Quantization Results

Results from different versions of the SmartWatch, BSPC & BSPC-GAP models

Dimitrios Yfantidis

The Examined Models

- 3 models: SmartWatch, BSPC, BSPC-GAP
- 2 categories: pre-LRF & pre-prunning
- 1 lean & 1 fat version each
- This gives a total of 12 models
- Each model is converted to TFLite format using 5 different methods, i.e. the 4 Quantization Schemes + conversion without optimizations (unquantized)
- The 4 Quantization Schemes are:
  - ✓ Dynamic Range Quantization (DRQ)
  - ✓ Full-Integer Quantization (FIQ) with unquantized (FLOAT32) I/O tensors
  - ✓ FIQ with INT8 I/O tensors
  - ✓ FIQ with UINT8 I/O tensors
- This results in 5 × 12 = 60 pipelines

```
> SMARTWATCH model
        > lean SMARTWATCH models
                > Smart_Watch_Lean : Smart_Watch_Lean
        > fat SMARTWATCH models
                > Smart_Watch_Fat : Smart_Watch_Fat
> BSPC model
        > lean BSPC models
                > BSPC_ST_CNN_5_V2 : BSPC_ST_CNN_5_V2
        > fat BSPC models
                > BSPC_seq_param4M : BSPC_seq_param50
> BSPC_GAP model
        > lean BSPC_GAP models
                > BSPC_res_param250K : BSPC_res_param250K
        > fat BSPC_GAP models
                > BSPC_res_param4M : BSPC_res_param4M
```

```
> SMARTWATCH model
        > lean SMARTWATCH models
                > model_after_pruning_lean_0.65 : model
        > fat SMARTWATCH models
                > model_after_pruning_fat_0.65 : model
> BSPC model
        > lean BSPC models
                > BSPC_seq_param100K_pruned_90 : BSPC_seq_param100K_pruned_90
        > fat BSPC models
                > BSPC_seq_param4M_pruned_80 : BSPC_seq_param4M_pruned_80
> BSPC_GAP model
        > lean BSPC_GAP models
                > BSPC_res_param250K_pruned_85 : BSPC_res_param250K_pruned_85
        > fat BSPC_GAP models
                > BSPC_res_param4M_pruned_85 : BSPC_res_param4M_pruned_85
```

Performance Metrics

The following slides contain the performance results of the 3 models' different versions for each Quantization Scheme.

SmartWatch Model – Prediction Accuracy

Post-Training Quantization (PTQ)

	Before LRF (After Pruning)		Before Pruning (Original Models)	
Quantization Scheme	Lean	Fat	Lean	Fat
Unquantized model	92.3%	94.0%	93.4%	94.9%
DRQ	92.3%	94.0%	93.4%	94.8%
FIQ – Float32	87.2%	92.7%	88.8%	91.2%
FIQ – Int8	77.6%	83.0%	80.9%	82.1%
FIQ – UInt8	75.5%	80.3%	80.0%	76.6%

SmartWatch Model – Prediction Accuracy

Quantization Aware Training (QAT)

	Before LRF (After Pruning)		Before Pruning (Original Models)	
Quantization Scheme	Lean	Fat	Lean	Fat
Unquantized model	90.8%↓	91.9%↓	91.7%↓	91.4%↓
DRQ	90.4%↓	91.8%↓	90.4%↓	91.8%↓
FIQ – Float32	90.5%↑	93.3%↑	91.5%↑	91.5%↑
FIQ – Int8	62.4%↓	70.0%↓	79.3%↓	75.0%↓
FIQ – UInt8	68.9%↓	73.4%↓	68.7%↓	71.9%↓

BSPC Model – Prediction Accuracy

Post-Training Quantization (PTQ)

	Before LRF (After Pruning)		Before Pruning (Original Models)	
Quantization Scheme	Lean	Fat	Lean	Fat
Unquantized model	85.1%	86.3%	86.2%	87.8%
DRQ	85.1%	86.4%	86.1%	87.8%
FIQ – Float32	84.9%	86.5%	85.0%	87.7%
FIQ – Int8	84.6%	85.8%	84.3%	87.4%
FIQ – UInt8	84.3%	86.5%	85.1%	87.6%

BSPC Model – Prediction Accuracy

Quantization Aware Training (QAT)

	Before LRF (After Pruning)		Before Pruning (Original Models)	
Quantization Scheme	Lean	Fat	Lean	Fat
Unquantized model	84.2%↓	85.3%↓	87.8%↑	85.7%↓
DRQ	84.5%↓	85.8%↓	87.3%↑	87.1%↓
FIQ – Float32	85.0%↑	85.9%↑	87.7%↑	87.1%↓
FIQ – Int8	82.6%↓	84.1%↓	84.4%↑	85.5%↓
FIQ – UInt8	82.9%↓	84.2%↓	85.1%~	86.5%↓

BSPC-GAP Model – Prediction Accuracy

Post-Training Quantization (PTQ)

	Before LRF (After Pruning)		Before Pruning (Original Models)	
Quantization Scheme	Lean	Fat	Lean	Fat
Unquantized model	84.0%	85.1%	85.9%	87.5%
DRQ	83.8%	85.1%	86.1%	87.6%
FIQ – Float32	82.7%	85.1%	84.4%	86.7%
FIQ – Int8	81.6%	85.0%	84.1%	86.6%
FIQ – UInt8	82.9%	85.0%	84.6%	86.3%

BSPC-GAP Model – Prediction Accuracy

Quantization Aware Training (QAT)

	Before LRF (After Pruning)		Before Pruning (Original Models)	
Quantization Scheme	Lean	Fat	Lean	Fat
Unquantized model	83.1%↓	86.0%↑	84.8%↓	86.1%↓
DRQ	84.1%↑	85.6%↑	86.4%↑	87.4%↓
FIQ – Float32	83.8%↑	85.1%~	87.3%↑	86.0%↓
FIQ – Int8	82.4%↑	85.0%~	84.0%↓	83.8%↓
FIQ – UInt8	83.7%↑	84.0%↓	81.7%↑	86.2%↓

Unquantized & DRQ models

Why are the unquantized and the DRQ TFLite models almost identical in performance?

Dimitrios Yfantidis

Example: Lean, pre-LRF SmartWatch Model

ORG

UINT8 tensors:

INT8 tensors:

INT16 tensors:

INT32 tensors:
> T#9(model/flatten/Const) shape:[2], type:INT32 RO 8 bytes, buffer: 10, data:[-1, 168]

FLOAT32 tensors:
> T#0(serving_default_x:0) shape:[1, 100, 6, 1], type:FLOAT32
> T#1(model/conv2d/BiasAdd/ReadVariableOp) shape:[4], type:FLOAT32 RO 16 bytes
> T#2(model/conv2d_1/BiasAdd/ReadVariableOp) shape:[6], type:FLOAT32 RO 24 bytes
> T#3(model/conv2d_2/BiasAdd/ReadVariableOp) shape:[3], type:FLOAT32 RO 12 bytes
> T#4(model/conv2d_3/BiasAdd/ReadVariableOp) shape:[4], type:FLOAT32 RO 16 bytes
> T#5(model/conv2d/Conv2D) shape:[4, 2, 2, 1], type:FLOAT32 RO 64 bytes
> T#6(model/conv2d_1/Conv2D) shape:[6, 2, 2, 4], type:FLOAT32 RO 384 bytes
> T#7(model/conv2d_2/Conv2D) shape:[3, 2, 2, 6], type:FLOAT32 RO 288 bytes
> T#8(model/conv2d_3/Conv2D) shape:[4, 2, 2, 3], type:FLOAT32 RO 192 bytes
> T#10(model/dense_1/BiasAdd/ReadVariableOp) shape:[6], type:FLOAT32 RO 24 bytes
> T#11(model/dense/BiasAdd/ReadVariableOp) shape:[8], type:FLOAT32 RO 32 bytes
> T#12(model/dense/MatMul) shape:[8, 168], type:FLOAT32 RO 5376 bytes
> T#13(model/dense_1/MatMul) shape:[6, 8], type:FLOAT32 RO 192 bytes
> T#14(model/.../conv2d/BiasAdd/ReadVariableOp) shape:[1, 100, 6, 4], type:FLOAT32
> T#15(model/max_pooling2d/MaxPool) shape:[1, 50, 6, 4], type:FLOAT32
> T#16(model/.../conv2d_1/BiasAdd/ReadVariableOp) shape:[1, 50, 6, 6], type:FLOAT32
> T#17(model/max_pooling2d_1/MaxPool) shape:[1, 25, 6, 6], type:FLOAT32
> T#18(model/.../conv2d_2/BiasAdd/ReadVariableOp) shape:[1, 25, 6, 3], type:FLOAT32
> T#19(model/max_pooling2d_2/MaxPool) shape:[1, 13, 6, 3], type:FLOAT32
> T#20(model/.../conv2d_3/BiasAdd/ReadVariableOp) shape:[1, 13, 6, 4], type:FLOAT32
> T#21(model/max_pooling2d_3/MaxPool) shape:[1, 7, 6, 4], type:FLOAT32
> T#22(model/flatten/Reshape) shape:[1, 168], type:FLOAT32
> T#23(model/…/dense/BiasAdd) shape:[1, 8], type:FLOAT32
> T#24(model/dense_1/MatMul;model/dense_1/BiasAdd) shape:[1, 6], type:FLOAT32
> T#25(StatefulPartitionedCall:0) shape:[1, 6], type:FLOAT32

DRQ

UINT8 tensors:

INT8 tensors:
> T#13(model/dense/MatMul) shape:[8, 168], type:INT8 RO 1344 bytes

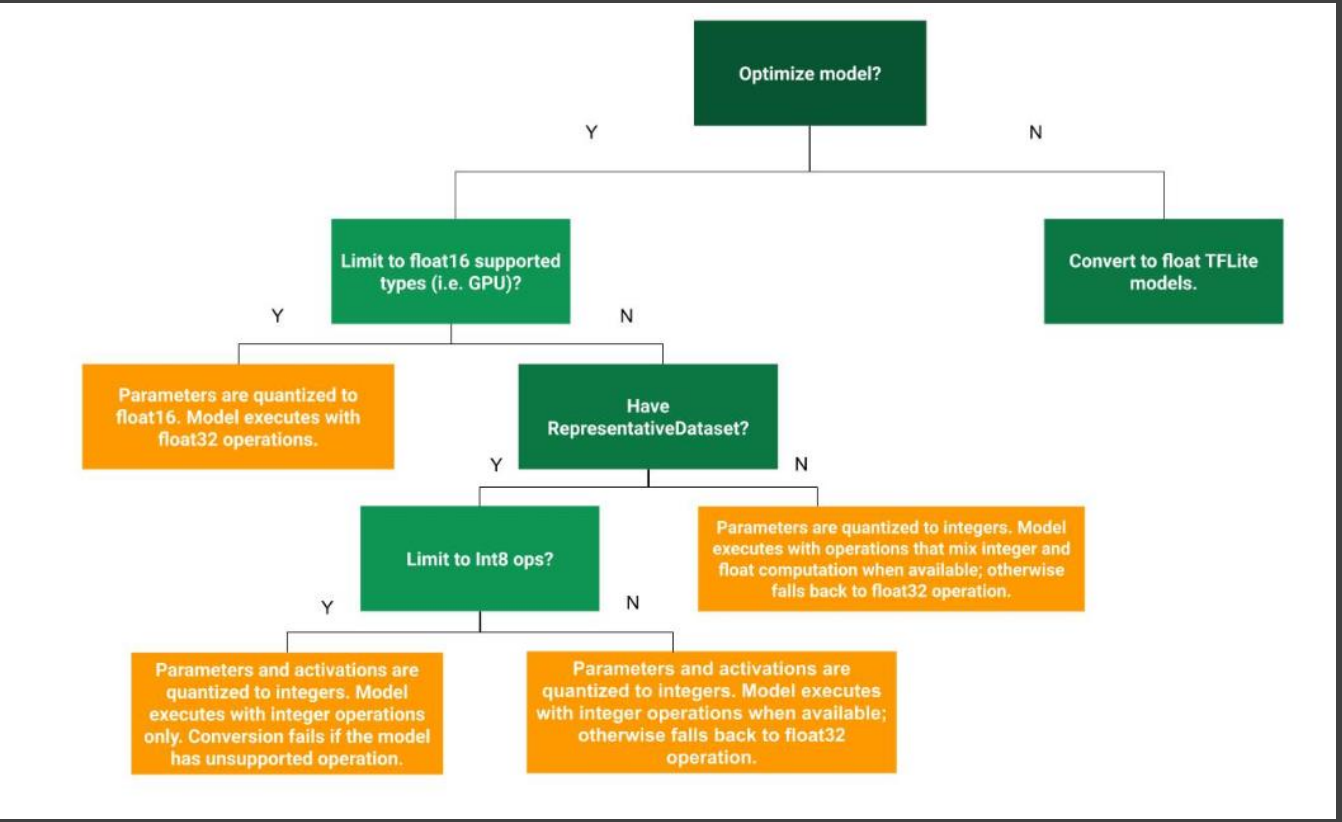
INT16 tensors:

INT32 tensors:
> T#10(model/flatten/Const) shape:[2], type:INT32 RO 8 bytes

FLOAT32 tensors:
> T#0(serving_default_x:0) shape:[1, 100, 6, 1], type:FLOAT32
> T#1(model/dense_1/MatMul) shape:[6, 8], type:FLOAT32 RO 192 bytes
> T#2(model/conv2d/BiasAdd/ReadVariableOp) shape:[4], type:FLOAT32 RO 16 bytes
> T#3(model/conv2d_1/BiasAdd/ReadVariableOp) shape:[6], type:FLOAT32 RO 24 bytes
> T#4(model/conv2d_2/BiasAdd/ReadVariableOp) shape:[3], type:FLOAT32 RO 12 bytes
> T#5(model/conv2d_3/BiasAdd/ReadVariableOp) shape:[4], type:FLOAT32 RO 16 bytes
> T#6(model/conv2d/Conv2D) shape:[4, 2, 2, 1], type:FLOAT32 RO 64 bytes, buffer: 7
> T#7(model/conv2d_1/Conv2D) shape:[6, 2, 2, 4], type:FLOAT32 RO 384 bytes, buffer: 8
> T#8(model/conv2d_2/Conv2D) shape:[3, 2, 2, 6], type:FLOAT32 RO 288 bytes, buffer: 9
> T#9(model/conv2d_3/Conv2D) shape:[4, 2, 2, 3], type:FLOAT32 RO 192 bytes, buffer: 10
> T#11(model/dense_1/BiasAdd/ReadVariableOp) shape:[6], type:FLOAT32 RO 24 bytes
> T#12(model/dense/BiasAdd/ReadVariableOp) shape:[8], type:FLOAT32 RO 32 bytes
> T#14(model/.../conv2d/BiasAdd/ReadVariableOp) shape:[1, 100, 6, 4], type:FLOAT32
> T#15(model/max_pooling2d/MaxPool) shape:[1, 50, 6, 4], type:FLOAT32
> T#16(model/.../conv2d_1/BiasAdd/ReadVariableOp) shape:[1, 50, 6, 6], type:FLOAT32
> T#17(model/max_pooling2d_1/MaxPool) shape:[1, 25, 6, 6], type:FLOAT32
> T#18(model/.../conv2d_2/BiasAdd/ReadVariableOp) shape:[1, 25, 6, 3], type:FLOAT32
> T#19(model/max_pooling2d_2/MaxPool) shape:[1, 13, 6, 3], type:FLOAT32
> T#20(model/.../conv2d_3/BiasAdd/ReadVariableOp) shape:[1, 13, 6, 4], type:FLOAT32
> T#21(model/max_pooling2d_3/MaxPool) shape:[1, 7, 6, 4], type:FLOAT32
> T#22(model/flatten/Reshape) shape:[1, 168], type:FLOAT32
> T#23(model/…/dense/BiasAdd) shape:[1, 8], type:FLOAT32
> T#24(model/dense_1/MatMul;model/dense_1/BiasAdd) shape:[1, 6], type:FLOAT32
> T#25(StatefulPartitionedCall:0) shape:[1, 6], type:FLOAT32

Technical Explaination

- Follow the Y → N → N path.
- That path leads to Dynamic Range Quantization.
- From the TF Guide's description we can expect that it is not impossible for a model to be consisting solely of float tensors.
- Thus, the DRQ models that had the same performance with the original models:
  - ✓ Received limited quantization
  - ✓ Were quantized perfectly
  - ✓ Combination of both



Inference Code Snippet

The model is loaded and the TFLite Interpreter is initialized from it, along with input and output details (used for inferencing the given data).

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

Inference Code Snippet

Determine the number of the model's labels from the output details and shape the prediction buffer accordingly (if multiclass).

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

Inference Code Snippet

The midpoint of the output data type is derived (used in multilabel classification).

If output dtype is float, then it is not quantized and, thus, the DNN outputs sigmoidal scalars in [0, 1]. In this case its midpoint is 0.5.

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

Inference Code Snippet

The midpoint of the output data type is derived (used in multilabel classification).

If output dtype is int8, then the output tensor has been quantized and, thus, the DNN outputs integer scalars in [-128, 127]. In this case its midpoint is |(-128 + 127) / 2| + 1 = 0.

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

Inference Code Snippet

The midpoint of the output data type is derived (used in multilabel classification).

If output dtype is uint8, then the output tensor has been quantized and, thus, the DNN outputs integer scalars in [0, 255]. In this case its midpoint is |(0+ 255) / 2| + 1 = 128.

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

Inference Code Snippet

Callibrates the input data, X (which should always be float), by performing the affine quantization mapping:

$$r = S(q - Z)$$

Which inversely gives:

$$q = \frac{r}{S} + Z$$

If the input tensor was not quantized, then $S = 1$ and $Z = 0$.

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

Inference Code Snippet

Standard procedure for inferencing data in TensorFlow Lite (setting input sample, invoking interpreter and fetching the prediction).

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

Inference Code Snippet

In case of multilabel classification, the midpoint is used for thresholding the output, regardless of quantization status.

E.g. float: [0.6, 0.1, 0.3, 0.8, 0.9] becomes [1, 0, 0, 1, 1]
E.g. int8: [-128, -110, 127, 80, -128] becomes [0, 0, 1, 1, 0]
E.g. uint8: [255, 230, 0, 0, 1] becomes [1, 1, 0, 0, 0]

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

Inference Code Snippet

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

In the case of single label classification for n_classes > 2, the result is much simpler to compute by argmax-ing the DNN's output vector (which is midpoint agnostic)

Inference Code Snippet

```python
# Runs inference on the given dataset using the specified classifier, if provided.
# If the model is not provided, then the examined model of the TFLiteManager's session is used.
# This allows for a flexible interface to run inference on other, irrelevant classifiers as well.
def run_tflite_model(self, X: np.ndarray, model_content: bytes|None=None, is_multiclass: bool=False) -> np.ndarray:
    # Standard code segment for running inference
    # using a TFLite model using the TFLite Interpreter
    if model_content is None:
        model_content = self._tflite_model
    interpreter = tf.lite.Interpreter(model_content=model_content)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()[0]
    output_details = interpreter.get_output_details()[0]
    n_classes = output_details['shape'][-1]
    # Initialize the output matrix
    out_dtype = output_details['dtype']
    predictions = np.zeros(
        shape=(X.shape[0], n_classes) if is_multiclass else X.shape[0],
        dtype=out_dtype
    )
    # It is assumed that float models produce output vectors where each of their scalars are in the range [0, 1]. Thus
    # the mid-point is arbitrarily assumed to be 1/2. In case that this isn't true (e.g.: output in [0, 10] or [-1, 1]),
    # this method, as well as the `evaluate_tflite_model` method, must be refactored accordingly.
    mid_point = .5 if np.issubdtype(out_dtype, np.floating) else (np.iinfo(out_dtype).max + np.iinfo(out_dtype).min) // 2 + 1
    for i, sample in enumerate(X):
        # Callibrate the quantized input
        input_scale, input_zero_point = input_details["quantization"] if input_details["quantization"] != (0.0, 0) else (1.0, 0)
        sample = sample / input_scale + input_zero_point
        # Morph the initial sample in an acceptable vector for the Interpreter
        sample = np.expand_dims(sample, axis=0).astype(input_details["dtype"])
        interpreter.set_tensor(input_details["index"], sample)
        interpreter.invoke()
        output = interpreter.get_tensor(output_details["index"])[0]
        # Callibrate the quantized output
        # output_scale, output_zero_point = output_details["quantization"] if output_details["quantization"] != (0.0, 0) else (1.0, 0)
        # output = output_scale * (output - output_zero_point)
        if is_multiclass:
            # one-hot encoded output
            idx = (output >= mid_point)
            output[idx] = 1
            output[~idx] = 0
            predictions[i] = output
        else:
            # Assumes > 2 classes; Binary classification demands thresholding
            # the single sigmoid output, not argmax-ing the softmax/logits outputs.
            predictions[i] = output.argmax()
    return predictions
```

NOTE: This code should be slightly modified to support single-label, binary classification as, in this case, the output is fetched from thresholding the single sigmoid.

In the case of single label classification for n_classes > 2, the result is much simpler to compute by argmax-ing the DNN's output vector (which is midpoint agnostic)

Quantization Debugger

The following slides contain the Quantization Debugger's output metrics charts for the SmartWatch model's different versions.

Dimitrios Yfantidis

Quantization Debugger

The following slides contain the Quantization Debugger's output metrics charts for the SmartWatch model's different versions.

NOTE #1: Using the Quantization Debugger is valid only for the FIQ-INT8 models of the Post-Training Quantization strategy.

Dimitrios Yfantidis

Quantization Debugger

The following slides contain the Quantization Debugger's output metrics charts for the SmartWatch model's different versions.

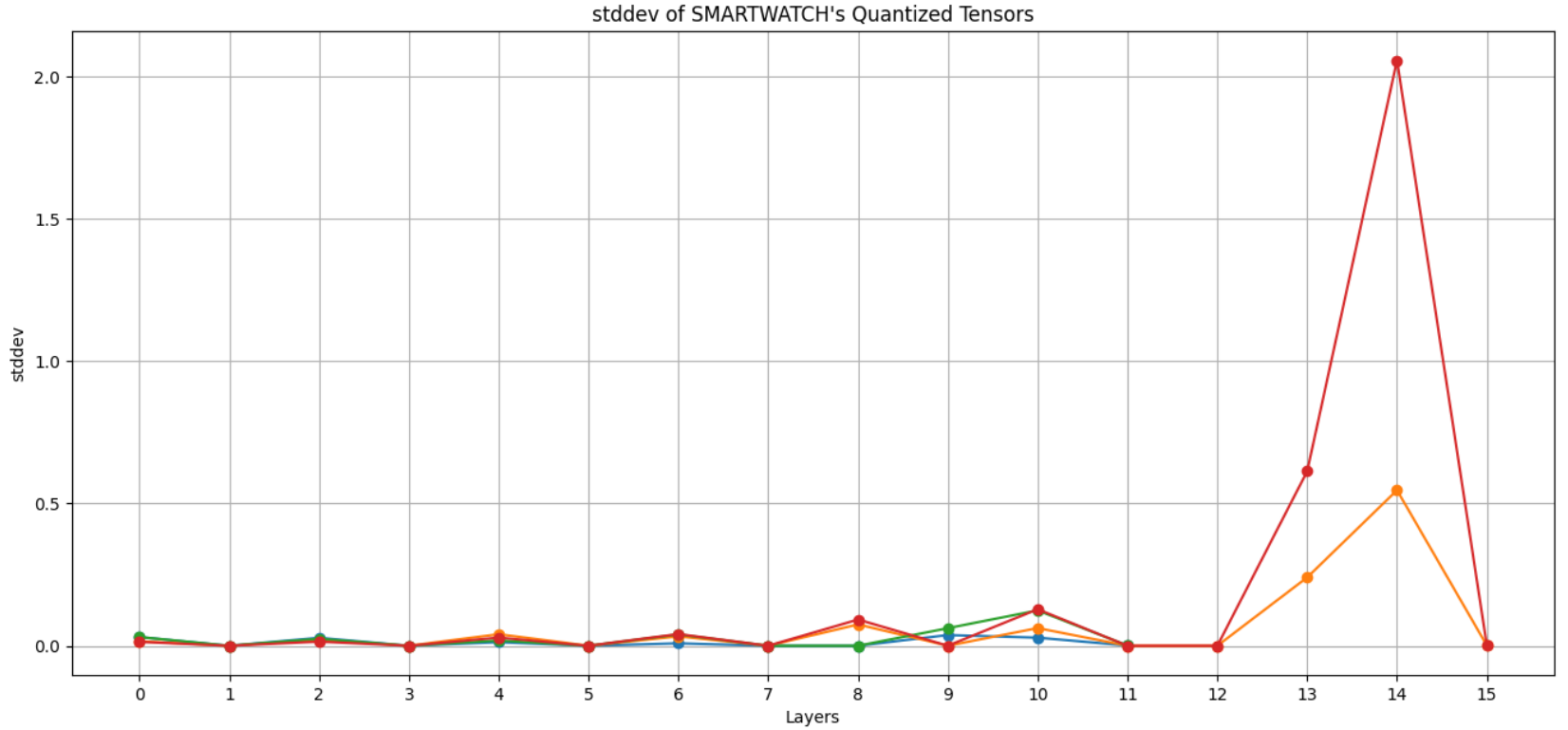
NOTE #1: Using the Quantization Debugger is valid only for the FIQ-INT8 models of the Post-Training Quantization strategy.

NOTE #2: The utilization of the Quantization Debugger and the Quantization Aware Training strategy are mutually exclusive options in the scope of the TF library.

Dimitrios Yfantidis

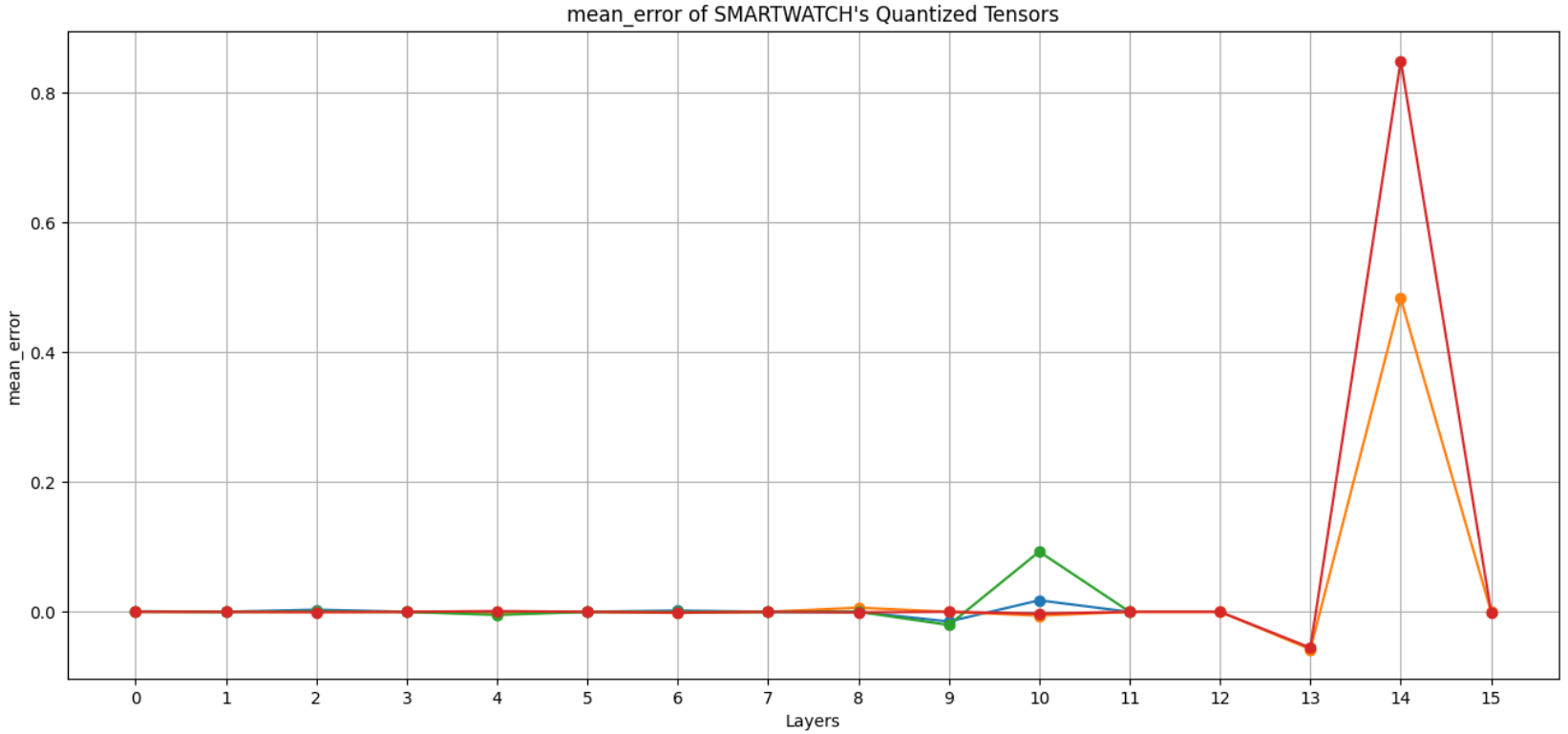
SmartWatch Model – Quantization Debugger (1)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



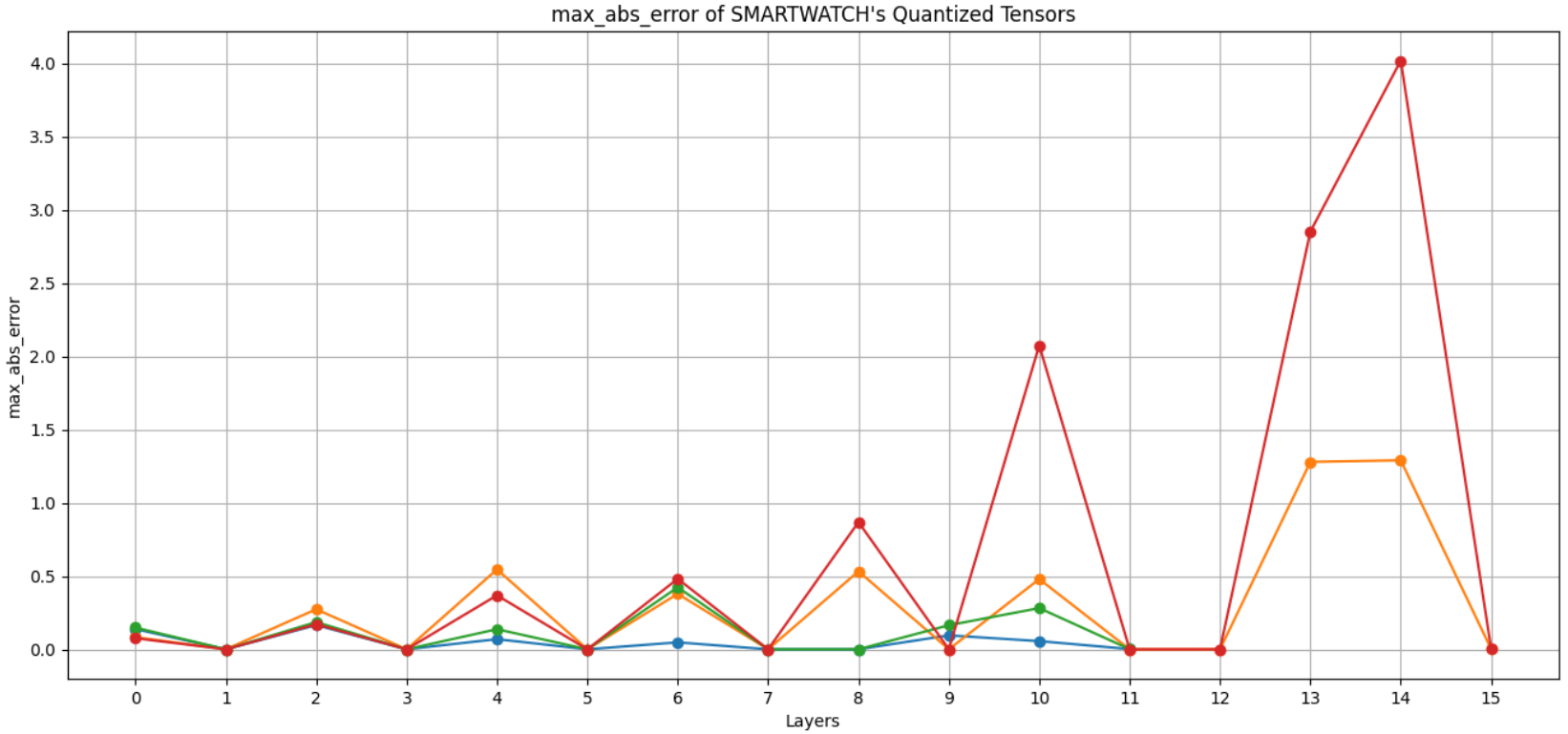
SmartWatch Model – Quantization Debugger (2)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



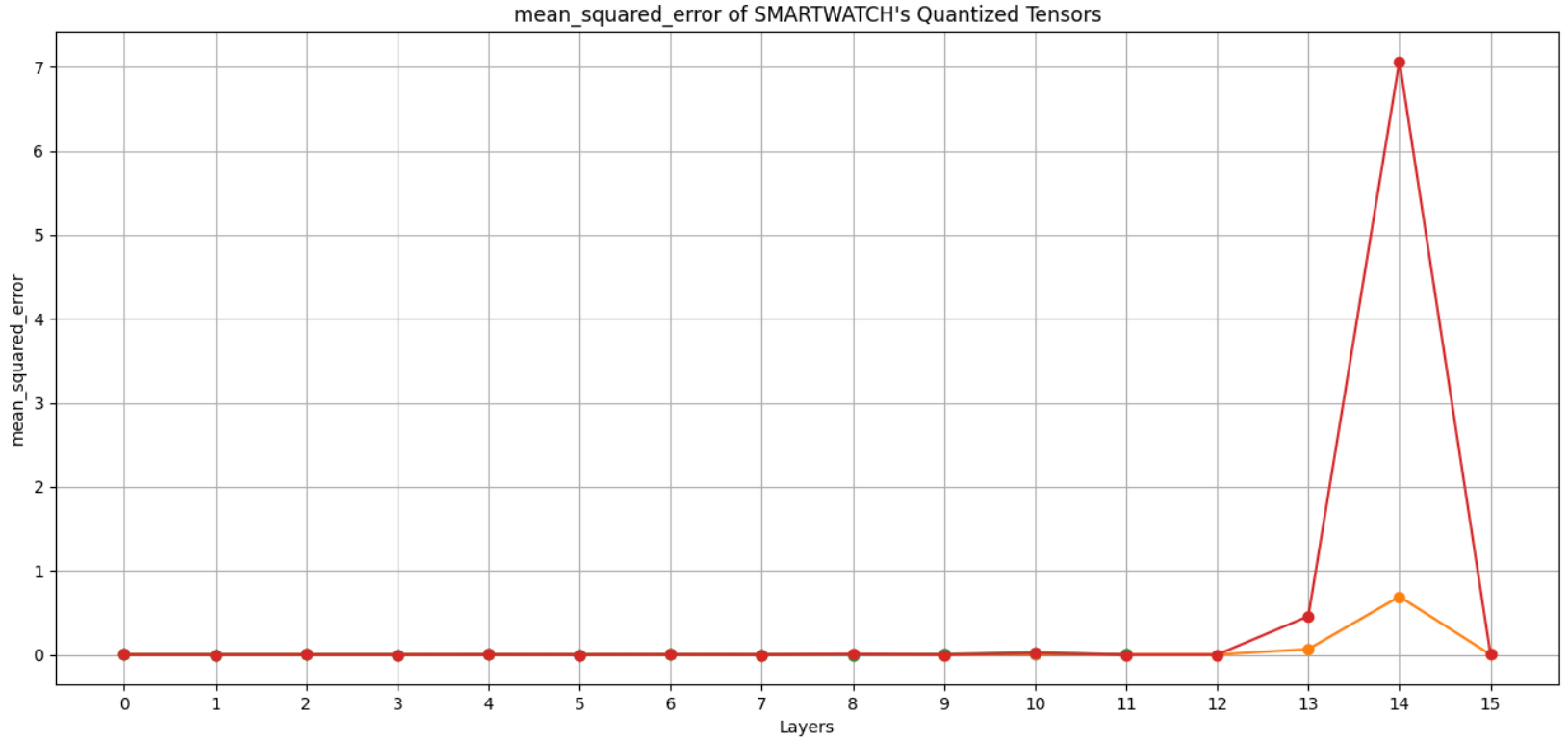
SmartWatch Model – Quantization Debugger (3)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



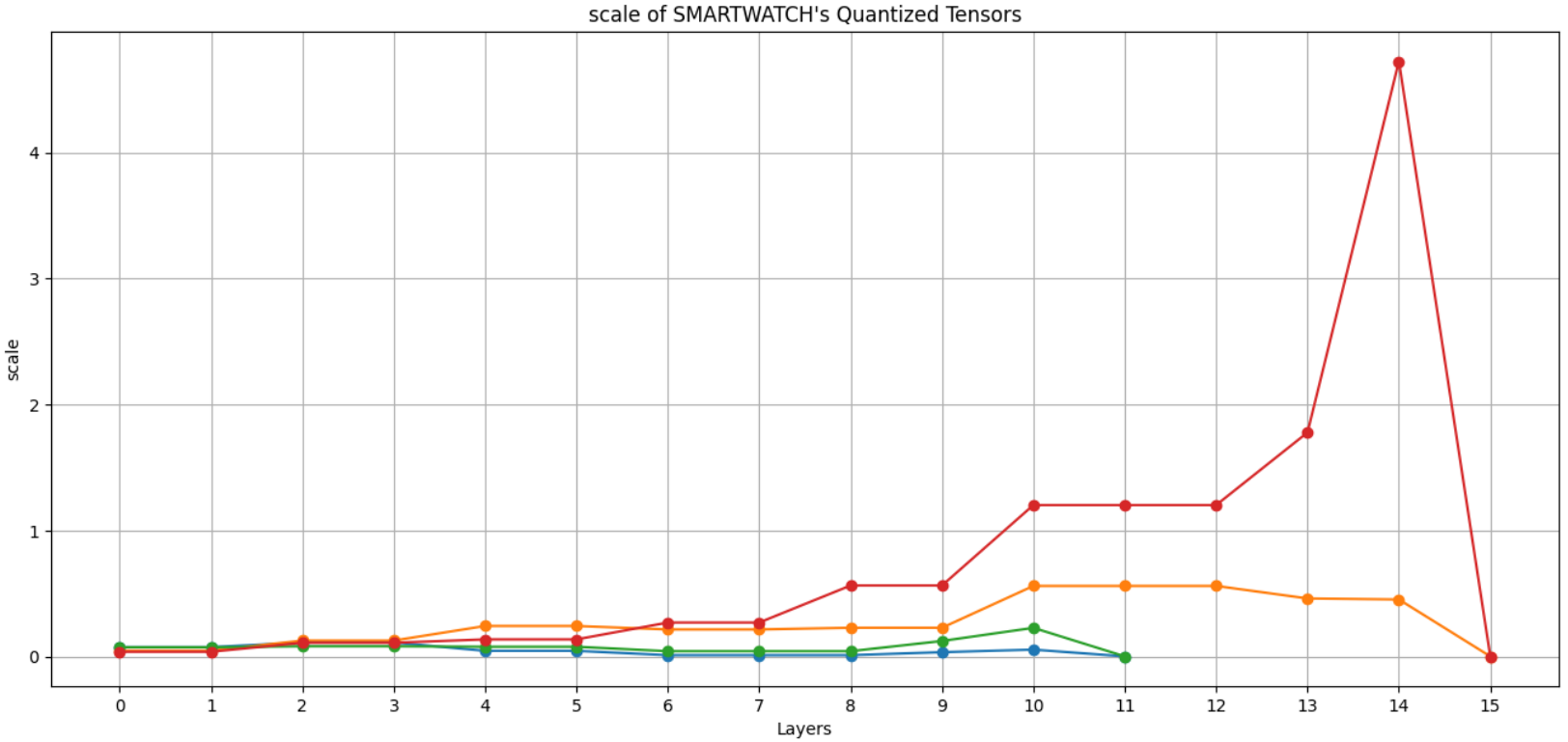
SmartWatch Model – Quantization Debugger (4)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



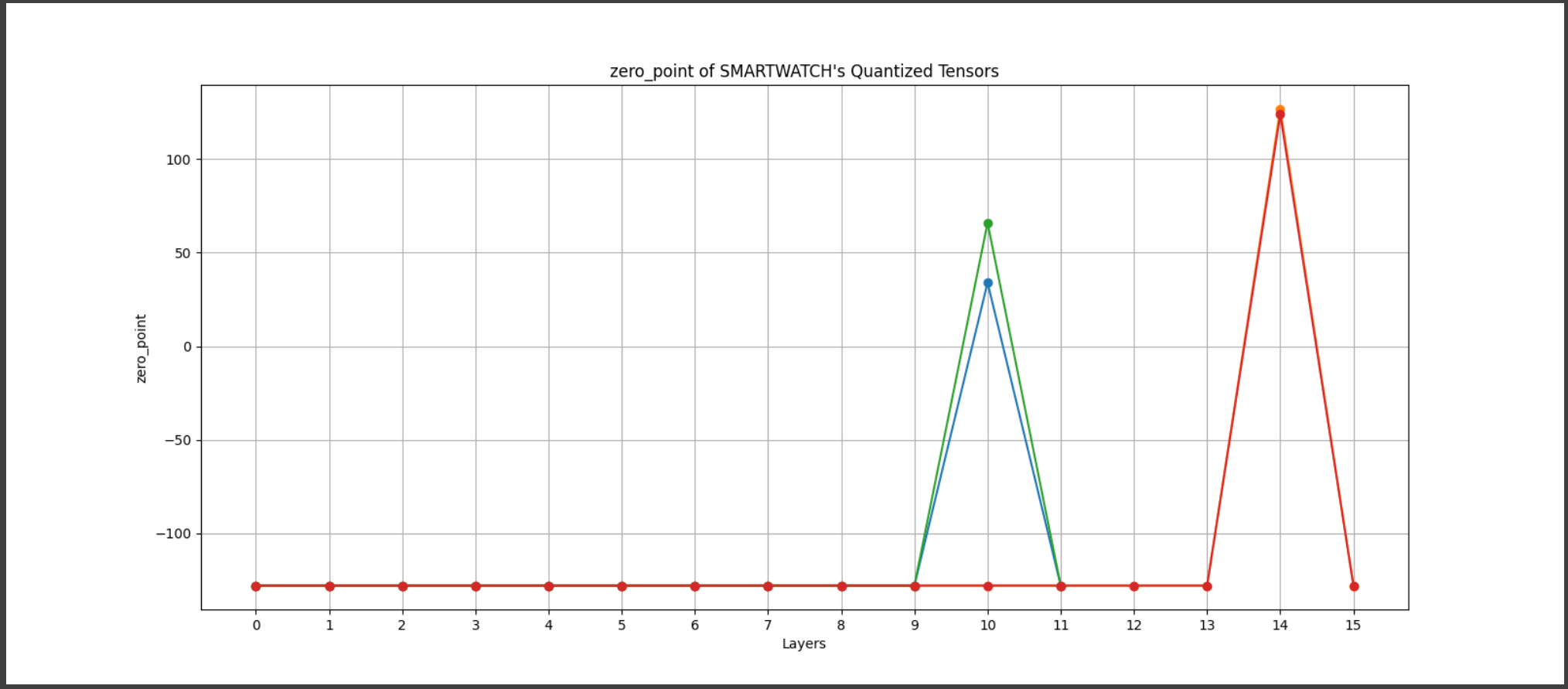
SmartWatch Model – Quantization Debugger (5)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



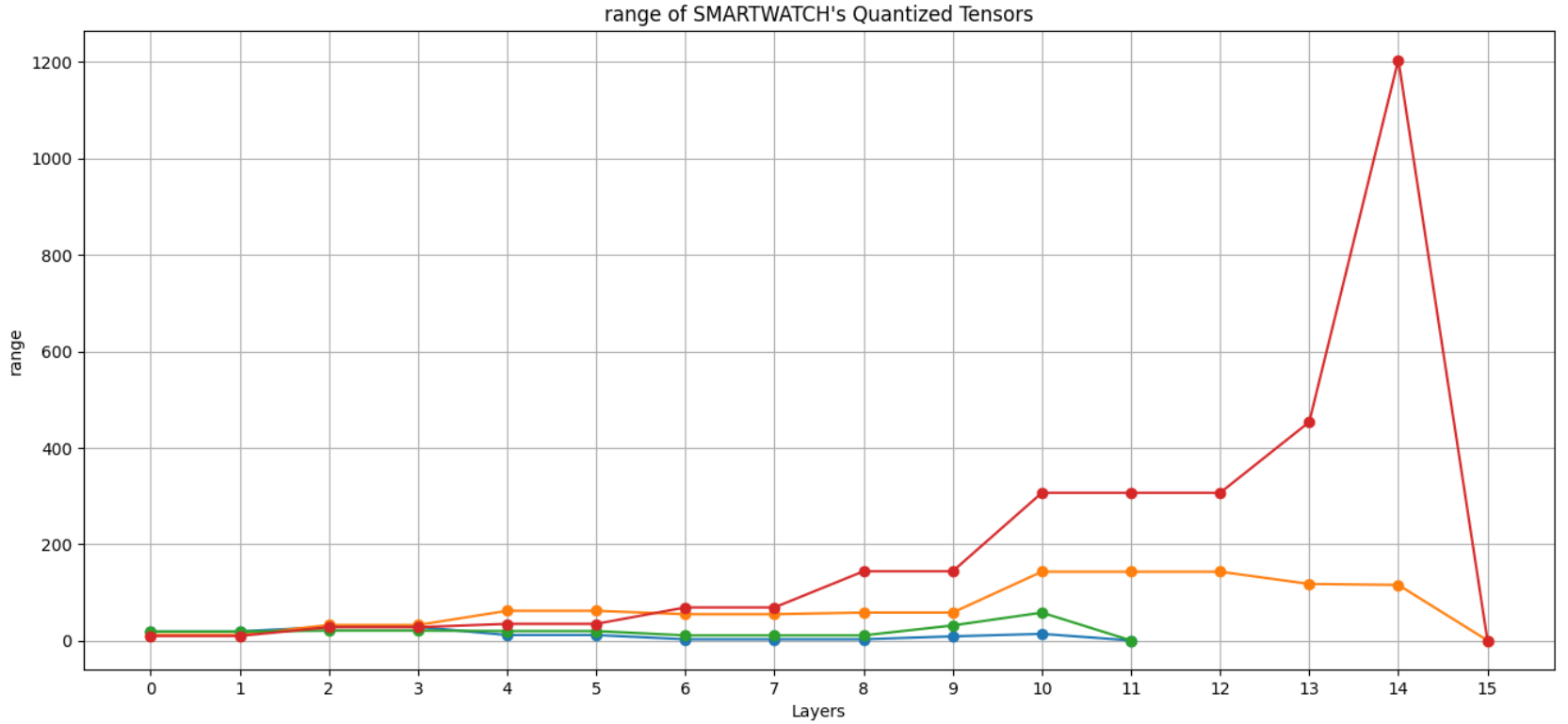
SmartWatch Model – Quantization Debugger (6)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



SmartWatch Model – Quantization Debugger (7)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



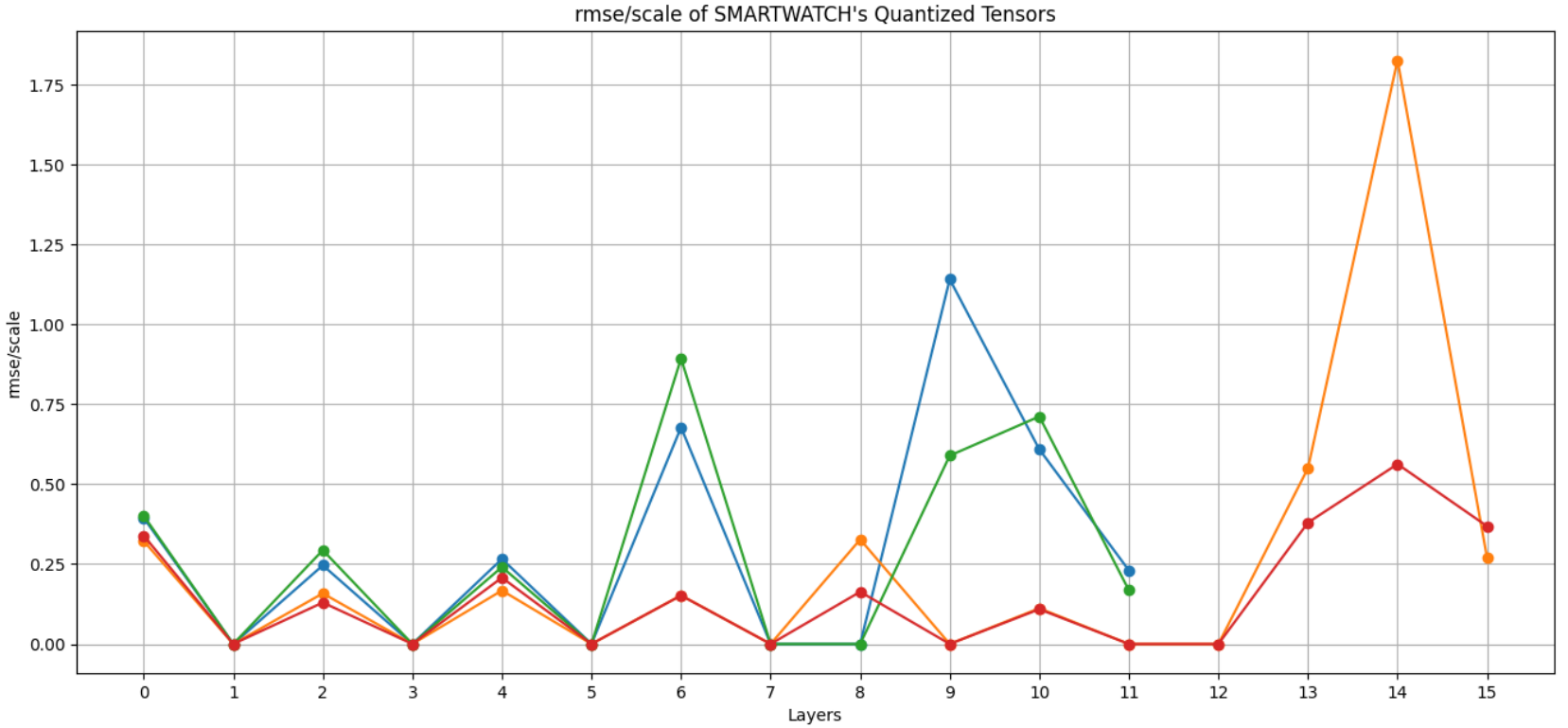
SmartWatch Model – Quantization Debugger (8)

Pre-LRF LEAN ver.

Pre-LRF FAT ver.

Pre-pruning LEAN ver.

Pre-pruning FAT ver.



Layer Indexing

The following slide maps the horizontal axis' indices to the layers of each SmartWatch model version. This contributes to the better interpretation of the charts.

Dimitrios Yfantidis

SmartWatch Model – Layer Indexing

Pre-LRF LEAN ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	RESHAPE
9	FULLY_CONNECTED
10	FULLY_CONNECTED
11	SOFTMAX

Pre-LRF FAT ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MAX_POOL_2D
12	RESHAPE
13	FULLY_CONNECTED
14	FULLY_CONNECTED
15	SOFTMAX

Pre-pruning LEAN ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	RESHAPE
9	FULLY_CONNECTED
10	FULLY_CONNECTED
11	SOFTMAX

Pre-pruning FAT ver.

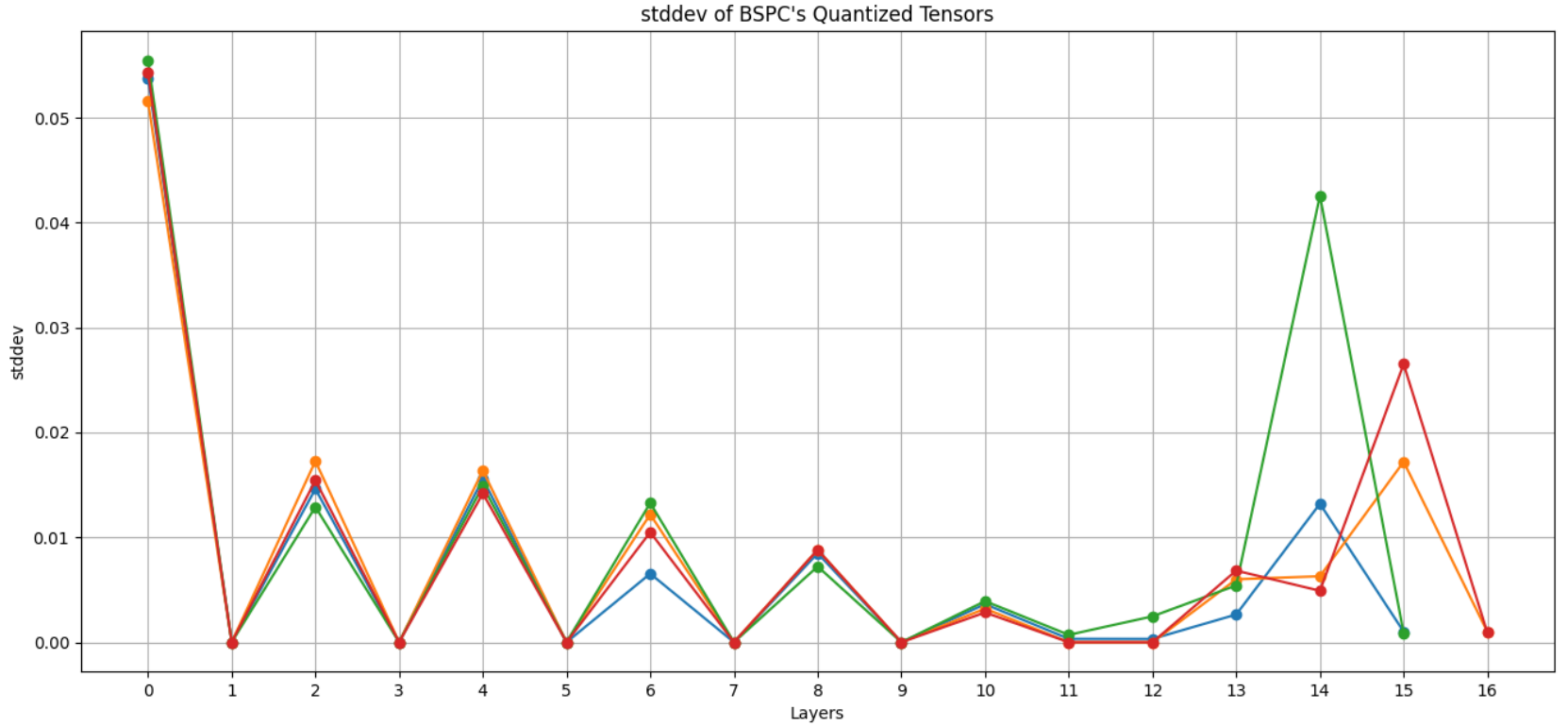
	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MAX_POOL_2D
12	RESHAPE
13	FULLY_CONNECTED
14	FULLY_CONNECTED
15	SOFTMAX

Next Models

The previous slides are repeated twice for the BSPC and BSPC-GAP models.

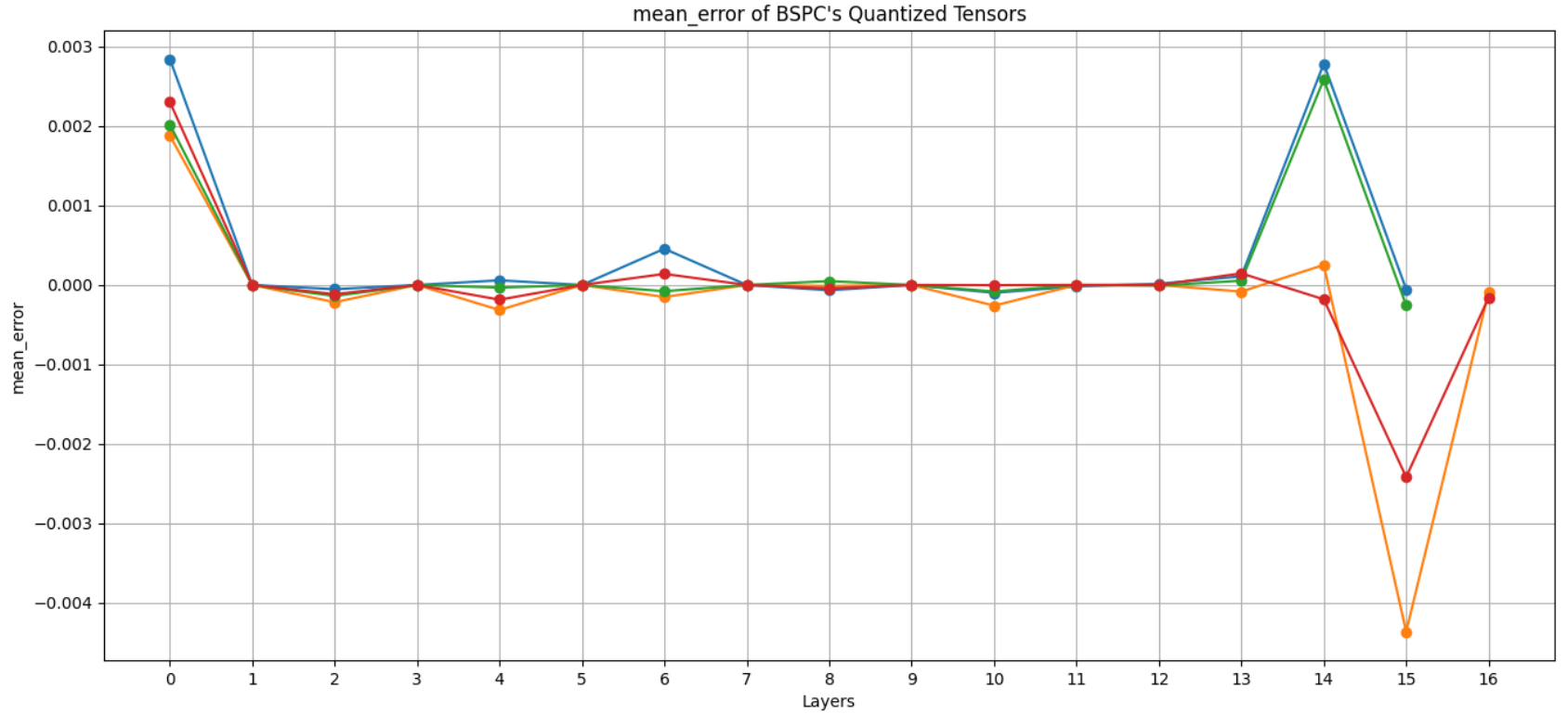
BSPC Model – Quantization Debugger (1)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



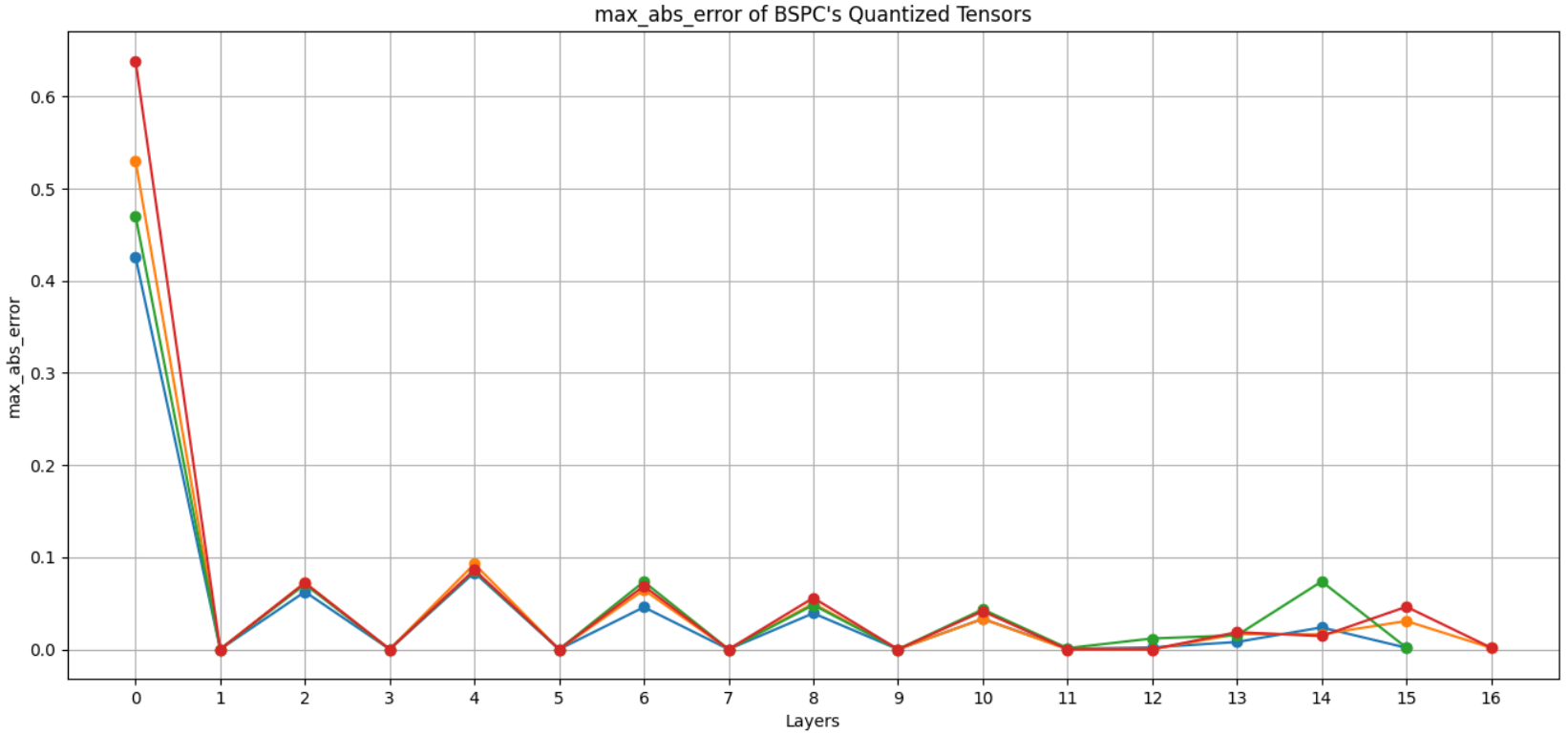
BSPC Model – Quantization Debugger (2)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



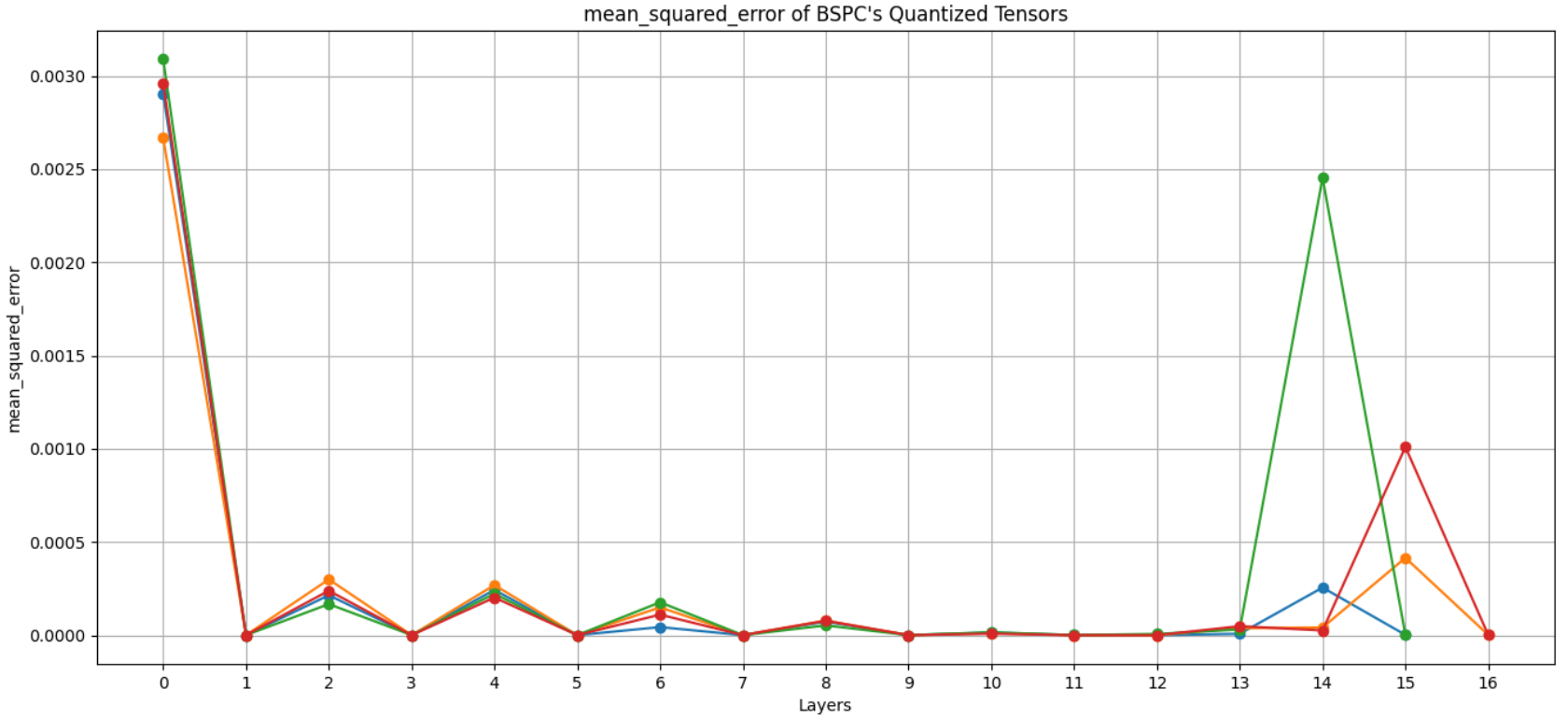
BSPC Model – Quantization Debugger (3)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



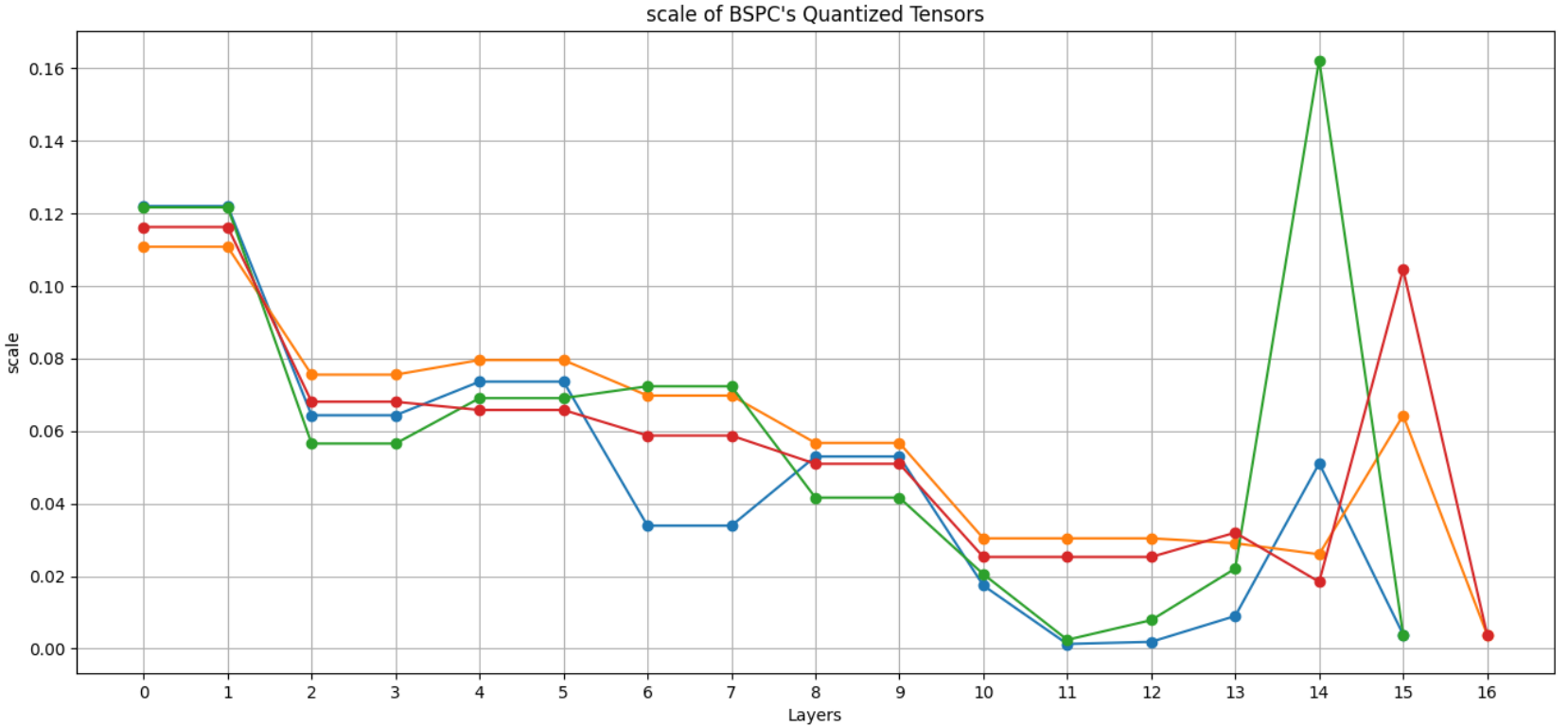
BSPC Model – Quantization Debugger (4)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



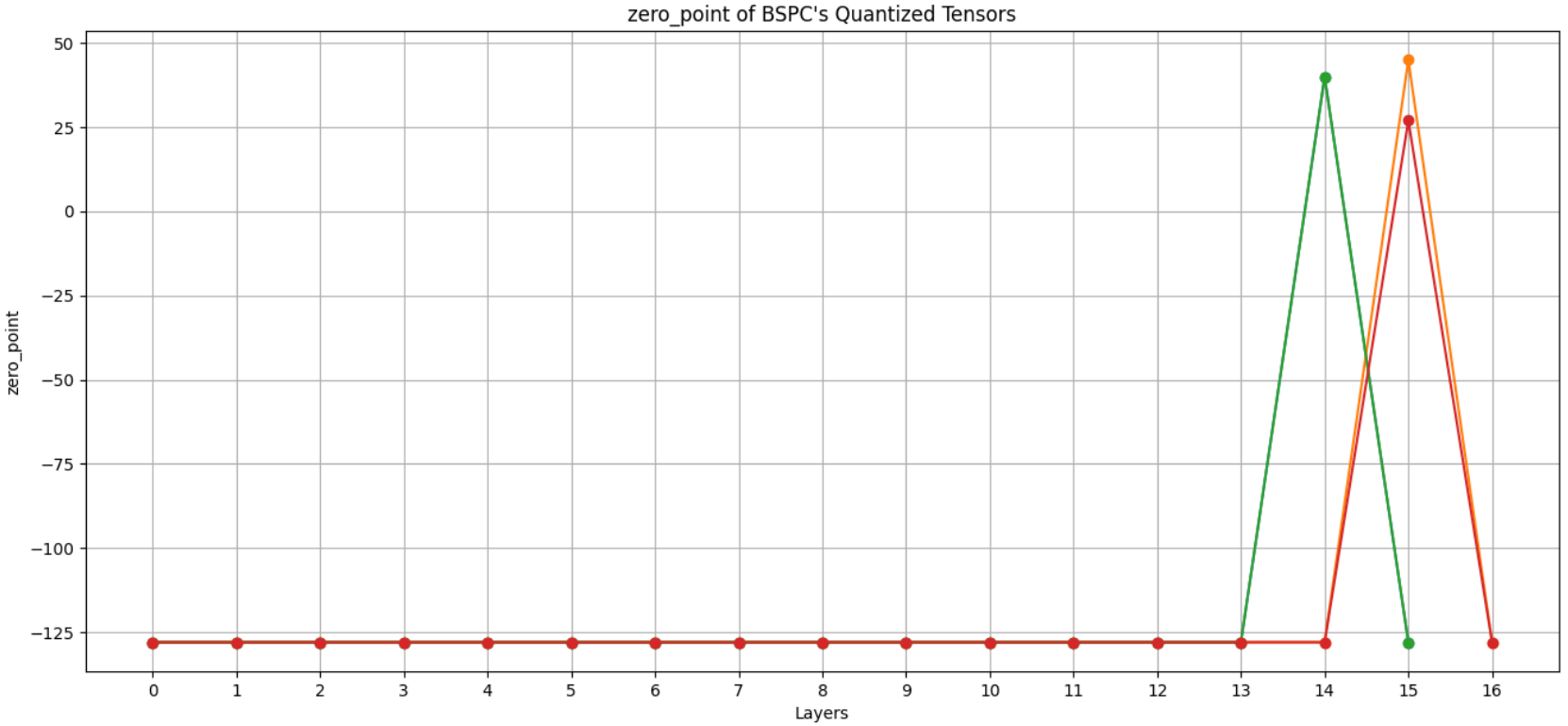
BSPC Model – Quantization Debugger (5)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



BSPC Model – Quantization Debugger (6)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



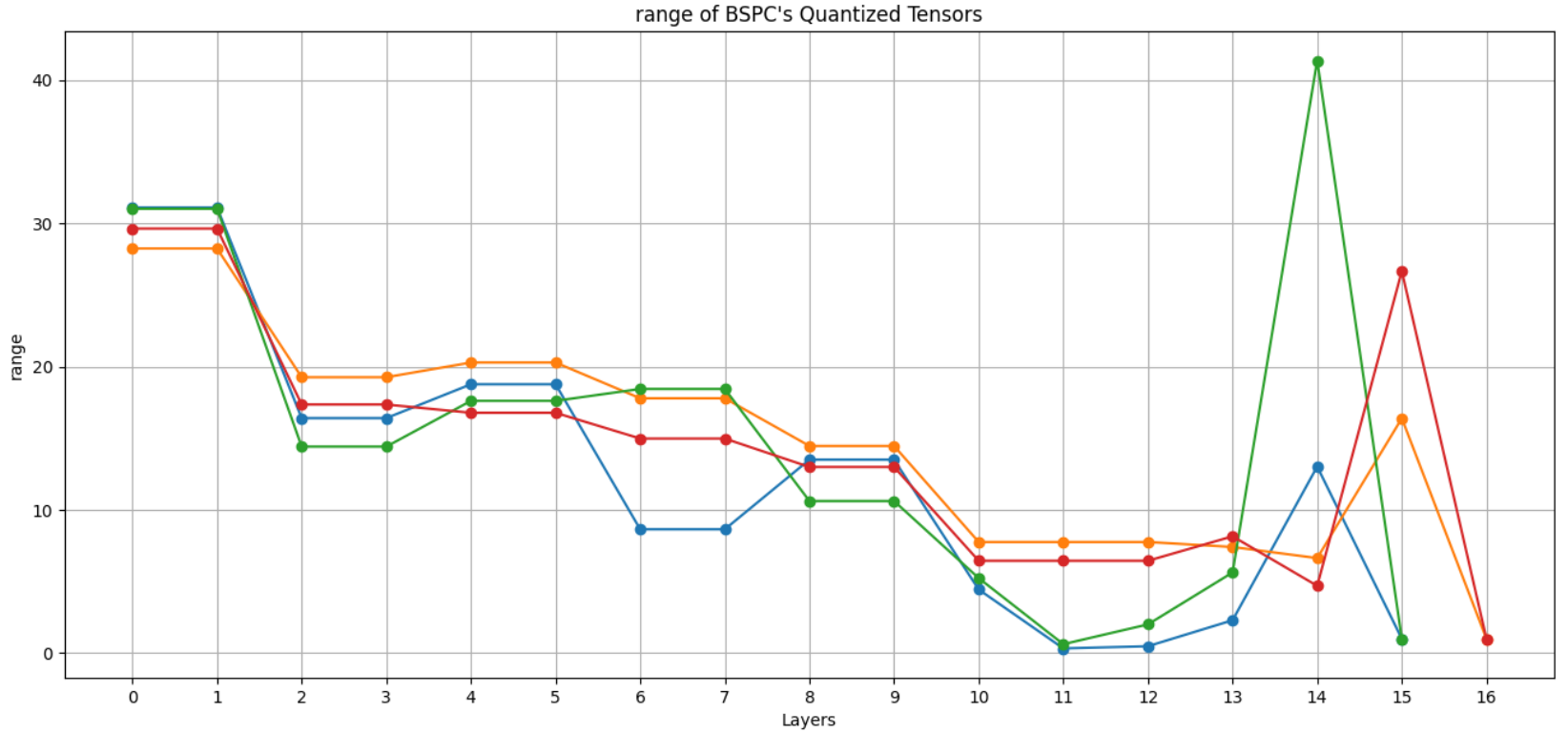
BSPC Model – Quantization Debugger (7)

Pre-LRF LEAN ver.

Pre-LRF FAT ver.

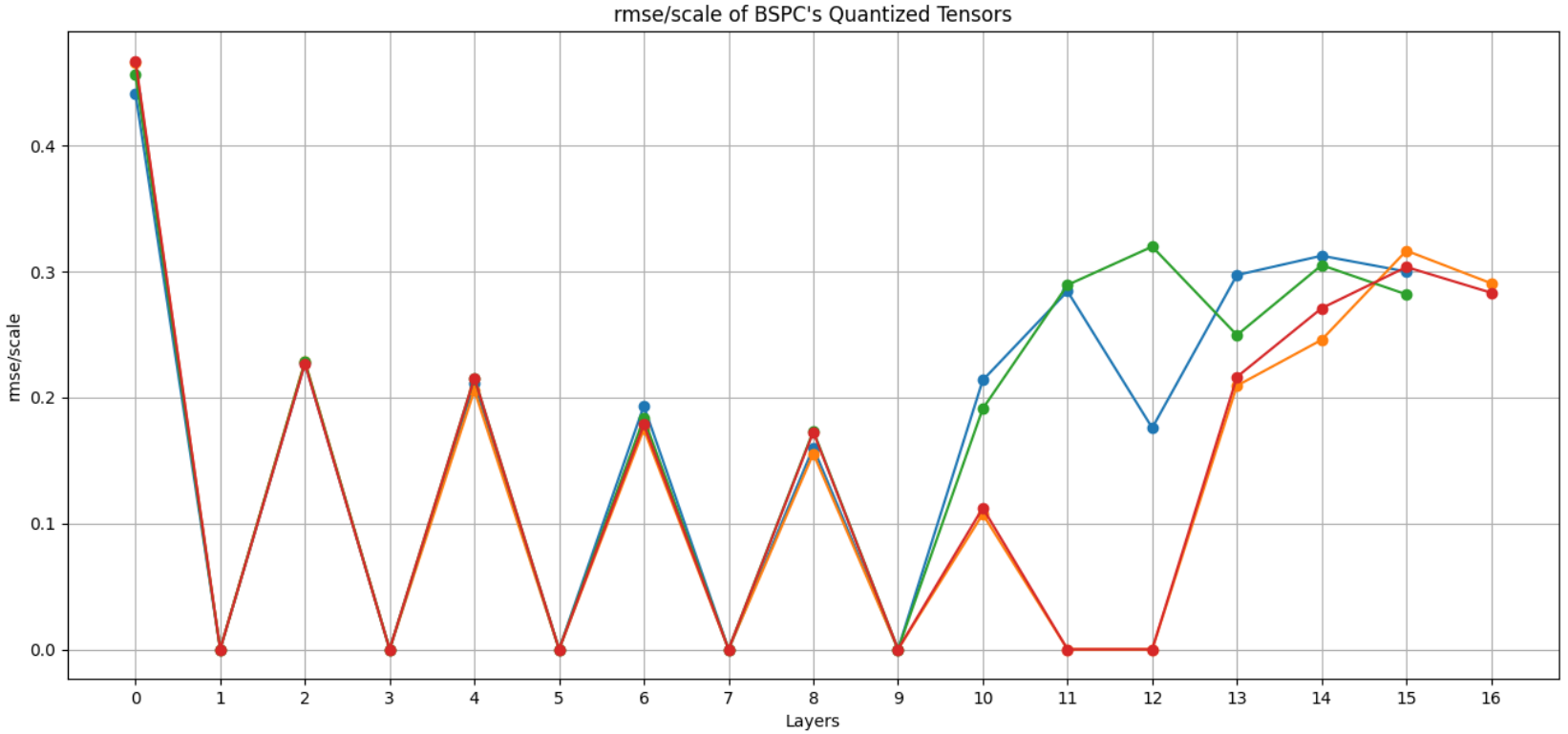
Pre-pruning LEAN ver.

Pre-pruning FAT ver.



BSPC Model – Quantization Debugger (8)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



BSPC Model – Layer Indexing

Pre-LRF LEAN ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MEAN
12	FULLY_CONNECTED
13	FULLY_CONNECTED
14	FULLY_CONNECTED
15	LOGISTIC

Pre-LRF FAT ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MAX_POOL_2D
12	RESHAPE
13	FULLY_CONNECTED
14	FULLY_CONNECTED
15	FULLY_CONNECTED
16	LOGISTIC

Pre-pruning LEAN ver.

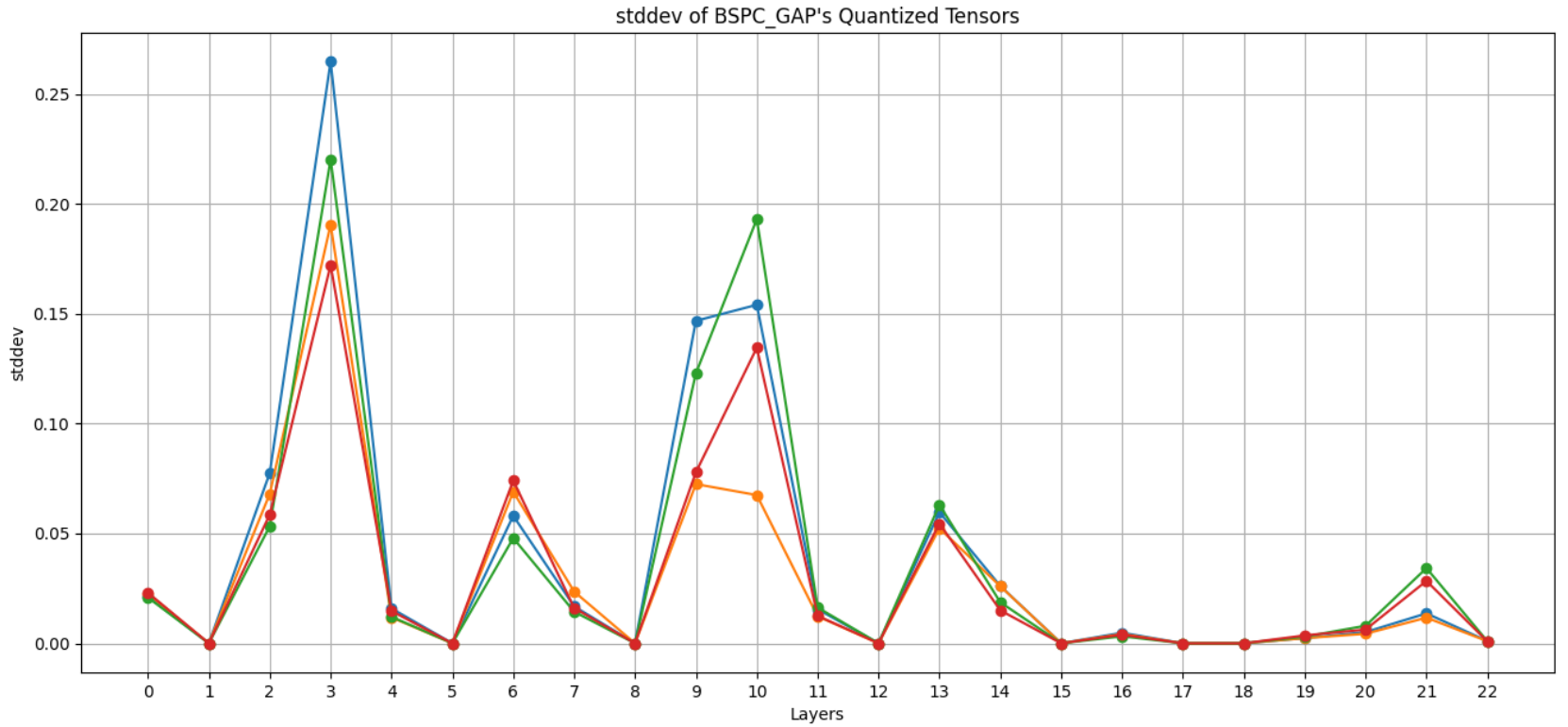
	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MEAN
12	FULLY_CONNECTED
13	FULLY_CONNECTED
14	FULLY_CONNECTED
15	LOGISTIC

Pre-pruning FAT ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MAX_POOL_2D
12	RESHAPE
13	FULLY_CONNECTED
14	FULLY_CONNECTED
15	FULLY_CONNECTED
16	LOGISTIC

BSPC-GAP Model – Quantization Debugger (1)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



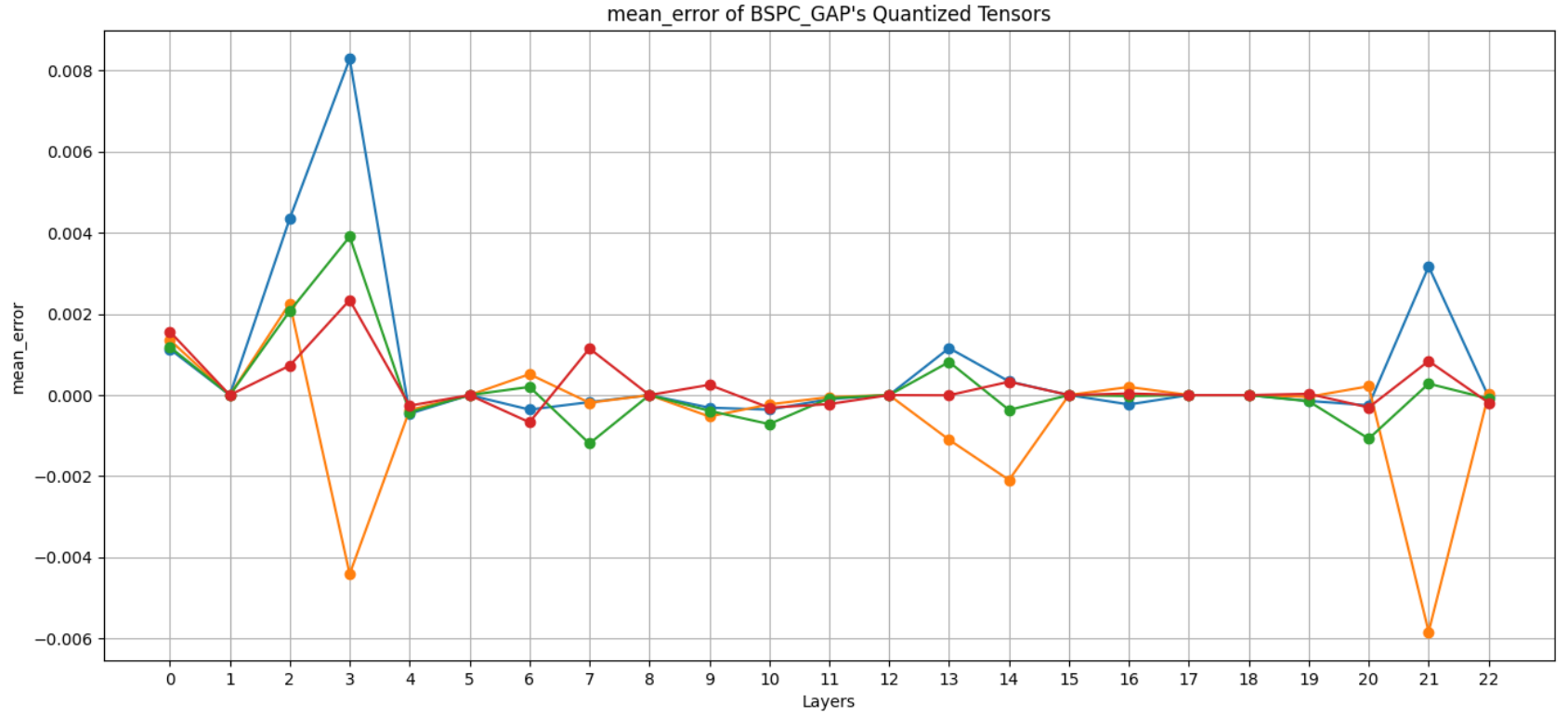
BSPC-GAP Model – Quantization Debugger (2)

Pre-LRF LEAN ver.

Pre-LRF FAT ver.

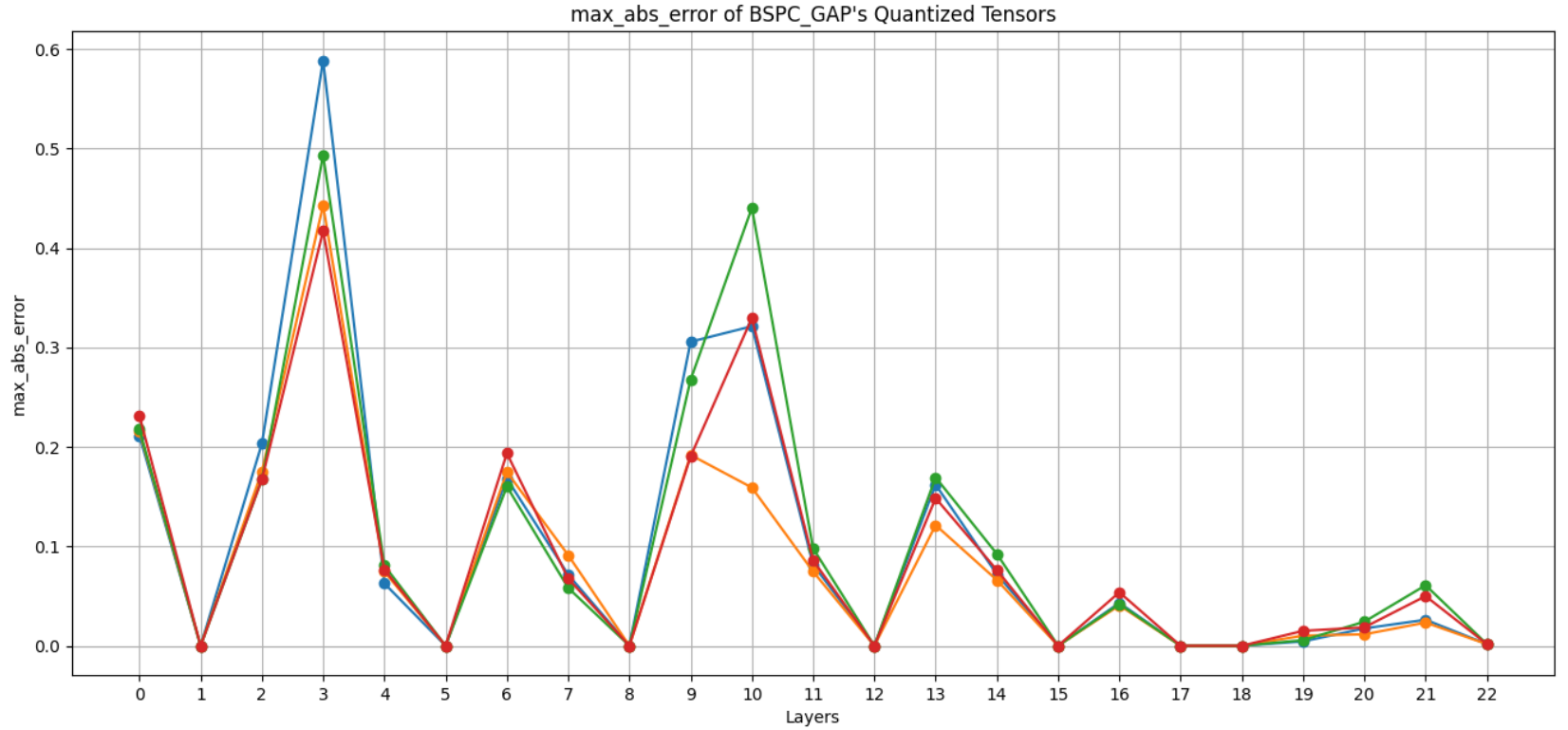
Pre-pruning LEAN ver.

Pre-pruning FAT ver.



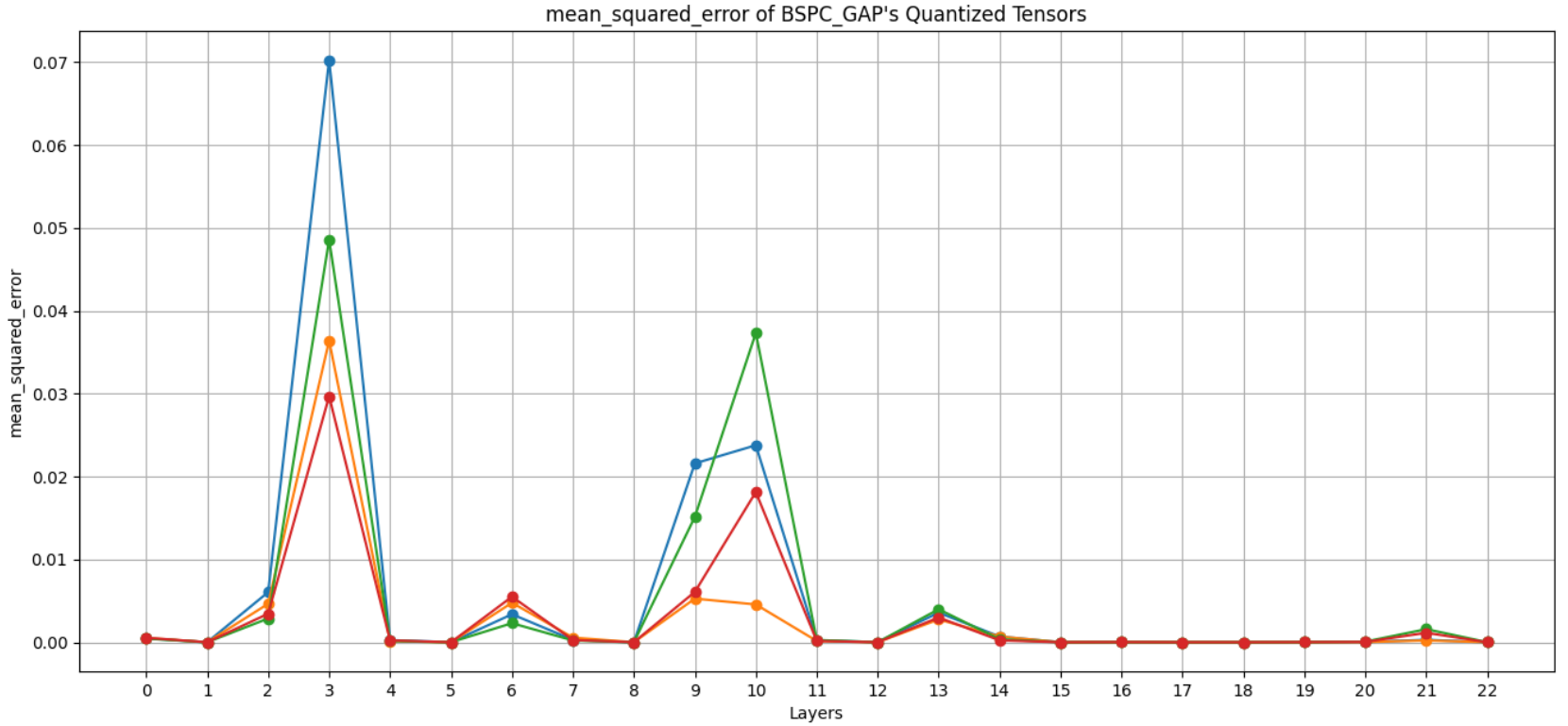
BSPC-GAP Model – Quantization Debugger (3)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



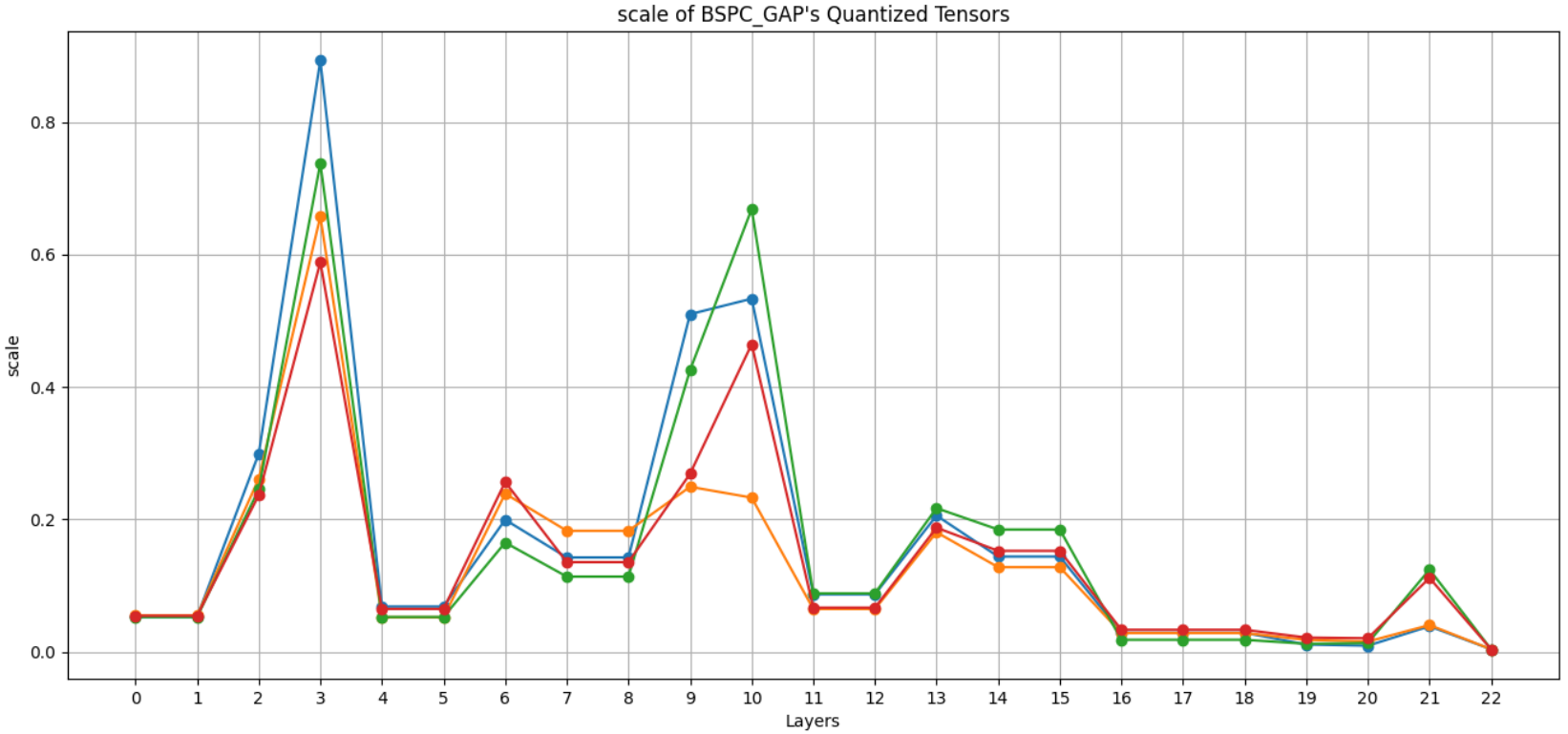
BSPC-GAP Model – Quantization Debugger (4)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



BSPC-GAP Model – Quantization Debugger (5)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



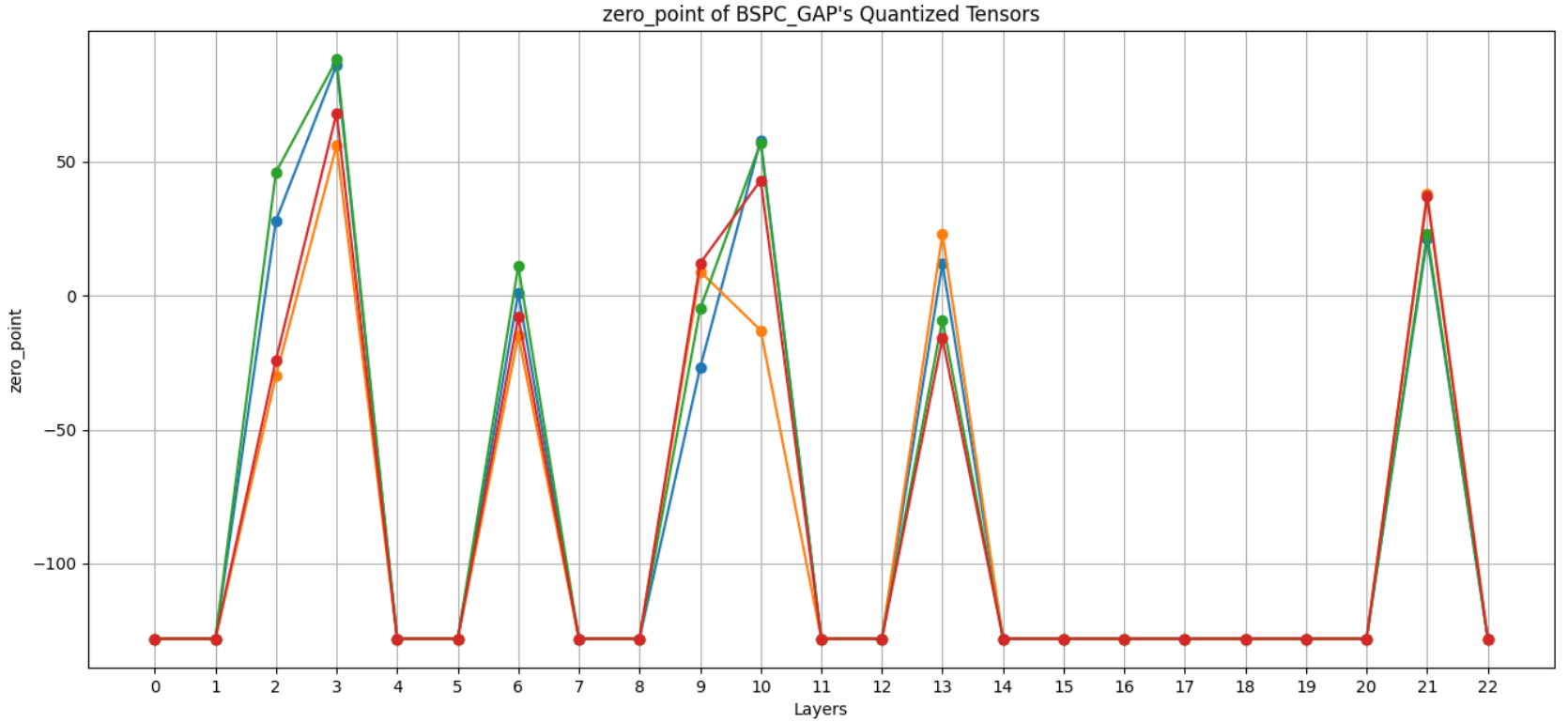
BSPC-GAP Model – Quantization Debugger (6)

Pre-LRF LEAN ver.

Pre-LRF FAT ver.

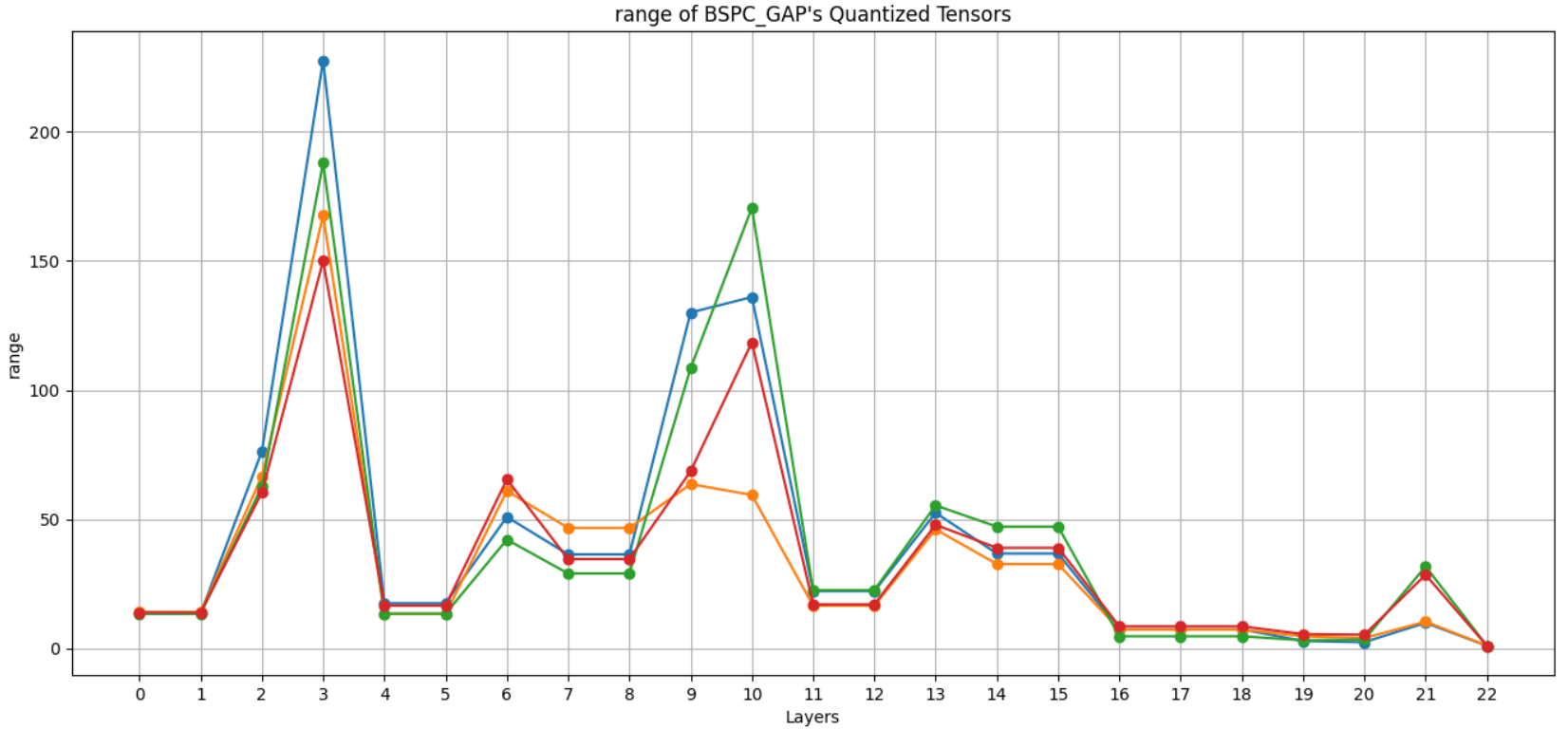
Pre-pruning LEAN ver.

Pre-pruning FAT ver.



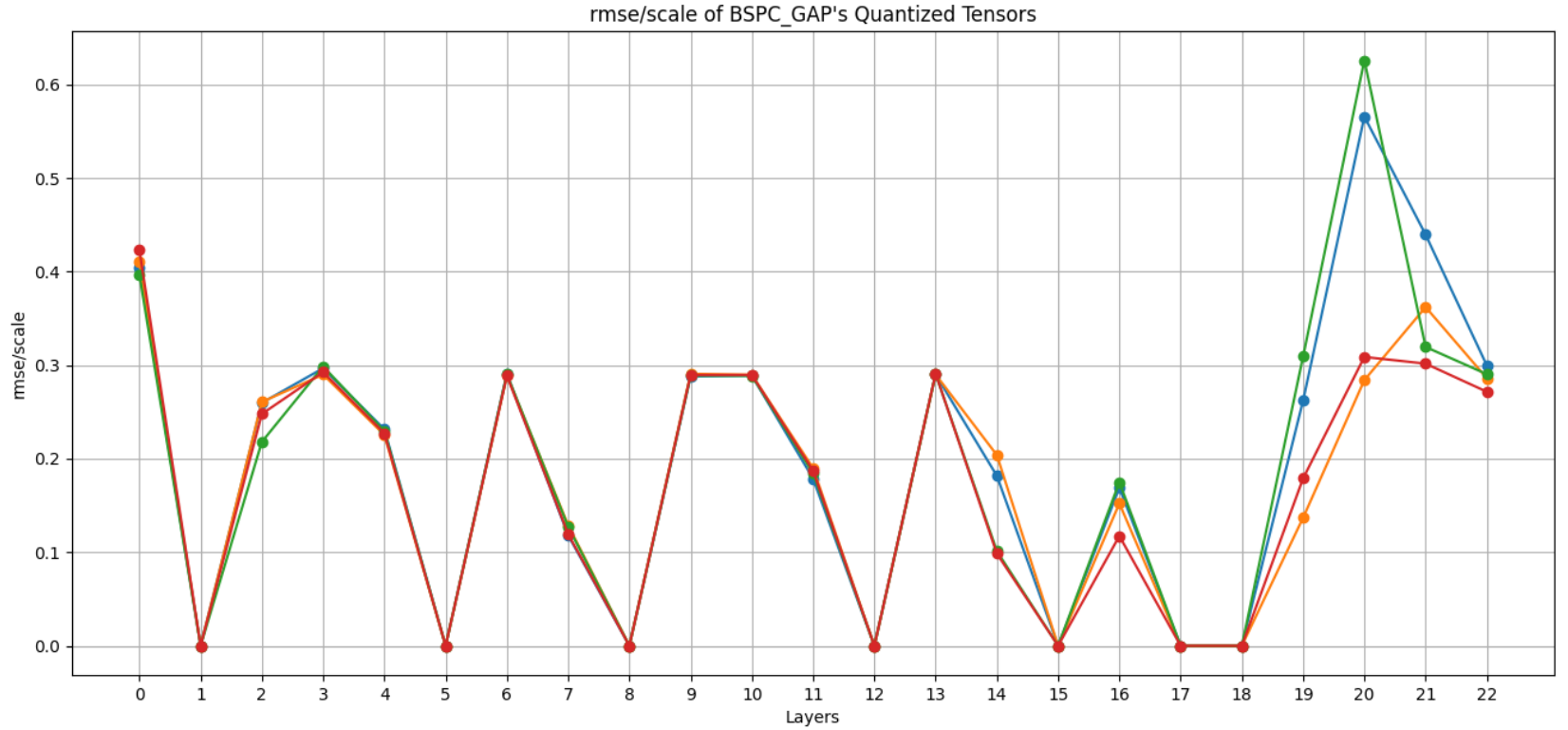
BSPC-GAP Model – Quantization Debugger (7)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



BSPC-GAP Model – Quantization Debugger (8)

- Pre-LRF LEAN ver.
- Pre-LRF FAT ver.
- Pre-pruning LEAN ver.
- Pre-pruning FAT ver.



BSPC-GAP Model – Layer Indexing

Pre-LRF LEAN ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	CONV_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	ADD
8	MAX_POOL_2D
9	CONV_2D
10	CONV_2D
11	CONV_2D
12	MAX_POOL_2D
13	CONV_2D
14	ADD
15	MAX_POOL_2D
16	CONV_2D
17	MAX_POOL_2D
18	RESHAPE
19	FULLY_CONNECTED
20	FULLY_CONNECTED
21	FULLY_CONNECTED
22	LOGISTIC

Pre-LRF FAT ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	CONV_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	ADD
8	MAX_POOL_2D
9	CONV_2D
10	CONV_2D
11	CONV_2D
12	MAX_POOL_2D
13	CONV_2D
14	ADD
15	MAX_POOL_2D
16	CONV_2D
17	MAX_POOL_2D
18	RESHAPE
19	FULLY_CONNECTED
20	FULLY_CONNECTED
21	FULLY_CONNECTED
22	LOGISTIC

Pre-pruning LEAN ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	CONV_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	ADD
8	MAX_POOL_2D
9	CONV_2D
10	CONV_2D
11	CONV_2D
12	MAX_POOL_2D
13	CONV_2D
14	ADD
15	MAX_POOL_2D
16	CONV_2D
17	MAX_POOL_2D
18	RESHAPE
19	FULLY_CONNECTED
20	FULLY_CONNECTED
21	FULLY_CONNECTED
22	LOGISTIC

Pre-pruning FAT ver.

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	CONV_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	ADD
8	MAX_POOL_2D
9	CONV_2D
10	CONV_2D
11	CONV_2D
12	MAX_POOL_2D
13	CONV_2D
14	ADD
15	MAX_POOL_2D
16	CONV_2D
17	MAX_POOL_2D
18	RESHAPE
19	FULLY_CONNECTED
20	FULLY_CONNECTED
21	FULLY_CONNECTED
22	LOGISTIC

The Final Models

Experimental results for the three models after successively applying Parameter Pruning and Low-Rank Factorization.

Lean & Fat Models

- 5 SmartWatch versions (3 lean & 2 fat)
- 2 BSPC versions (1 lean & 1 fat)
- 2 BSPC-GAP versions (1 lean & 1 fat)
- This gives a total of 9 models
- Each model is converted to TFLite format using 5 different methods, i.e. the 4 Quantization Schemes + conversion without optimizations (unquantized)
- The 4 Quantization Schemes are:
  - ✓ Dynamic Range Quantization (DRQ)
  - ✓ Full-Integer Quantization (FIQ) with unquantized (FLOAT32) I/O tensors
  - ✓ FIQ with INT8 I/O tensors
  - ✓ FIQ with UINT8 I/O tensors
- This results in 5 × 9 = 45 pipelines

```
> SMARTWATCH model
        > lean SMARTWATCH models
                > 0_3_bottom_left : Smart_Watch_0.65_lean
                > 0_5_top_left : Smart_Watch_0.65_lean
                > 4_7_bottom_left : Smart_Watch_0.65_lean
        > fat SMARTWATCH models
                > 0_0_bottom_left : Smart_Watch_0.65_fat
                > 7_7_bottom_right : Smart_Watch_0.65_fat
> BSPC model
        > lean BSPC models
                > 0_0_bottom_right : BSPC_seq_param100K_pruned_90
        > fat BSPC models
                > 0_7_top_right : BSPC_seq_param4M_pruned_80
> BSPC_GAP model
        > lean BSPC_GAP models
                > 0_0_bottom_left : BSPC_res_param250K_pruned_85
        > fat BSPC_GAP models
                > 0_6_bottom_left : BSPC_res_param4M_pruned_85
```

Performance Metrics

Sadly, the final models cannot be fine-tuned by the Quantization Aware Training strategy.

SmartWatch Model (final) – Performance Metrics

Smart_Watch_0.65_lean (0_3_bottom_left)

Quantization Scheme	Accuracy	Precision	Recall	F1	Balanced Accuracy	Cohen's Kappa
Unquantized model	0.916	0.917	0.917	0.917	0.917	0.899
DRQ	0.916~	0.917~	0.917~	0.917~	0.917~	0.899~
FIQ – Float32	0.861~	0.880~	0.856~	0.862~	0.856~	0.833~
FIQ – Int8	0.770↓	0.817↓	0.754↓	0.742↓	0.754↓	0.722↓
FIQ – UInt8	0.769↓	0.815↓	0.753↓	0.743↓	0.753↓	0.720↓

SmartWatch Model (final) – Performance Metrics

Smart_Watch_0.65_lean (0_5_top_left)

Quantization Scheme	Accuracy	Precision	Recall	F1	Balanced Accuracy	Cohen's Kappa
Unquantized model	0.925	0.927	0.926	0.926	0.926	0.910
DRQ	0.925~	0.927~	0.926~	0.926~	0.926~	0.910~
FIQ – Float32	0.891~	0.907~	0.887~	0.892~	0.887~	0.868~
FIQ – Int8	0.780↓	0.837↓	0.763↓	0.745↓	0.763↓	0.733↓
FIQ – UInt8	0.784↓	0.833↓	0.768↓	0.758↓	0.768↓	0.738↓

SmartWatch Model (final) – Performance Metrics

Smart_Watch_0.65_lean (4_7_bottom_left)

Quantization Scheme	Accuracy	Precision	Recall	F1	Balanced Accuracy	Cohen's Kappa
Unquantized model	0.928	0.931	0.928	0.929	0.928	0.913
DRQ	0.928~	0.931~	0.928~	0.929~	0.928~	0.913~
FIQ – Float32	0.893~	0.907~	0.891~	0.895~	0.891~	0.871~
FIQ – Int8	0.809↓	0.851↓	0.796↓	0.793↓	0.796↓	0.769↓
FIQ – UInt8	0.800↓	0.846↓	0.787↓	0.781↓	0.787↓	0.759↓

SmartWatch Model (final) – Performance Metrics

Smart_Watch_0.65_fat (0_0_bottom_left)

Quantization Scheme	Accuracy	Precision	Recall	F1	Balanced Accuracy	Cohen's Kappa
Unquantized model	0.942	0.943	0.941	0.941	0.941	0.930
DRQ	0.942~	0.943~	0.941~	0.941~	0.941~	0.930~
FIQ – Float32	0.924~	0.927~	0.921~	0.922~	0.921~	0.908~
FIQ – Int8	0.794↓	0.835↓	0.780↓	0.780↓	0.780↓	0.751↓
FIQ – UInt8	0.807↓	0.842↓	0.793↓	0.794↓	0.793↓	0.768↓

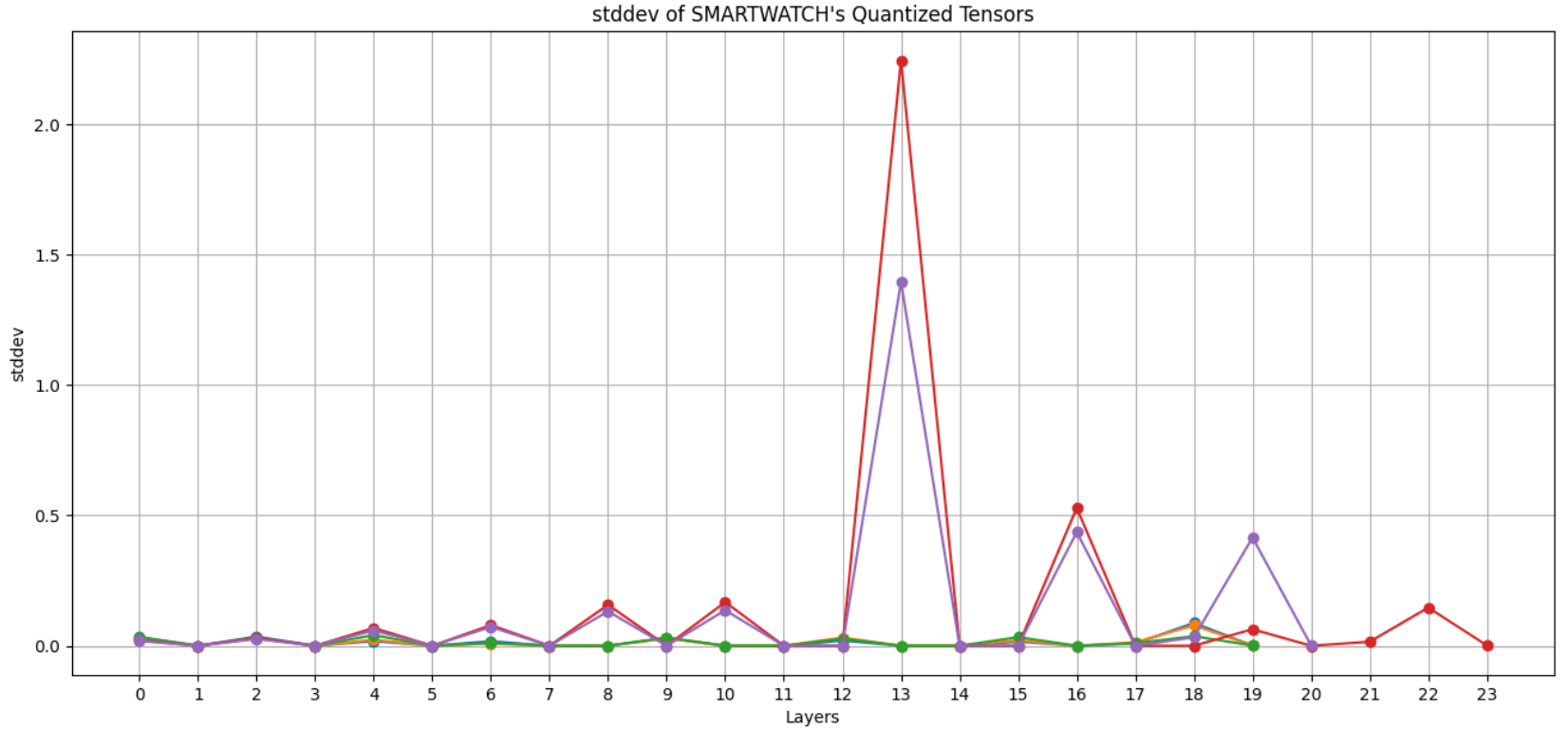
SmartWatch Model (final) – Performance Metrics

Smart_Watch_0.65_fat (7_7_bottom_right)

Quantization Scheme	Accuracy	Precision	Recall	F1	Balanced Accuracy	Cohen's Kappa
Unquantized model	0.952	0.953	0.952	0.952	0.952	0.942
DRQ	0.951~	0.953~	0.951~	0.952~	0.951~	0.942~
FIQ – Float32	0.938~	0.941~	0.936~	0.937~	0.936~	0.926~
FIQ – Int8	0.816↓	0.856↓	0.803↓	0.805↓	0.803↓	0.777↓
FIQ – UInt8	0.836↓	0.863↓	0.823↓	0.824↓	0.823↓	0.803↓

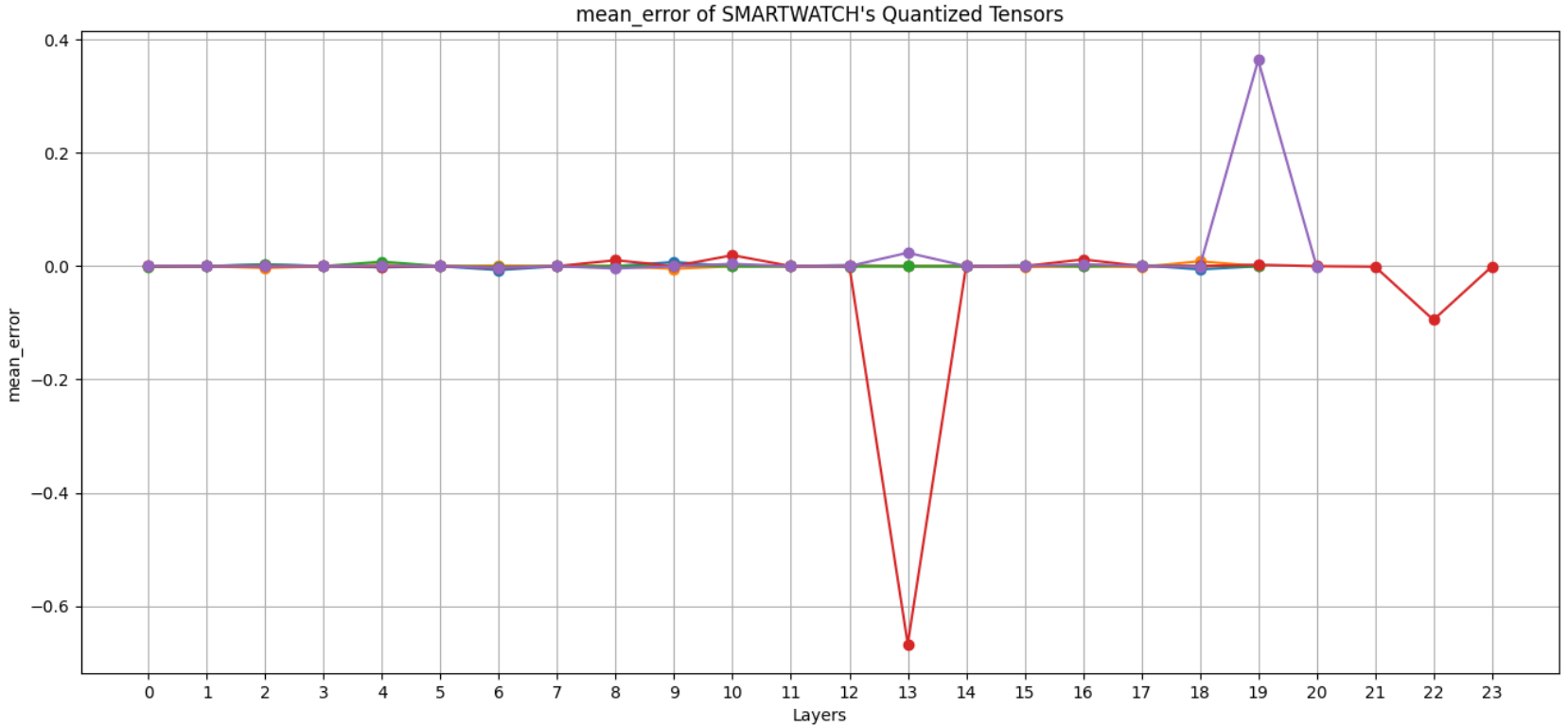
SmartWatch Model (final) – Quantization Debugger (1)

0_3_bottom_left
0_5_top_left
4_7_bottom_left
0_0_bottom_left
7_7_bottom_right



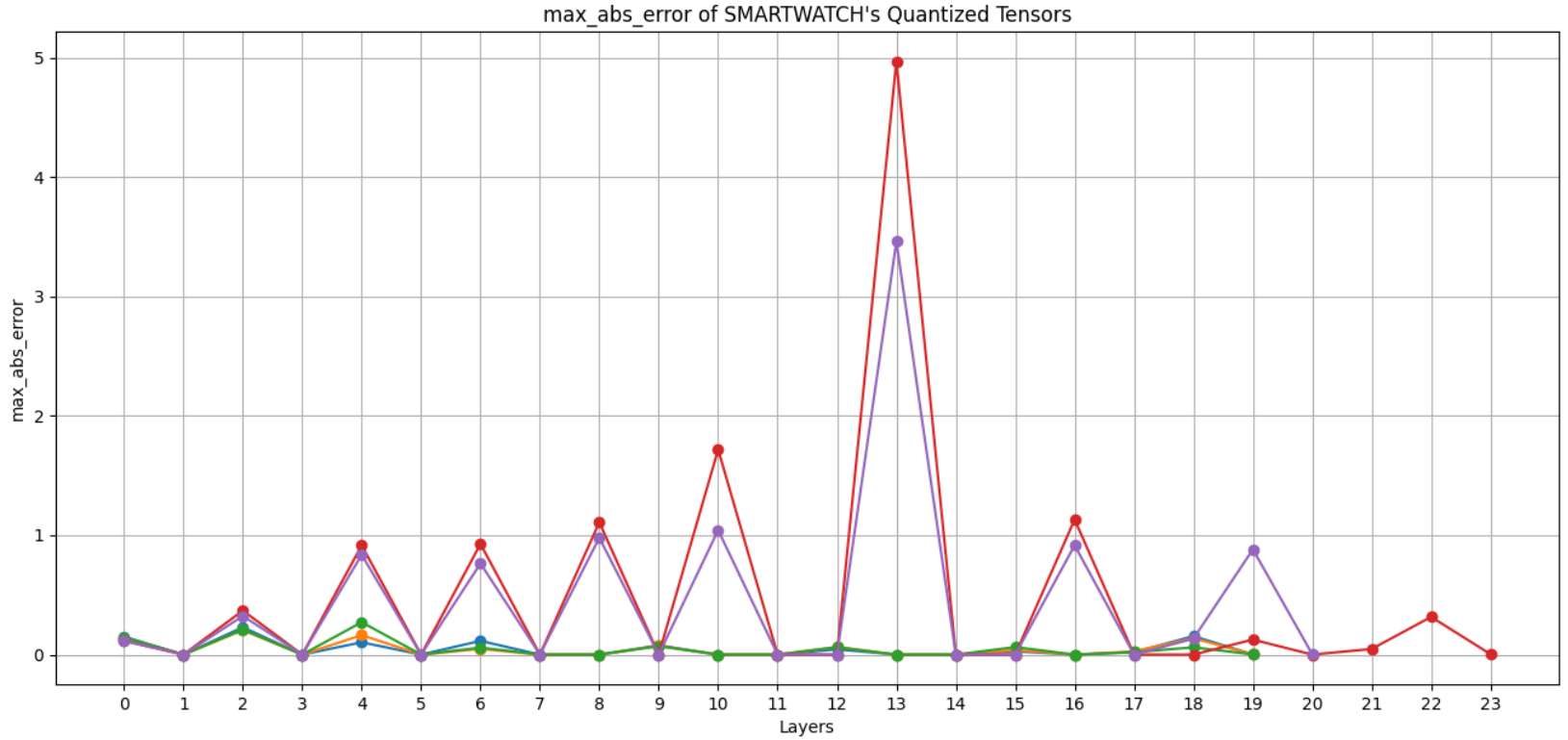
SmartWatch Model (final) – Quantization Debugger (2)

0_3_bottom_left
0_5_top_left
4_7_bottom_left
0_0_bottom_left
7_7_bottom_right



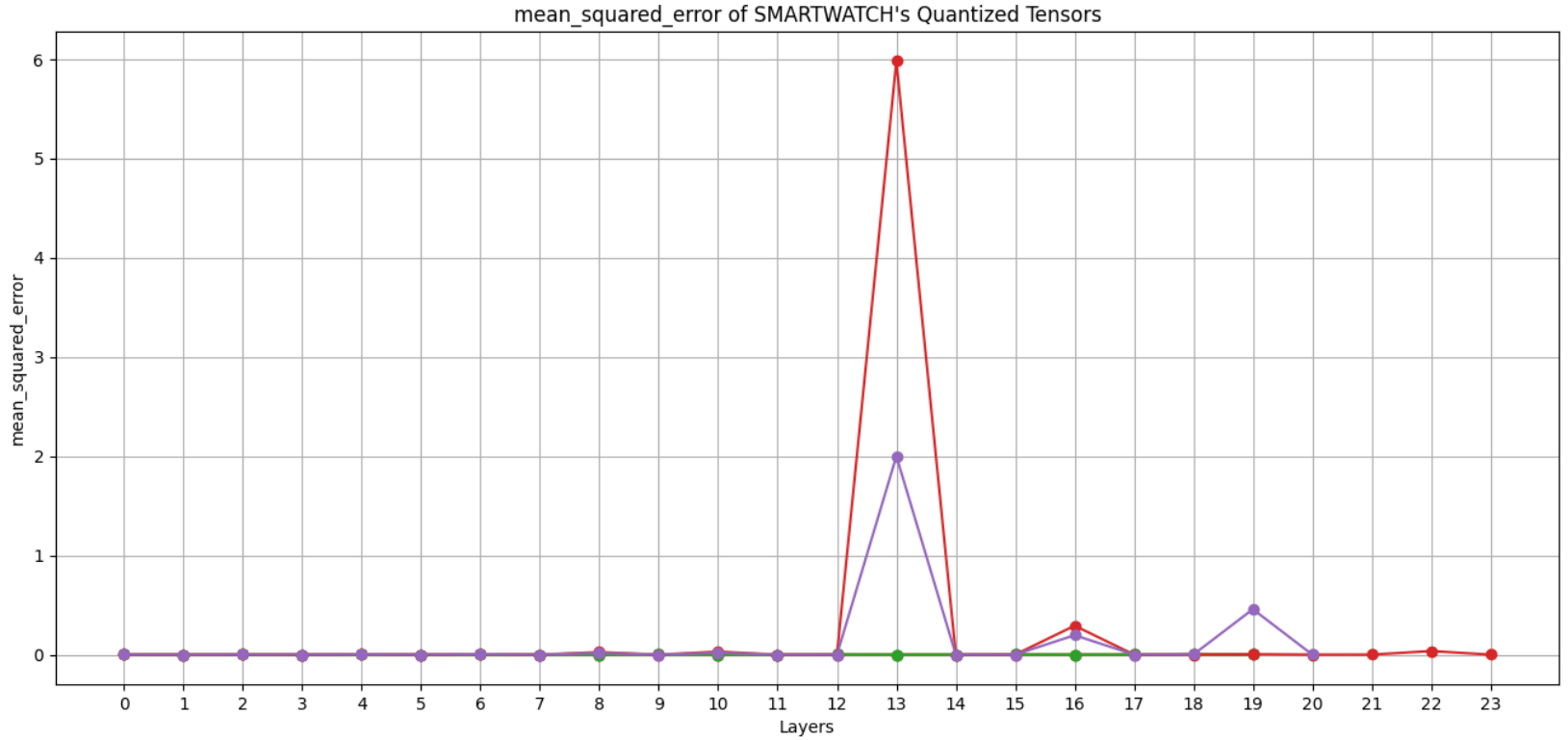
SmartWatch Model (final) – Quantization Debugger (3)

0_3_bottom_left
0_5_top_left
4_7_bottom_left
0_0_bottom_left
7_7_bottom_right



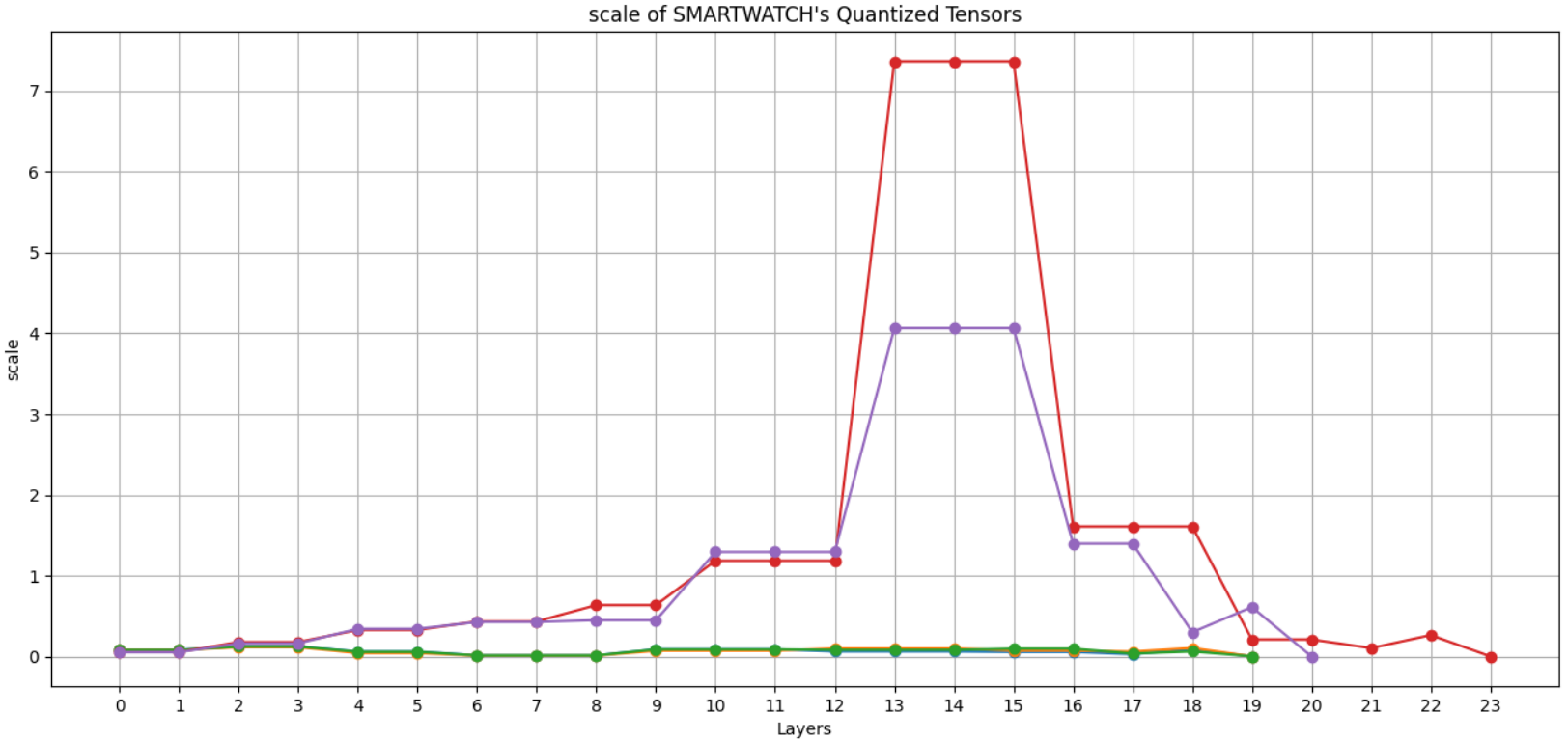
SmartWatch Model (final) – Quantization Debugger (4)

0_3_bottom_left
0_5_top_left
4_7_bottom_left
0_0_bottom_left
7_7_bottom_right



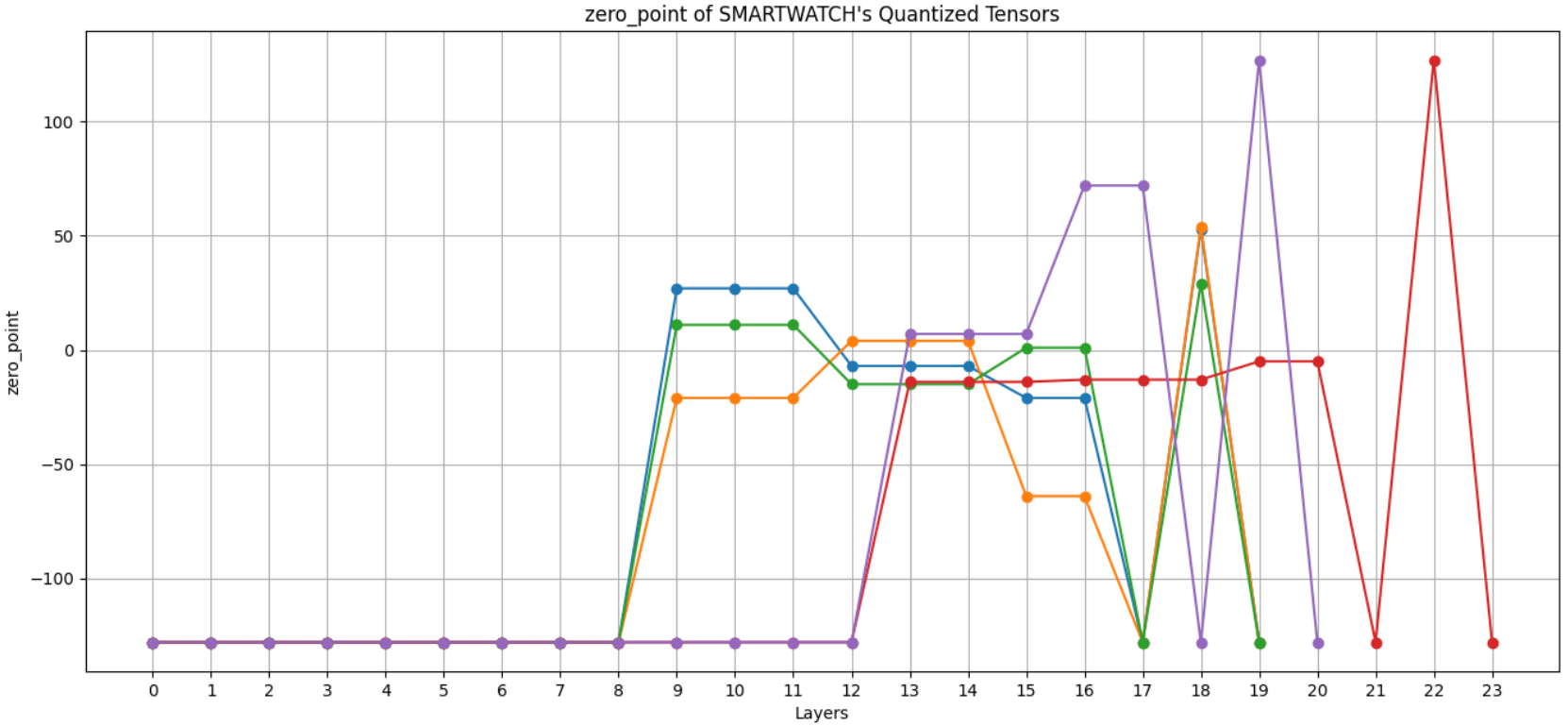
SmartWatch Model (final) – Quantization Debugger (5)

0_3_bottom_left
0_5_top_left
4_7_bottom_left
0_0_bottom_left
7_7_bottom_right



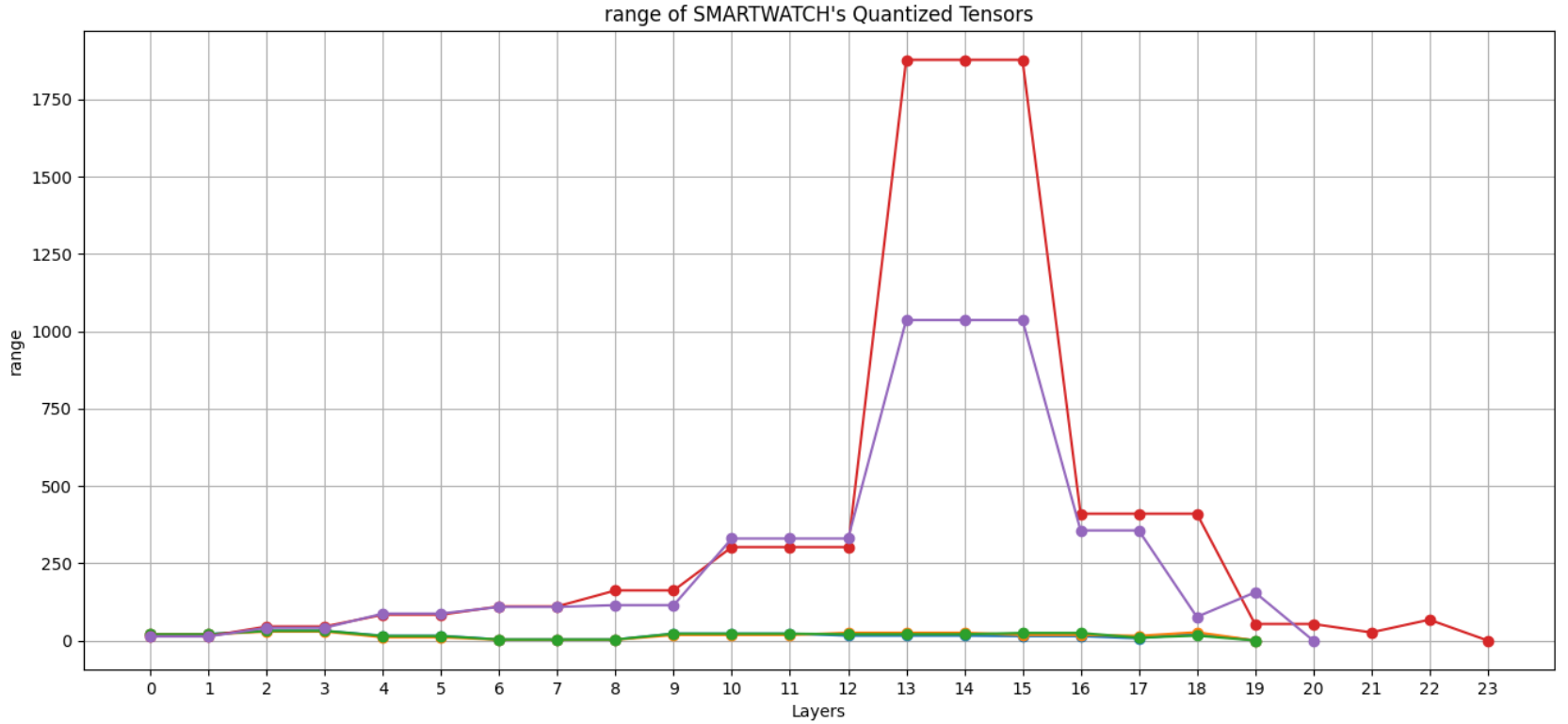
SmartWatch Model (final) – Quantization Debugger (6)

0_3_bottom_left
0_5_top_left
4_7_bottom_left
0_0_bottom_left
7_7_bottom_right



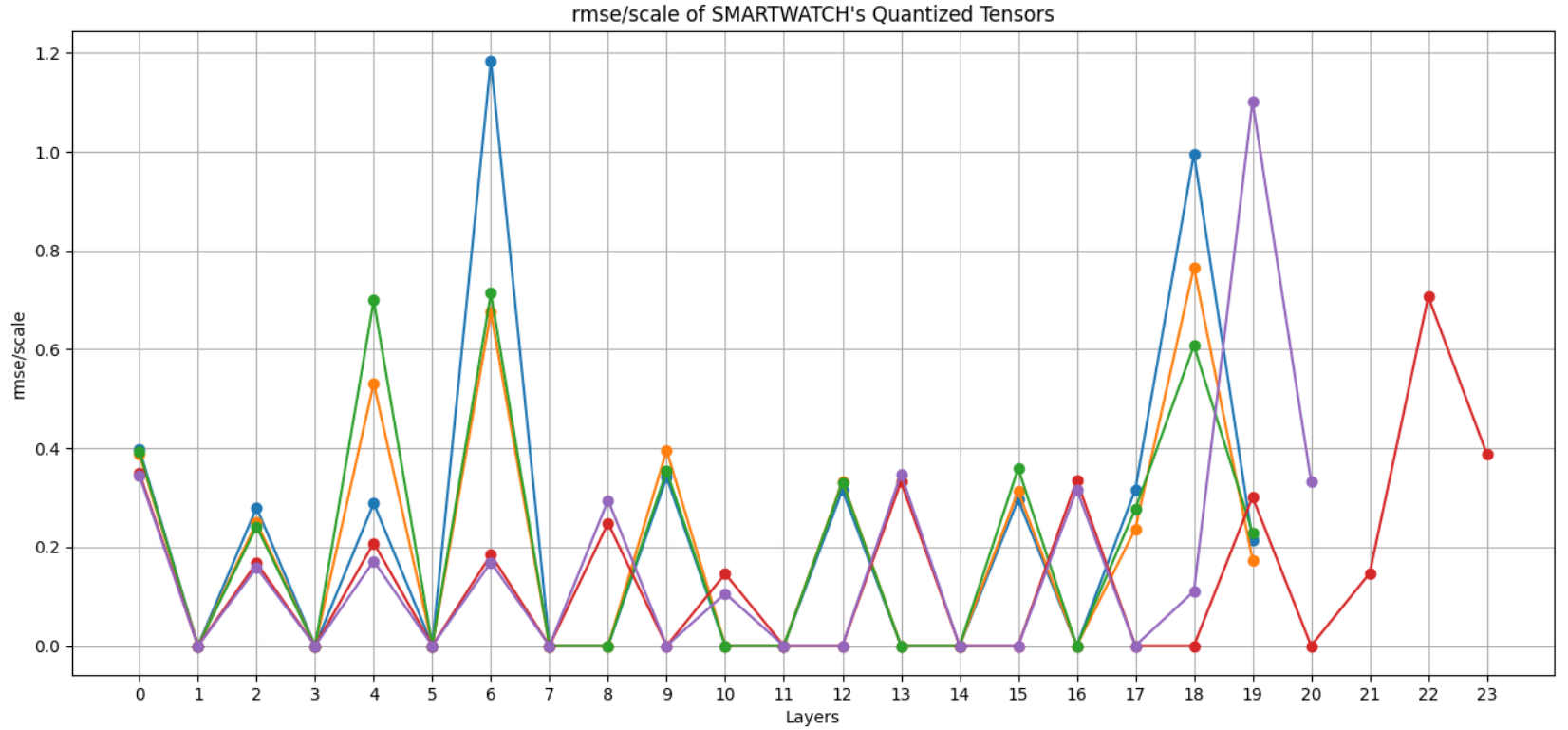
SmartWatch Model (final) – Quantization Debugger (7)

0_3_bottom_left
0_5_top_left
4_7_bottom_left
0_0_bottom_left
7_7_bottom_right



SmartWatch Model (final) – Quantization Debugger (8)

0_3_bottom_left
0_5_top_left
4_7_bottom_left
0_0_bottom_left
7_7_bottom_right



SmartWatch Model (final) – Layer Indexing

0_3_bottom_left

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	RESHAPE
9	BATCH_MATMUL
10	TRANSPOSE
11	RESHAPE
12	BATCH_MATMUL
13	TRANSPOSE
14	RESHAPE
15	BATCH_MATMUL
16	RESHAPE
17	ADD
18	FULLY_CONNECTED
19	SOFTMAX

0_5_top_left

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	RESHAPE
9	BATCH_MATMUL
10	TRANSPOSE
11	RESHAPE
12	BATCH_MATMUL
13	TRANSPOSE
14	RESHAPE
15	BATCH_MATMUL
16	RESHAPE
17	ADD
18	FULLY_CONNECTED
19	SOFTMAX

4_7_bottom_left

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	RESHAPE
9	BATCH_MATMUL
10	TRANSPOSE
11	RESHAPE
12	BATCH_MATMUL
13	TRANSPOSE
14	RESHAPE
15	BATCH_MATMUL
16	RESHAPE
17	ADD
18	FULLY_CONNECTED
19	SOFTMAX

0_0_bottom_left

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MAX_POOL_2D
12	RESHAPE
13	BATCH_MATMUL
14	TRANSPOSE
15	RESHAPE
16	BATCH_MATMUL
17	TRANSPOSE
18	RESHAPE
19	BATCH_MATMUL
20	RESHAPE
21	ADD
22	FULLY_CONNECTED
23	SOFTMAX

7_7_bottom_right

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MAX_POOL_2D
12	RESHAPE
13	BATCH_MATMUL
14	TRANSPOSE
15	RESHAPE
16	BATCH_MATMUL
17	RESHAPE
18	ADD
19	FULLY_CONNECTED
20	SOFTMAX

BSPC Model (final) – Performance Metrics

BSPC_seq_param100K_pruned_90 (0_0_bottom_right)

Quantization Scheme	Binary Accuracy	Precision	Recall	F1	Hamming Loss	Jaccard Score
Unquantized model	0.869	0.807	0.572	0.630	0.131	0.485
DRQ	0.869~	0.808~	0.571~	0.630~	0.139~	0.520~
FIQ – Float32	0.846~	0.757↓	0.548~	0.614~	0.154↑	0.453~
FIQ – Int8	0.827~	0.725↓	0.523~	0.580~	0.173↑	0.416~
FIQ – UInt8	0.852~	0.775~	0.544~	0.613~	0.148~	0.457~

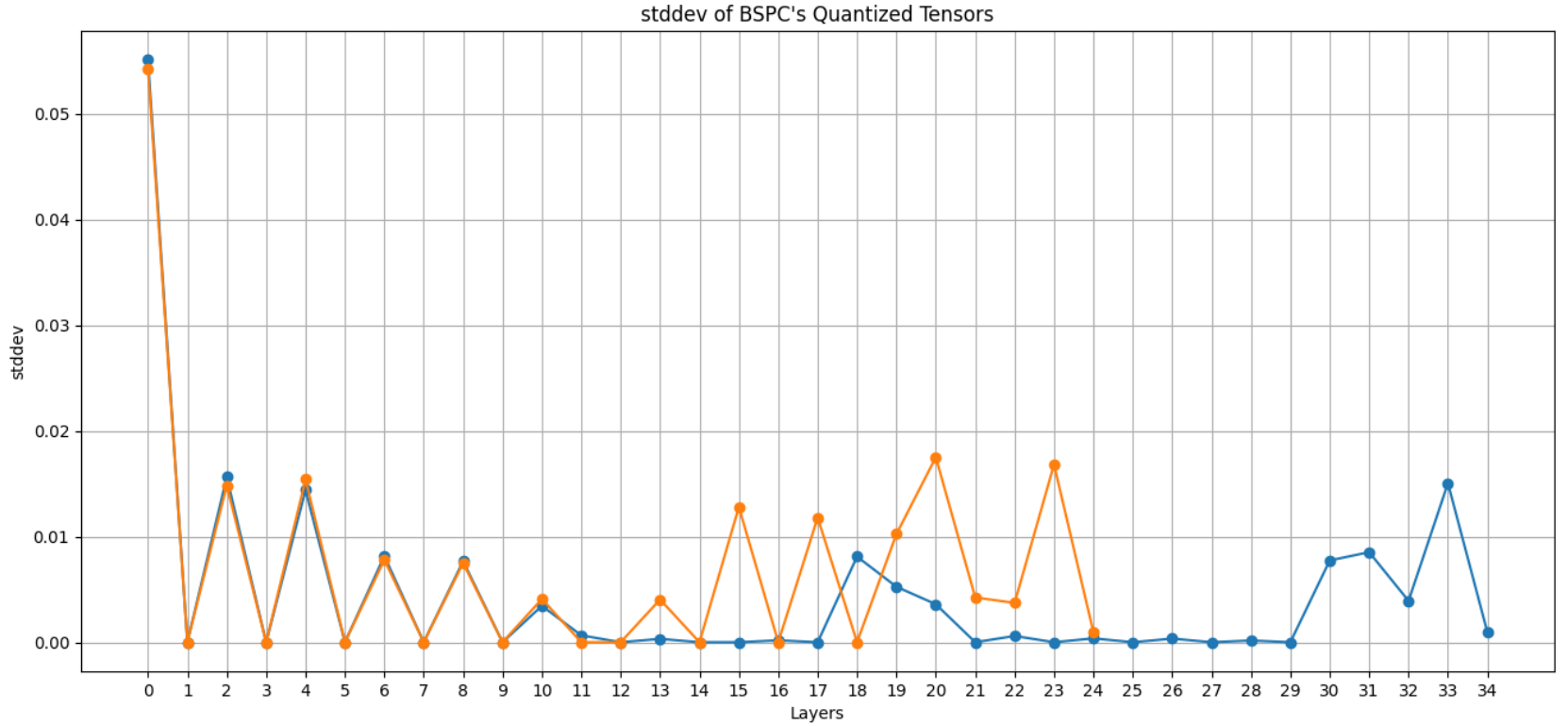
BSPC Model (final) – Performance Metrics

BSPC_seq_param4M_pruned_80 (0_7_top_right)

Quantization Scheme	Binary Accuracy	Precision	Recall	F1	Hamming Loss	Jaccard Score
Unquantized model	0.887	0.807	0.661	0.715	0.113	0.567
DRQ	0.887~	0.806~	0.658~	0.713~	0.113~	0.566~
FIQ – Float32	0.886~	0.805~	0.665~	0.716~	0.114~	0.568~
FIQ – Int8	0.882~	0.807~	0.639~	0.697~	0.118~	0.547~
FIQ – UInt8	0.886~	0.793~	0.678~	0.721~	0.114~	0.574~

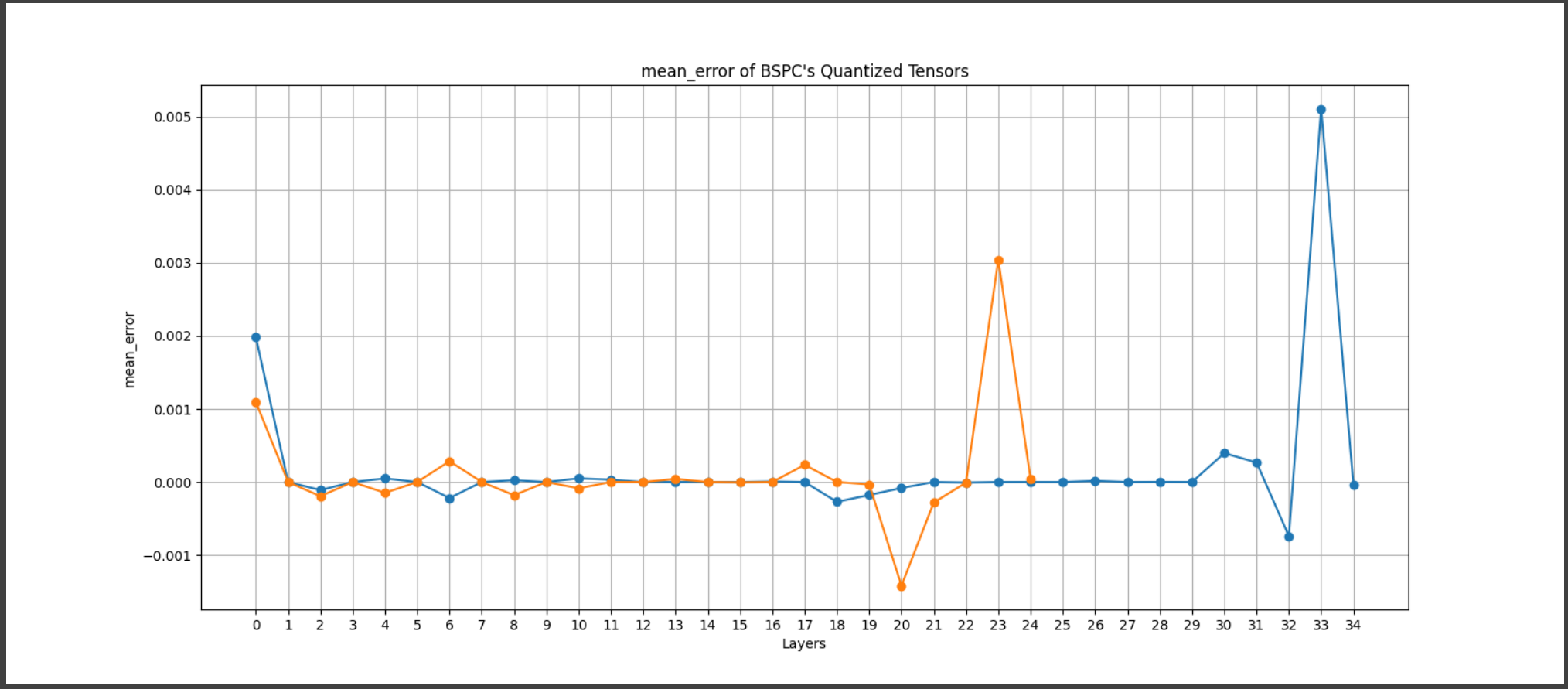
BSPC Model (final) – Quantization Debugger (1)

- 0_0_bottom_right (BSPC_seq_param100K_pruned_90)
- 0_7_top_right (BSPC_seq_param4M_pruned_80)



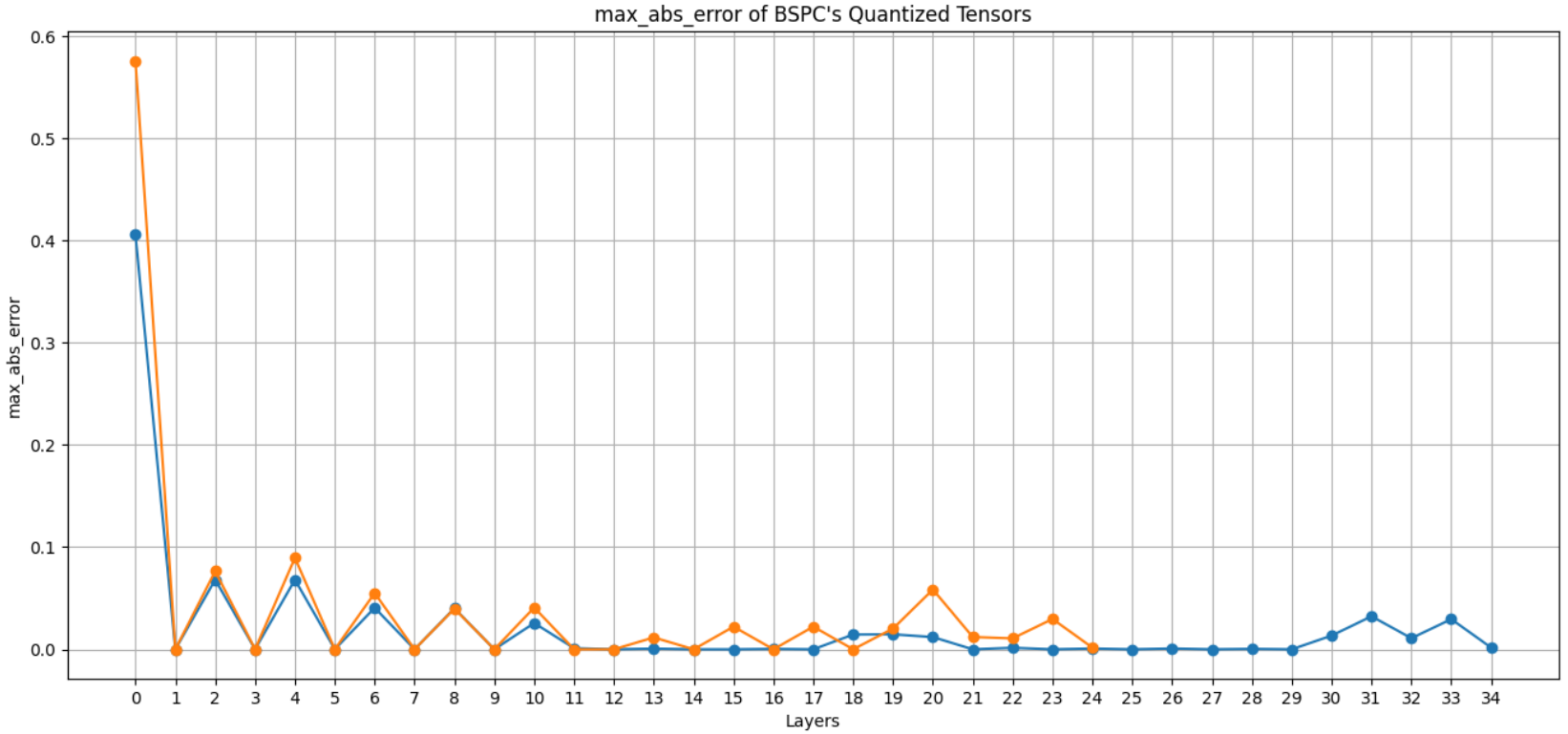
BSPC Model (final) – Quantization Debugger (2)

- 0_0_bottom_right (BSPC_seq_param100K_pruned_90)
- 0_7_top_right (BSPC_seq_param4M_pruned_80)



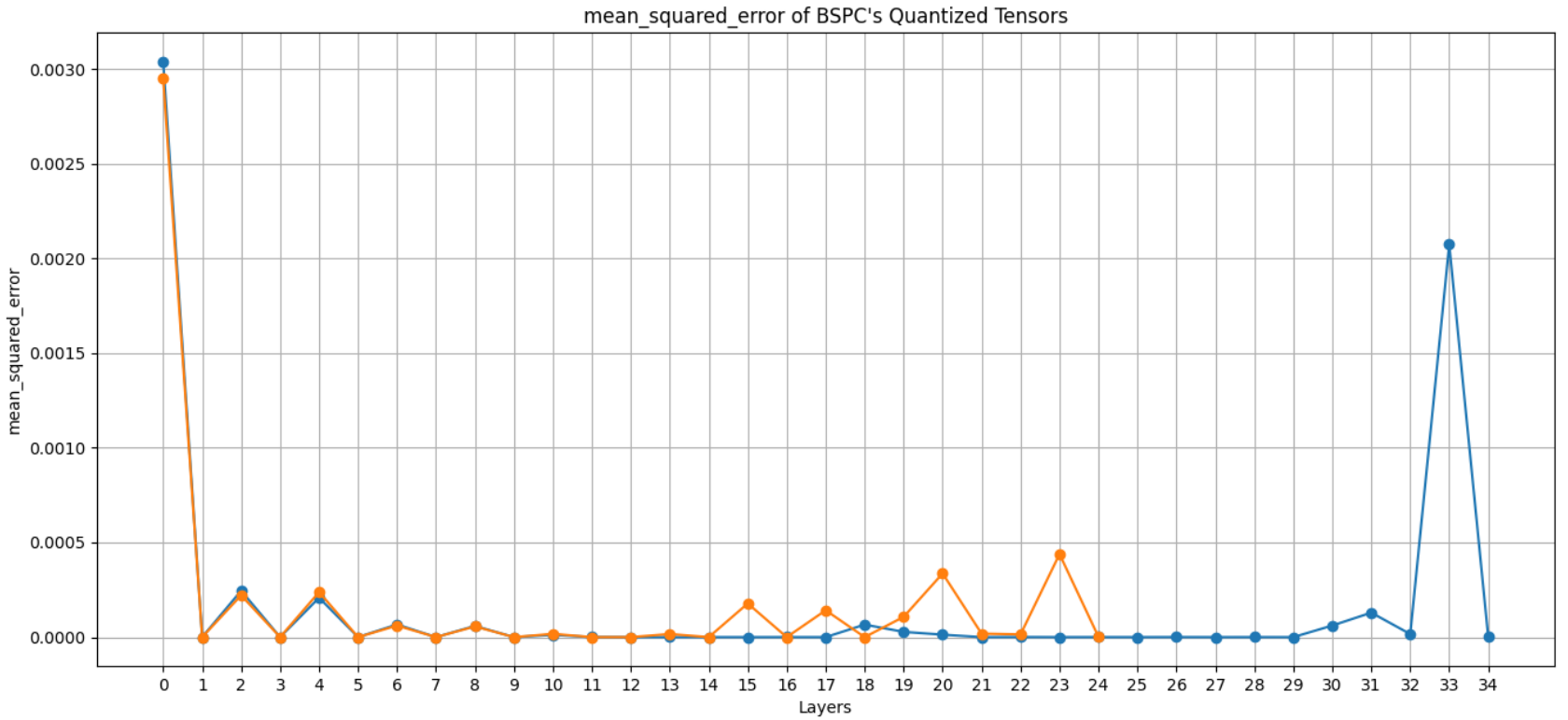
BSPC Model (final) – Quantization Debugger (3)

- 0_0_bottom_right (BSPC_seq_param100K_pruned_90)
- 0_7_top_right (BSPC_seq_param4M_pruned_80)



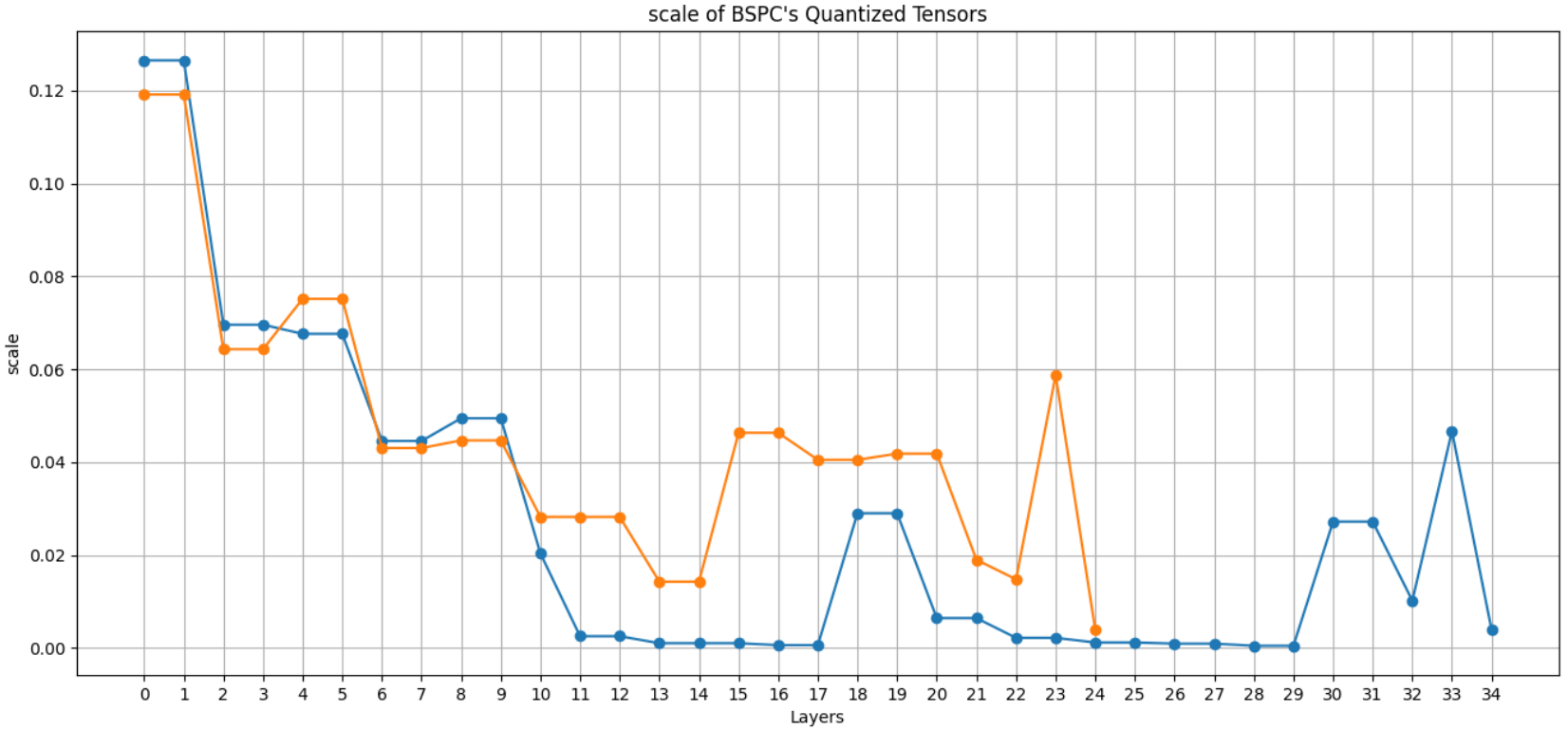
BSPC Model (final) – Quantization Debugger (4)

- 0_0_bottom_right (BSPC_seq_param100K_pruned_90)
- 0_7_top_right (BSPC_seq_param4M_pruned_80)



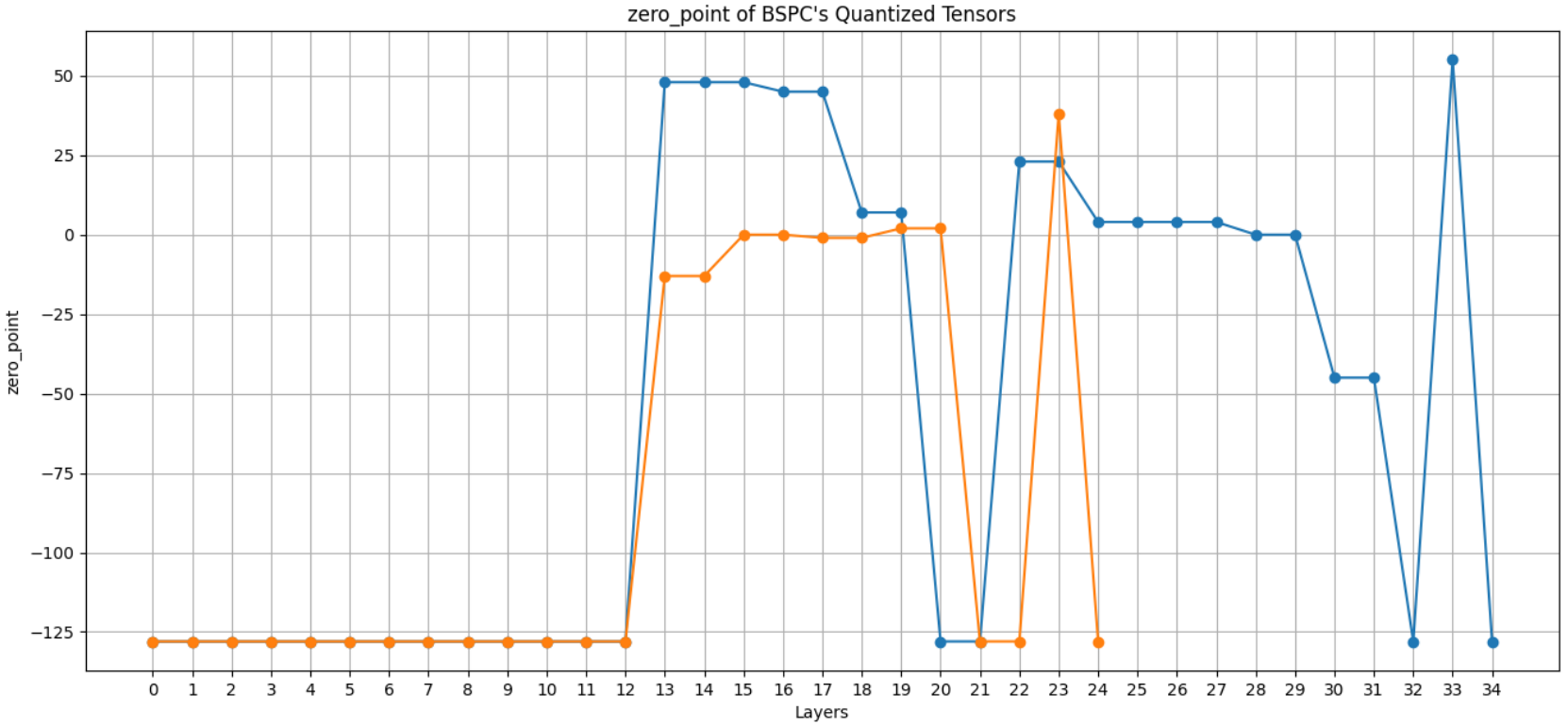
BSPC Model (final) – Quantization Debugger (5)

- 0_0_bottom_right (BSPC_seq_param100K_pruned_90)
- 0_7_top_right (BSPC_seq_param4M_pruned_80)



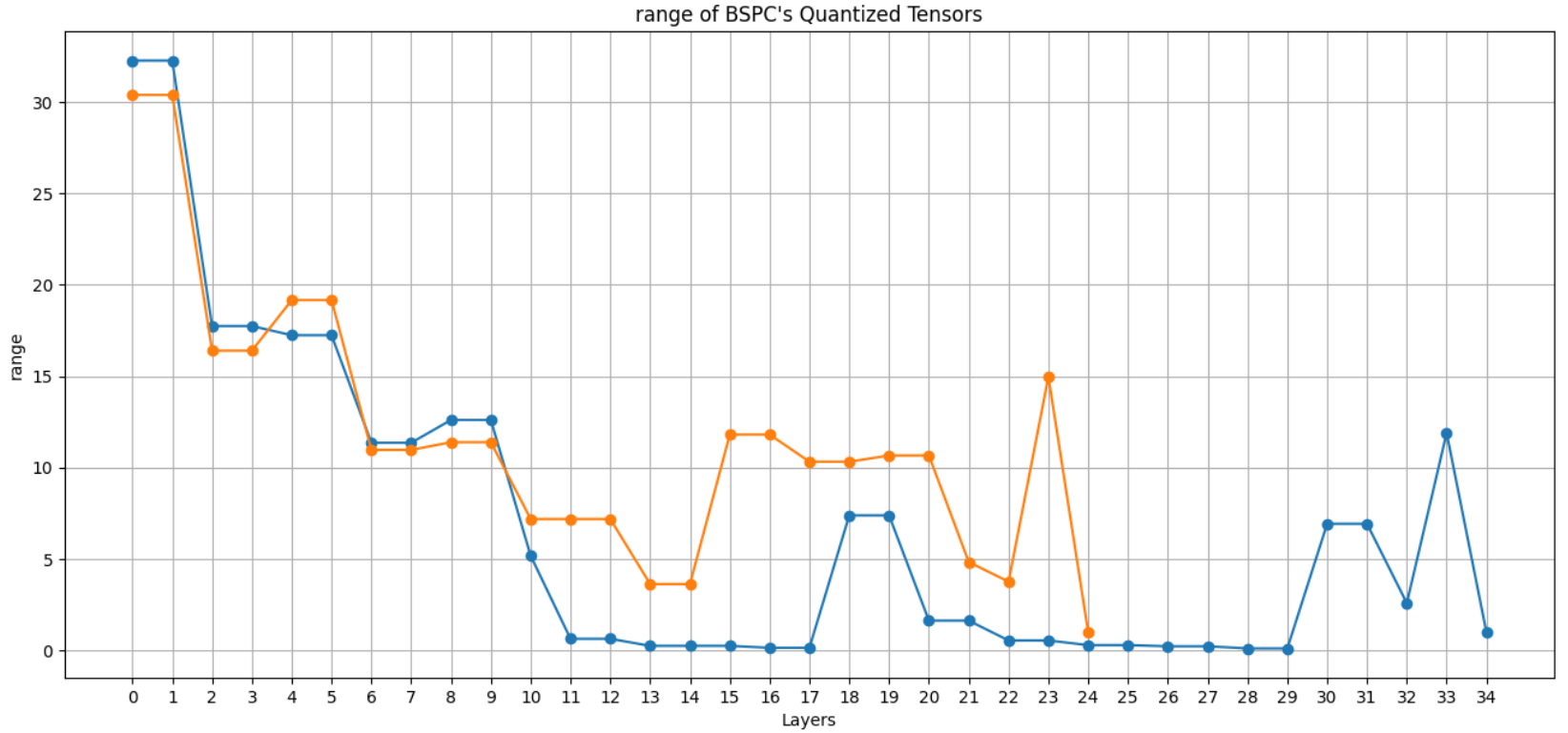
BSPC Model (final) – Quantization Debugger (6)

- 0_0_bottom_right (BSPC_seq_param100K_pruned_90)
- 0_7_top_right (BSPC_seq_param4M_pruned_80)



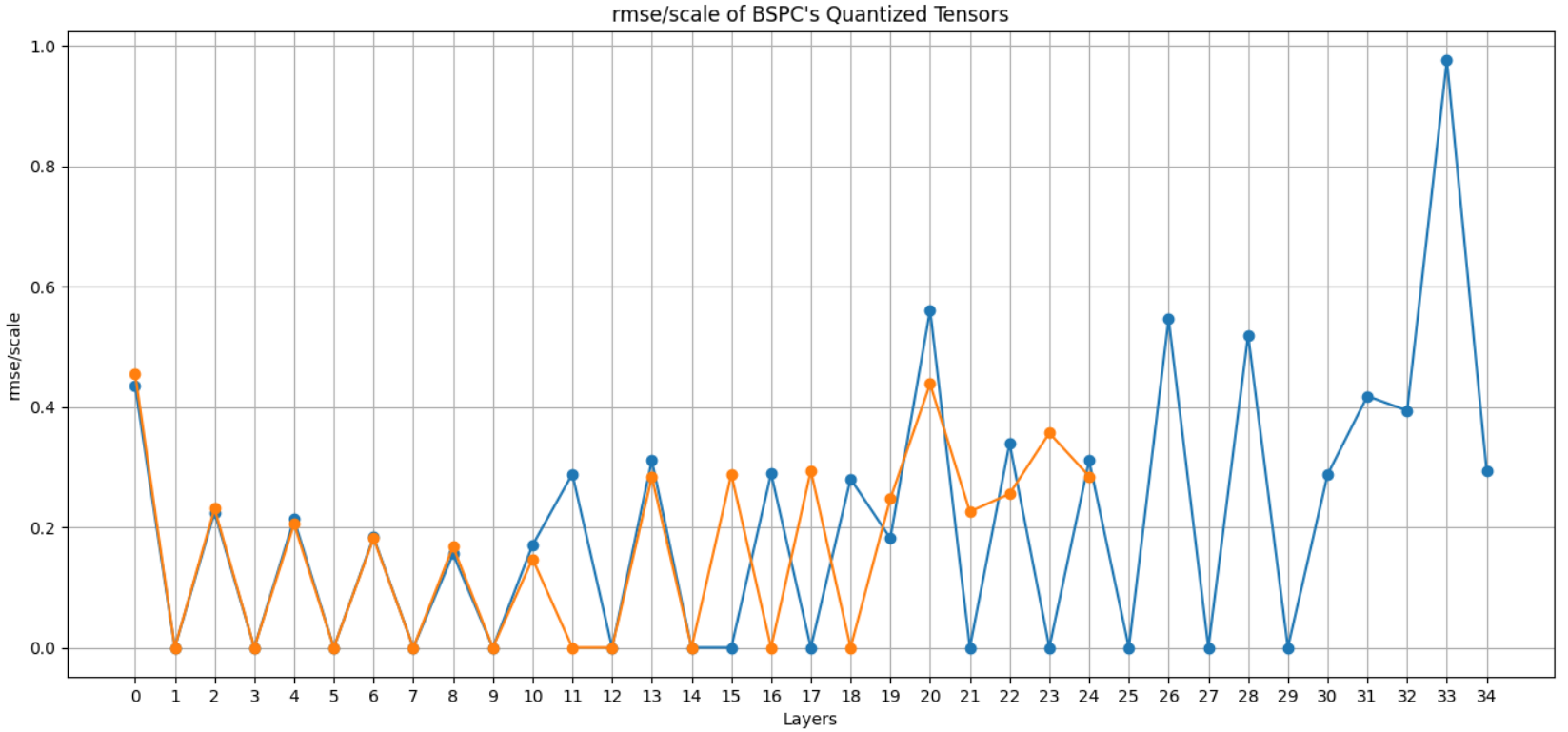
BSPC Model (final) – Quantization Debugger (7)

- 0_0_bottom_right (BSPC_seq_param100K_pruned_90)
- 0_7_top_right (BSPC_seq_param4M_pruned_80)



BSPC Model (final) – Quantization Debugger (8)

- 0_0_bottom_right (BSPC_seq_param100K_pruned_90)
- 0_7_top_right (BSPC_seq_param4M_pruned_80)



BSPC Model (final) – Layer Indexing

0_0_bottom_right

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MEAN
12	RESHAPE
13	BATCH_MATMUL
14	TRANSPOSE
15	RESHAPE
16	BATCH_MATMUL
17	RESHAPE
18	ADD
19	MUL
20	ADD
21	RESHAPE
22	BATCH_MATMUL
23	RESHAPE
24	BATCH_MATMUL
25	RESHAPE
26	BATCH_MATMUL
27	RESHAPE
28	BATCH_MATMUL
29	RESHAPE
30	ADD
31	MUL
32	ADD
33	FULLY_CONNECTED
34	LOGISTIC

0_7_top_right

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	MAX_POOL_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	MAX_POOL_2D
8	CONV_2D
9	MAX_POOL_2D
10	CONV_2D
11	MAX_POOL_2D
12	RESHAPE
13	BATCH_MATMUL
14	RESHAPE
15	BATCH_MATMUL
16	RESHAPE
17	BATCH_MATMUL
18	RESHAPE
19	ADD
20	MUL
21	ADD
22	FULLY_CONNECTED
23	FULLY_CONNECTED
24	LOGISTIC

BSPC-GAP Model (final) – Performance Metrics

BSPC_res_param250K_pruned_85 (0_0_bottom_left)

Quantization Scheme	Binary Accuracy	Precision	Recall	F1	Hamming Loss	Jaccard Score
Unquantized model	0.865	0.802	0.550	0.608	0.135	0.467
DRQ	0.864~	0.804~	0.547~	0.604~	0.136~	0.463~
FIQ – Float32	0.855~	0.793~	0.541~	0.596~	0.145~	0.450~
FIQ – Int8	0.855~	0.802~	0.529~	0.588~	0.145~	0.445~
FIQ – UInt8	0.854~	0.778~	0.542~	0.597~	0.146~	0.450~

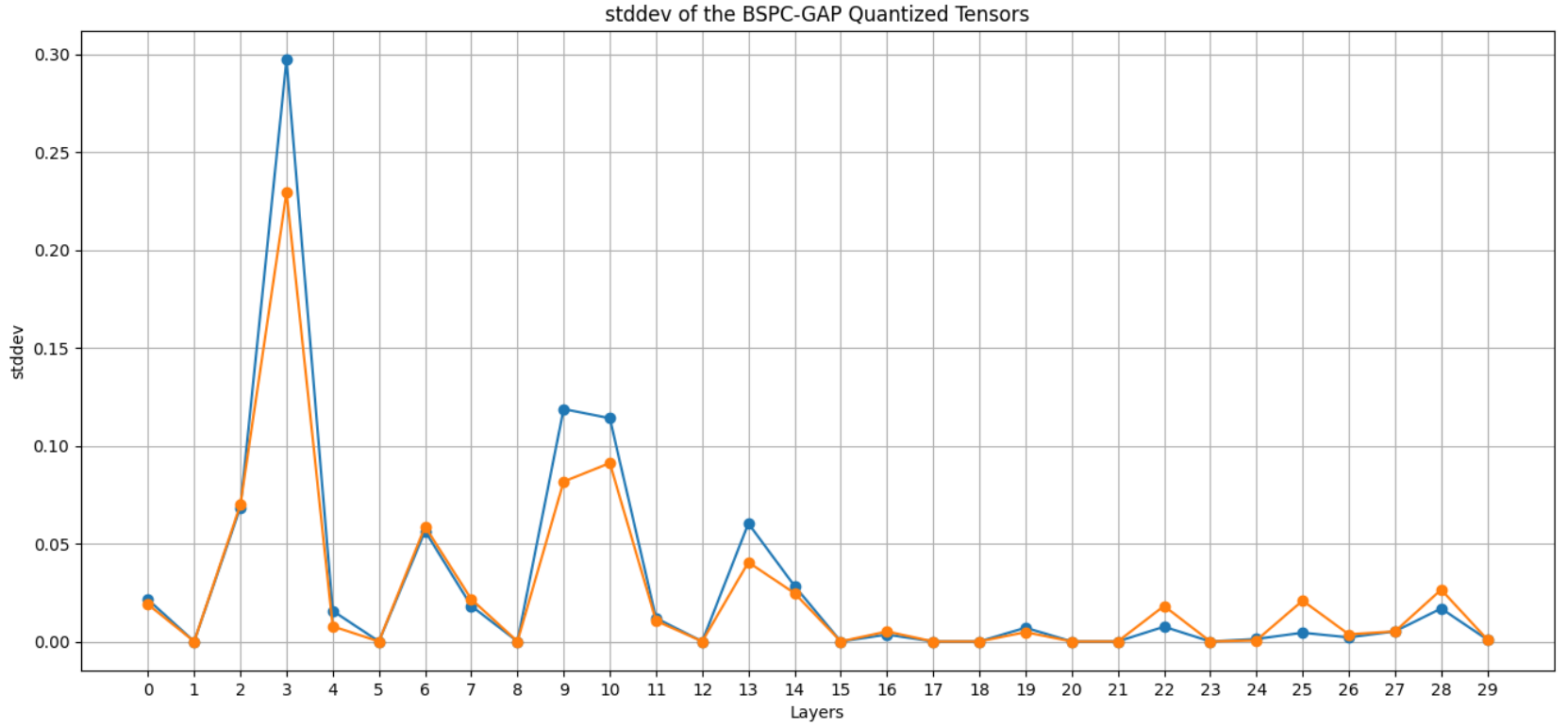
BSPC-GAP Model (final) – Performance Metrics

BSPC_res_param4M_pruned_85 (0_6_bottom_left)

Quantization Scheme	Binary Accuracy	Precision	Recall	F1	Hamming Loss	Jaccard Score
Unquantized model	0.883	0.786	0.676	0.713	0.117	0.565
DRQ	0.882~	0.786~	0.671~	0.710~	0.118~	0.561~
FIQ – Float32	0.883~	0.792~	0.674~	0.714~	0.117~	0.566~
FIQ – Int8	0.883~	0.792~	0.663~	0.709~	0.117~	0.560~
FIQ – UInt8	0.880~	0.786~	0.673~	0.708~	0.120~	0.559~

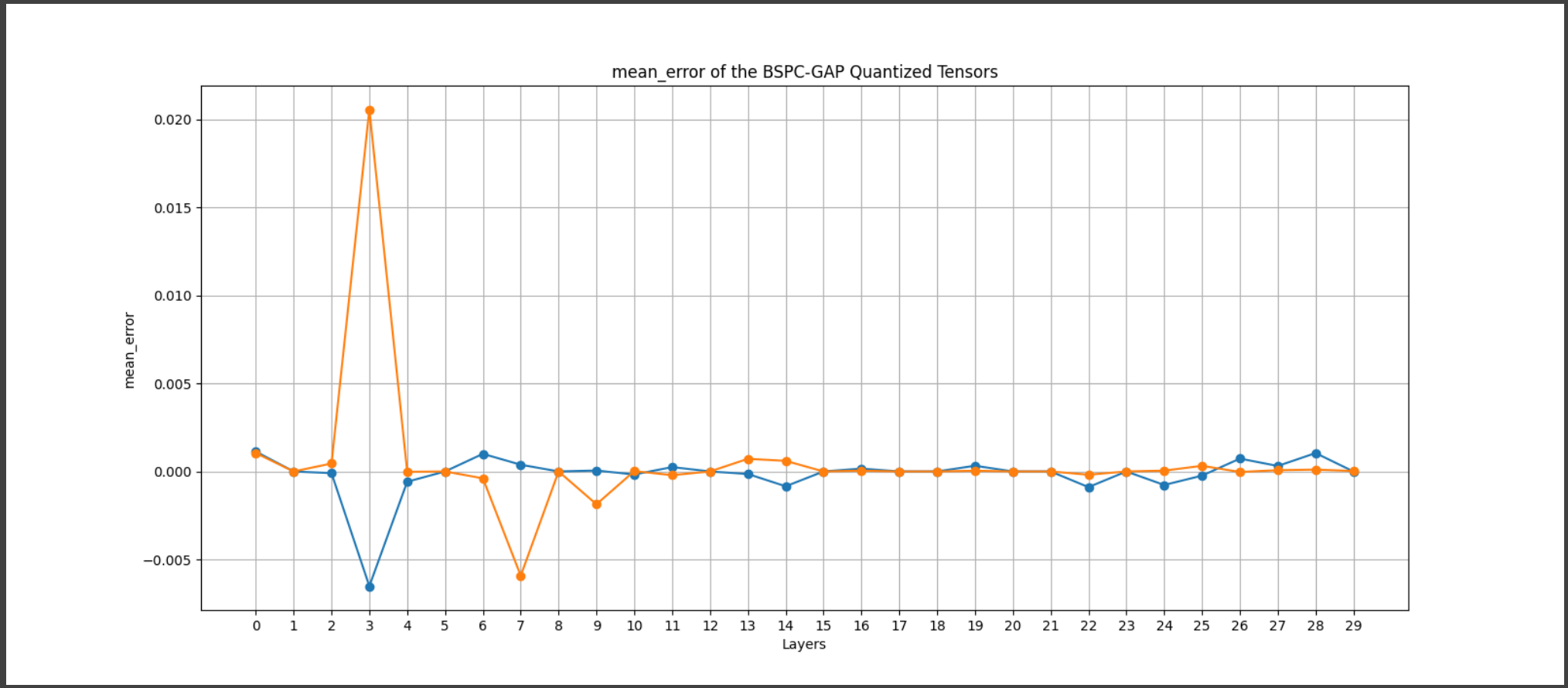
BSPC-GAP Model (final) – Quantization Debugger (1)

- 0_0_bottom_left (BSPC_res_param250K_pruned_85)
- 0_6_bottom_left (BSPC_res_param4M_pruned_85)



BSPC-GAP Model (final) – Quantization Debugger (2)

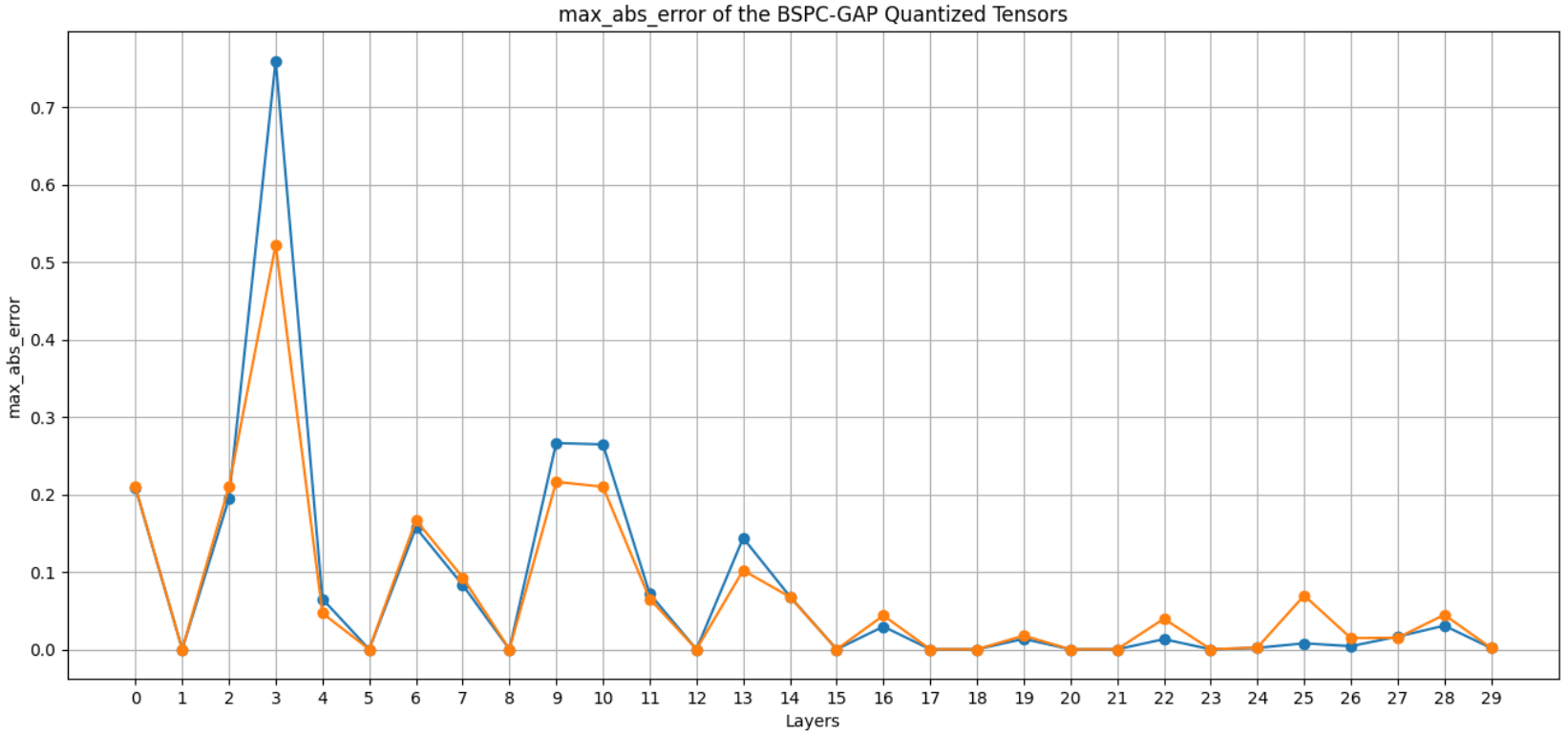
- 0_0_bottom_left (BSPC_res_param250K_pruned_85)
- 0_6_bottom_left (BSPC_res_param4M_pruned_85)



BSPC-GAP Model (final) – Quantization Debugger (3)

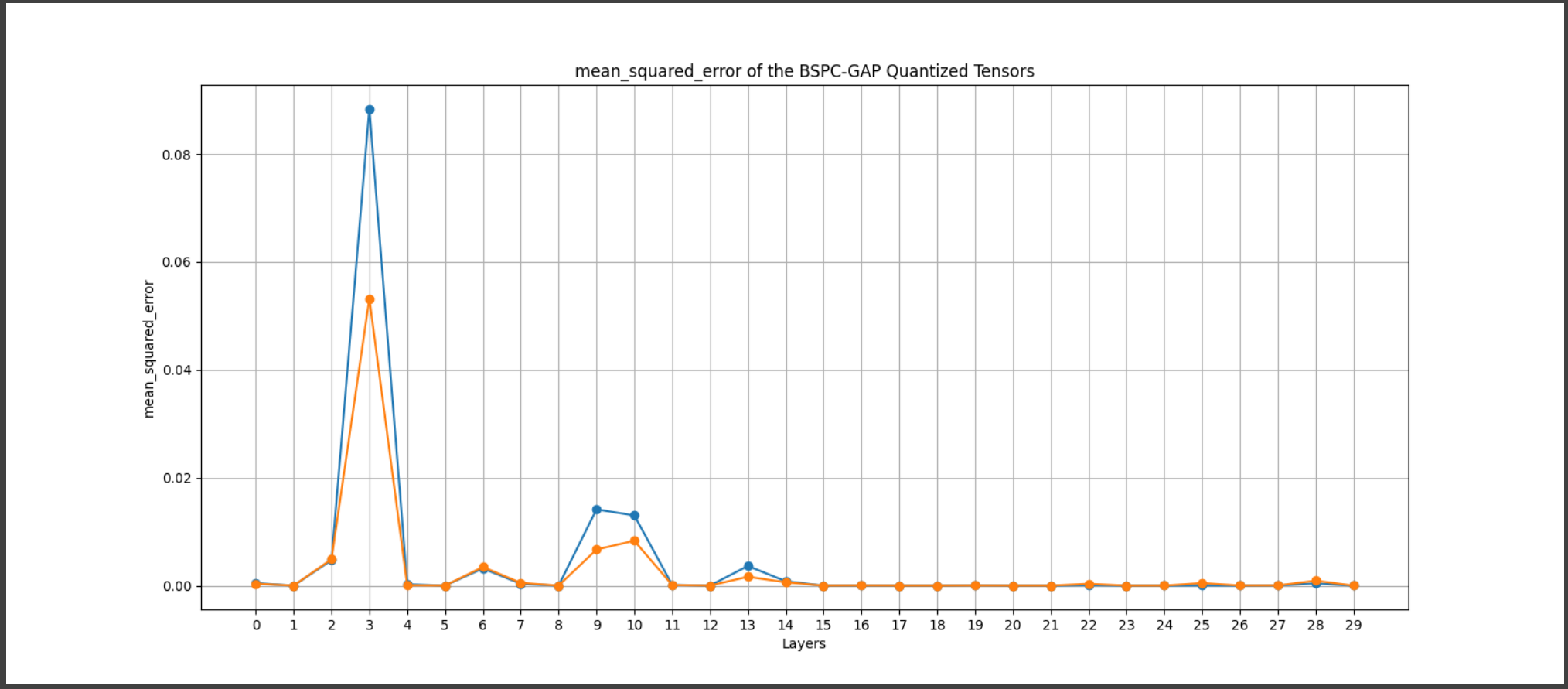
0_0_bottom_left (BSPC_res_param250K_pruned_85)

0_6_bottom_left (BSPC_res_param4M_pruned_85)



BSPC-GAP Model (final) – Quantization Debugger (4)

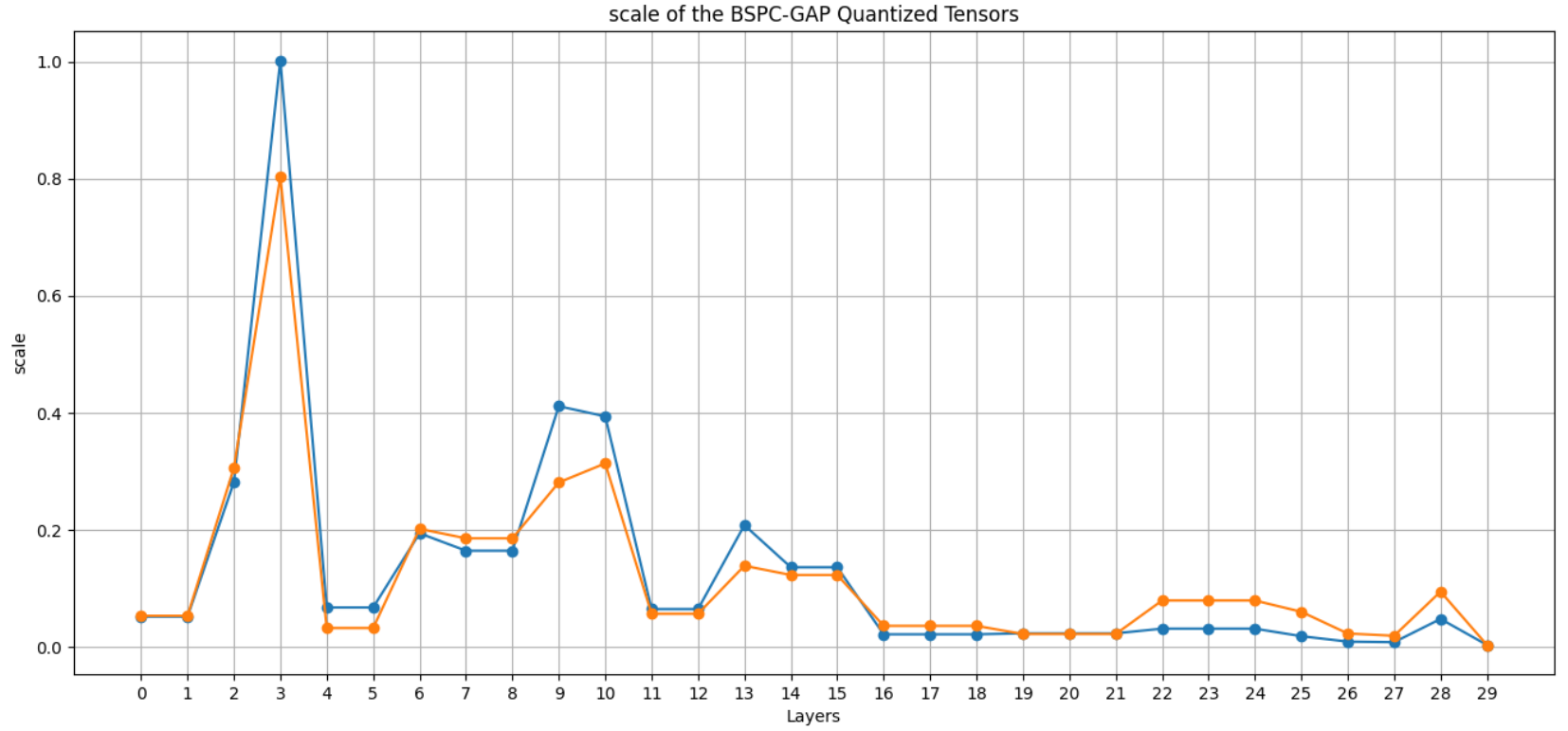
- 0_0_bottom_left (BSPC_res_param250K_pruned_85)
- 0_6_bottom_left (BSPC_res_param4M_pruned_85)



BSPC-GAP Model (final) – Quantization Debugger (5)

0_0_bottom_left (BSPC_res_param250K_pruned_85)

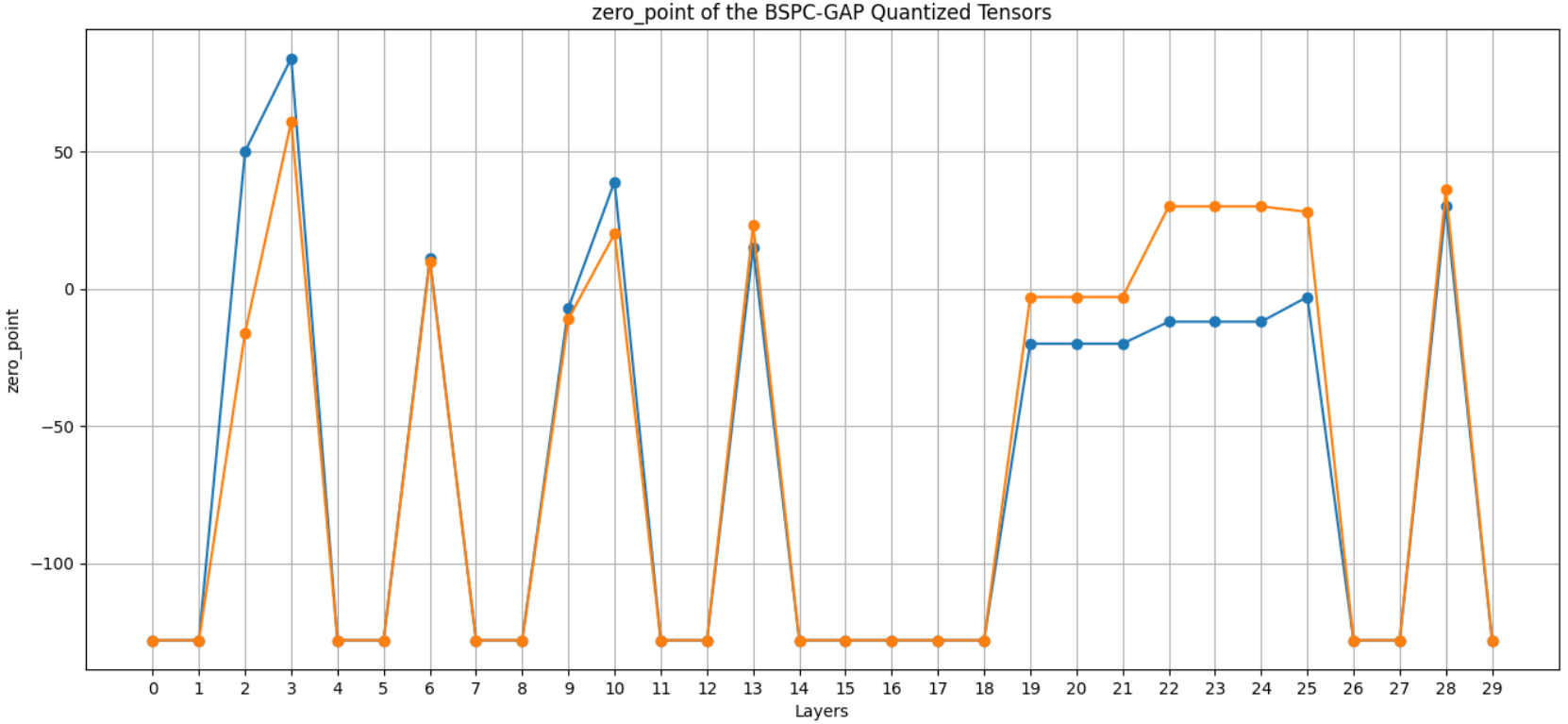
0_6_bottom_left (BSPC_res_param4M_pruned_85)



BSPC-GAP Model (final) – Quantization Debugger (6)

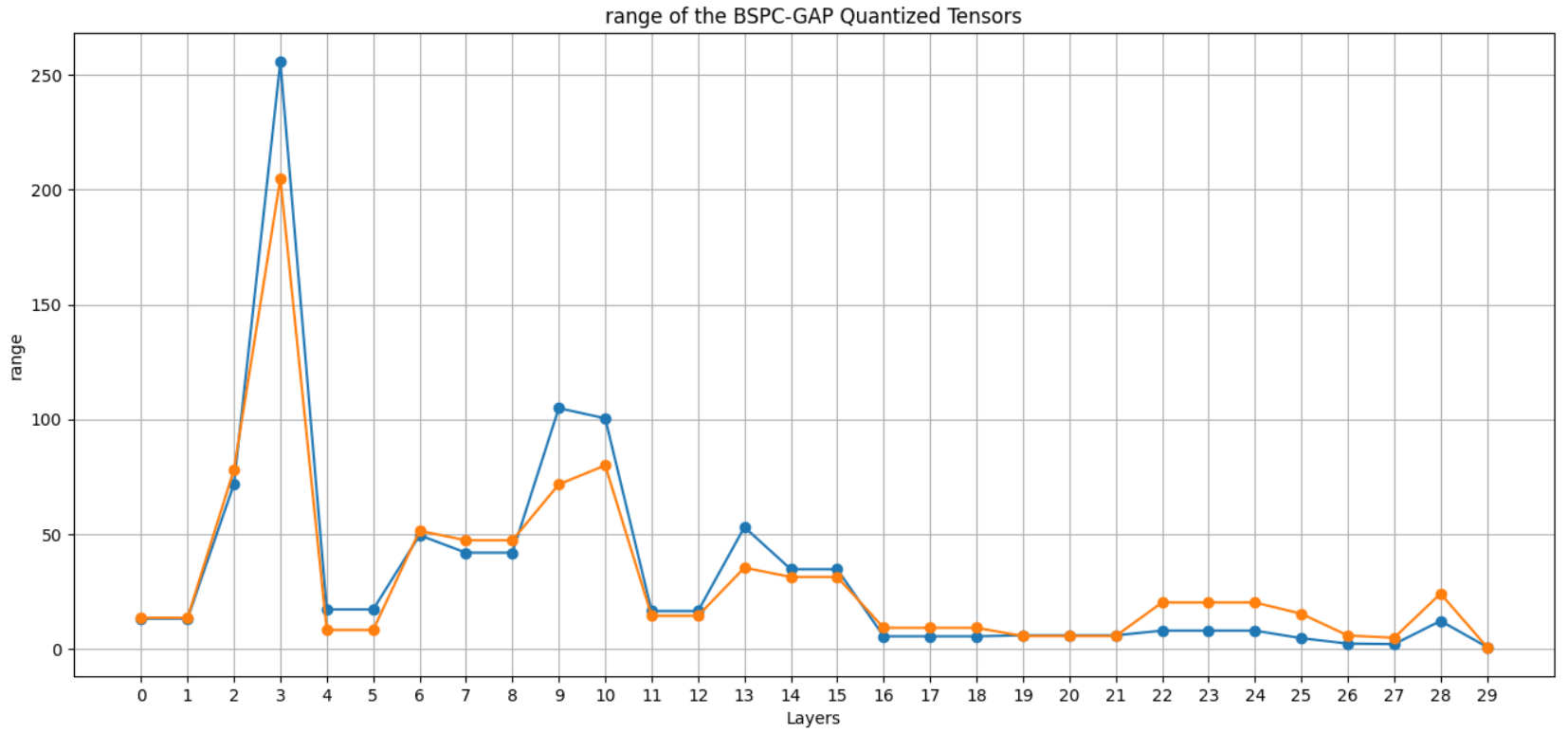
0_0_bottom_left (BSPC_res_param250K_pruned_85)

0_6_bottom_left (BSPC_res_param4M_pruned_85)



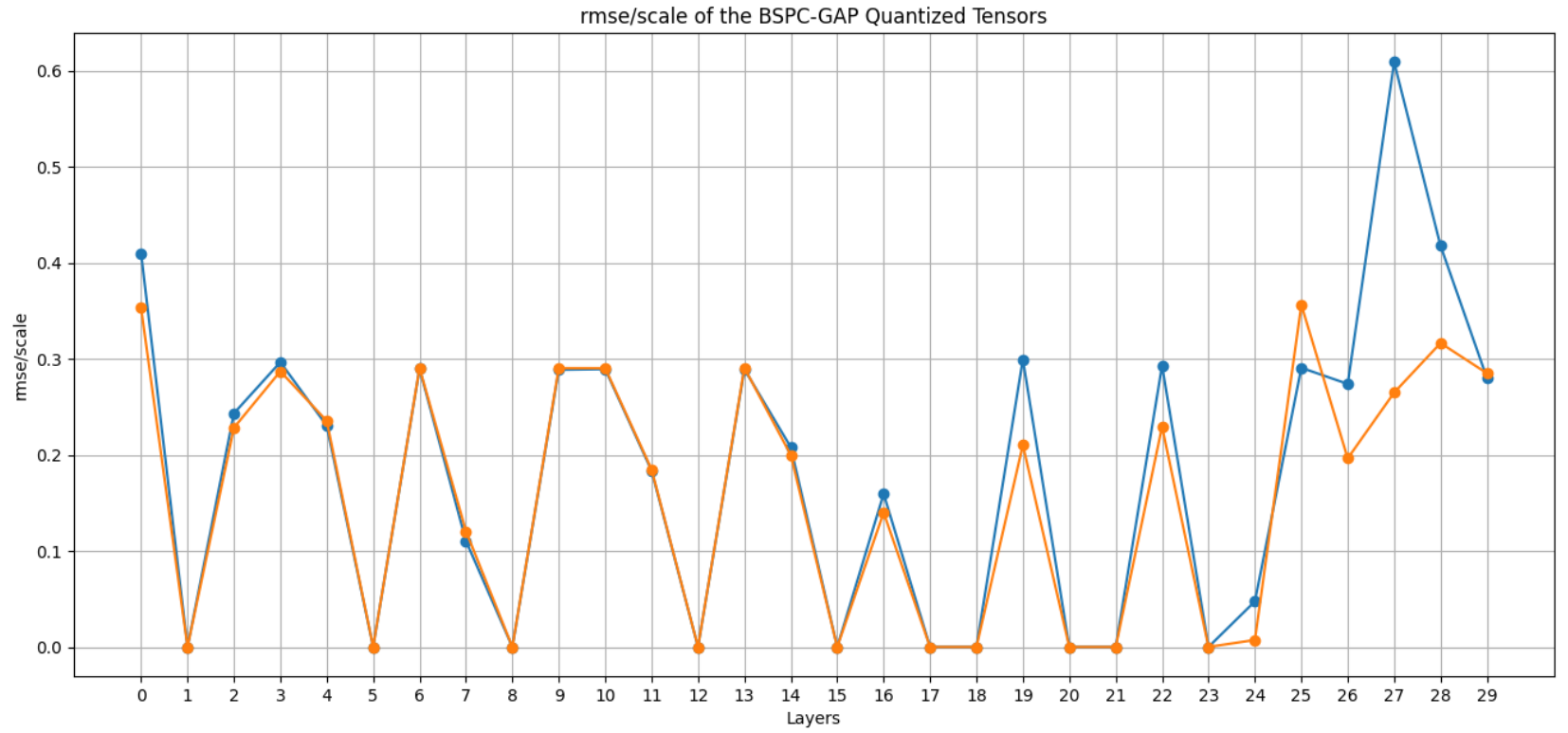
BSPC-GAP Model (final) – Quantization Debugger (7)

- 0_0_bottom_left (BSPC_res_param250K_pruned_85)
- 0_6_bottom_left (BSPC_res_param4M_pruned_85)



BSPC-GAP Model (final) – Quantization Debugger (8)

- 0_0_bottom_left (BSPC_res_param250K_pruned_85)
- 0_6_bottom_left (BSPC_res_param4M_pruned_85)



BSPC-GAP Model (final) – Layer Indexing

0_0_bottom_left

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	CONV_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	ADD
8	MAX_POOL_2D
9	CONV_2D
10	CONV_2D
11	CONV_2D
12	MAX_POOL_2D
13	CONV_2D
14	ADD
15	MAX_POOL_2D
16	CONV_2D
17	MAX_POOL_2D
18	RESHAPE
19	BATCH_MATMUL
20	TRANSPOSE
21	RESHAPE
22	BATCH_MATMUL
23	RESHAPE
24	ADD
25	MUL
26	ADD
27	FULLY_CONNECTED
28	FULLY_CONNECTED
29	LOGISTIC

0_6_bottom_left

	op_name
0	CONV_2D
1	MAX_POOL_2D
2	CONV_2D
3	CONV_2D
4	CONV_2D
5	MAX_POOL_2D
6	CONV_2D
7	ADD
8	MAX_POOL_2D
9	CONV_2D
10	CONV_2D
11	CONV_2D
12	MAX_POOL_2D
13	CONV_2D
14	ADD
15	MAX_POOL_2D
16	CONV_2D
17	MAX_POOL_2D
18	RESHAPE
19	BATCH_MATMUL
20	TRANSPOSE
21	RESHAPE
22	BATCH_MATMUL
23	RESHAPE
24	ADD
25	MUL
26	ADD
27	FULLY_CONNECTED
28	FULLY_CONNECTED
29	LOGISTIC