WebNN Small Language Model (SLM) Performance Optimization Case Study

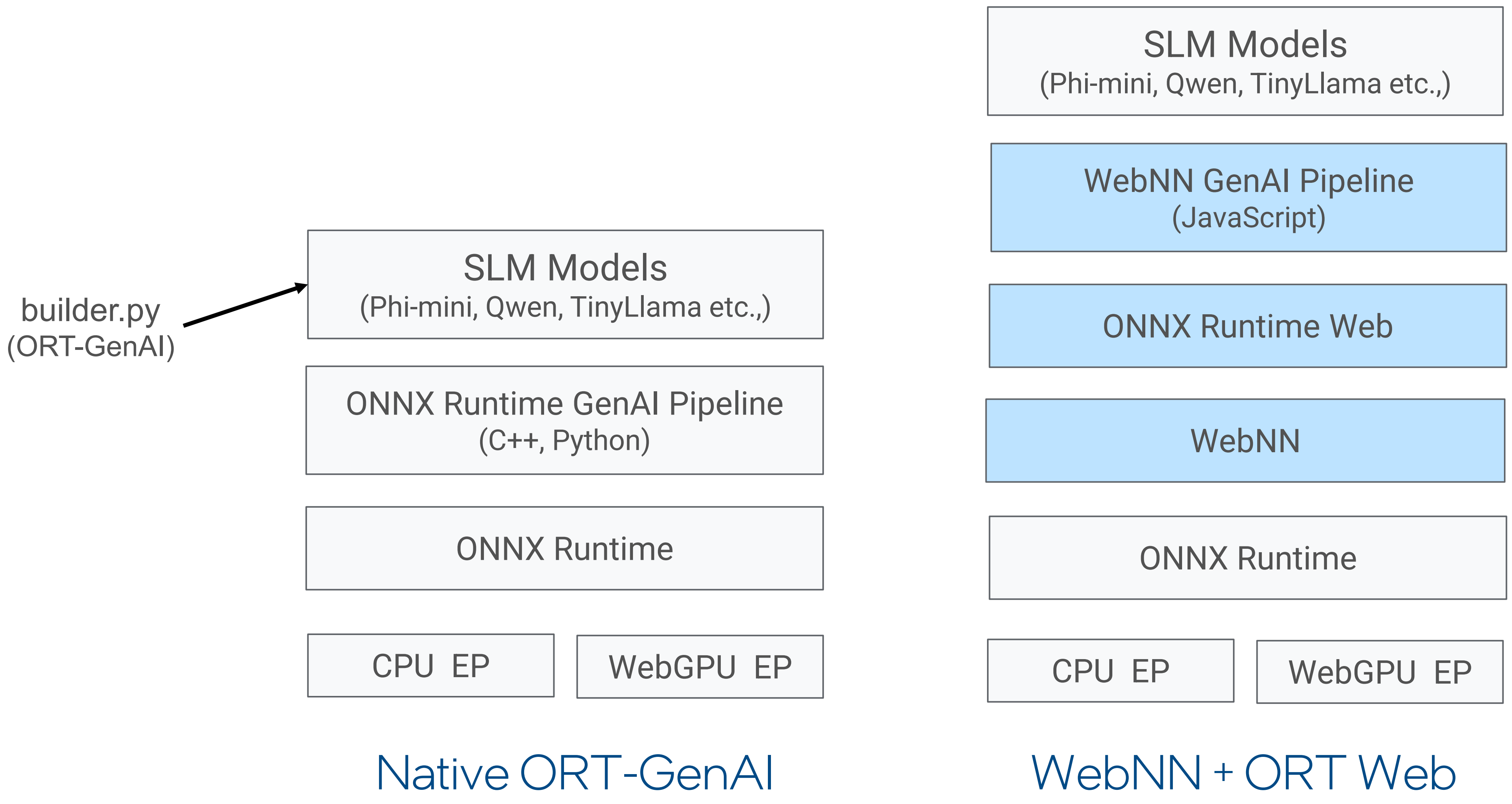
Yuheng Wei, Wei Wang, Wanming Lin, Jonathan Ding, Ningxin Hu – Intel

Nov 2025, @TPAC

Agenda

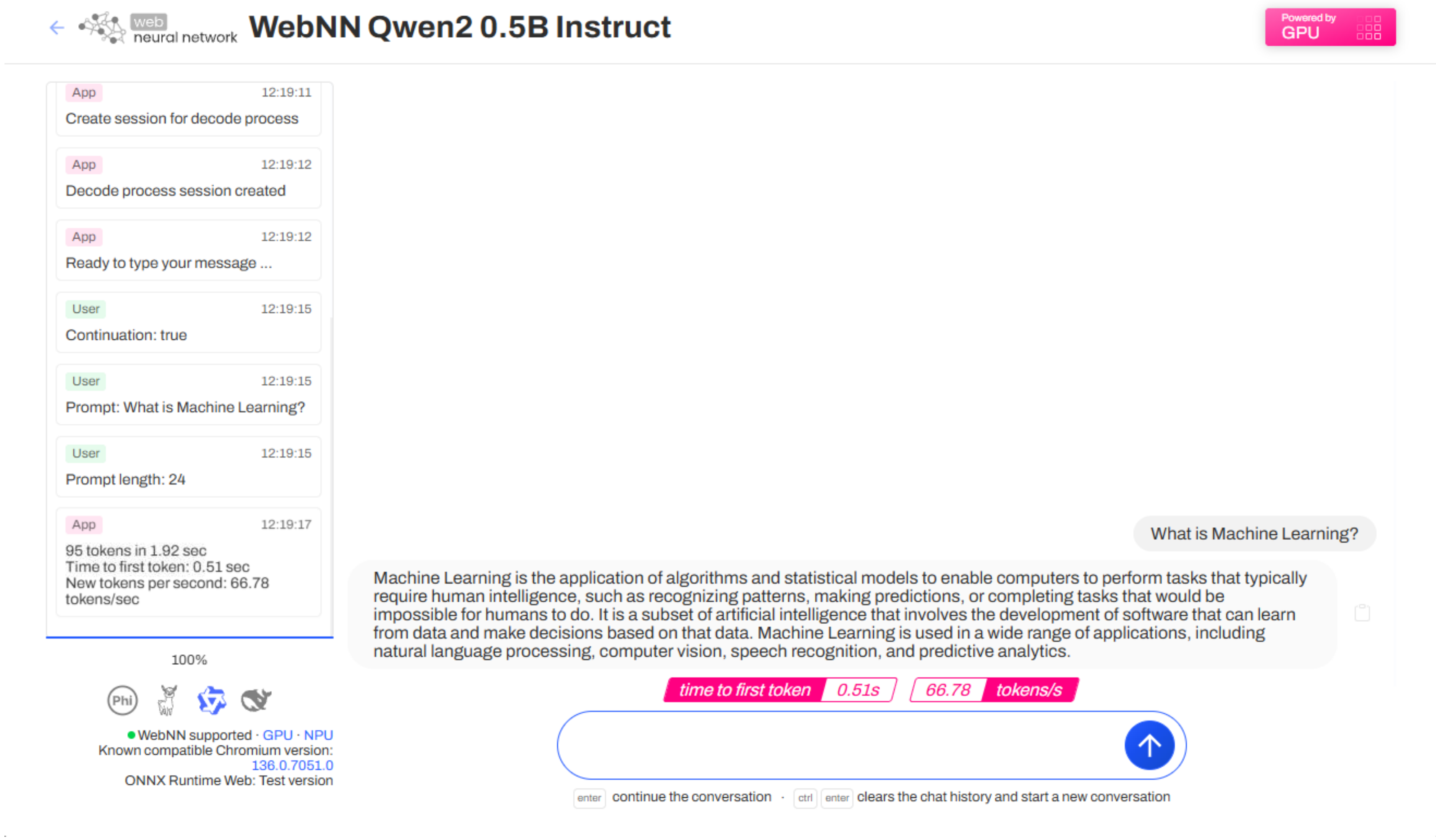
- WebNN SLM support and challenges
- Operators' fusion
- On-device KV cache
- Tensor binding
- Dynamic shape
- Results and discussion

Run ORT-GenAI SLM on WebNN: Stack Overview



WebNN Chat Demo

Use SLMs generated by ORT GenAI — fused and quantized ONNX models

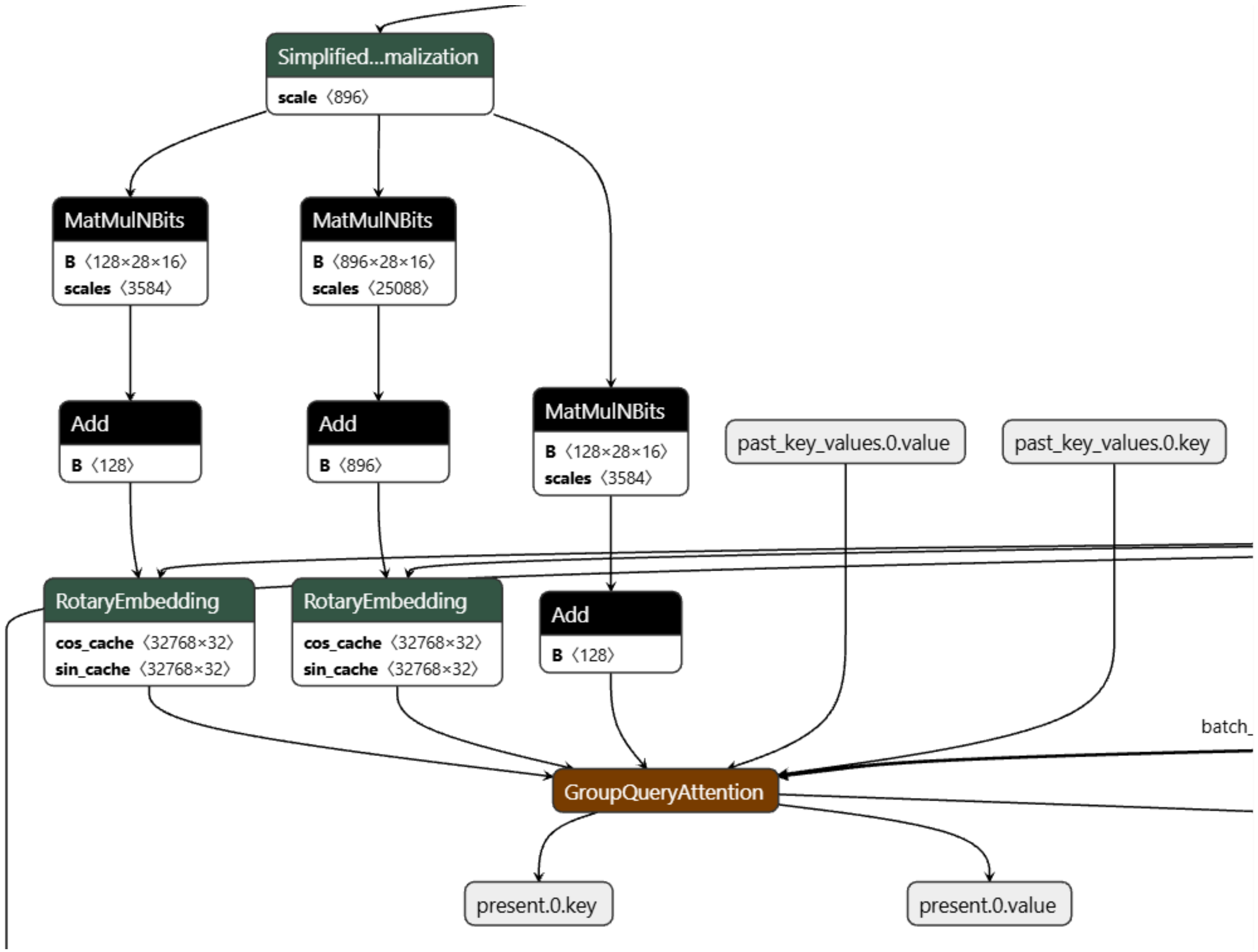


☐ Supported SLMs:

- Phi-4 Mini Instruct 3.8B (2.5G)
- Qwen2 0.5B (555M)
- TinyLLama 1.1B (714M)
- DeepSeek R1 Distill Qwen 1.5B (1.5G)

Key Building Blocks of ORT-GenAI SLM Models

- MatMulNBits
- GroupQueryAttention (GQA)
- SkipSimplifiedLayerNorm / SimplifiedLayerNorm
- RotaryEmbedding
- KV Caches
- ...



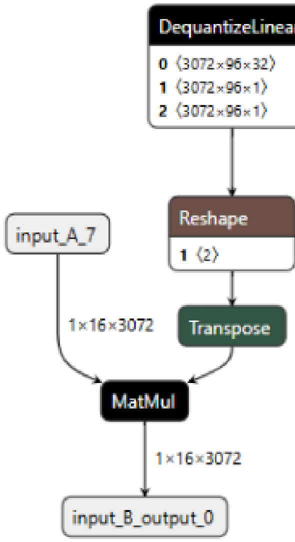
Profiling of ORT-GenAI SLM Models

- Profiling Qwen2-0.5B on native ORT-GenAI WebGPU EP
- Macro ONNX ops like MatmulNBits, GQA etc., take **74.3%** of total inference time

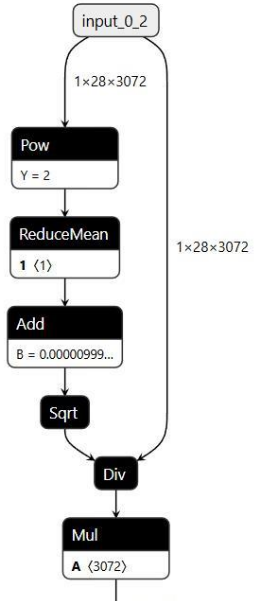
op_type	provide	pct
MatMulNBits.float16	WebGpu	**36.4**
GroupQueryAttention.float16	WebGpu	**20.7**
Add.float16	WebGpu	12.3
SkipSimplifiedLayerNormalization.float16	WebGpu	**9.2**
RotaryEmbedding.float16	WebGpu	**8**
Mul.float16	WebGpu	7
Sigmoid.float16	WebGpu	3.8
MemcpyFromHost.int64	WebGpu	0.7
MemcpyFromHost.int32	WebGpu	0.6
Gather.float16	WebGpu	0.2
SimplifiedLayerNormalization.float16	WebGpu	0.2
Gather.int64	CPU	0.2
ReduceSum.int64	CPU	0.1
Cast.int64	CPU	0.1
Shape.int64	CPU	0.1
Sub.int64	CPU	0.1
Concat.int64	CPU	0.1
Unsqueeze.int64	CPU	0.1
Reshape.int64	CPU	0.1

ONNX Ops Decomposition by WebNN Ops

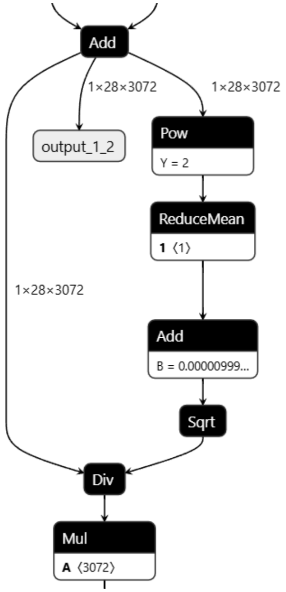
MatMulNBits



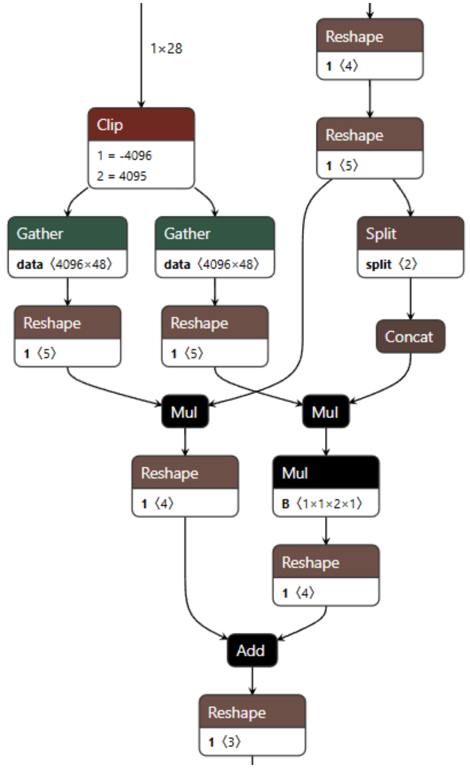
SimplifiedLayerNorm



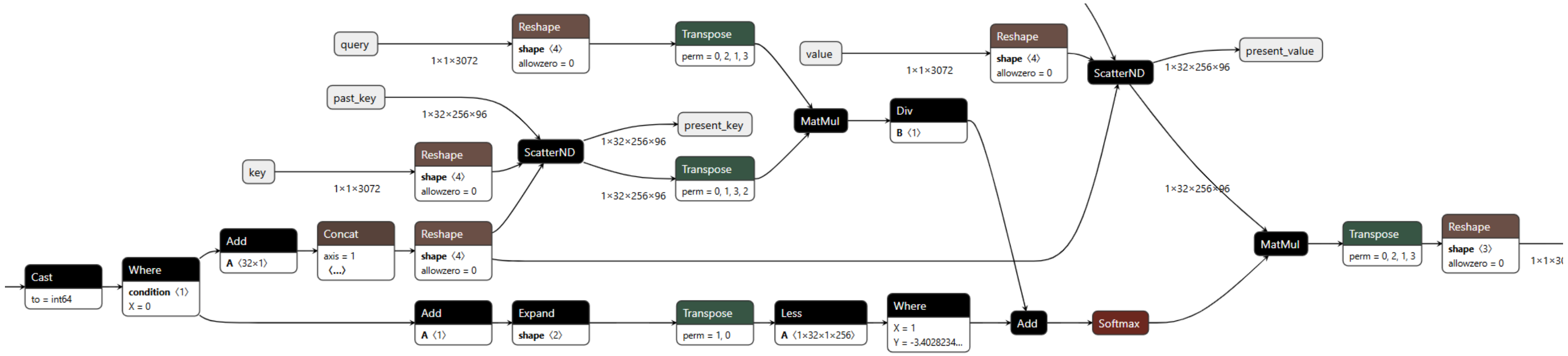
SkipSimplifiedLayerNorm



RotaryEmbedding



GroupQueryAttention



No operators' fusion: baseline performance

- Profiling Qwen2-0.5B on WebNN — the baseline
- Decomposition increase the ops count: **446 -> 2421 ops**
- Inference is blocked by DequantizeLinear (CPU kernel) and CPU-GPU data copy (83.9%): > **100X slower** than native ORT-Gen AI

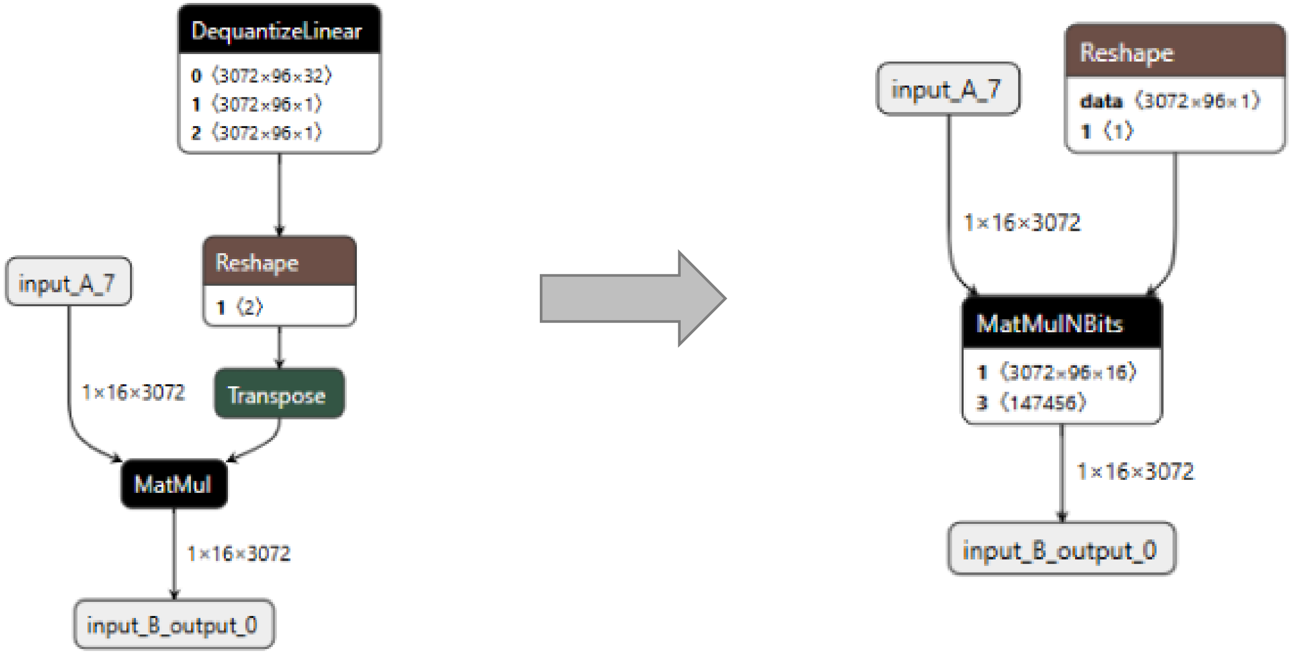
op_type	provider	pct
DequantizeLinear.UInt4x2	CPU	69.6
MemcpyFromHost.float16	WebGpu	14.3
Where.bool	WebGpu	3.6
Transpose.float16	WebGpu	2.3
MatMul.float16	WebGpu	1.3
Mul.float16	WebGpu	0.8
MatMul.float	WebGpu	0.7
Gemm.float16	WebGpu	0.6
Softmax.float	WebGpu	0.5
MemcpyToHost.int32	WebGpu	0.5
Mul.float	WebGpu	0.4
Pow.float	WebGpu	0.4
Add.float	WebGpu	0.4
Add.int32	WebGpu	0.4
Transpose.float	WebGpu	0.3
Concat.float16	WebGpu	0.3

WebNN Operation Fusion

- WebNN spec is built upon a foundation of core primitive operators
- Framework macro-ops need to be decomposed into primitive op subgraph
  - sub-optimal inference perf, increased memory pressure
- **Experiment**: Fuse performance critical macro-ops in WebNN implementation

MatMulNBits Fusion

- **26%** graph node reduction
- TTFT (time to first token): **~5x speedup**, TPS (token per second) : **~73x speedup**

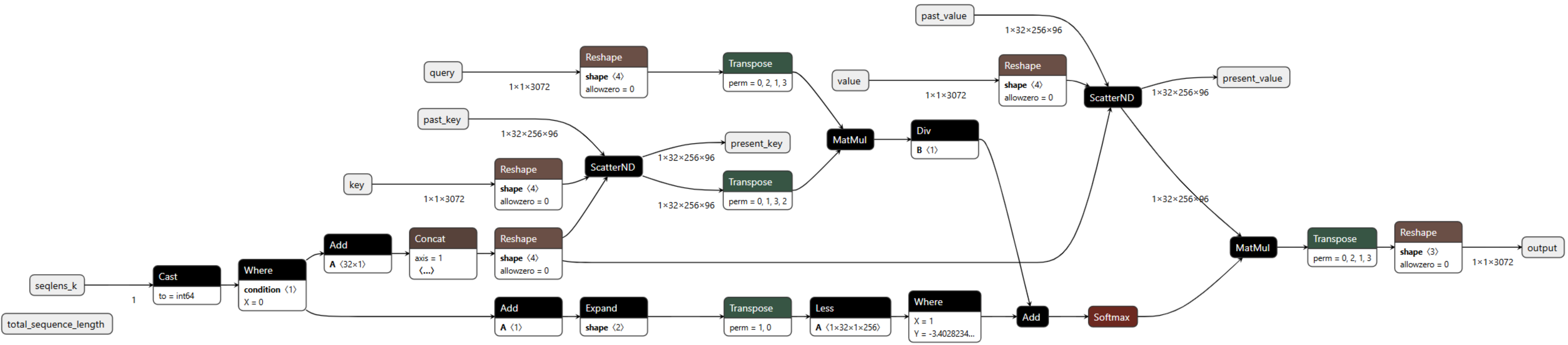


Qwen2 0.5B max_lenth = 512	time to first token (%)	tps (%)	model operations count	count rate (%)
Baseline	100.00%	100.00%	2821	100.00%
Baseline + MatMulNBits	19.12%	7333.33%	2073	73.48%

- Baseline = decomposed primitive op graph

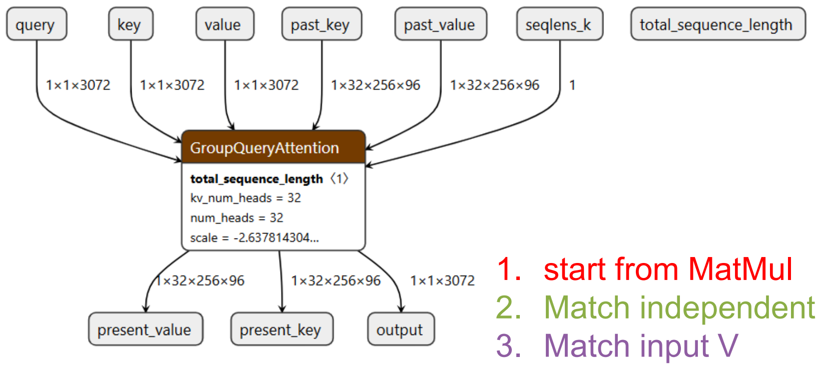
GQA Fusion

- Decomposed GQA subgraph in WebNN, 1 node → subgraph with 24 nodes

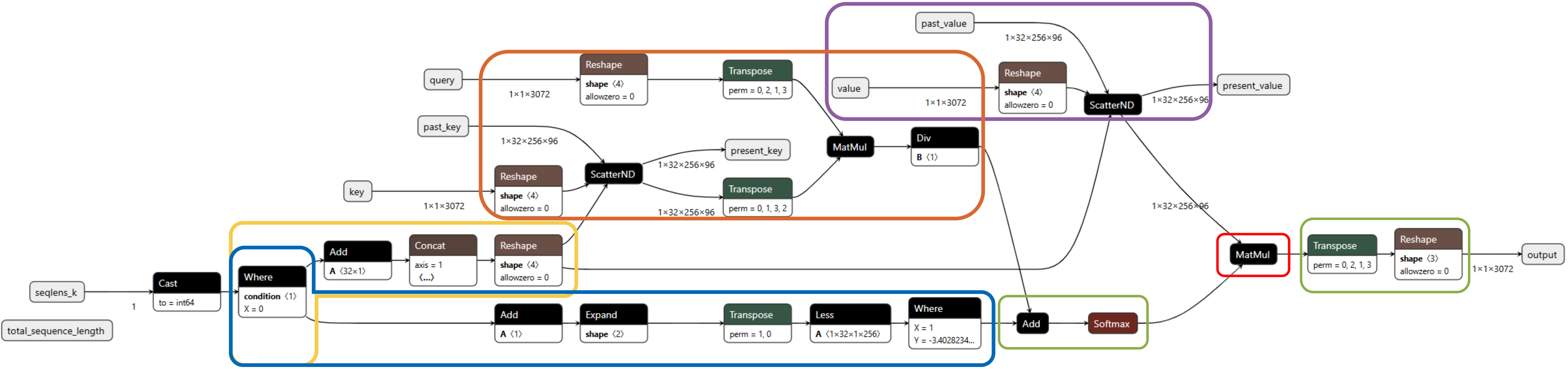


GQA Fusion

- GQA fusion: a Subgraph-Aware DFS
- **~6X speedup** over subgraph inference



1. start from MatMul
2. Match independent nodes
3. Match input V
4. Match scatter indices subgraph
5. Match attention bias subgraph
6. Match Q * K subgraph



WebNN Operation Fusion Summary

- **> 100x speedup** vs. no fusion
- **60% speedup** vs. MatMulNBits fusion only
- TPS: ~**52%** of native ORT-GenAI

Graph Node Count

Qwen2 0.5B max_lenth = 512	**model operations count**	**count rate (%)**
Origin model	446	/
Baseline	2821	100.00%
Baseline + MatMulNBits	2073	73.48%
Baseline + MatMulNBits + GQA	1413	50.09%
Baseline + MatMulNBits + GQA + others	443	15.70%

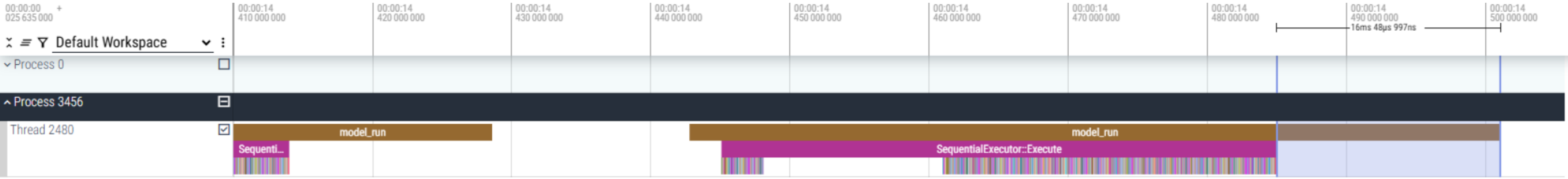
Performance

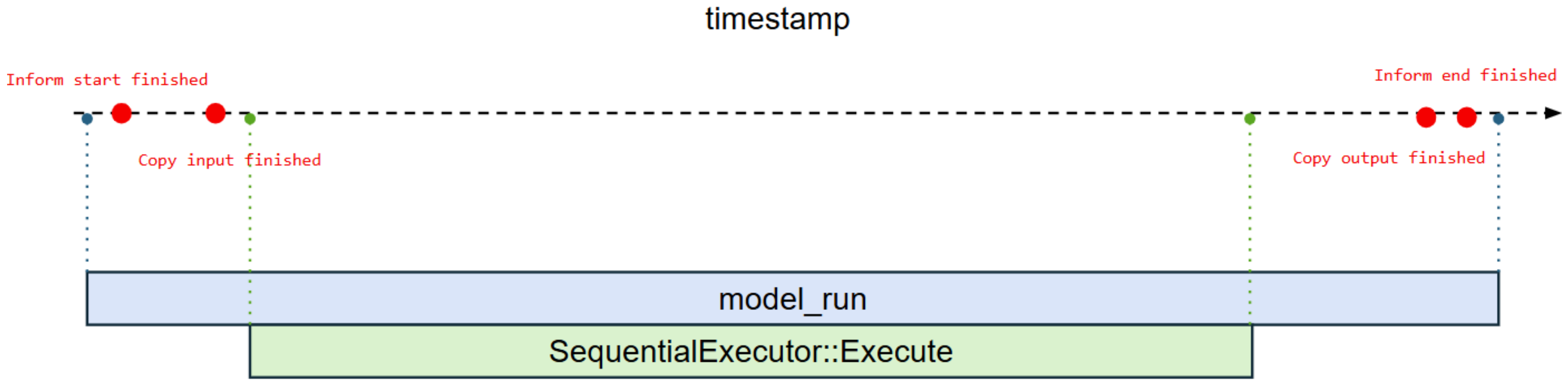
Qwen2 0.5B max_lenth = 512	**time to first token (%)**	**tps (%)**
Baseline	523.12%	1.36%
Baseline + MatMulNBits	100.00%	100.00%
Baseline + MatMulNBits + GQA	95.38%	115.39%
Baseline + MatMulNBits + GQA + others	68.21%	160.72%
Native WebGPU	39.88%	307.50%

- others = RotaryEmbedding + [Skip]SimplifiedLayerNormalization fusion

CPU-based KV Cache Overhead

- Tensor data copy takes longer than operator execution





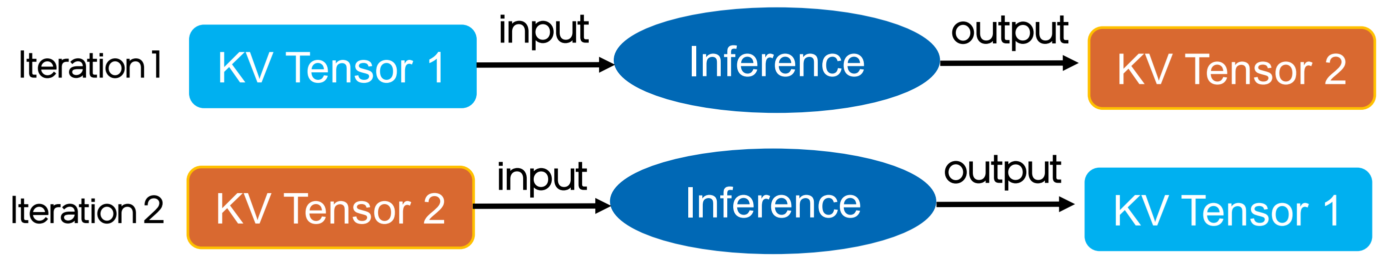
Use On-Device Tensor for KV Cache

- Brings another **~50% speedup**
- TPS: ~**78%** of native ORT-GenAI

Qwen2 0.5B max_lenth = 512	time to first token (%)	tps (%)
Baseline	523.12%	1.36%
Baseline + MatMulNBits	100.00%	100.00%
Baseline + MatMulNBits + GQA	95.38%	115.39%
Baseline + MatMulNBits + GQA + others	68.21%	160.72%
Baseline + all fusion + device tensor	76.30%	241.56%
Native WebGPU	39.88%	307.50%

- others = RotaryEmbedding + [Skip]SimplifiedLayerNormalization fusion
- all fusion = MatMulNBits + GQA + others

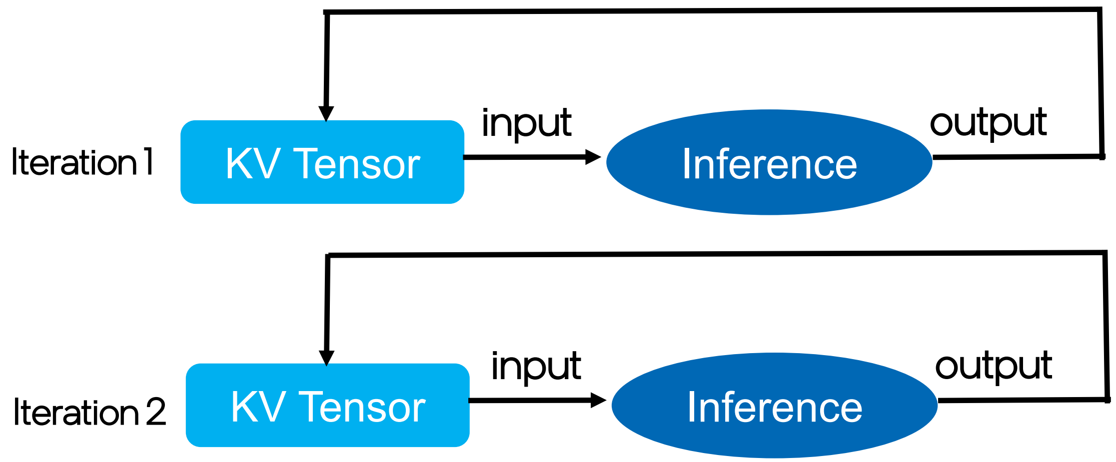
Allow same tensor for input and output



Before using same tensor

The `dispatch(graph, inputs, outputs)` method steps are:

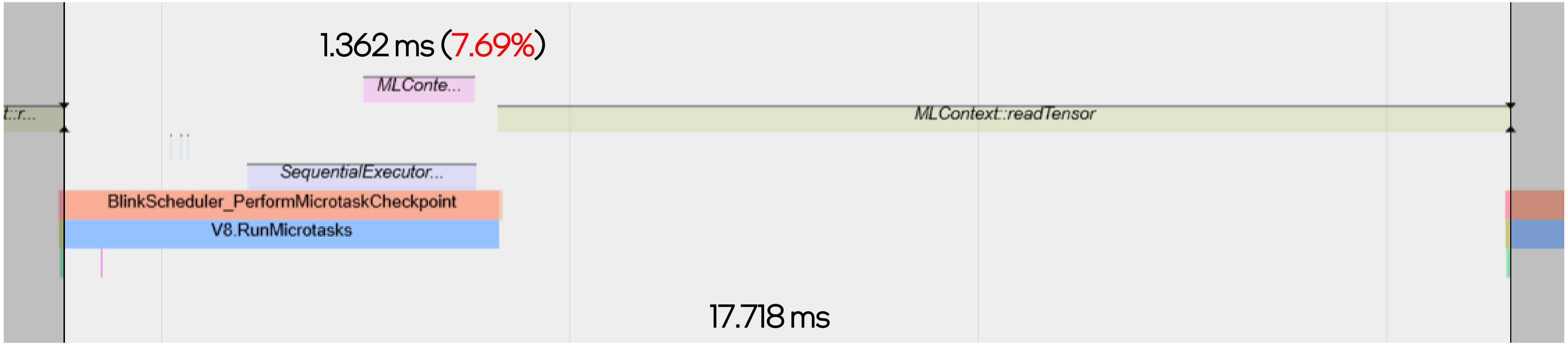
1. If *graph*.`[[context]]` is not this, then throw a `TypeError`.
2. If *graph*.`[[isDestroyed]]` is true, then throw an "`InvalidStateError`" `DOMException`.
3. Let *allTensors* be a list of `MLTensor`s consisting of *inputs*'s values extended by *outputs*'s values.
4. If *allTensors* contains any duplicate items, then throw a `TypeError`.



After using same tensor

- No need to swap KV cache tensors.
- Less # tensors -> less footprint
- Less tensor bindings for dispatch

Tensor binding overhead



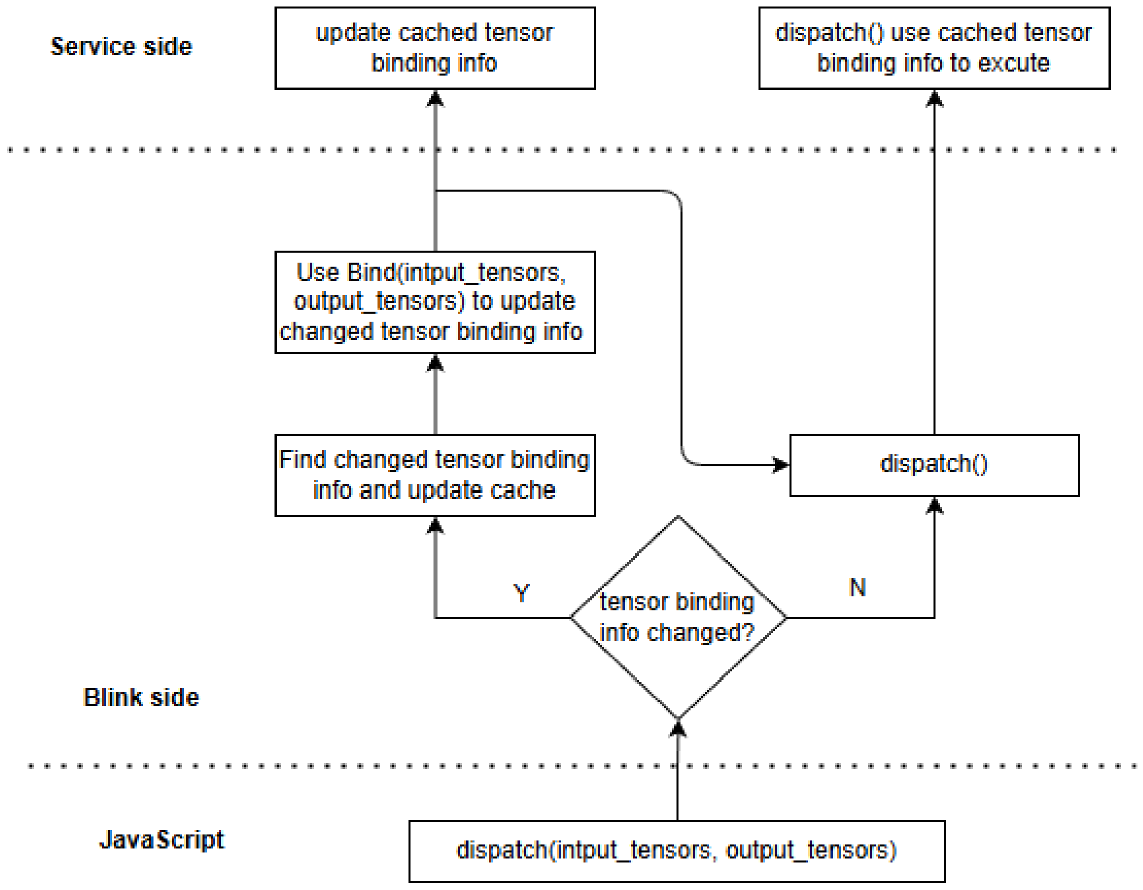
Qwen2 0.5B model:

- Input Tensors:
  - Input ids tensor
  - Attention mask tensor
  - Position ids tensor
  - 48 k-v tensors
- Output Tensors:
  - logits tensor
  - 48 k-v tensors
- Totally 100 Tensors

Issue: The tensor binding information (name -> tensor token) is transmitted at each dispatch, which also takes time.

Dispatch interface optimization

- **Option 1:** Split the dispatch IDL interface into two IDL interfaces (bind + dispatch).
  - Bind: update tensor binding information.
  - Dispatch: trigger graph execution using tensors bound via the Bind interface.
- **Option 2:** Cache tensor binding information in implementation.



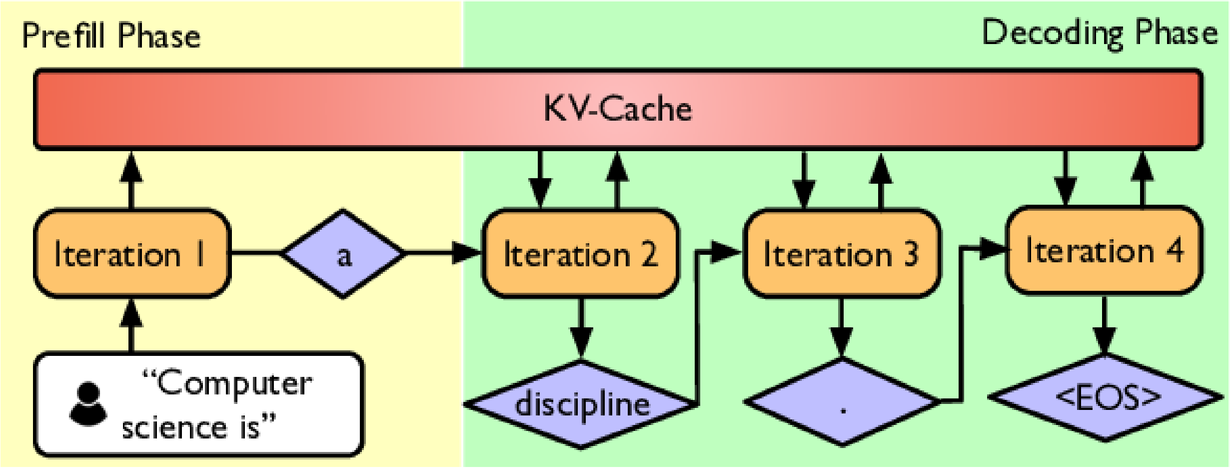
Dispatch optimization result

- Brings another **~10% speedup**
- TPS: ~**86%** of native ORT-GenAI

Qwen2 0.5B max_lenth = 512	time to first token (%)	tps (%)
Baseline	523.12%	1.36%
Baseline + MatMulNBits	100.00%	100.00%
Baseline + MatMulNBits + GQA	95.38%	115.39%
Baseline + MatMulNBits + GQA + others	68.21%	160.72%
Baseline + all fusion + device tensor	76.30%	241.56%
Baseline + all fusion + device tensor + dispatch	75.94%	266.68%
Native WebGPU	39.88%	307.50%

Dynamic shape usage for SLM

name:	**input_ids**
tensor:	`int64[batch_size,sequence_length]`
name:	**attention_mask**
tensor:	`int64[batch_size,total_sequence_length]`
name:	**position_ids**
tensor:	`int64[batch_size,sequence_length]`



- One model for both Prefill and Decoding phases
- Prefill Phase:
  - The **`input_ids`**, **`position_ids`**, and **`attention_mask`** in shape of `[batch_size, sequence_length]`
  - **`sequence_length`** is the length of the prompt which is dynamic
- Decoding Phase:
  - The **`input_ids`** and **`position_ids`** are shaped `[batch_size, 1]` because only one token is processed per step
  - The **`attention_mask`** grows dynamically to `[batch_size, total_sequence_length]`
  - **`total_sequence_length`** `= sequence_length + t` (prompt length + generated tokens) which is dynamic.

Adapt SLM to static shape

- **Static Models**:
  - Need separate static models for the **Prefill** and **Decoding** stages.
  - Each model is compiled with fixed input shapes.
- **Fixed Sequence Length**:
  - The sequence length (`sequence_length`) is set to a constant value, typically `max_sequence_length`.
  - This applies to `input_ids`, `position_ids`, and `attention_mask`.

Cons

- **Higher Memory Usage**: one more static model required
- **Double Complication Time Cost**
- **Inefficiency for Variable-Length Inputs in Prefill phase**: All inputs must be padded to `max_sequence_length`, leading to inefficiency.
- **Increased Deployment Complexity**: Managing multiple static models increases operational overhead.

Discussion

- Accessing to optimized macro ops is key for SLM performance
  - MatMulNBits, GQA etc.,
  - Support in spec or fusion in implementation?
- Support dynamic input shapes?
- Allow same tensor for input and output?
- Decouple tensor binding and graph dispatch?