ONNX.js

A Javascript library to run ONNX models in browsers and Node.js

Emma Ning| Senior product manager, Microsoft

JS-based Client-side applications

- Website
- Electron APP
  - A framework for building cross platform desktop apps with JavaScript, HTML, and CSS
  - Compatible with Mac, Windows, and Linux

GitHub Desktop

Skype

Visual Studio Code

Slack

JS-based Client-side Machine Learning

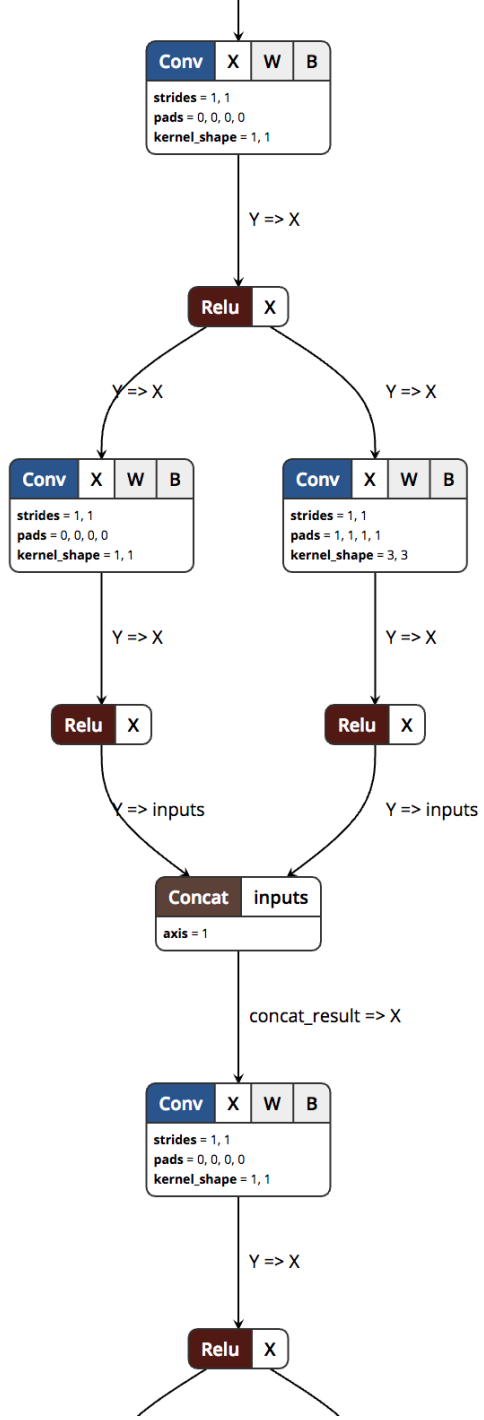
- Benefits of Client-side Machine Learning
  - Privacy protection
  - Real-time Analysis
  - Offline Capability
- More with JS-based Client-side Machine Learning
  - Cross-platform support
  - GPU acceleration
  - Easy Access and distribution

ONNX.JS Background - ONNX

ONNX Open Neural Network Exchange

An open standard for representing machine learning models

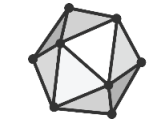
- Consisting of:
  - common Intermediate Representation
  - full operator spec
- Model = graph composed of computational nodes
- Supports both DNN and traditional ML
- Backward compatible with comprehensive versioning



ONNX converters for popular frameworks

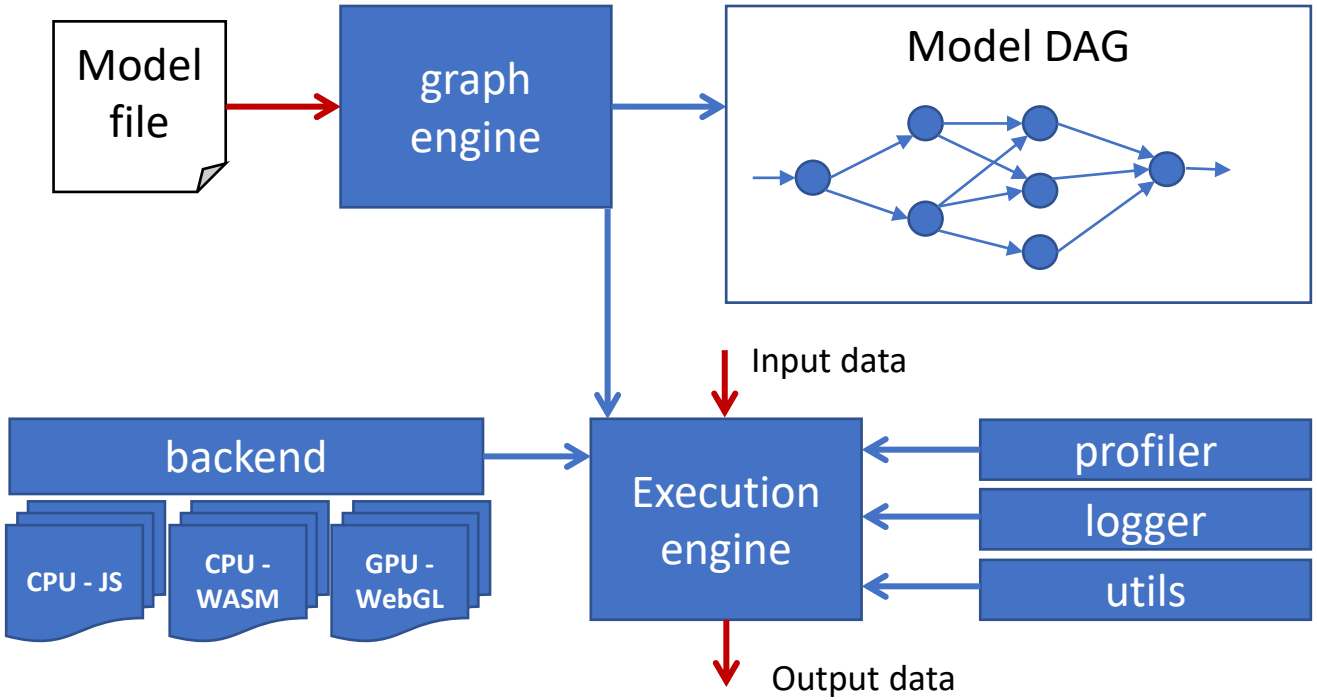
- Native Support
  - Pytorch
  - CNTK
- Open sourced Converter Tools
  - Tensorflow: onnx/tensorflow-onnx
  - Keras: onnx/keras-onnx
  - Scikit-learn: onnx/sklearn-onnx
  - CoreML: onnx/onnxmltools
  - LightGBM: onnx/onnxmltools
  - LibSVM: onnx/onnxmltools
  - XGBoost: onnx/onnxmltools


ONNX Community



ONNX.JS

- A pure JavaScript implementation of ONNX framework
- Optimize ONNX model inference on both CPUs and GPUs
- Support a variety of browsers on major OSes



Desktop Platforms

OS/Browser	Chrome	Edge	FireFox	Safari	Opera	Electron	Node.js
Windows 10	✔	✔	✔	-	✔	✔	✔
macOS	✔	-	✔	✔	✔	✔	✔
Ubuntu LTS 18.04	✔	-	✔	-	✔	✔	✔

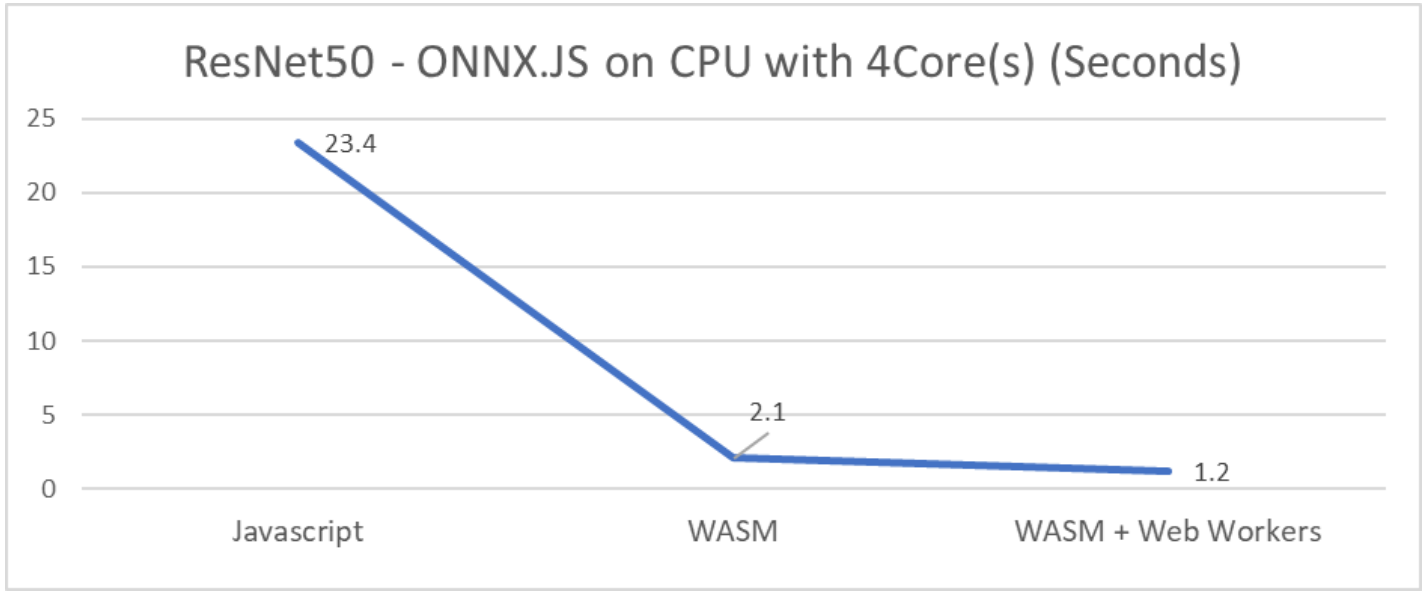
Mobile Platforms

OS/Browser	Chrome	Edge	FireFox	Safari	Opera
iOS	✔	✔	✔	✔	✔
Android	✔	✔	Coming soon	-	✔

ONNX.js

For running on CPU

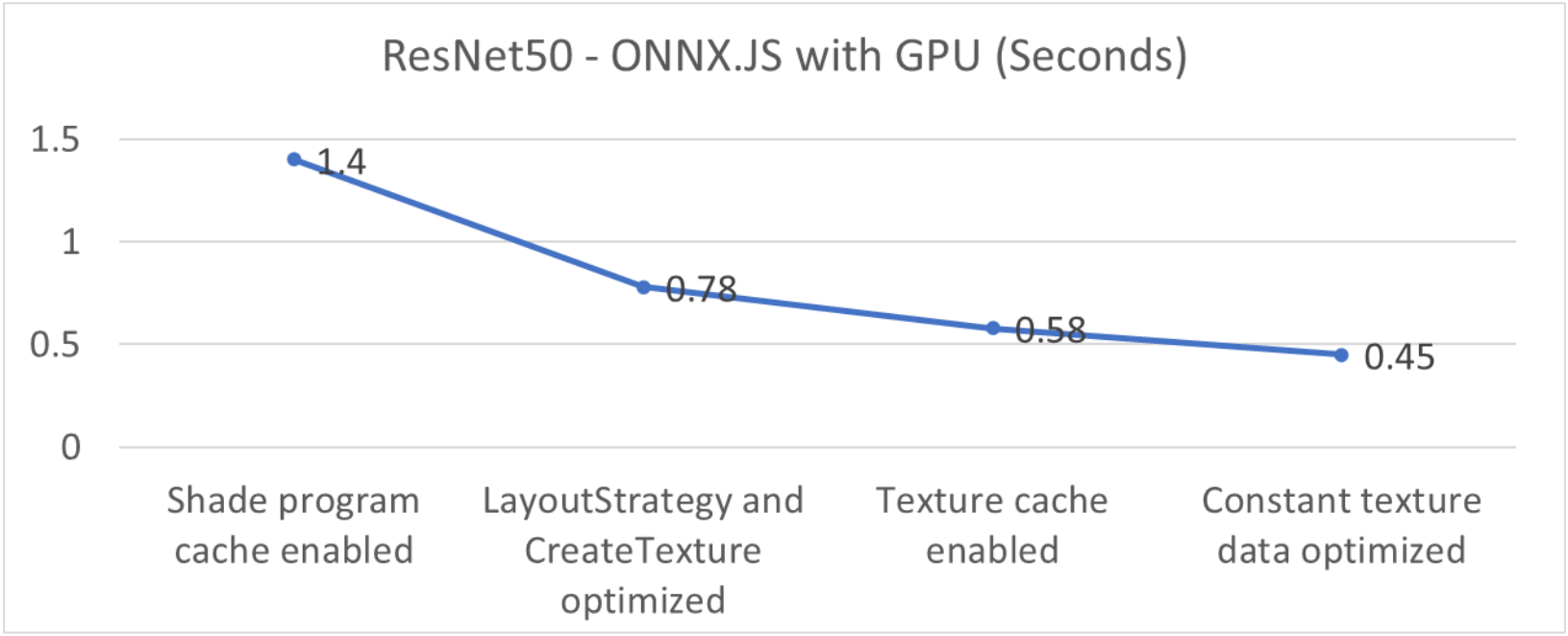
- Leverage WebAssembly to execute model at near-native speed
- Leverage Web Workers to provide a "multi-threaded" environment for parallelization in operators



ONNX.js

For running on GPUs

- Leverage WebGL, a popular standard for accessing GPU capabilities
- Apply novel optimizations to reduce GPU processing cycles and data transfer between CPU and GPU

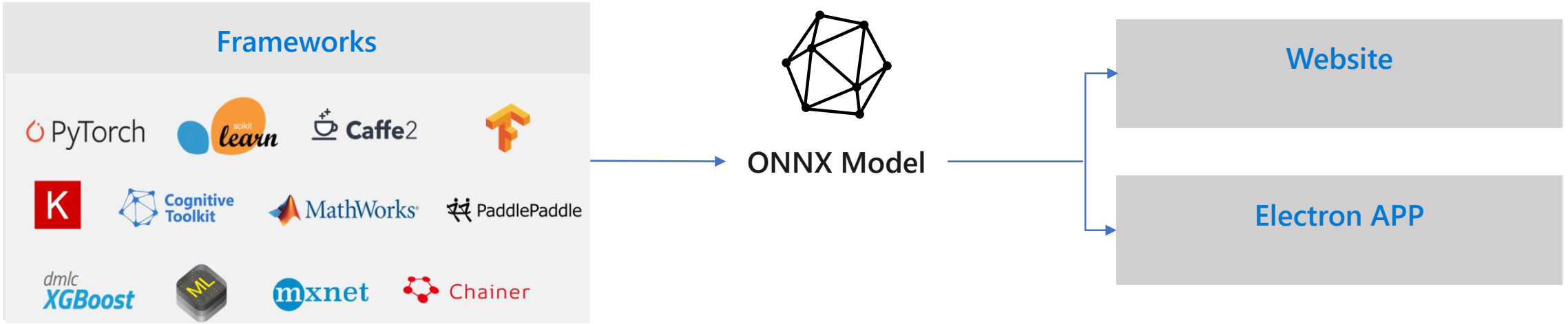


Model inference with ONNX.JS

Train models with various **frameworks or services**

Convert into ONNX with **ONNX Converters**

Model inference with **ONNX.JS**



HTML example to use ONNX.js

```
<html>
  <head>
  </head>

  <body>
    <!-- Load ONNX.js -->
    <script src="https://cdn.jsdelivr.net/npm/onnxjs/dist/onnx.min.js"></script>
    <!-- Code that consume ONNX.js -->
    <script>
      // create a session
      const myOnnxSession = new onnx.InferenceSession();
      // load the ONNX model file
      myOnnxSession.loadModel("./my-model.onnx").then(()=>{
        // generate model input
        const inferenceInputs = getInputs();
        // execute the model
        session.run(inferenceInputs).then(output=>{
          // consume the output
          const outputTensor = output.values().next().value;
          console.log(`model output tensor: ${outputTensor.data}.`);
        });
      })
    </script>
  </body>
</html>
```

Using NPM and bundling tools to use ONNX.js

1. Import `Tensor` and `InferenceSession`.

```
import {Tensor, InferenceSession} from 'onnxjs';
```

2. Create an instance of `InferenceSession`.

```
const session = new InferenceSession();
```

3. Load the ONNX.js model

```
// use the following in an async method
const url = './data/models/resnet/model.onnx';
await session.loadModel(url);
```

4. Create your input Tensor(s) similar to the example below. You need to do any pre-processing required by your model at this stage. For that refer to the documentation of the model you have:

```
// creating an array of input Tensors is the easiest way. For other options see t
const inputs = [new Tensor(new Float32Array([1.0,2.0,3.0,4.0]), 'float32', [2,2])
```

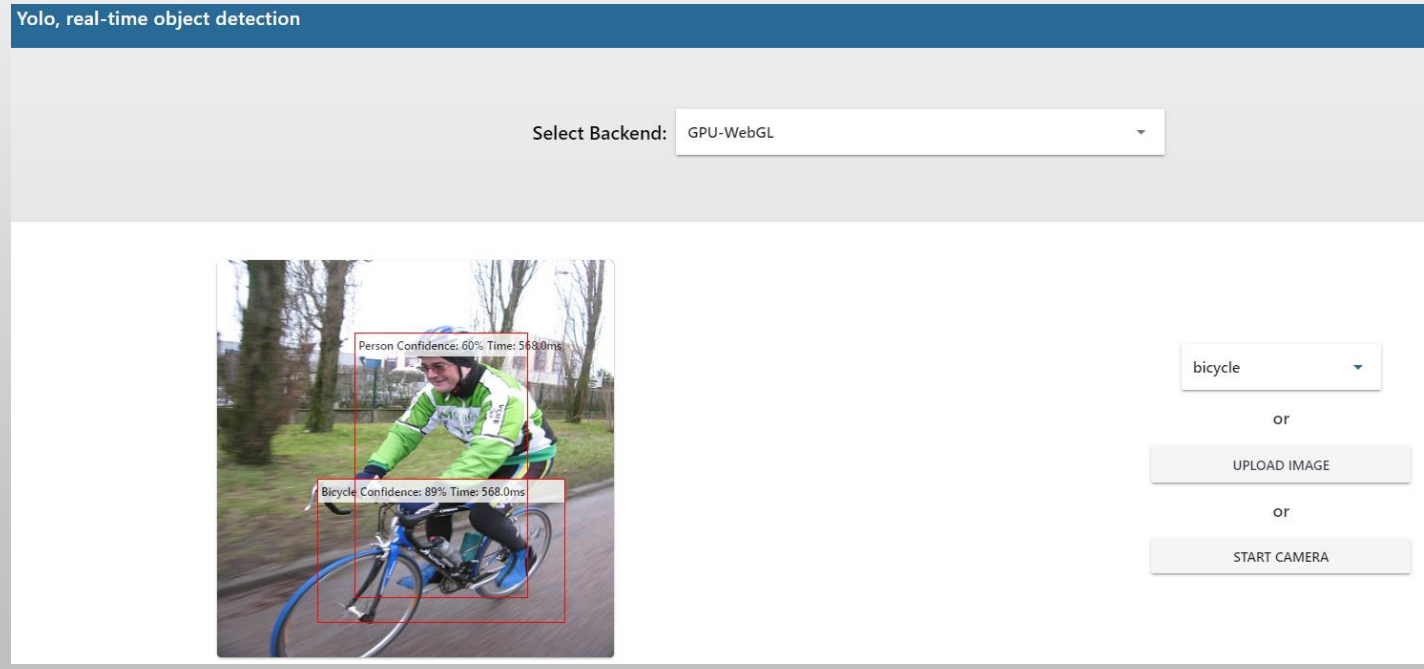
5. Run the model with the input Tensors. The output Tensor(s) are available once the run operation is complete:

```
// run this in an async method:
const outputMap = await session.run(inputs);
const outputTensor = outputMap.values().next().value;
```

ONNX.js Demo

- Models supported
  - **Resnet-50** - a deep convolutional networks for image classification
  - **Squeezenet** - a light-weight convolutional networks for image classification
  - **Emotion_ferplus** - a deep convolutional neural network for emotion recognition in faces
  - **Mnist** - a convolutional neural network to predict handwritten digits
  - **Tiny_yolov2** - a real-time neural network for object detection

[ONNX.js demo website]



We’d love to embrace your contribution

Operator Coverage Improvement

Performance improvement

More demos with more models

Thanks