_options::quantize_int8_options (C++ function), 127
migraphx::quantize_op_names (C++ struct), 127
migraphx::quantize_op_names::add (C++ function), 127
migraphx::quantize_op_names::quantize_op_names (C++ function), 127
migraphx::raw_data (C++ struct), 34
migraphx::raw_data::at (C++ function), 35
migraphx::raw_data::auto_cast (C++ struct), 35
migraphx::raw_data::auto_cast::get_data_type (C++ type), 36
migraphx::raw_data::auto_cast::is_data_ptr (C++ type), 36
migraphx::raw_data::auto_cast::matches (C++ function), 36
migraphx::raw_data::auto_cast::operator T (C++ function), 36
migraphx::raw_data::auto_cast::operator T* (C++ function), 36
migraphx::raw_data::auto_cast::self (C++ member), 36
migraphx::raw_data::cast (C++ function), 35
migraphx::raw_data::get (C++ function), 35
migraphx::raw_data::implicit (C++ function), 35
migraphx::raw_data::operator<< (C++ function), 35
migraphx::raw_data::operator>> (C++ function), 35
migraphx::raw_data::single (C++ function), 35
migraphx::raw_data::to_string (C++ function), 35
migraphx::raw_data::visit (C++ function), 34, 35
migraphx::raw_data::visit_at (C++ function), 34
migraphx::rewrite_rnn (C++ struct), 110
migraphx::rewrite_rnn::apply (C++ function), 110
migraphx::rewrite_rnn::name (C++ function), 110
migraphx::save (C++ function), 129
migraphx::schedule (C++ struct), 110
migraphx::schedule::apply (C++ function), 110
migraphx::schedule::enable (C++ member), 110
migraphx::schedule::model (C++ member), 110
migraphx::schedule::name (C++ function), 110
migraphx::shape (C++ struct), 122
migraphx::shape::broadcasted (C++ function), 123
migraphx::shape::bytes (C++ function), 122
migraphx::shape::compute_index (C++ function), 123
migraphx::shape::dyn_dims (C++ function), 122
migraphx::shape::dynamic (C++ function), 122
migraphx::shape::element_space (C++ function), 123

```

- `migraphx::shape::elements` (C++ function), 122, 123
- `migraphx::shape::get_shape` (C++ function), 123
- `migraphx::shape::index` (C++ function), 123
- `migraphx::shape::index_array` (C++ type), 122
- `migraphx::shape::lengths` (C++ function), 122
- `migraphx::shape::lens` (C++ member), 123
- `migraphx::shape::multi` (C++ function), 123
- `migraphx::shape::operator!=` (C++ function), 123
- `migraphx::shape::operator==` (C++ function), 123
- `migraphx::shape::operator<<` (C++ function), 123
- `migraphx::shape::packed` (C++ function), 123
- `migraphx::shape::shape` (C++ function), 122, 123
- `migraphx::shape::shape_type` (C++ type), 122
- `migraphx::shape::single` (C++ function), 123
- `migraphx::shape::skips` (C++ function), 123
- `migraphx::shape::standard` (C++ function), 122, 123
- `migraphx::shape::strides` (C++ function), 122
- `migraphx::shape::strides` (C++ member), 123
- `migraphx::shape::transposed` (C++ function), 123
- `migraphx::shape::type` (C++ function), 122
- `migraphx::simplify_algebra` (C++ struct), 110
- `migraphx::simplify_algebra::apply` (C++ function), 111
- `migraphx::simplify_algebra::name` (C++ function), 111
- `migraphx::simplify_resshapes` (C++ struct), 111
- `migraphx::simplify_resshapes::apply` (C++ function), 111
- `migraphx::simplify_resshapes::depth` (C++ member), 111
- `migraphx::simplify_resshapes::name` (C++ function), 111
- `migraphx::target` (C++ struct), 124
- `migraphx::target::target` (C++ function), 125
- `migraphx_compile_options` (C++ struct), 125
- `migraphx_compile_options::migraphx_compile_options` (C++ function), 126
- `migraphx_compile_options::object` (C++ member), 126
- `migraphx_shape_datatype_t` (C++ enum), 121
- `migraphx_shape_datatype_t::migraphx_shape_bool_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_double_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_float_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_fp8e4m3fnuz_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_half_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_int16_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_int32_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_int64_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_int8_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_tuple_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_uint16_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_uint32_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_uint64_type` (C++ enumerator), 121
- `migraphx_shape_datatype_t::migraphx_shape_uint8_type` (C++ enumerator), 121
- `migraphx-driver-perf` command line option
  - `--iterations`, 21
  - `-n`, 21
- `migraphx-driver-verify` command line option
  - `--atol`, 21
  - `--per-instruction`, 21
  - `--reduce`, 21
  - `--ref-use-double`, 21
  - `--rms-tol`, 21
  - `--rtol`, 21
  - `-i`, 21
  - `-r`, 21
- module
  - `migraphx`, 129
- N
  - `ndim()` (in module `migraphx`), 132
- P
  - `packed()` (in module `migraphx`), 132
  - `parse_onnx()` (in module `migraphx`), 136
  - `parse_tf()` (in module `migraphx`), 137
  - `print()` (in module `migraphx`), 134
  - `program` (class in `migraphx`), 135
- Q
  - `quantize_fp16()` (in module `migraphx`), 136
  - `quantize_int8()` (in module `migraphx`), 136
- R
  - `run()` (in module `migraphx`), 136
- S
  - `scalar()` (in module `migraphx`), 132
  - `shape` (class in `migraphx`), 131
  - `sort()` (in module `migraphx`), 136

`standard()` (*in module `migraphx`*), [132](#)  
`strides()` (*in module `migraphx`*), [131](#)

## T

`target` (*class in `migraphx`*), [134](#)  
`tolist()` (*in module `migraphx`*), [133](#)  
`transposed()` (*in module `migraphx`*), [132](#)  
`type()` (*in module `migraphx`*), [131](#)  
`type_size()` (*in module `migraphx`*), [131](#)