Shared Externally

# Update on Model Loader API

The chromeOS MI Team

Jan, 2022

# Contents

# Updates on API Design

Main updates on API design,

1. Align with the WebNN API.
2. Improve the **compute** interface.
3. Always let user download the model.
4. The model loading and inference calls are both asynchronous.

# Update API: Create Context

```
// First, create an MLContext. This is consistent with the WebNN API. And we will
// add two new fields, “numThread” and "modelFormat".
const context = await navigator.ml.createContext(
                                        { devicePreference: "cpu",
                                          powerPreference: "low-power",
                                          numThreads: 0,   // the default 0 means
                                                           // "decide automatically".
                                         modelFormat: "tflite" });
```

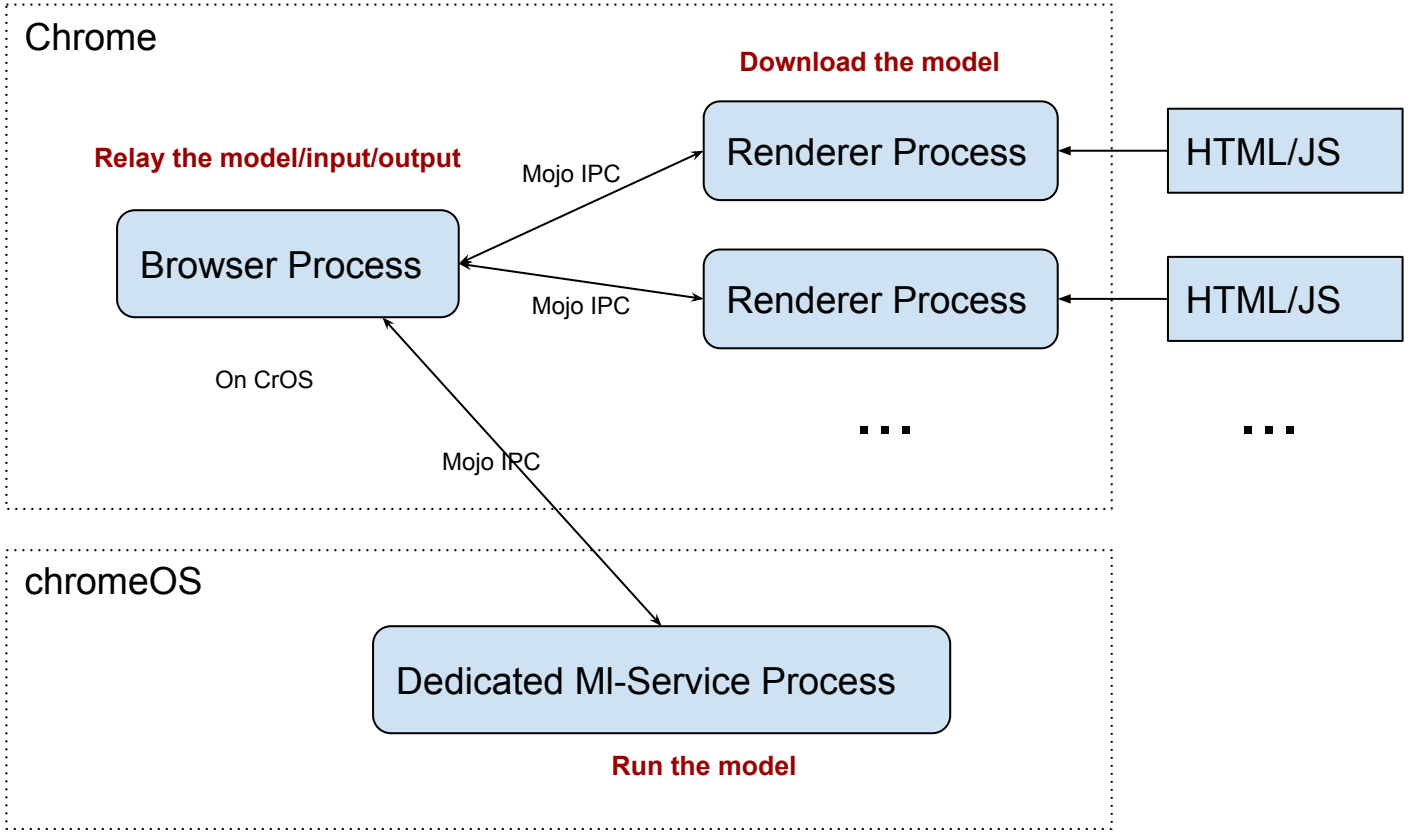
# Update API: Load Model

```
// Then create the model loader using the ML context.
loader = new MLModelLoader(context);
// In the first version, we only support loading models from ArrayBuffers. We
// believe this covers most of the usage cases. Web developers can download the
// model, e.g., by the fetch API. We can add new "load" functions in the future
// if they are really needed.
const modelUrl = 'https://path/to/model/file';
const modelBuffer = await fetch(modelUrl)
                            .then(response => response.arrayBuffer());
// Load the model.
model = await loader.load(modelBuffer);
```

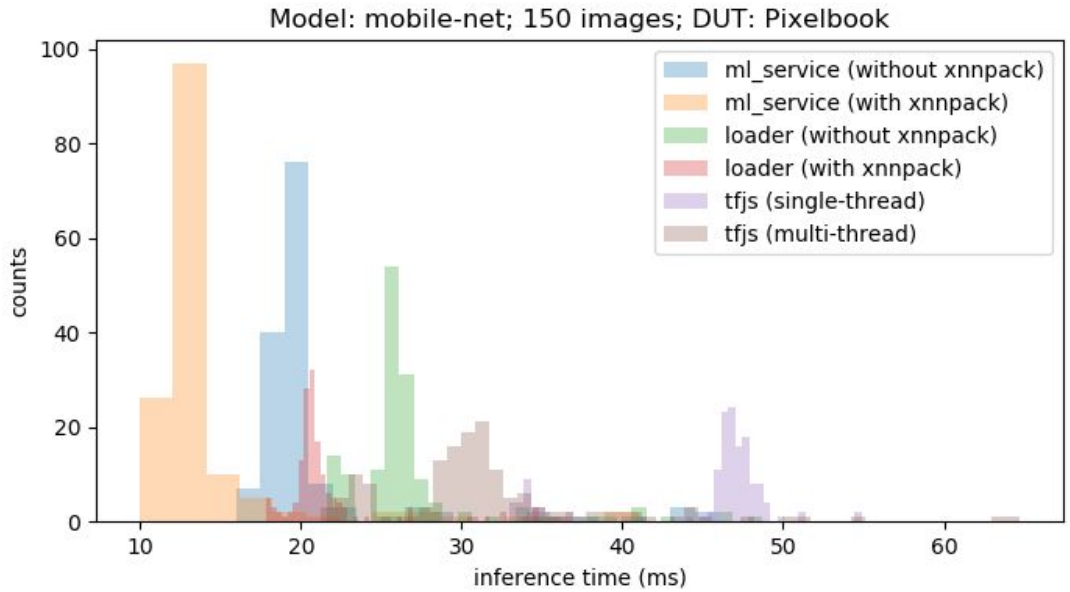
# Update API: Improve the “compute” interface

```
// When there is only input tensor, no need to specify the name.
z = await model.compute({ data: new Float32Array([10]),
                          dimensions: [1]) });

// Specify the input tensors by names.
z = await model.compute({ x: { data: new Float32Array([10]), dimensions: [1] },
                          y: { data: new Float32Array([20]), dimensions: [1] } });

// Can specify the output tensor too.
z_buffer = ml.tensor({ data: new Float32Array(1), dimensions: [1] });
await model.compute({ data: new Float32Array([10]), dimensions: [1] },
                    z_buffer);
```

# Current Prototype (CL)



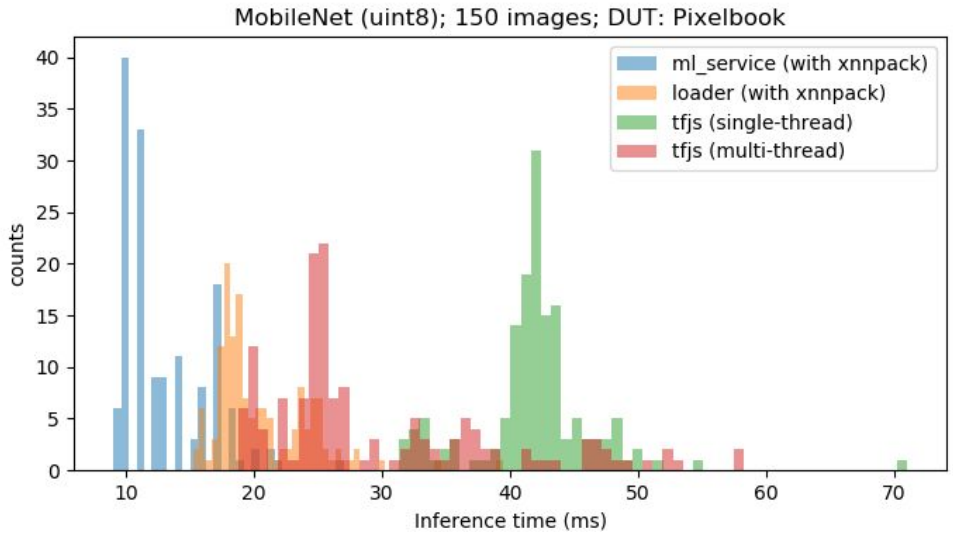
# Benchmarks: MobileNet v2, Float32




- **ml_service**: the backend.
- **loader**: the mode loader API.
- **tfjs**: TensorFlow.js TFLite runner.

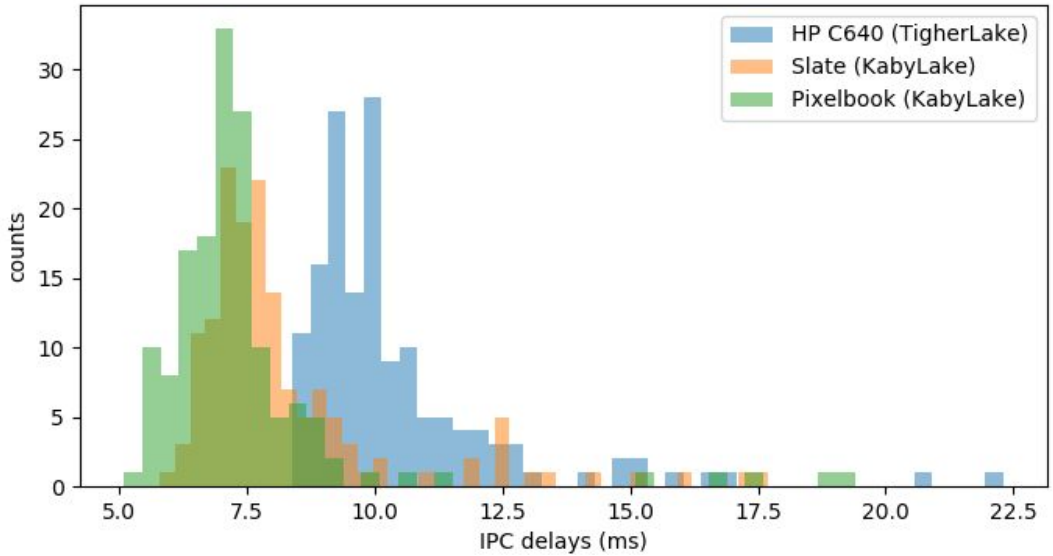
**Notice that different #bin are used to better show each histogram*

# Benchmarks: MobileNet v2, UInt8




*Notice that different #bin are used to better show each histogram

# Benchmark: IPC cost

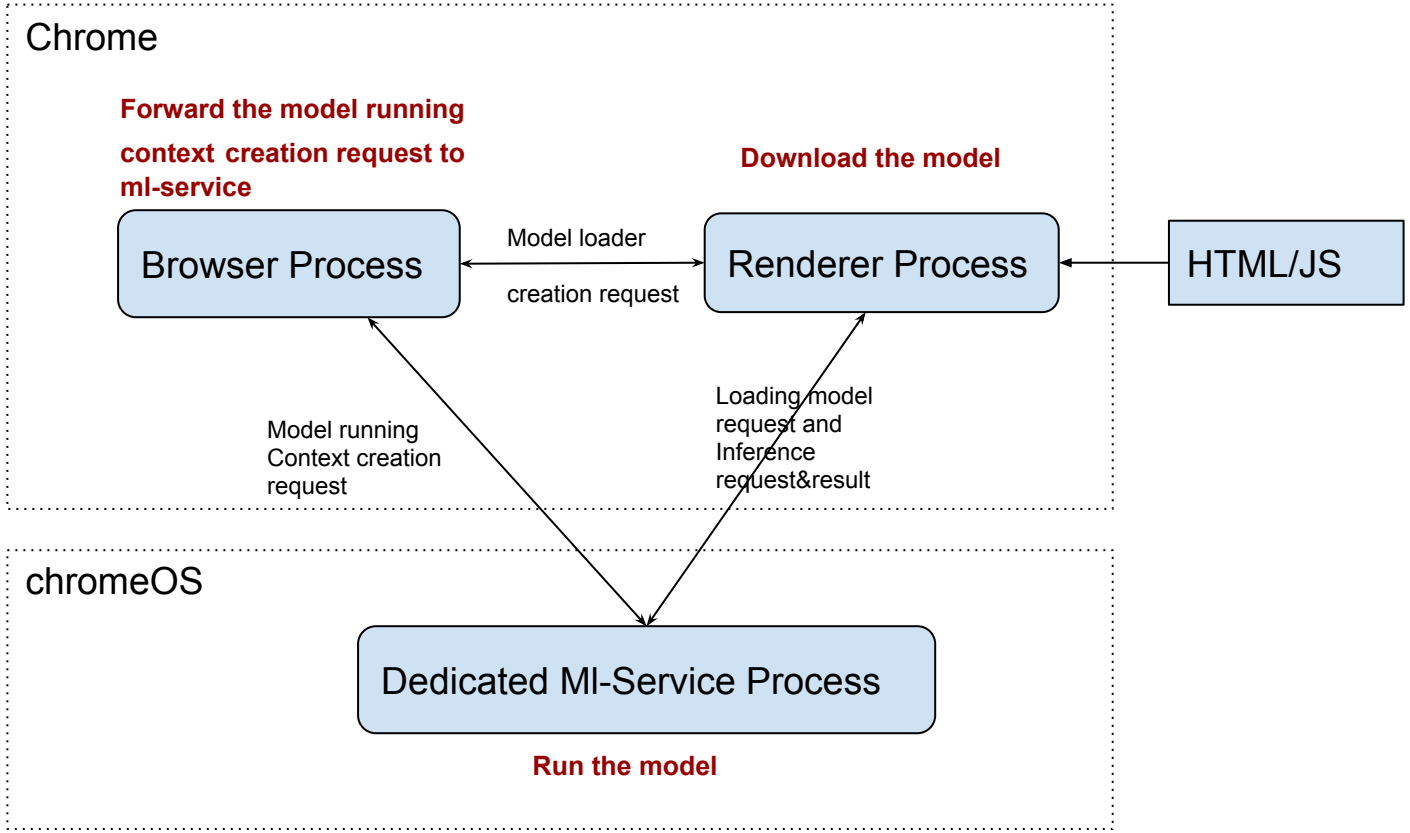



- Main workload is a 224x224x3 float32 array representing the image.

# Todos

1. Complete the “intent-to-prototype” review.
2. More benchmarks (e.g. more types of models)?
3. Improve ml-service. Create a dedicated TFLite running stack for Web Model Loader API.
4. Enable XNNPACK and tune it to achieve best performance in ml-service (for both float and quantized models).
5. Hardening TFLite. (Not needed for prototype)
6. Reduce the IPC cost.
7. Explore hardware acceleration.
8. Use permission policy to restrict feature availability.

# Reduce IPC Cost: Example



# Open Questions

1. What will the interface look like when hardware acceleration is supported?
    a. What if the data of a tensor is on GPU/TPU?
    b. Shall we let users know which ML accelerator is available on the platform? (Performance v.s. Privacy)
2. How to version the API? (Supported OPs and formats)
3. Coordinate with WebNN? e.g. How to organize the folder structure,
    a. Both under “third_party/blink/renderer/modules/**ml**/”
    b. Common files (MLContext etc.) under “ml/common/”
    c. Model loader under “ml/model_loader/”
    d. WebNN under “ml/webnn/”
4. Dedicated interface for streaming inputs (e.g. buffered inputs for video, audio)?
5. Model formats

# Thanks!