Transformer models via WebNN in ORT& Chromimum

2023-06-29

Dwayne Robinson

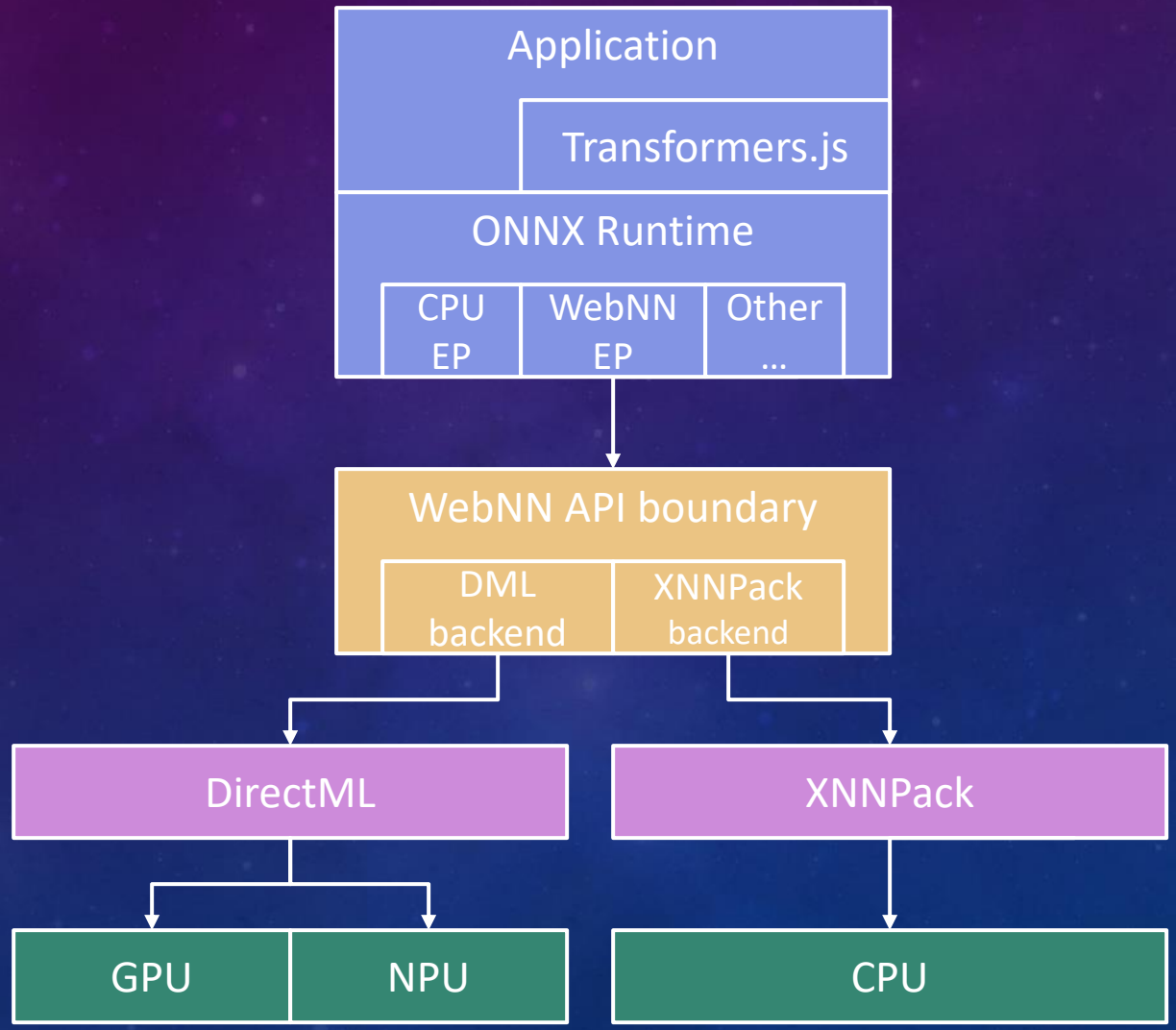
WebNN Architecture

Web Application

Browser

OS

Hardware



Working Areas

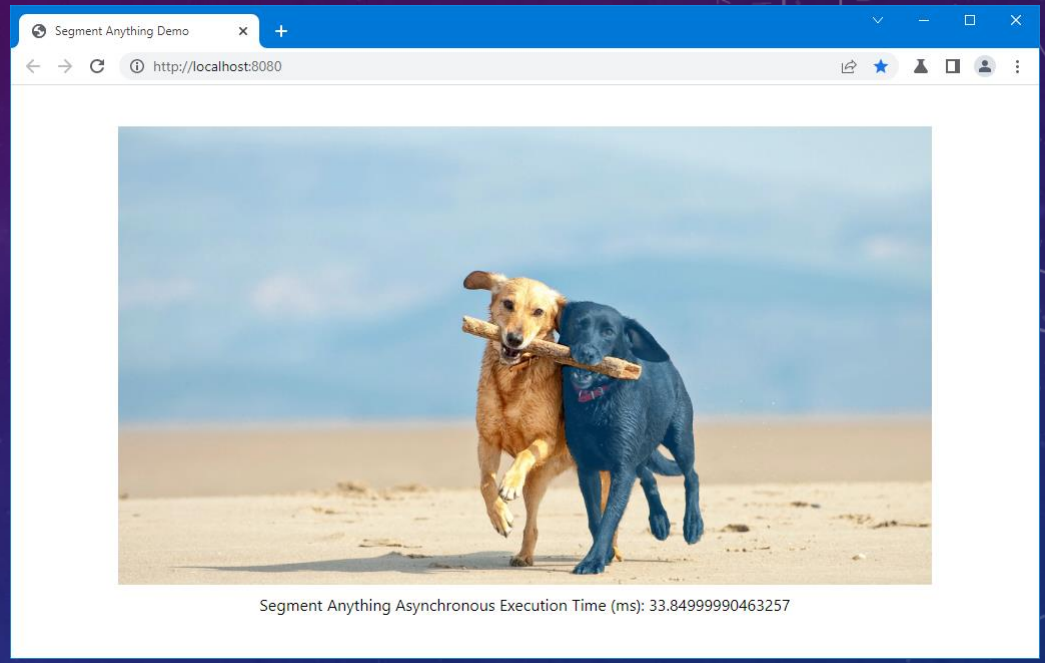
- ONNX Runtime (in main branch)
  - V1 ops - 34/60
  - V2 ops - 14/20 (falls back to CPU when browser lacks support)
- Chromium DML backend operator kernels (in fork, diff)
  - V1 ops - 44/60
  - V2 ops - 19/20

Model Targets

- ☑ Segment Anything
- ⏳ Stable Diffusion
- ⏳ Whisper Tiny
- ⏳ Other customer models…
- (eventually larger ones like Dolly)

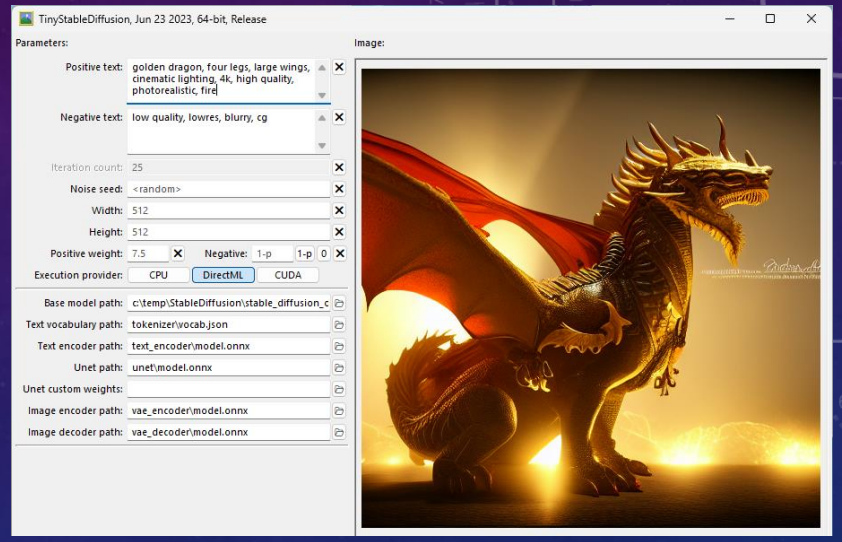
Model Targets – Segment Anything

- 16MBs
- 39 operators
- Coordinates in → Mask out
- ☑ Works on current ORT for the Web on Chromium fork
- Static shape assumption



Model Targets – Stable Diffusion

- Float32=5.1GBs, Float16=2.3GBs (1.7GB unet)
- 38 operators
- 5 different models orchestrated:
  - Tokenization (word/fragment → numeric id)
  - CLIP text encoder (id → embedding vector)
  - VAE image encoder (image → latent)
  - Unet noise predictor (latent → latent)
  - Denoising algorithm (not an NN):
    1. weight factor scheduling per iteration
    2. sample update equation
  - VAE image decoder (latent → image)
  - Safety checker (probability of erotic or violent content notes)



Model Targets – Stable Diffusion

- ✅ Native ORT+DML runs SD effectively already (huggingface diffusers, Axodox ML, MS //build demos)
- ⏳ WebNN prototype needs a little more operator work still (ORTW +5 ops and Chromium backend +2 ops).
- Other challenges:
  - 4GB limit in WASM 32 (Memory64 might be our salvation, but it has bugs – commence onion peeling exercise)
  - large file cache eviction causes redownload of large models

Operators – v2

- argMax / argMin – find value in tensor, return index
- cast – change data type (essential gap)
- equal / greater / lesser – compare elementwise
- logicalNot – invert boolean elementwise
- erf – Gauss error function
- expand – broadcast up to new size (used by ConstantOfShape)
- unsqueeze – missing corollary to Squeeze
- flattenTo2d – kin to squeeze/unsqueeze
- gather – collect values from indices
- identity – placeholder and direct mapping for callers
- fillSequence – fill numeric sequence from/to/step
- triangularMatrix – fill upper or lower triangular part of matrix
- elementwiseIf – per-element source selection
- meanVarianceNormalization – encompasses instance/norm/layer/groupedChannel…
- shape getter – essential for sanity debugging and printing results
- reciprocal – dedicated to op (avoid 1/div work-around)
- sqrt – dedicated op (avoid 0.5 extra tensor)

- Special notes later for: MHA, spatial+grouped channel normalization, gelu

Operators – v2 IDL

```
// Elementwise binary logical comparison operations
[RaisesException] MLOperand equal(MLOperand a, MLOperand b);
[RaisesException] MLOperand greater(MLOperand a, MLOperand b);
[RaisesException] MLOperand lesser(MLOperand a, MLOperand b);

// Elementwise unary operations
[RaisesException] MLOperand identity(MLOperand input);
[RaisesException] MLOperand sqrt(MLOperand input);
[RaisesException] MLOperand erf(MLOperand input);
[RaisesException] MLOperand reciprocal(MLOperand input);
[RaisesException] MLOperand logicalNot(MLOperand input);

// This trinary elementwise operator corresponds to the ONNX where operator.
[RaisesException] MLOperand elementwiseIf(MLOperand condition, MLOperand trueValue, MLOperand falseValue);

// Reshaping operations
// Belongs with reshape(MLOperand input, sequence<unsigned long> newShape)
[RaisesException] MLOperand squeeze(MLOperand input, optional MLSqueezeOptions options = {});
[RaisesException] MLOperand unsqueeze(MLOperand input, MLSqueezeOptions options);
[RaisesException] MLOperand flattenTo2d(MLOperand input, unsigned long axis);

[RaisesException] MLOperand expand(MLOperand input, sequence<unsigned long> newShape);
[RaisesException] MLOperand gather(MLOperand input, MLOperand indices, optional MLGatherOptions options = {});

[RaisesException] MLOperand argMax(MLOperand input, optional MLArgMinMaxOptions options = {});
[RaisesException] MLOperand argMin(MLOperand input, optional MLArgMinMaxOptions options = {});

[RaisesException] MLOperand cast(MLOperand input, MLOperandType operandType);

[RaisesException] MLOperand meanVarianceNormalization(MLOperand input, optional MLMeanVarianceNormalizationOptions options = {});
[RaisesException] MLOperand fillSequence(MLOperandType operandType, sequence<unsigned long> outputShape, optional MLFillSequenceOptions options = {});

[RaisesException] MLOperand triangularMatrix(MLOperand input, optional MLTriangularMatrixOptions options = {});

// Shape retrieval.
[RaisesException] sequence<unsigned long> shape(MLOperand input);
```

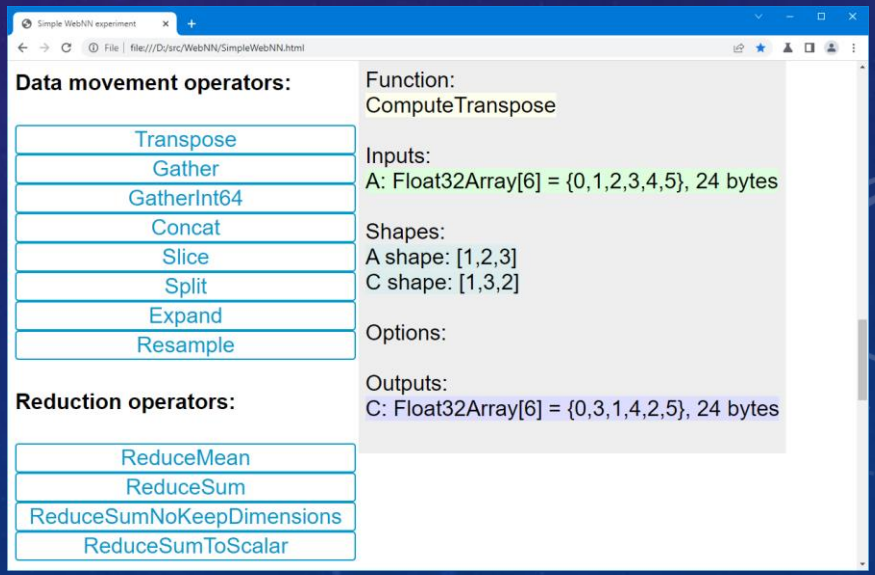
https://github.com/fdwr/chromium-src-webnn-dml/blob/dml_sd/third_party/blink/renderer/modules/ml/webnn/ml_graph_builder.idl#L166-L267

Operators – missing MLOperand*Data*Type's

- bool8: elementwiseIf, equal / greater / lesser, logicalNot
- int64: gather, argMin / argMax, (future ops - scatter, oneHot)
  - used heavily in ONNX for all things indexish
  - should accept int64 directly to avoid casts of index tensors
  - backends can emulate using lower 32-bits
- Note other frameworks have various "*operand types*": maps, sequences, sparse tensors… so current naming will have an interesting conflict if you ever try to extend WebNN. 😉

Operators – missing concepts

- 0D scalars
  - Every ML framework supports them… except WebNN (issue). Treating scalars like 1D arrays with 1 element is not equivalent, breaking gather's shape inference, and reduceSum with keepDimensions=false and all axes.
- Retrieving the built shape (issue)
  - WebNN *knows* the built shape so far, but it offers no way to report it.
  - Devs will need this facility for basic sanity checking, debugging, resource size estimation, and informative output. Really use WebNN, and you'll feel MLOperand::shape()'s absence.

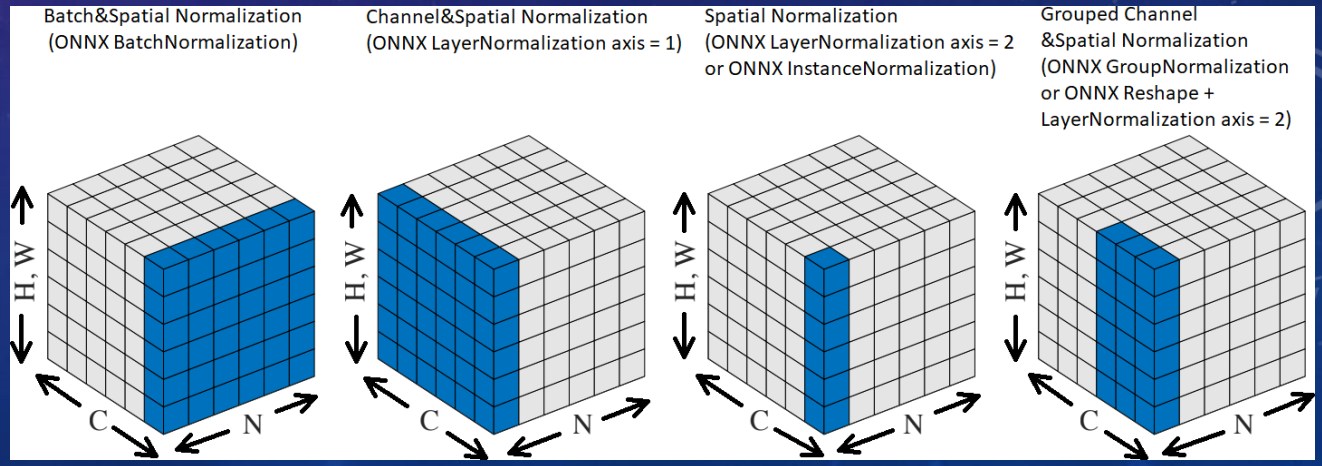


Operators – reshaping ops

- Has squeeze but is missing counterpart unsqueeze 🙃 (issue).
- flattenTo2d needed by models too
- However 🤔, all 3 are just reshape(), no?
- We should either complete the currently awkward family, or (since they can be easily resolved by the calling framework) just have reshape() to implement them (deleting squeeze)

Operators – MeanVarianceNormalization, the new normal

- Pointlessly many ways to arrange axes:
  - spatial normalization
  - spatial+batch normalization
  - spatial+channel normalization
  - spatial+grouped channel normalization…
- Just add MVN and take axes
- Backends can map that to an optimized version
- Delete instance/batch/ layerNorm and layout field



Operators – dedicated ops for efficiency & semantics

- Pow(x, 0.5) is not a substitute for dedicated sqrt
- Div(1/x) is not a substitute for true reciprocal
- Implementations have dedicated instructions
- Avoid the pointless temporaries to create and upload

Operators – omitted optimized operators

- OLive optimizations rearrange the model, fusing new ops:
  - Gelu – x * ½ * (1 + erf(x / sqrt(2)) (ORT MS op)
  - MultiheadAttention (ORT MS op)
  - Spatial+grouped channel normalization (ORT MS op)
- Operators exist in DML and ORT MS contrib ops, but not in ONNX, and there is variation in frameworks
- Given so many variations, let the community settle on blessed approach
- In the meantime, can recognize and fuse patterns in backend

Links

- WebNN spec: https://www.w3.org/TR/webnn/
  - Brief Explainer: https://github.com/webmachinelearning/webnn/blob/main/explainer.md
- Microsoft ONNX Runtime for the Web
  - Quick start https://github.com/microsoft/onnxruntime-inference-examples/blob/main/js/quick-start_onnxruntime-web-script-tag/index.html (Tutorial https://onnxruntime.ai/docs/tutorials/web/)
  - WebNN EP https://github.com/microsoft/onnxruntime/tree/main/onnxruntime/core/providers/webnn Builder: https://github.com/microsoft/onnxruntime/blob/main/onnxruntime/core/providers/webnn/builders/op_builder_factory.cc
- Chromium fork:
  - Dwayne's (includes v2 ops)
  - Mingming's (based off older fork, supports mobilenet).
- Microsoft OLive Optimizer: https://github.com/microsoft/OLive

Questions?

Appendix follows

Operators – v1, plus all potential v2 candidates

WebNN v1 operator	Priority	DML	XNN Pack	Chrome DML	ORTW	Misc notes	ONNX Operators	Usage count
Supported by WebNN…								
abs		✔	✔				Abs	2
add		✔	✔	✔	✔		Add	68
averagePool2d		✔	✔	✔			AveragePool	14
batchNormalization		✔		✓			BatchNormalization	14
ceil		✔	✔	✔	✔		Ceil	2
clamp		✔	✔	✔	✔		Clip	16
concat		✔	✔	✔	✔		Concat	60
constant		✔	✔	✔	✔		Constant	40
conv2d		✔	✔	✔	✔		Conv	51
convTranspose2d		✔	✔	✔	✔		ConvTranspose	9
cos		✔		✔	✔		Cos	2
div		✔	✔	✔	✔		Div	37
(nop in inference)		NA					Dropout	2
elu		✔	✔				Elu	1
exp		✔		✔	✔		Exp	7
floor		✔	✔	✔	✔		Floor	5
gemm		✔	✔	✔	✔		Gemm	19
(averagePool2d large kernel)		✔	✔	✔	✔		GlobalAveragePool	7
(averageMax2d large kernel)		✔	✔	✔	✔		GlobalMaxPool	
gru		✔					GRU	4
gruCell		✔					---	0
hardSigmoid		✔					HardSigmoid	0
hardSwish		✓	✔	✔			---	0
instanceNormalization		✔		✔			InstanceNormalization	6
l2Pool2d		✔	✔				LpPool	0
leakyRelu		✔	✔		✔		LeakyRelu	10
linear		✔					---	0
log		✔					Log	4
lstm		✔					LSTM	0
matmul		✔			✔		MatMul	37
max		✔		✔			Max	2
maxPool2d		✔	✔	✔	✔		MaxPool	22
min		✔		✔			Min	4
mul		✔	✔	✔	✔		Mul	51
neg		✔	✔				Neg	1
pad		✔	✔	⌛			Pad	11
pow		✔	✔	✔	✔		Pow	20
prelu		✔	✔				Prelu	
(div 1/x)		✔		✔	✔		Reciprocal	3
reduceL1		✔		✔			ReduceL1	0
reduceL2		✔		✔			ReduceL2	1
reduceLogSum		✔		✔			ReduceLogSum	0
reduceLogSumExp		✔		✔			ReduceLogSumExp	0
reduceMax		✔		✔	✔		ReduceMax	2
reduceMean		✔		✔	✔		ReduceMean	18
reduceMin		✔		✔			ReduceMin	5
reduceProduct		✔		✔			ReduceProd	2
reduceSum		✔		✔			ReduceSum	12
reduceSumSquare		✔		✔			ReduceSumSquare	0
relu		✔	✔	✔	✔		Relu	41
resample2d		✔	✔	✔	✔		Resize	17
reshape		✔	✔	✔	✔		Reshape	59
sigmoid		✔	✔	✔	✔		Sigmoid	29
sin		✔		✔	✔		Sin	2
slice		✔	✔	✔	✔		Slice	40
softmax		✔	✔	✔	✔		Softmax	41
softplus		✔					Softplus	0
softsign		✔					Softsign	0
split		✔	✔	✔	✔		Split	16
(pow with 0.5)		✔		✔	✔		Sqrt	22
squeeze		✔		✔	✔		Squeeze	34
sub		✔	✔	✔			Sub	36
tan		✔		✔	✔		Tan	0
tanh		✔	✔				Tanh	15
transpose		✔	✔	✔	✔		Transpose	53
(resample)		✔					Upsample	4

Potential Future WebNN	Priority	DML	XNN Pack	Chrome DML	ORTW	Misc notes	ONNX Operators	Usage count
						= Mul + Add	Affine	1
		✔					And	3
argMax		✔		✔	✔		ArgMax	7
argMin		✔		✔	✔		ArgMin	0
cast	1	✔		✔	✔		Cast	43
	1	✔		✔	⌛		ConstantOfShape	28
		✔				= Slice	Crop	1
cumulativeSummation	1	✔					CumSum	4
		✔					DepthToSpace	3
		✔					DequantizeLinear	10
		✔				= QuantizeLinear, ReduceMin, ReduceMax	DynamicQuantizeLinear	5
		✓					EinSum	1
equal	1	✔		✔	✔		Equal	22
erf	1	✔		✔	✔		Erf	11
expand	1	✔		✔	✔		Expand	21
flattenTo2d	1	✔		✔	✔		Flatten	7
gather	1	✔		✔	✔		Gather	47
		✔					GatherElements	3
		✔					GatherND	3
greater	1	✔		✔			Greater	8
	1	✔					GreaterOrEqual	3
		⌛					GridSample	1
identity		✔		✔	✔		Identity	14
		✕					If	5
		✓					ImageScaler	0
		✔					IsInf	2
		✔					IsNaN	2
via meanVarianceNormalization	1	✔		✔	✔		LayerNormalization	5
lesser	1	✔		✔			Less	7
		✔					LogSoftmax	1
		✕				Can unroll	Loop	4
		✔					LRN	2
		✔				= Cast, MatMul	MatMulInteger	2
meanVarianceNormalization	1	✔		✔			MeanVarianceNormalization	
		✕					NonMaxSuppression	4
		✔					NonZero	10
logicalNot	1	✔		✔	✔		Not	7
		✔					OneHot	2
		✔	✔				PRelu	1
		✔					QLinearConv	1
		✔					QuantizeLinear	1
fillSequence	1	✔					Range	7
		✔					RoiAlign	2
		✔					Round	1
		✔					Scatter	2
		✔					ScatterElements	2
		✔					ScatterND	4
		NA				Can split to separate tensors	SequenceEmpty	2
		NA					SequenceLength	1
shape op or getter	1	NA		✔	✔		Shape	47
		✔					Sign	2
		⌛					STFT	2
		✔					Tile	5
		✔					TopK	8
triangularMatrix	1	✔		⌛			Trilu	1
unsqueeze	1*	NA		✔	✔		Unsqueeze	49
elementwiself	1	✔		✔	✔	= bool input, Cast, Mul x 2, Add	Where	18

WebNN Sample

```
const context = await navigator.ml.createContext();
const builder = new MLGraphBuilder(context);

// 1. Create a computational graph 'C = 0.2 * A + B'.
const tensorDescription = {type: 'float32', dimensions: [2, 2]};
const constant = builder.constant(0.2);
const A = builder.input('A', tensorDescription);
const B = builder.input('B', tensorDescription);
const C = builder.add(builder.mul(A, constant), B);

// 2. Compile it into an executable graph.
const graph = await builder.build({'C': C});

// 3. Bind inputs to the graph and execute for the result.
const bufferA = new Float32Array(4).fill(1.0);
const bufferB = new Float32Array(4).fill(0.8);
const bufferC = new Float32Array(4);
const inputs = {'A': bufferA, 'B': bufferB};
const outputs = {'C': bufferC};
const result = await context.compute(graph, inputs, outputs);

// Print the computed result in the output buffer: [[1, 1], [1, 1]]
console.log("Output value: " + result.outputs.C + ", " + result.outputs.C.shape());
```

Chrome WebNN backend IPC