Using the OpenVINO™ Toolkit for Deploying Accelerated Deep Learning Applications – Part2 [2021.4]

July 2021

intel®

Agenda

Part 1: OpenVINO Workshop (110mins):

- Demos on DevCloud
- Post-Training Optimization Tool
- DL Workbench
- DL Streamer

- **Part2: Q & A(10mins)**

Notices and Disclaimers

- Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.
- Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.
- Your costs and results may vary.
- Intel technologies may require enabled hardware, software or service activation.
- All product plans and roadmaps are subject to change without notice.
- Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.
-

Intel® DevCloud for the Edge Demo

https://software.intel.com/content/www/us/en/develop/tools/devcloud/edge/build/sample-apps.html

July 2021

intel®

Post-Training Optimization Tool

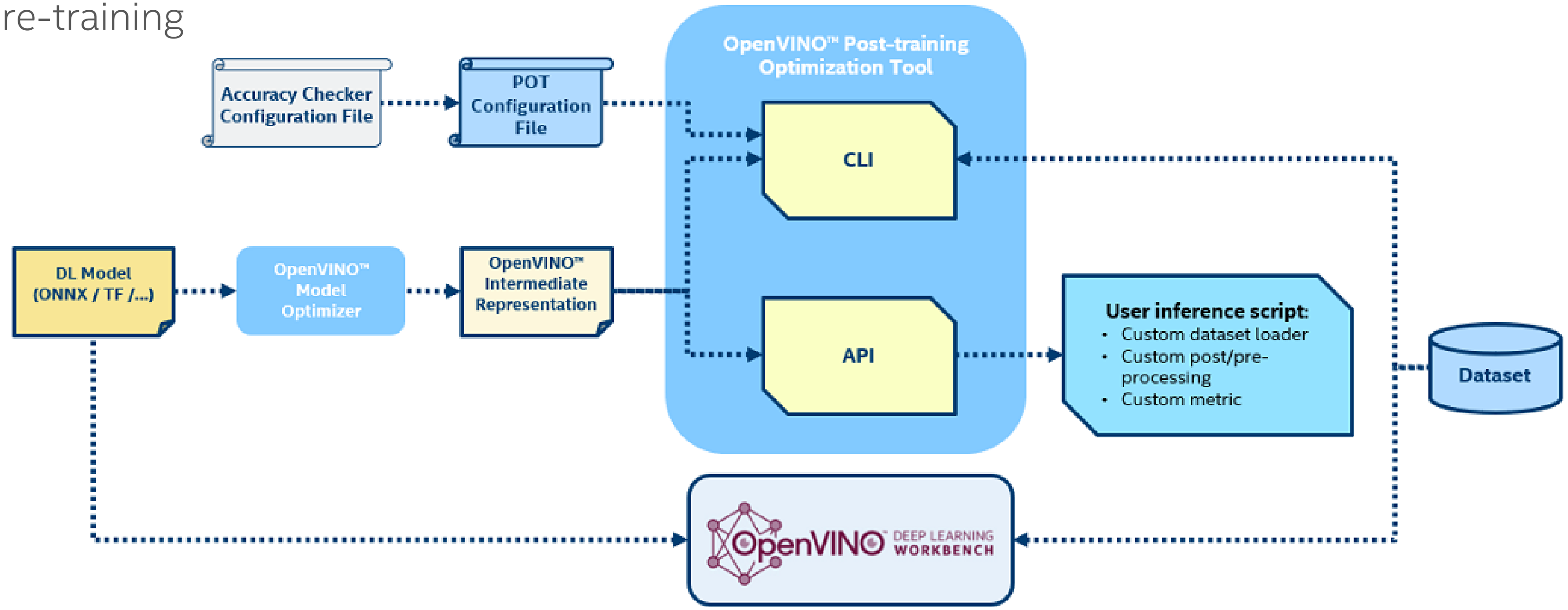
July 2021

intel®

Post-Training Optimization Tool

https://docs.openvinotoolkit.org/latest/pot_README.html

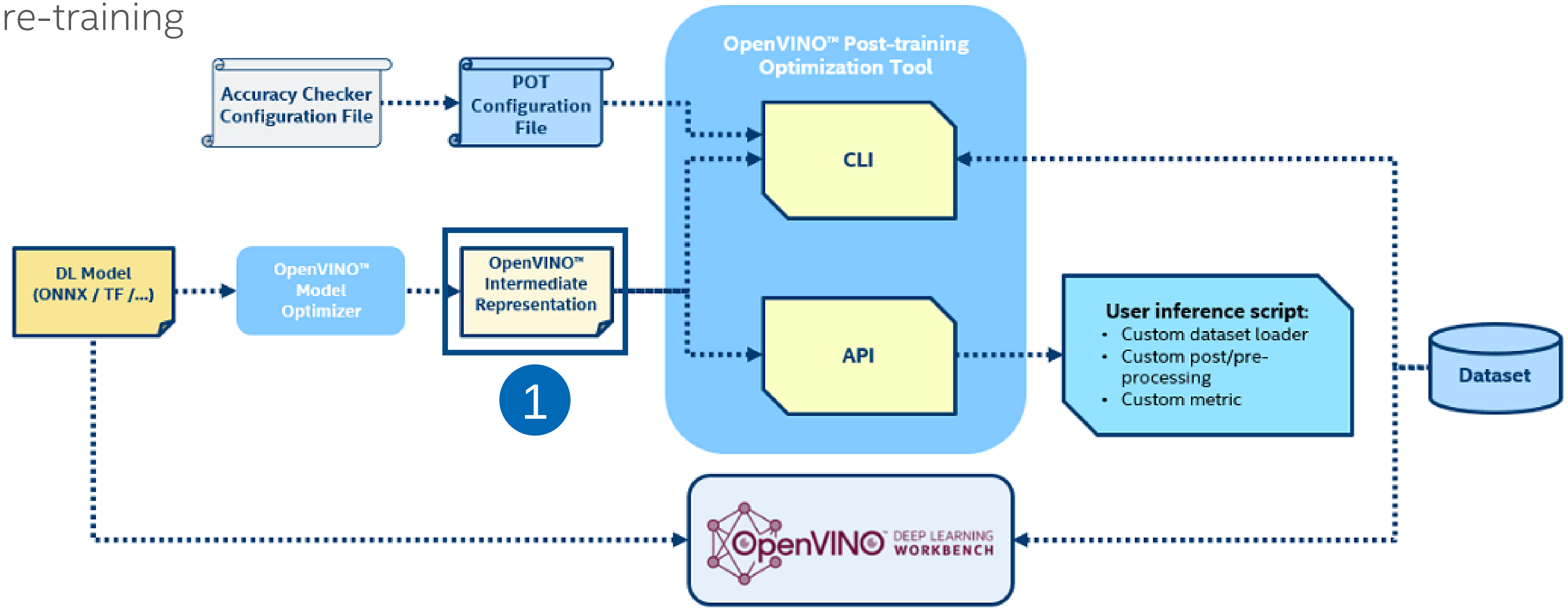
- Using the Python* API, the Post-training Optimization Tool integrates with the Model Optimizer, DL Workbench and accuracy checker tools to streamline the development process
- Enables a conversion technique of deep learning model that **reduces model size into low precision data types**, such as INT8, without re-training
- Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.
- Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Post-Training Optimization Tool

https://docs.openvinotoolkit.org/latest/pot_README.html

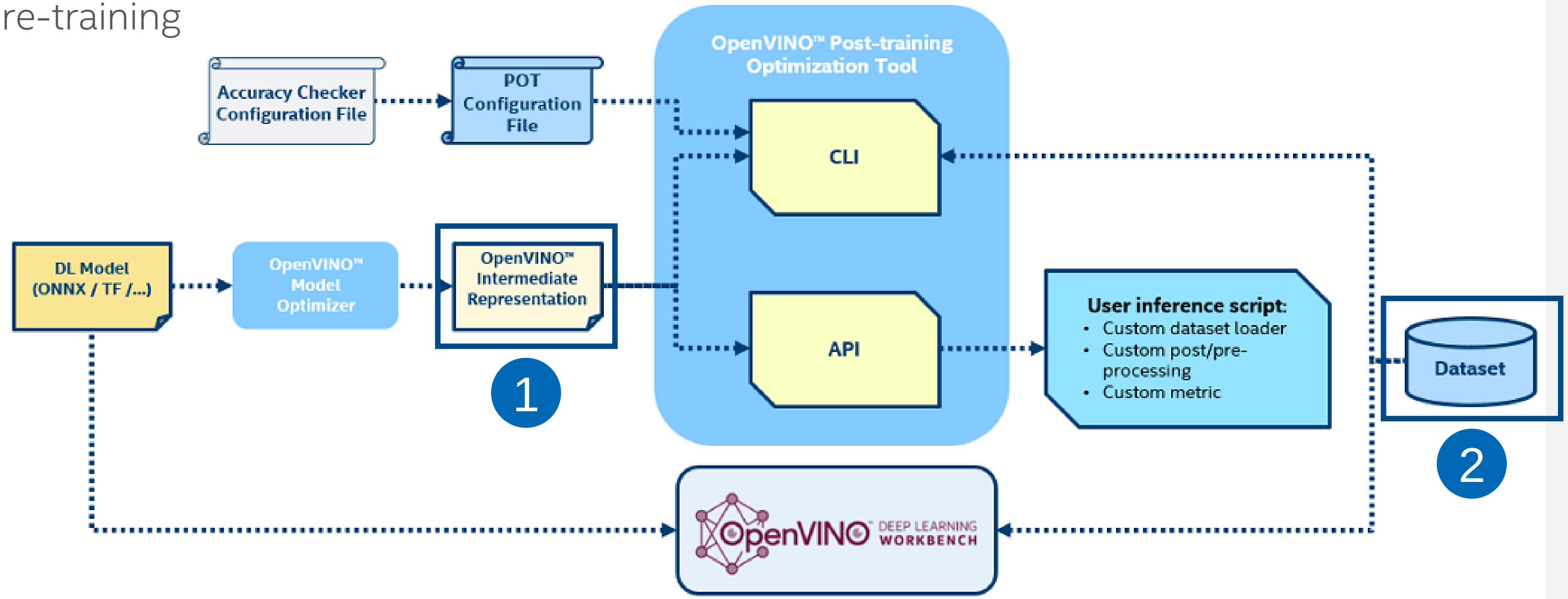
- Using the Python* API, the Post-training Optimization Tool integrates with the Model Optimizer, DL Workbench and accuracy checker tools to streamline the development process
- Enables a conversion technique of deep learning model that **reduces model size into low precision data types**, such as INT8, without re-training
- Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.
- Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Post-Training Optimization Tool

https://docs.openvinotoolkit.org/latest/pot_README.html

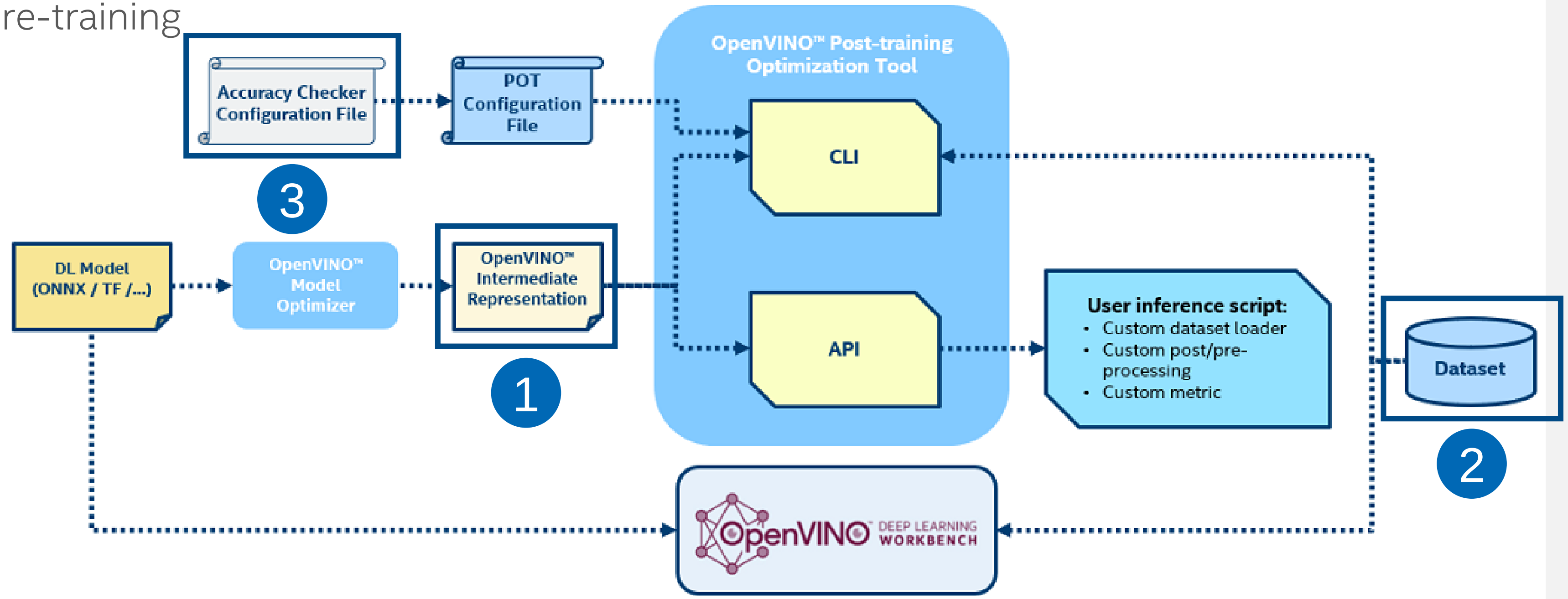
- Using the Python* API, the Post-training Optimization Tool integrates with the Model Optimizer, DL Workbench and accuracy checker tools to streamline the development process
- Enables a conversion technique of deep learning model that **reduces model size into low precision data types**, such as INT8, without re-training
- Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.
- Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Post-Training Optimization Tool

https://docs.openvinotoolkit.org/latest/pot_README.html

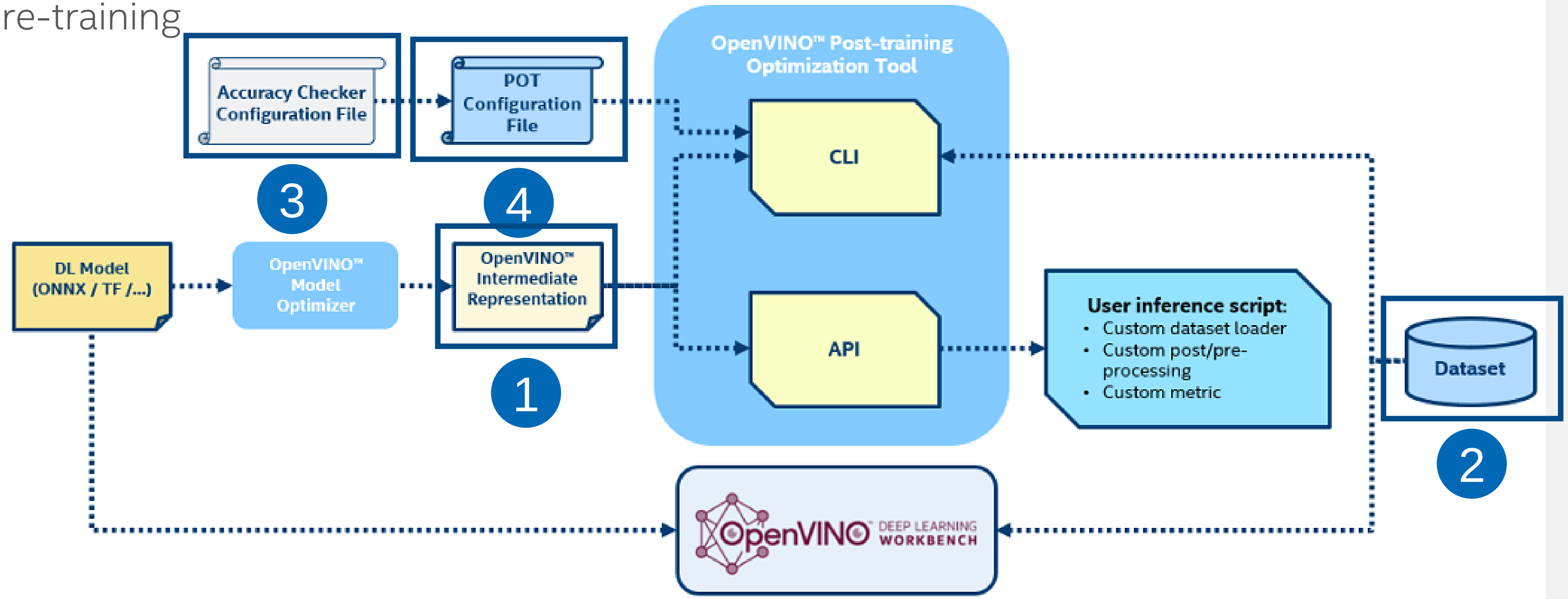
- Using the Python* API, the Post-training Optimization Tool integrates with the Model Optimizer, DL Workbench and accuracy checker tools to streamline the development process
- Enables a conversion technique of deep learning model that **reduces model size into low precision data types**, such as INT8, without re-training
- Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.
- Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Post-Training Optimization Tool

https://docs.openvinotoolkit.org/latest/pot_README.html

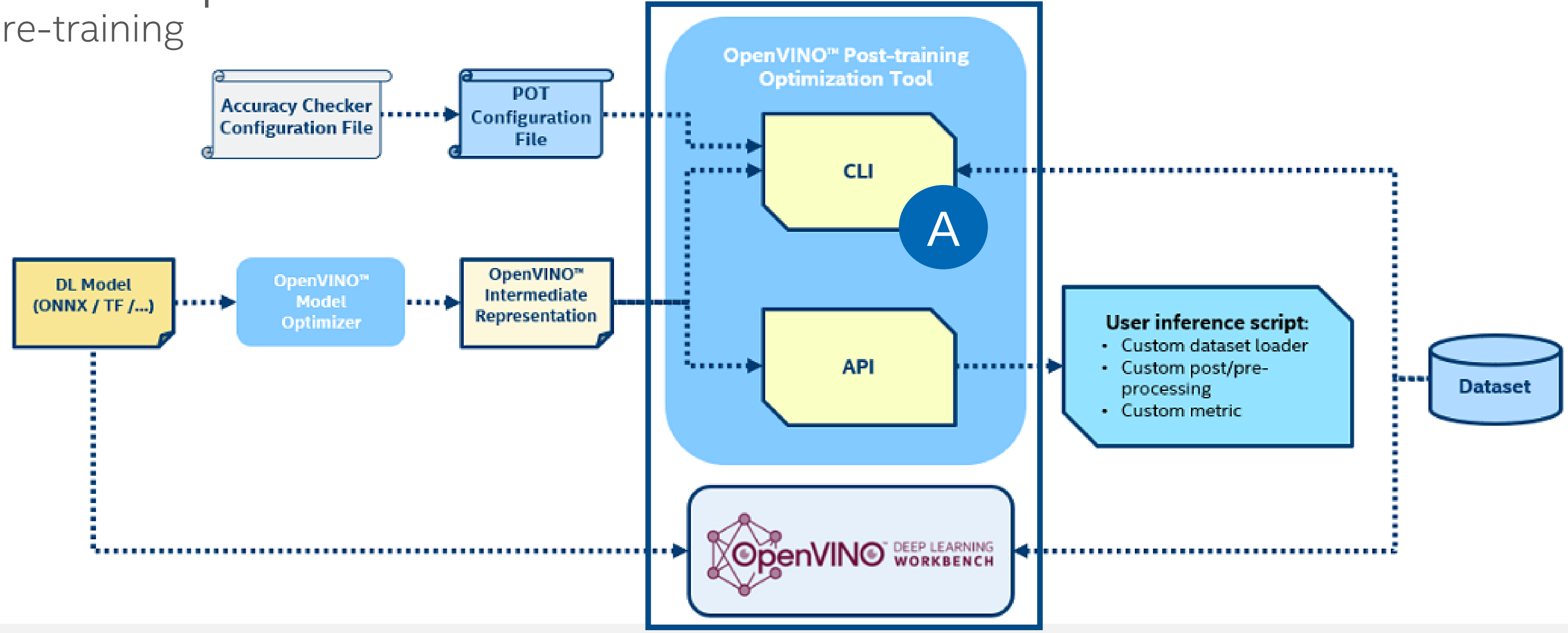
- Using the Python* API, the Post-training Optimization Tool integrates with the Model Optimizer, DL Workbench and accuracy checker tools to streamline the development process
- Enables a conversion technique of deep learning model that **reduces model size into low precision data types**, such as INT8, without re-training
- Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.
- Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Post-Training Optimization Tool

https://docs.openvinotoolkit.org/latest/pot_README.html

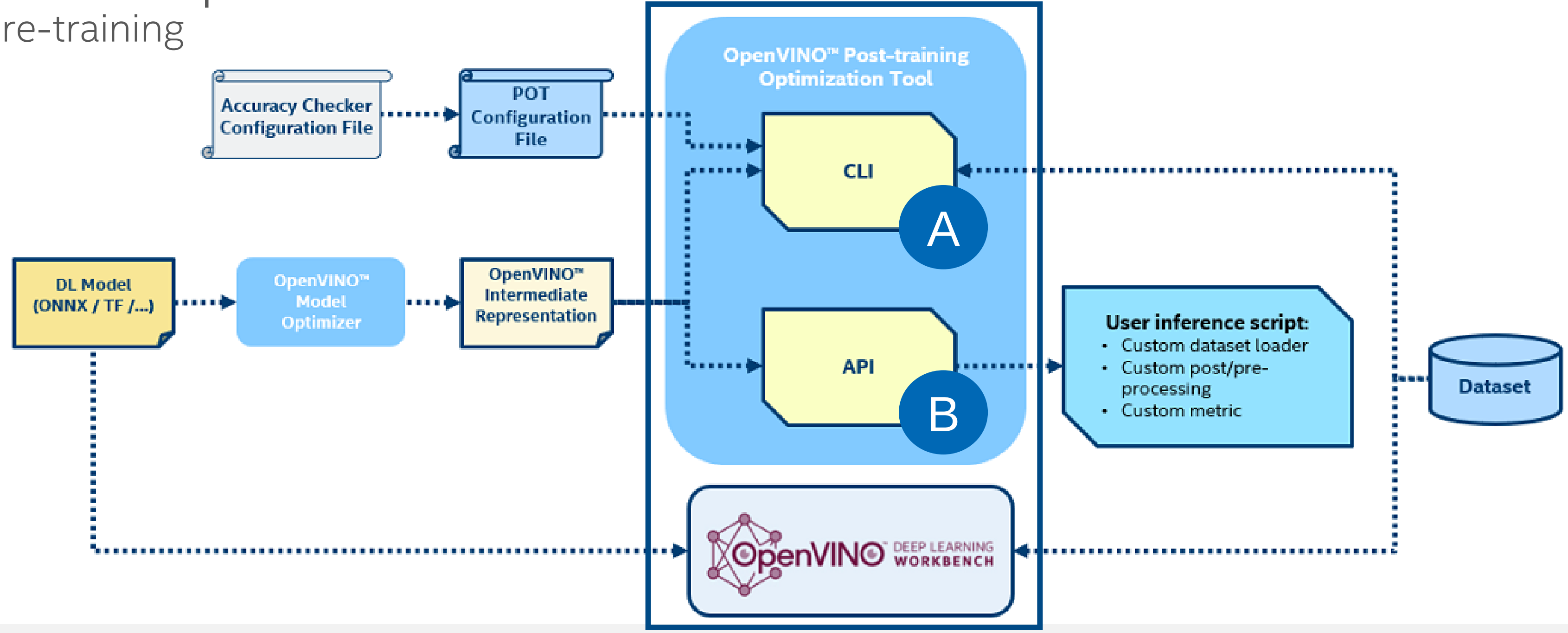
- Using the Python* API, the Post-training Optimization Tool integrates with the Model Optimizer, DL Workbench and accuracy checker tools to streamline the development process
- Enables a conversion technique of deep learning model that **reduces model size into low precision data types**, such as INT8, without re-training
- Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.
- Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Post-Training Optimization Tool

https://docs.openvinotoolkit.org/latest/pot_README.html

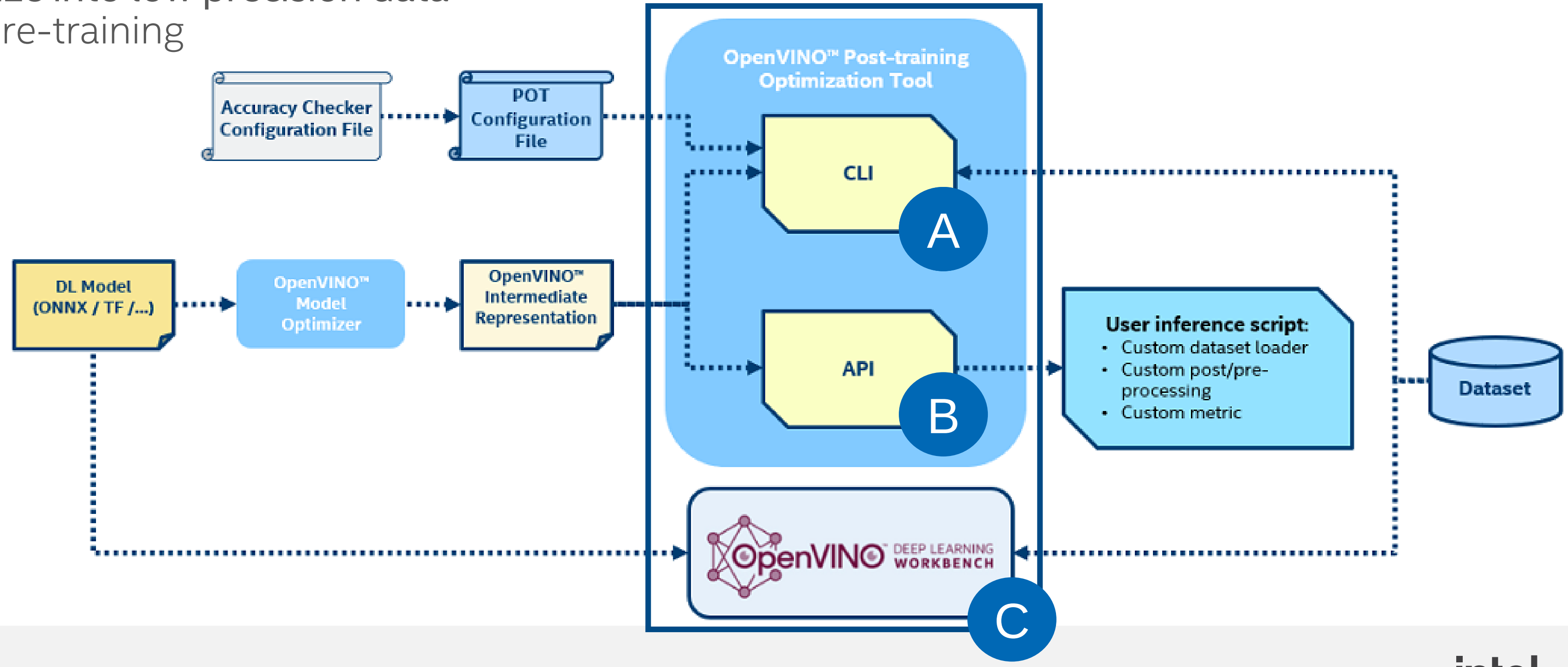
- Using the Python* API, the Post-training Optimization Tool integrates with the Model Optimizer, DL Workbench and accuracy checker tools to streamline the development process
- Enables a conversion technique of deep learning model that **reduces model size into low precision data types**, such as INT8, without re-training
- Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.
- Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Post-Training Optimization Tool

https://docs.openvinotoolkit.org/latest/pot_README.html

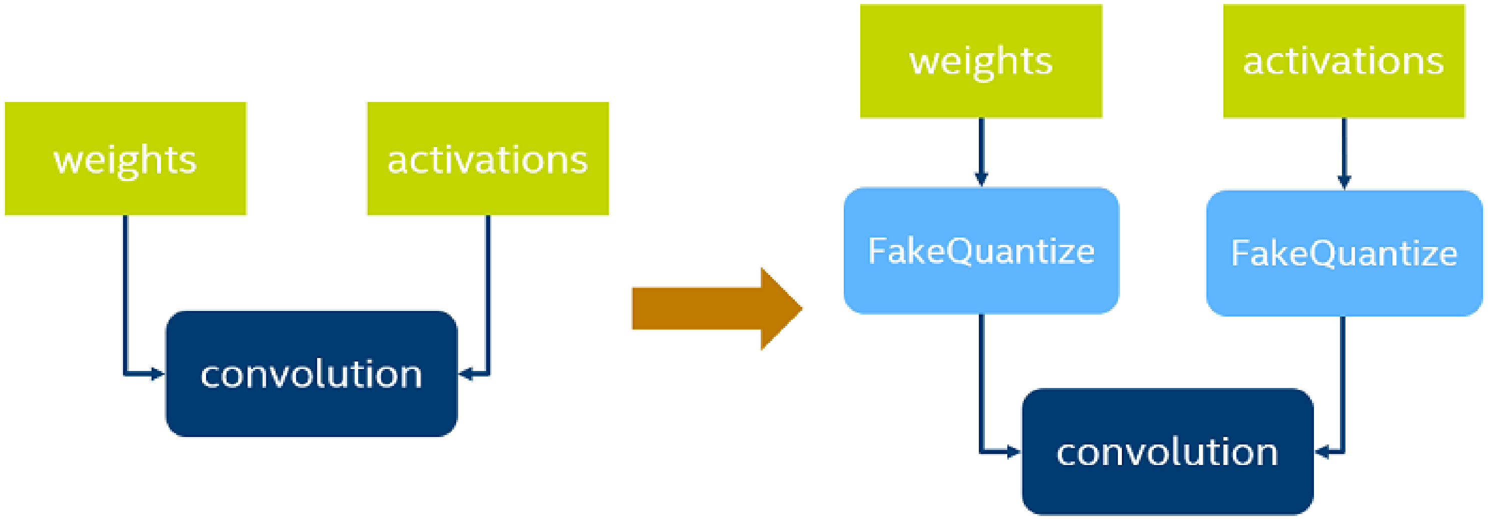
- Using the Python* API, the Post-training Optimization Tool integrates with the Model Optimizer, DL Workbench and accuracy checker tools to streamline the development process
- Enables a conversion technique of deep learning model that **reduces model size into low precision data types**, such as INT8, without re-training
- Reduces model size **while also improving latency, with little degradation** in model accuracy and without model re-training.
- Different optimization approaches are supported: quantization algorithms, sparsity, etc.



Post-Training Optimization Tool

Quantization Algorithms

https://docs.openvinotoolkit.org/latest/pot_compression_algorithms_quantization_README.html



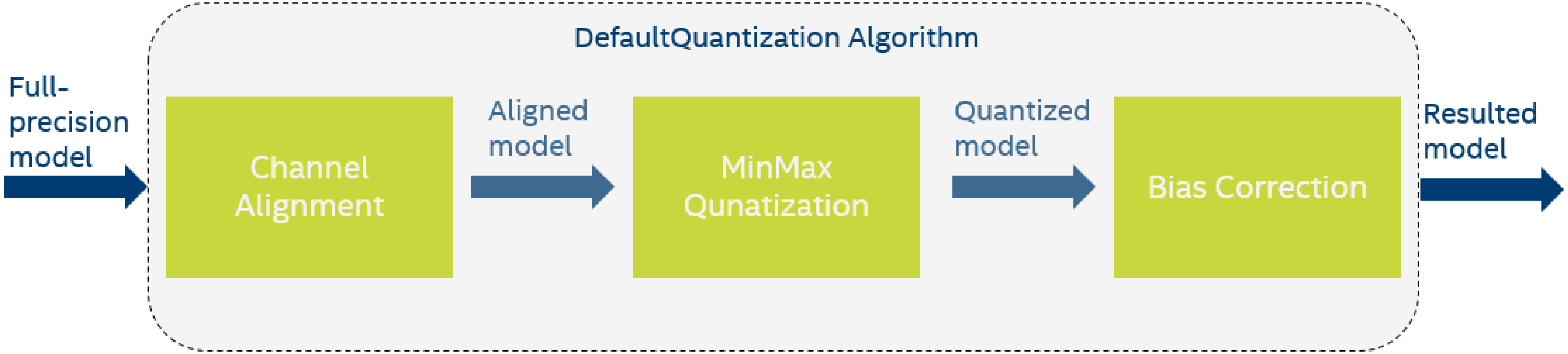
3. **Tree-Structured Parzen Estimator (TPE)** tries to provide best possible performance improvement. It requires even more time for quantization than **AccuracyAwareQuantization** but may lead to better performance improvement.

1. **DefaultQuantization** is a default method that provides fast and, in most cases, accurate results for 8-bit quantization.

2. **AccuracyAwareQuantization** enables remaining at a predefined range of accuracy drop after quantization at the cost of performance improvement. It may require more time for quantization.

Post-Training Optimization Tool – Best Practices

Start with **DefaultQuantization** Algorithm with default settings




```
{
    /* Model parameters */

    "model": {
        "model_name": "model_name", // Model name
        "model": "<MODEL_PATH>", // Path to model (.xml format)
        "weights": "<PATH_TO_WEIGHTS>" // Path to weights (.bin format)
    },

    /* Parameters of the engine used for model inference */

    "engine": {
        "config": "<CONFIG_PATH>" // Path to Accuracy Checker config
    },
```

default_quantization_template.json

```

    /* Optimization hyperparameters */

    "compression": {
        "target_device": "ANY", // Target device, the specificity of which will be taken
                                // into account during optimization
        "algorithms": [
            {
                "name": "DefaultQuantization", // Optimization algorithm name
                "params": {
                    "preset": "performance", // Preset [performance, mixed, accuracy] which control the quantization
                                             // mode (symmetric, mixed (weights symmetric and activations asymmetric)
                                             // and fully asymmetric respectively)

                    "stat_subset_size": 300  // Size of subset to calculate activations statistics that can be used
                                             // for quantization parameters calculation
                }
            }
        ]
    }
}
```

Post-Training Optimization Tool – Best Practices

Tuning Hyperparameters of the **DefaultQuantization**

- `"preset"` - preset which controls the quantization mode (symmetric and asymmetric). It can take two values:
  - `"performance"` (default) - stands for symmetric quantization of weights and activations. This is the most performant across all the HW.
  - `"mixed"` - symmetric quantization of weights and asymmetric quantization of activations. This mode can be useful for quantization of NN which has both negative and positive input values in quantizing operations, e.g. non-ReLU based CNN.
- `"stat_subset_size"` - size of subset to calculate activations statistics used for quantization. The whole dataset is used if no parameter specified. We recommend using not less than 300 samples.

- `"ignored"` - NN subgraphs which should be excluded from the optimization process
  - `"scope"` - list of particular nodes to exclude

Post-Training Optimization Tool – Best Practices

Tuning Hyperparameters of the **DefaultQuantization**

- `"preset"` - preset which controls the quantization mode (symmetric and asymmetric). It can take two values:
  - `"performance"` (default) - stands for symmetric quantization of weights and activations. This is the most performant across all the HW.
  - `"mixed"` - symmetric quantization of weights and asymmetric quantization of activations. This mode can be useful for quantization of NN which has both negative and positive input values in quantizing operations, e.g. non-ReLU based CNN.

- `"stat_subset_size"` - size of subset to calculate activations statistics used for quantization. The whole dataset is used if no parameter specified. We recommend using not less than 300 samples.

- `"ignored"` - NN subgraphs which should be excluded from the optimization process
  - `"scope"` - list of particular nodes to exclude

Post-Training Optimization Tool – Best Practices

Tuning Hyperparameters of the **DefaultQuantization**

- `"preset"` - preset which controls the quantization mode (symmetric and asymmetric). It can take two values:
  - `"performance"` (default) - stands for symmetric quantization of weights and activations. This is the most performant across all the HW.
  - `"mixed"` - symmetric quantization of weights and asymmetric quantization of activations. This mode can be useful for quantization of NN which has both negative and positive input values in quantizing operations, e.g. non-ReLU based CNN.
- `"stat_subset_size"` - size of subset to calculate activations statistics used for quantization. The whole dataset is used if no parameter specified. We recommend using not less than 300 samples.

- `"ignored"` - NN subgraphs which should be excluded from the optimization process
  - `"scope"` - list of particular nodes to exclude

Post-Training Optimization Tool – Best Practices

Tuning Hyperparameters of the **DefaultQuantization**

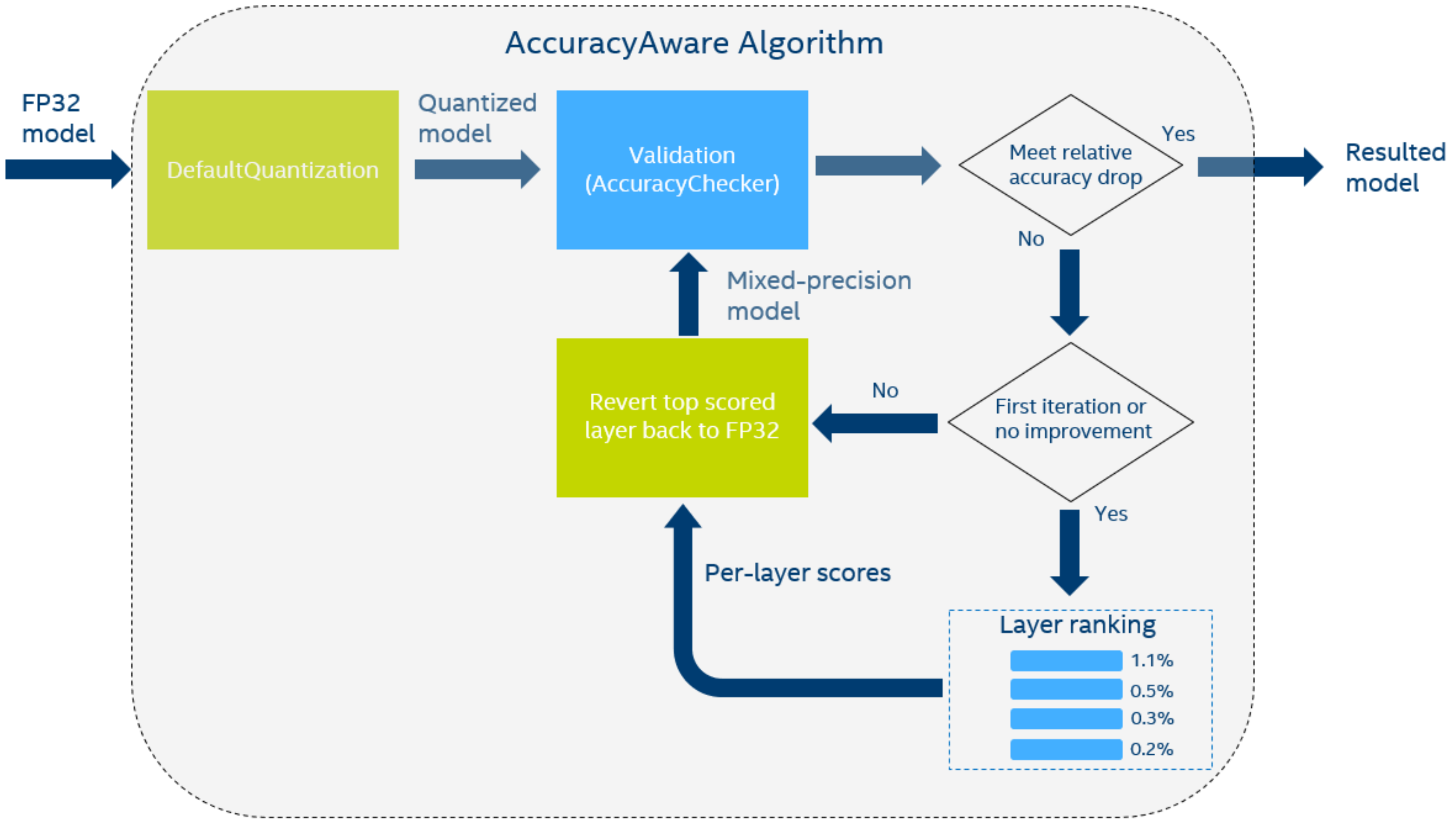
- `"range_estimator"` - this section describes parameters of range estimator that is used in MinMaxQuantization method to get the quantization ranges and filter outliers based on the collected statistics. These are the parameters that user can vary to get better accuracy results:
  - `"max"` - parameters to estimate top border of quantizing floating-point range:
    - `"type"` - type of the estimator:
      - `"max"` (default) - estimates the maximum in the quantizing set of value
      - `"quantile"` - estimates the quantile in the quantizing set of value
    - `"outlier_prob"` - outlier probability used in the "quantile" estimator
  - `"min"` - parameters to estimate bottom border of quantizing floating-point range:
    - `"type"` - type of the estimator:
      - `"min"` (default) - estimates the minimum in the quantizing set of value
      - `"quantile"` - estimates the quantile in the quantizing set of value
    - `"outlier_prob"` - outlier probability used in the "quantile" estimator

Learn more about DefaultQuantization:

https://docs.openvinotoolkit.org/latest/pot_compression_algorithms_quantization_default_README.html

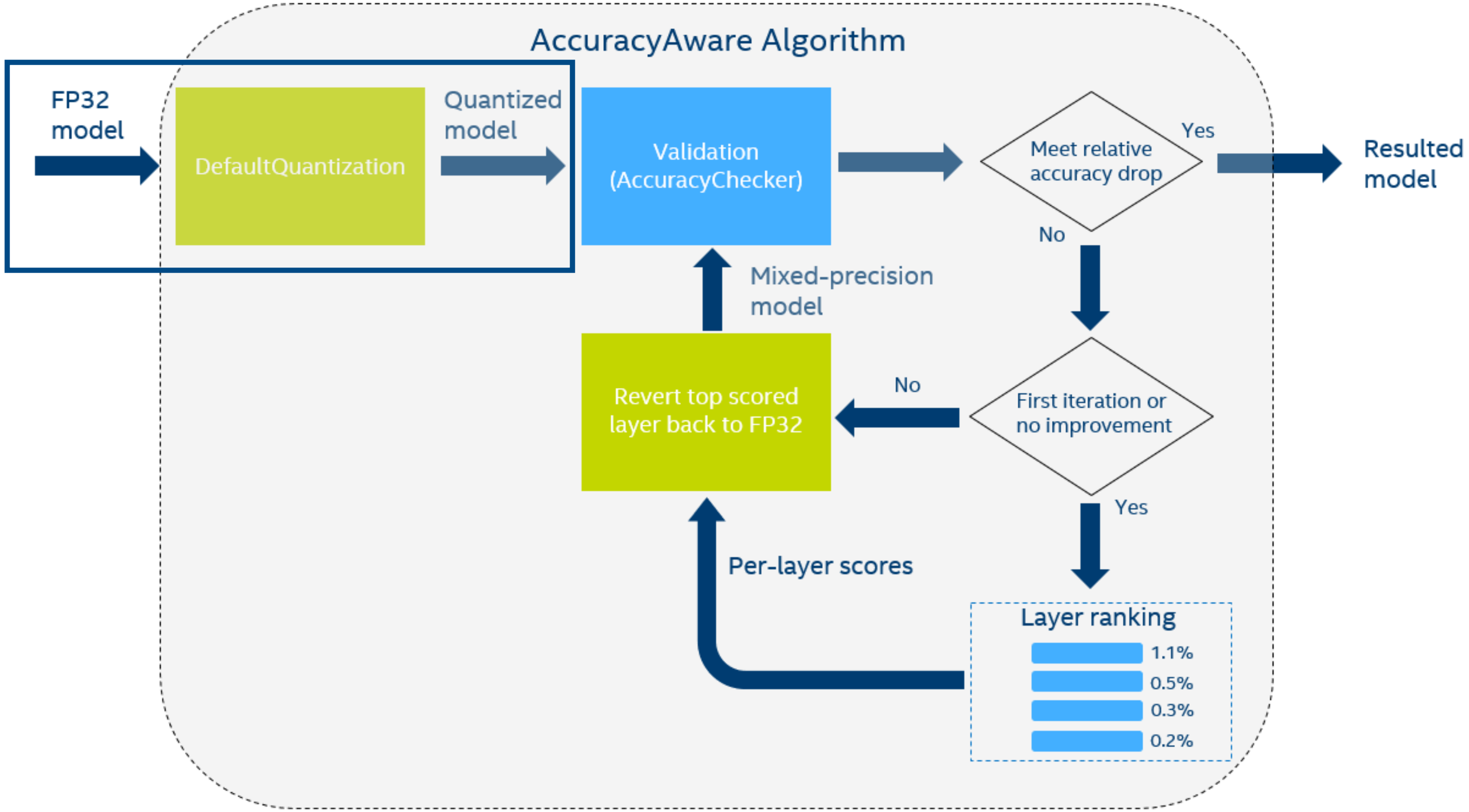
Post-Training Optimization Tool – Best Practices

Try **AccuracyAwareQuantization** Method if not satisfied with previous steps



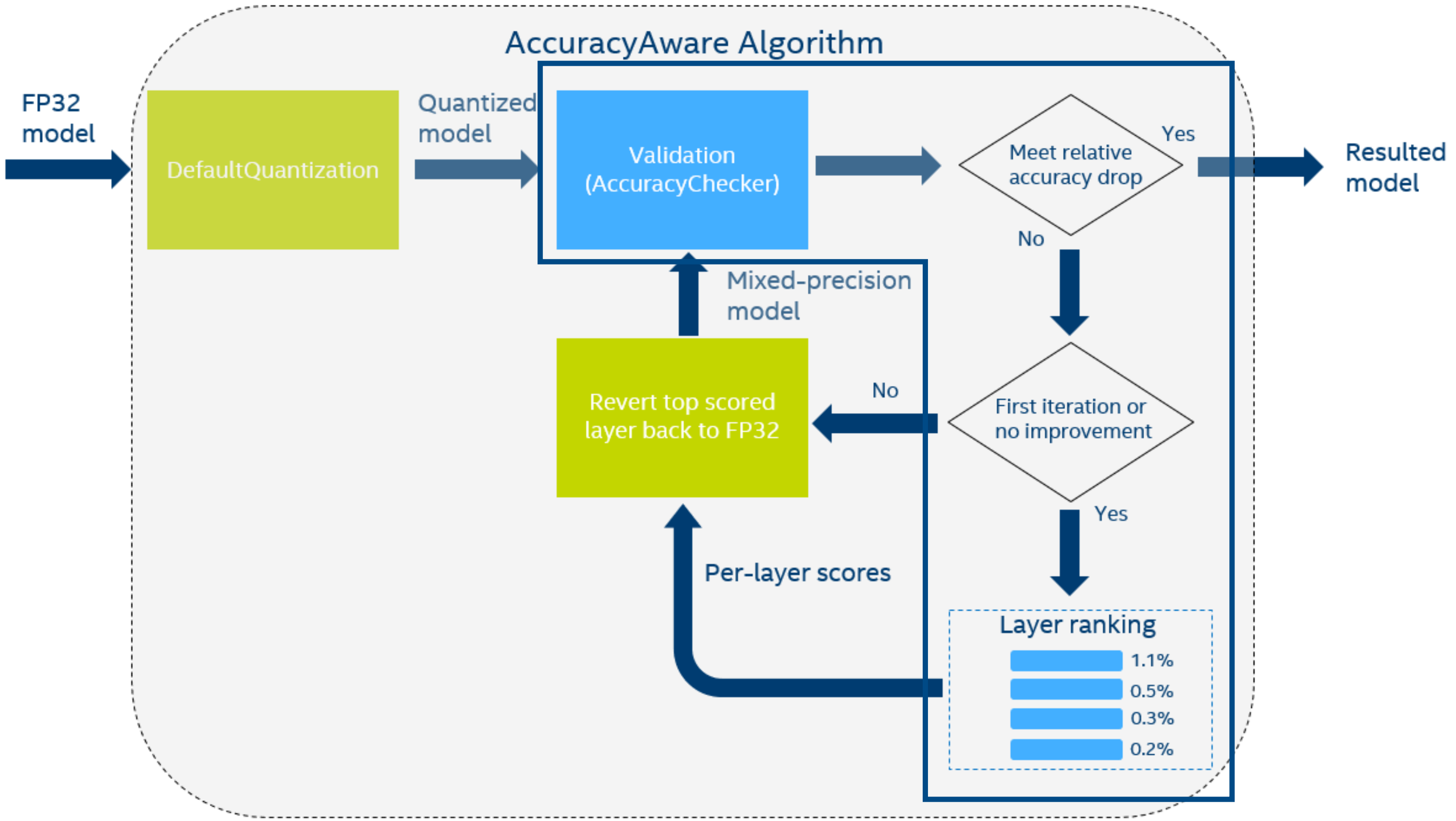
Post-Training Optimization Tool – Best Practices

Try **AccuracyAwareQuantization** Method if not satisfied with previous steps



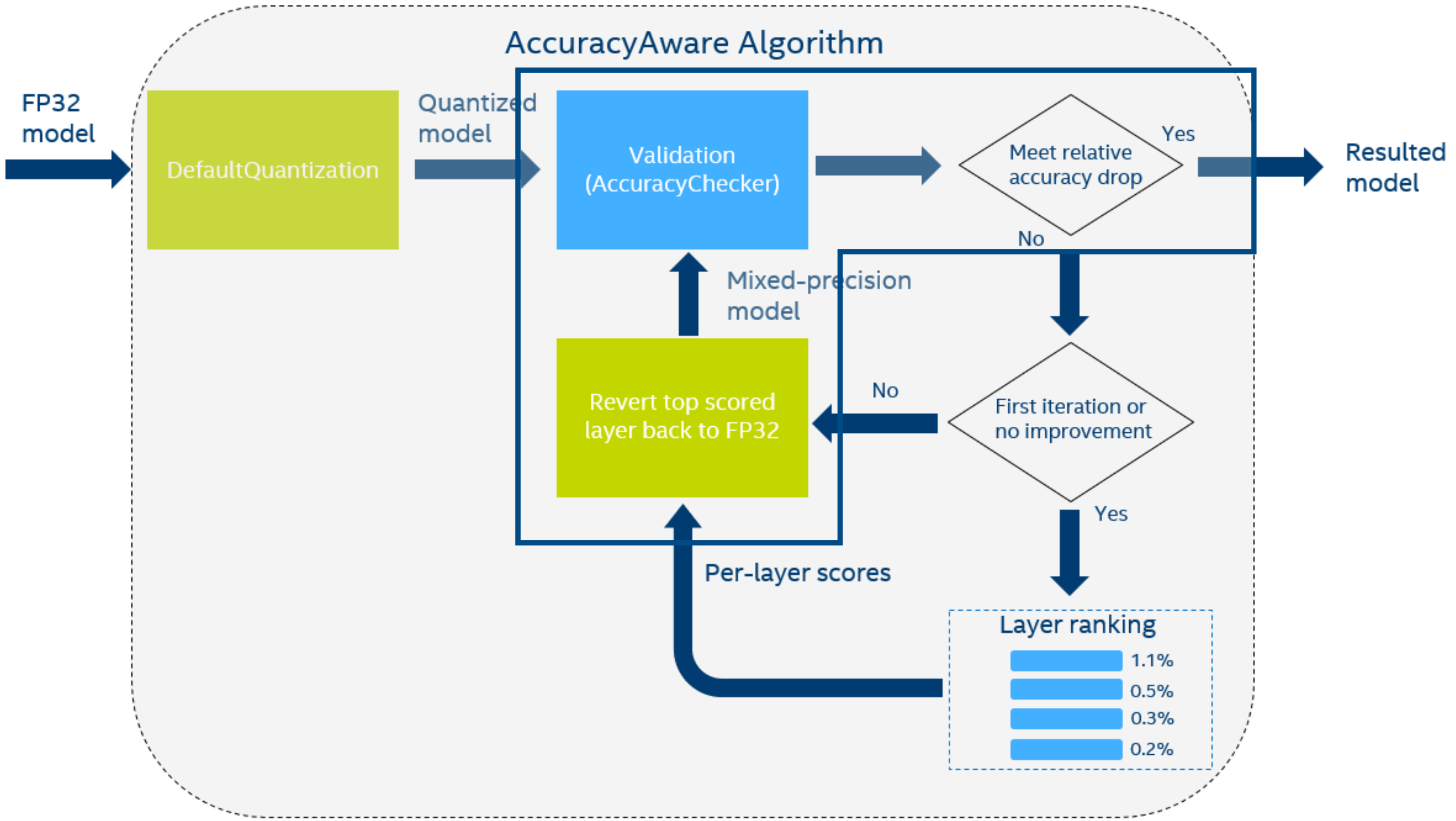
Post-Training Optimization Tool – Best Practices

Try **AccuracyAwareQuantization** Method if not satisfied with previous steps



Post-Training Optimization Tool – Best Practices

Try **AccuracyAwareQuantization** Method if not satisfied with previous steps



Post-Training Optimization Tool – Best Practices

Try **AccuracyAwareQuantization** Method if not satisfied with previous steps

```
{
    /* Model parameters */

    "model": {
        "model_name": "model_name", // Model name
        "model": "<MODEL_PATH>", // Path to model (.xml format)
        "weights": "<PATH_TO_WEIGHTS>" // Path to weights (.bin format)
    },

    /* Parameters of the engine used for model inference */

    "engine": {
        "config": "<CONFIG_PATH>" // Path to Accuracy Checker config
    },

    /* Optimization hyperparameters */

    "compression": {
        "target_device": "ANY", // Target device, the specificity of which will be taken
                                // into account during optimization
        "algorithms": [
            {
                "name": "AccuracyAwareQuantization", // Optimization algorithm name
                "params": {
                    "preset": "performance", // Preset [performance, mixed, accuracy] which control the quantization
                                             // mode (symmetric, mixed (weights symmetric and activations asymmetric)
                                             // and fully asymmetric respectively)

                    "stat_subset_size": 300, // Size of subset to calculate activations statistics that can be used
                                             // for quantization parameters calculation

                    "maximal_drop": 0.01, // Maximum accuracy drop which has to be achieved after the quantization
                    "tune_hyperparams": false // Whether to search the best quantization parameters for model
                }
            }
        ]
    }
```

accuracy_aware_quantization_template.json

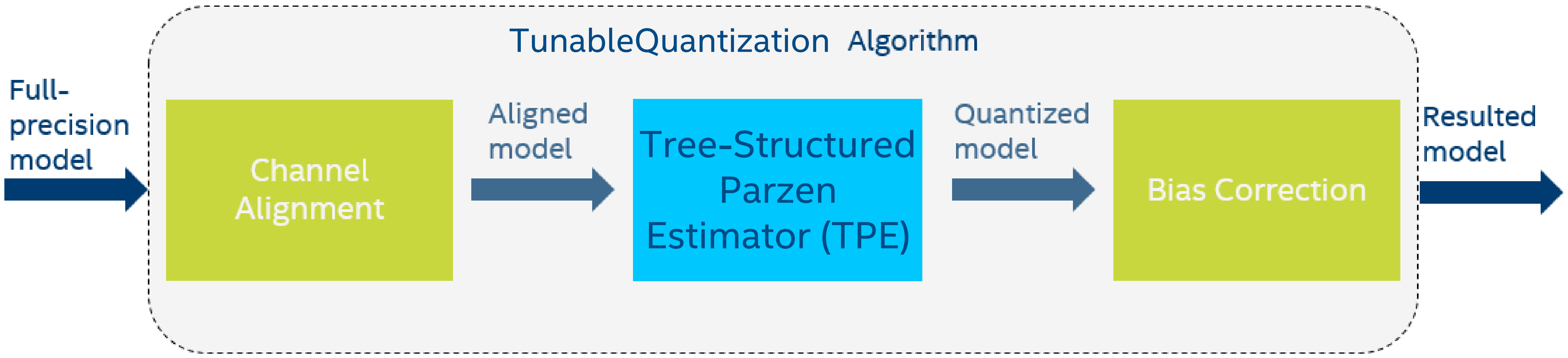
Learn more about AccuracyAwareQuantization :

https://docs.openvinotoolkit.org/latest/pot_compression_algorithms_quantization_accuracy_aware_README.html

Post-Training Optimization Tool – Best Practices

Lastly: **TunableQuantization** with layer-Wise Hyperparameters Tuning Using TPE (**Tree of Parzen Estimators**)

- TunableQuantization algorithm is a modified version (to support hyperparameters setting by **Tree-Structured Parzen Estimator (TPE)**) of the vanilla MinMaxQuantization quantization method that automatically inserts FakeQuantize operations into the model graph based on the specified target hardware and initializes them using statistics collected on the calibration dataset.



Post-Training Optimization Tool – Best Practices

Lastly: **TunableQuantization** with layer-Wise Hyperparameters Tuning Using TPE (**Tree of Parzen Estimators**)

```
    "model": {
        "model_name": "model_name", // Model name
        "model": "<MODEL_PATH>", // Path to a model (.xml format)
        "weights": "<PATH_TO_WEIGHTS>" // Path to weights (.bin format)
    },

    /* Parameters of the engine used for model inference. */

    "engine": {
        "config": "<CONFIG_PATH>" // Path to Accuracy Checker config
    },

    /* Optimizer used to find "optimal" hyperparameters */

    "optimizer": {
        "name": "Tpe", // Global optimizer name
        "params": {
            "max_trials": 200, // Maximum number of trails
            "trials_load_method": "cold_start", // Start from scratch or
            "accuracy_loss": 0.1, // Accuracy threshold (%)
            "latency_reduce": 1.5, // Target latency improvement versus o
            "accuracy_weight": 1.0, // Accuracy weight in loss function
            "latency_weight": 1.0, // Latency weight in loss function
            "benchmark": {
                // Latency measurement benchmark configuration (https://d
                "performance_count": false,
                "batch_size": 0,
                "nthreads": 4,
                "nstreams": 0,
                "nireq": 0,
                "api_type": "sync",
                "niter": 4,
                "duration_seconds": 30,
                "benchmark_app_dir": "<path_to_benchmark_app>" // Path to
reduce jitter in results.
```

```
    "compression": {
        "target_device": "ANY", // Target device, the spec
                                // into account during opt
        "algorithms": [
            {
                "name": "ActivationChannelAlignment",
                "params": {
                    "stat_subset_size": 300 // Size of sub
                                            // for quantiz
                }
            },
            {
                "name": "TunableQuantization",
                "params": {
                    /* Preset is a collection of optimizat
                    to improve which metric the algorithm
                    [performance, mixed, accuracy] presets
                    (symmetric, mixed(weights symmetric an
                    "preset": "performance",
                    "stat_subset_size": 300,    // Size of
                                                // for quan
                    "tuning_scope": ["layer"] // List of c
                                              // available
                }
            },
            {
                "name": "FastBiasCorrection",
                "params": {
                    "stat_subset_size": 300 // Size of sub
                                            // for quantiz
                }
            }
        ]
    }
}
```

Learn more about TunableQuantization :

https://docs.openvinotoolkit.org/latest/pot_compression_algorithms_quantization_tunable_quantization_README.html

Learn more about Tree-Structured Parzen Estimator (TPE) :

https://docs.openvinotoolkit.org/latest/pot_compression_optimization_tpe_README.html

Deep Learning Workbench

July 2021

intel®

Deep Learning Workbench

https://docs.openvinotoolkit.org/latest/workbench_docs_Workbench_DG_Introduction.html

- Web-based, **UI extension tool** of the Intel® Distribution of OpenVINO™ toolkit
- **Visualizes performance data for** topologies and layers to aid in model analysis
- **Automates analysis** for optimal performance configuration (streams, batches, latency)
- **Experiment with INT8 or Winograd calibration** for optimal tuning using the Post Training Optimization Tool
- Provide **accuracy informatio**n through accuracy checker
- **Direct access to models** from public set of Open Model Zoo
- Enables **remote profiling**, allowing the collection of performance data from multiple different machines without any additional set-up.

Development Guide ▸
https://docs.openvinotoolkit.org/latest/_docs_Workbench_DG_Introduction.html

Work with models in Deep Learning Workbench

- Obtain for specific use case
- Explore inference configurations
- Optimize for higher performance
- Measure accuracy
- Visualize model output
- Prepare for deployment

Learn OpenVINO in Deep Learning Workbench

- Analyze performance
- Explore model architecture
- Experiment with OpenVINO API
- Follow full OpenVINO workflow created for your model

Installation Methods

Run the Deep Learning Workbench Locally

This section contains instructions on how to run DL Workbench. Select your options and run the commands on your local machine. Please, ensure that you have met the prerequisites.

General Options

I have Docker installed	Yes		**No**	
My OS	**Linux**	Windows	macOS	
Accelerators on my machine	**CPU**	**GPU**	**NCS2**	HDDL

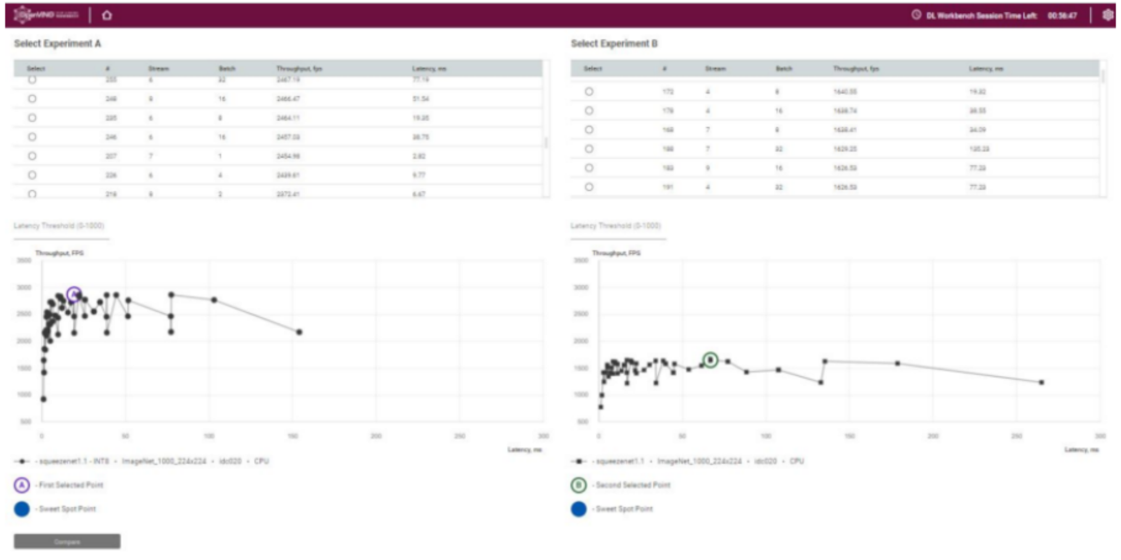
NCS2 and HDDL cannot be used simultaneously

Advanced Options

Start DL Workbench with	**Python wrapper**		Docker command
Use HTTP Proxy	Yes	**No**	Enter your proxy
Use HTTPS Proxy	Yes	**No**	Enter your proxy
Use No Proxy	Yes	**No**	Enter your proxy

Execute / Results

1. Install Docker	Follow instructions on configuring Docker on Linux
2. Install Python Wrapper	python3 -m pip install -U openvino-workbench
3. Start DL Workbench	openvino-workbench --image openvino/workbench:2021.4 --enable-gpu --enable-myriad



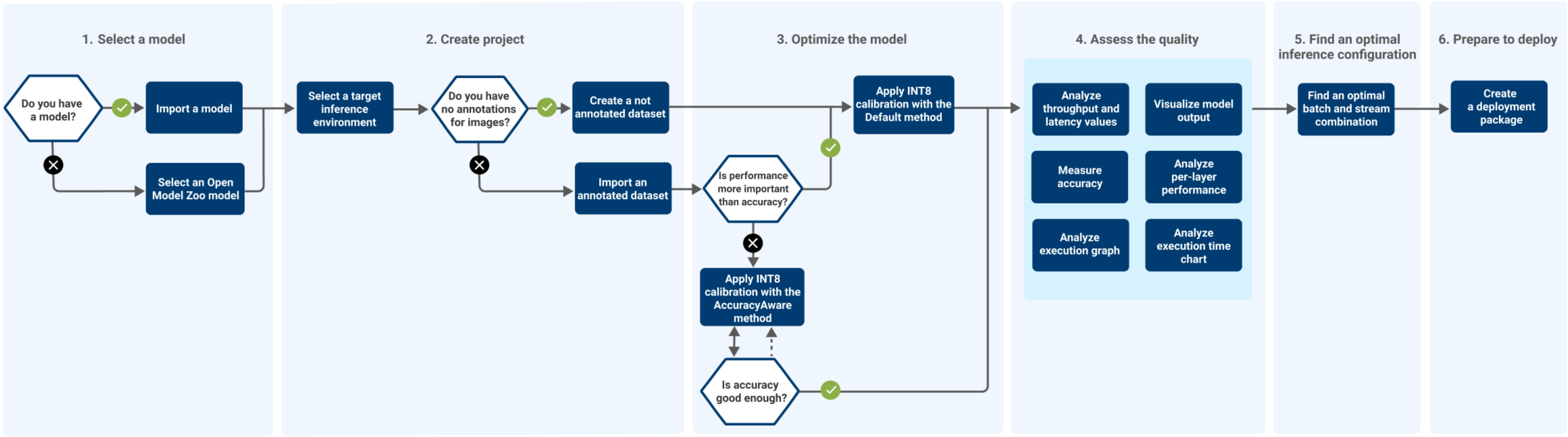
Deep Learning Workbench

The Deep Learning Workbench simplifies using the Intel® Distribution of OpenVINO toolkit to tune, visualize, and compare the performance of deep learning models on Intel® architecture.

Go There →

Note: To get full features of DL Workbench, please run it on local system

Deep Learning Workbench Workflow



DL Workbench Demo

Deep Learning Streamer

April 2021

intel®

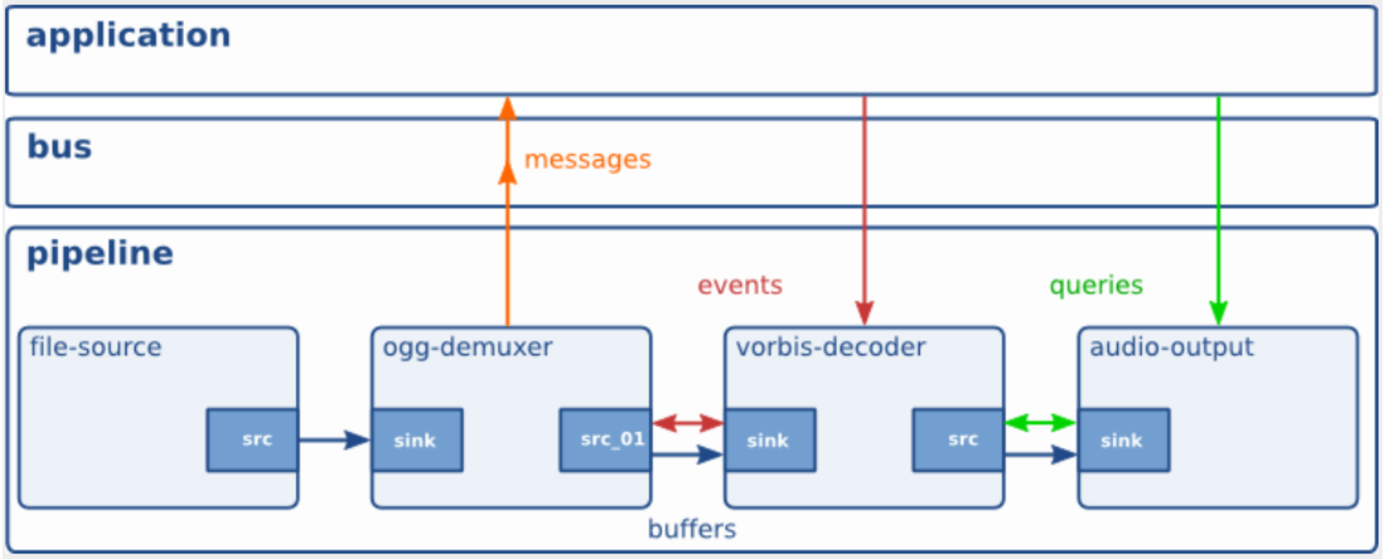
Introducing.. Dl streamer

- Intel® Distribution of OpenVINO™ toolkit Deep Learning (DL) Streamer, now part of the **default installation** package
- Enables developers to **create and deploy optimized streaming media analytics pipelines** across Intel® architecture from edge to cloud
- Optimal pipeline interoperability with a **familiar developer experience** built using the GStreamer multimedia framework

gstreamer + OpenVINO™

What is GStreamer?

- A pipeline consists of **connected processing elements**
- Each element is provided by **a plug-in** and can be **grouped into bins**
- Elements communicate by means of **pads** – source pad and sink pad
- Data buffers flow **from Source** element **to Sink** element & from source pad to sink pad



Ref:
https://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/manual.pdf

Under the hood: DL Streamer

Application

- Reference Application Designs
- GStreamer framework

GStreamer plugins

- GStreamer Media Plugins (Standard)
  - Decode
  - VPP
  - Encode
- DL Streamer - GStreamer Video Analytics (GVA) Plugin
  - Detect
  - Classify
  - Track
  - Publish

Runtime Libraries

- VAAPI
- Libav
- Intel® Distribution of OpenVINO™ toolkit Deep Learning Inference Engine
- OpenCV
- MQTT/ Kafka

Hardware

intel XEON PLATINUM inside™

intel CORE inside™

intel ATOM inside™

intel MOVIDIUS inside™

intel IRIS™ Pro GRAPHICS

Media Processing Pipeline

Video Pipeline – decode, convert, render

filesrc	—	decodebin	—	videoconvert	—	xvimagesink
input		HW/SW decode		convert		render on screen

```
gst-launch-1.0 filesrc location=/path/to/video.mp4 ! decodebin ! videoconvert ! xvimagesink
```

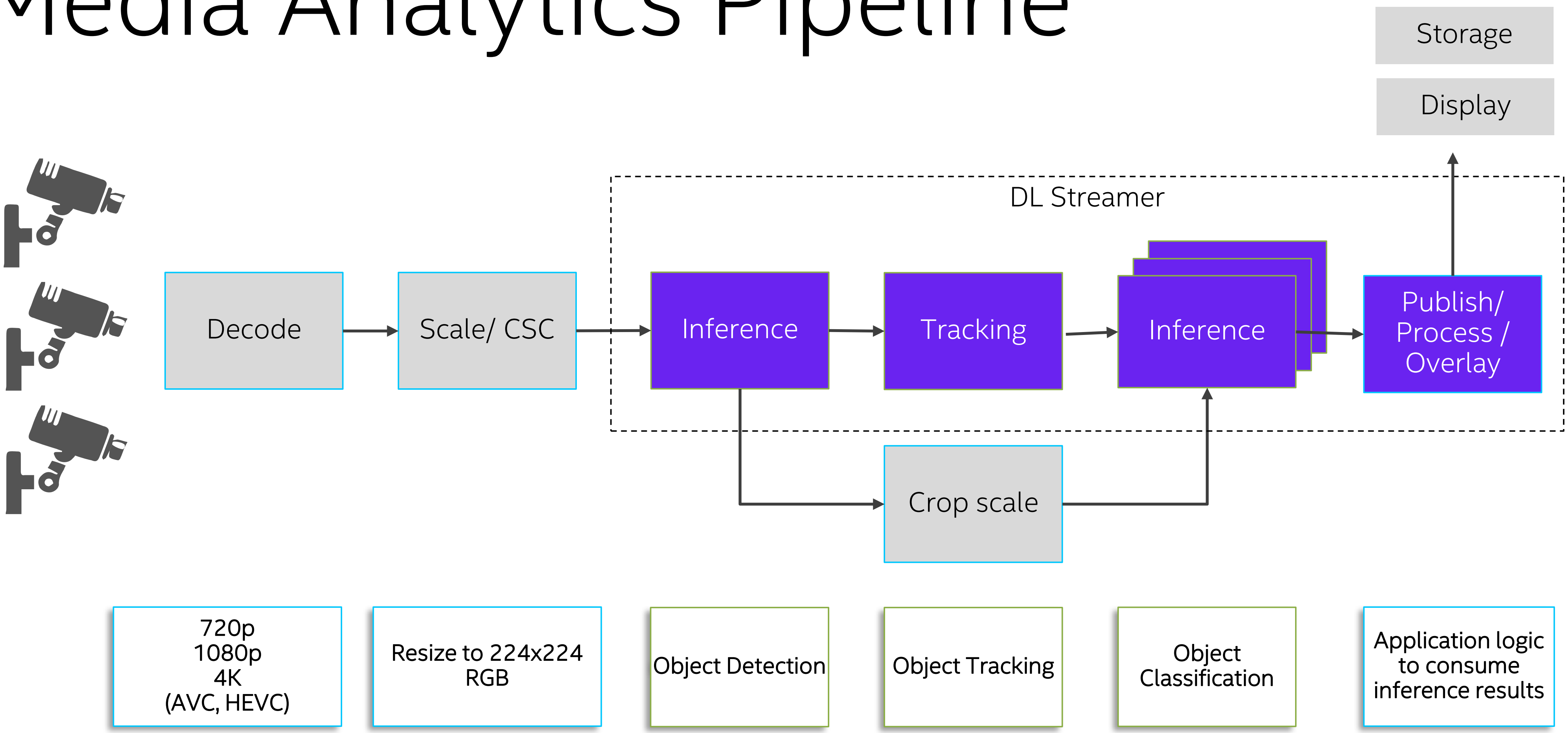
Using the DL Streamer

Video Analytics pipeline – person and vehicle detection, person, vehicle attributes classification

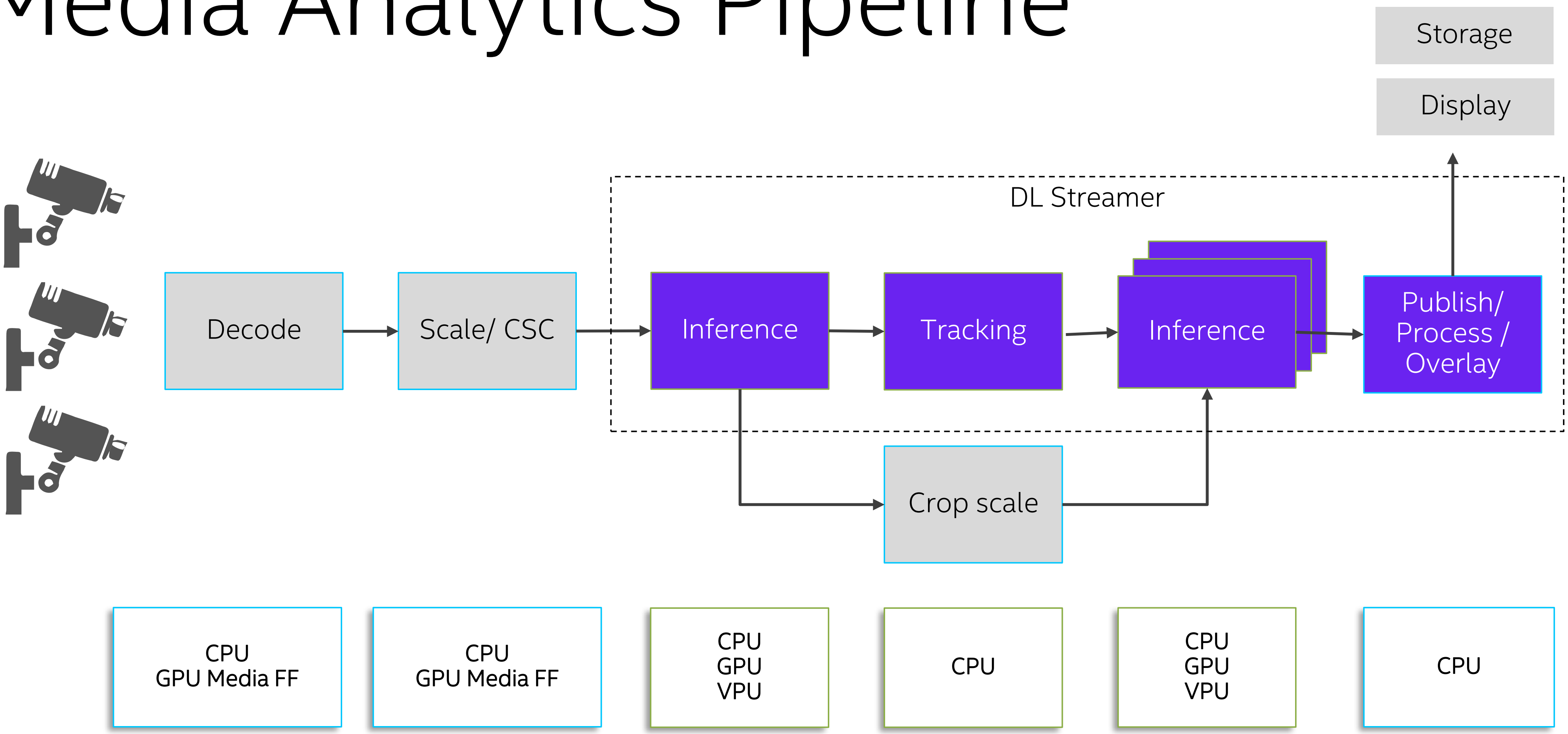
filesrc	decodebin	gvadetect	gvatrack	gvaclassify	gvaclassify	gvawatermark	xvimage sink
Input	HW/SW Decode	Person Vehicle Detection	Object Tracking	Person Attributes Recognition	Vehicle Attributes Recognition	Watermark	Render On Screen


```
 gst-launch-1.0 filesrc location=/path/to/video.mp4 !
 decodebin ! videoconvert ! video/x-raw,format=BGRx ! \
 gvadetect model=person-vehicle-bike-detection-crossroad-0078.xml model-proc=person-vehicle-bike-detection-
crossroad-0078.json inference-interval=10 threshold=0.6 device=CPU ! queue ! \
 gvatrack tracking-type="short-term" ! queue ! \
 gvaclassify model= person-attributes-recognition-crossroad-0230.xml model-proc= person-attributes-recognition-
crossroad-0230.json reclassify-interval=10 device=CPU object-class=person ! queue ! \
 gvaclassify model= vehicle-attributes-recognition-barrier-0039.xml model-proc= vehicle-attributes-recognition-
barrier-0039.json reclassify-interval=10 device=CPU object-class=vehicle ! queue ! \
 gvawatermark ! videoconvert ! fpsdisplaysink video-sink=xvimagesink sync=true
```

Media Analytics Pipeline



Media Analytics Pipeline



Audio Processing

DL Streamer for end-to-end audio analytics pipeline

- Intel® Distribution of OpenVINO™ toolkit Deep Learning (DL) Streamer, part of the default installation package
- Enables developers to create and deploy optimized streaming media analytics pipelines across Intel® architecture from edge to cloud
- Optimal pipeline interoperability with a familiar developer experience built using the GStreamer* multimedia framework
- Introduces gvaaudiodetect for audio event detection
  - Can be paired with alcnet public model for end-to-end audio analytics pipeline

DL Streamer Elements:

- gvaaudiodetect for audio event detection using ACLNet
- gvametaconvert for converting ACLNet detection results into JSON for further processing and display
- gvametapublish for printing detection results to stdout

Resources to Get Started

OpenVINO™

Intel® Distribution of OpenVINO™ Toolkit:

https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html

Intel® Edge Software Hub

Download prevalidated software to learn, develop, and test your solutions for the edge.

Intel® Edge Software Hub:

https://software.intel.com/content/www/us/en/develop/topics/iot/edge-solutions.html

Intel® DevCloud FOR THE EDGE

Intel® DevCloud for the Edge:

https://devcloud.intel.com/edge/home

To get access to the full video series, please complete the short form: http://intel.ly/38B9ix6

intel®