Accelerating Object Detection Throughput in Cloud and Edge Deployments by Optimizing YOLOv7 Model using Intel® Distribution of OpenVINO™ Toolkit

White Paper

July 2023

Author:

Ramesh Perumal
Chun Jieh Sow
Muhammad Nasih Ulwan Abd Wahab
Hoay Tien Teoh

Document number: 746745-1.0

Contents

Tables

Figures

Revision History

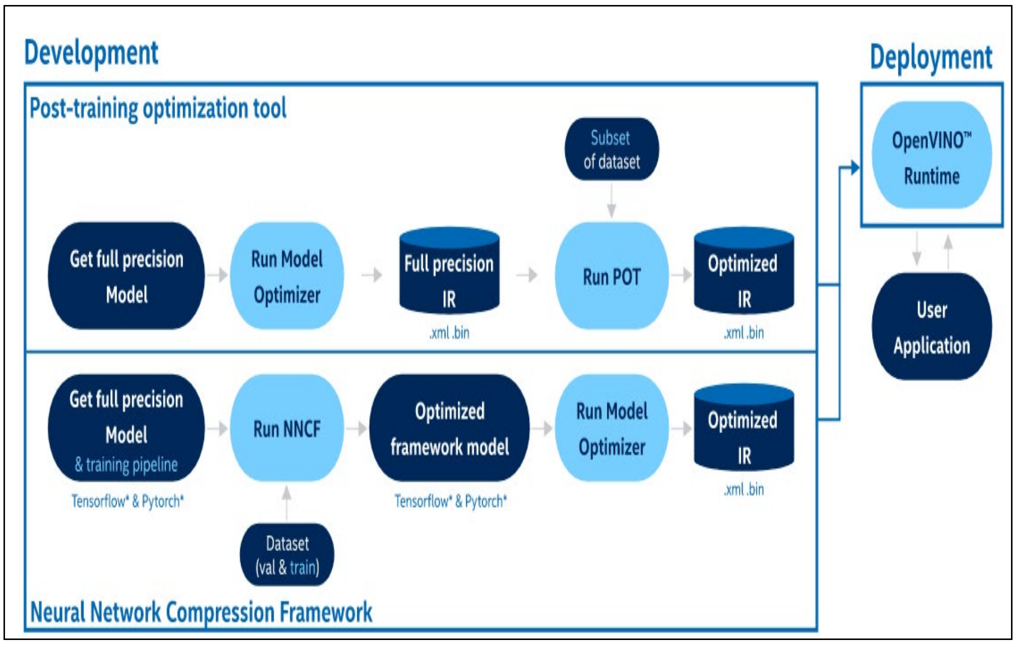
Date	Revision	Description
July 2023	1.0	Initial release

§

1.0 Introduction

This document presents the BKMs for optimizing and quantizing YOLOv7 model using Intel® Distribution of OpenVINO™ Toolkit. The object detection use case based on the optimized YOLOv7 model is evaluated on Intel platforms to demonstrate the improved throughput. Furthermore, the potential cost savings by the optimized model in cloud deployment is illustrated with the real-time use case of an ISV.

Figure 1. Model Optimization Workflow



The raw pre-trained model is converted into the optimized IR model using Model Optimizer that implements most of the optimization parameters to a model by default. It is further optimized by applying special optimization methods, such as quantization, pruning, and preprocessing optimization. Starting from OpenVINO™ 2022.2.0, Neural Network Compression Framework (NNCF) becomes the recommended tool for post-training and training-time optimization methods. Post-training Quantization (or Post-training Optimization Tool (POT) in previous versions of OpenVINO™) is designed to accelerate the inference of models by converting them into a more hardware-friendly representation (INT8) by applying specific methods that do not require re-training. It is limited in terms of achievable accuracy-performance trade-off for optimizing models. To overcome this, training-time optimization may give better results with methods, like Quantization-aware Training and Filter Pruning. This paper demonstrates the increased throughput of the object detection use case with the quantized YOLOv7 model in INT8 precision using post-training quantization.

1.1 Acronyms

Table 1. Acronyms

Term	Description
BKM	Best Known Method
OpenVINO™	Open Visual Inference & Neural Network Optimization
NNCF	Neural Network Compression Framework
POT	Post-training Optimization Tool
ISV	Independent Software Vendor
DUT	Device Under Test
8-bit Integer	INT8
16-bit Floating Point	FP16
32-bit Floating Point	FP32
FPS	Frames per second

1.2 Reference Documents

Log in to the Resource and Documentation Center (rdc.intel.com) to search and download the document numbers listed in the following table. Contact your Intel field representative for access.

Note: Third-party links are provided as a reference only. Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether the referenced data is accurate.

Table 2. Reference Document

Document	Document No./Location
OpenVINO™ Toolkit	https://software.seek.intel.com/openvino-toolkit
OpenVINO™ Docker Image	https://hub.docker.com/r/openvino/ubuntu20_dev
YOLOv7 GitHub Repository	https://github.com/WongKinYiu/yolov7
OpenVINO™ Notebook for Optimization and Quantization of YOLOv7 model	https://github.com/openvinotoolkit/openvino_notebooks/blob/main/notebooks/226-yolov7-optimization/226-yolov7-optimization.ipynb
Model Optimization Guide	https://docs.openvino.ai/latest/openvino_docs_model_optimization_guide.html
numactl	https://manpages.ubuntu.com/manpages/trusty/man8/numactl.8.html
taskset	https://manpages.ubuntu.com/manpages/jammy/man1/taskset.1.html
OpenVINO™ CPU plugin properties	https://docs.openvino.ai/2023.0/openvino_docs_OV_UG_supported_plugins_CPU.html
AWS EC2 Xeon Instances	https://aws.amazon.com/ec2/instance-types/m6i/
AWS EC2 Pricing	https://aws.amazon.com/emr/pricing/
PuTTY	https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
WinSCP	https://winscp.net/eng/index.php

Table 3. Devices Under Test

DUT-1: Core	Model	NUC11TNHv5
	CPU	11th Gen Intel® Core™ i5-1145G7 x 8
	GPU	Intel® Iris® Xe Graphics
	Memory	16 GB
	OS	Ubuntu 20.04 LTS
	Docker	20.10.16
	OpenVINO™	2022.2.0
	NNCF	2.4.0
DUT-2: Celeron	Model	iBOX-6305E
	CPU	Intel® Celeron® 6305E CPU x 2
	GPU	Intel® UHD Graphics
	Memory	16 GB
	OS	Ubuntu 20.04 LTS
	OpenVINO™	2022.2.0
DUT-3: 3rd Gen Xeon	CPU	Intel® Xeon® Silver 4316 CPU x 80
	Memory	256 GB
	OS	Ubuntu 20.04 LTS
	OpenVINO™	2022.2.0
DUT-4: AWS EC2 (2nd Gen Xeon)	Model	c5.2xlarge
	CPU	Intel® Xeon® Platinum 8275CL CPU x 8
	Memory	16 GB
	OS	Ubuntu 20.04 LTS
	OpenVINO™	2022.2.0

§

2.0 Model Optimization and Quantization

2.1 Prerequisites

a. Create a Python* virtual environment and upgrade the pip version in DUT-1

```
python -m venv yolov7_venv
source yolov7_venv/bin/activate
python -m pip install --upgrade pip
```

b. Clone the official YOLOv7 GitHub repository and install the dependencies

```
git clone https://github.com/WongKinYiu/yolov7.git
cd yolov7
pip install -r requirements.txt
pip install coremltools onnx onnx-simplifier onnxruntime
jupyterlab
```

c. Install OpenVINO™ and NNCF for model optimization and quantization

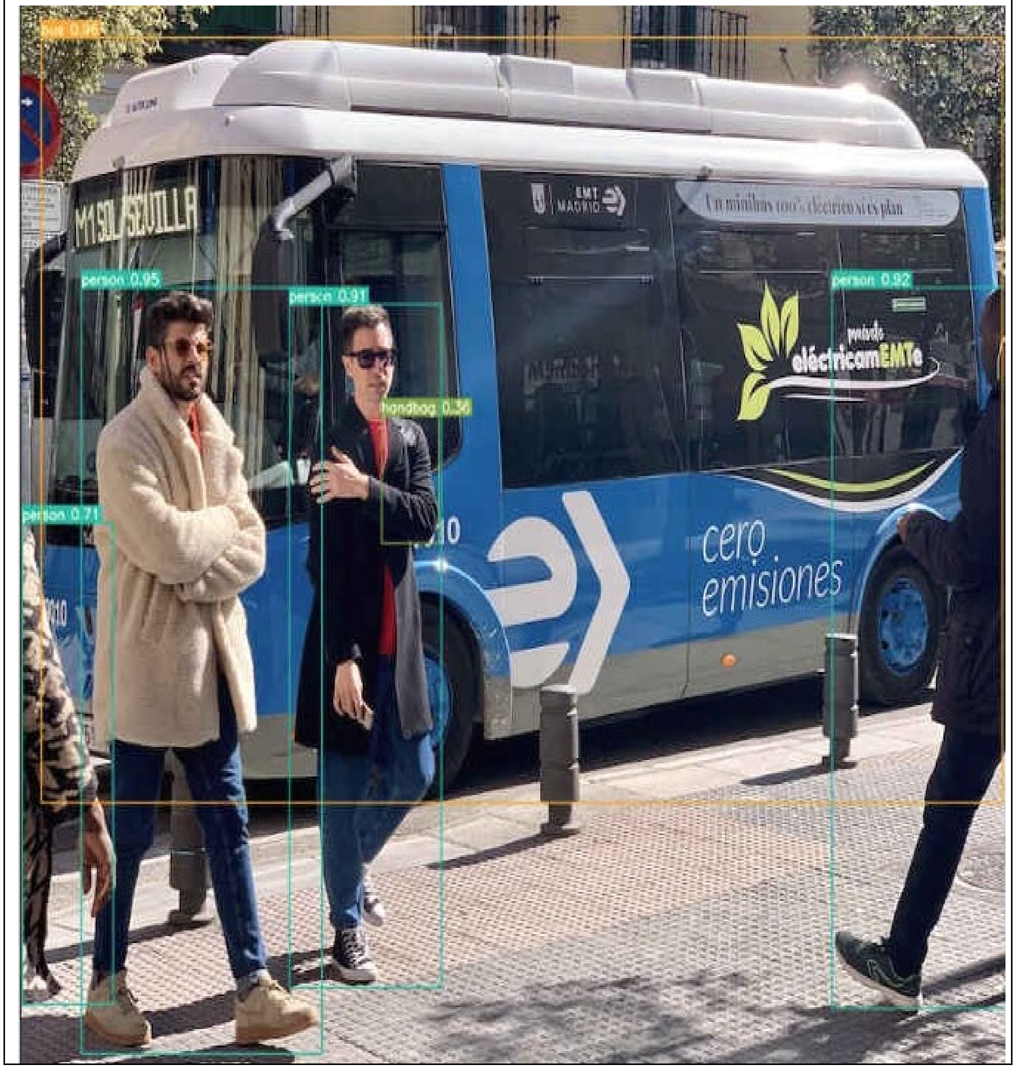
```
pip install openvino-dev==2022.2.0 nncf
```

d. Verify the object detection results of the pre-trained YOLOv7 model in PyTorch format (yolov7.pt) using the existing script detect.py in the YOLOv7 repository

```
python detect.py --weights yolov7.pt --conf 0.25 --img-size
384 --source ./inference/images/bus.jpg
```

The inference time of the detection results shown in Figure 2 is **211 ms** on CPU in DUT-1.

Figure 2. Detection Results of YOLOv7 Model in PyTorch Framework



2.2 Model Optimization

a. Export the YOLOv7 model from PyTorch into ONNX format

```
python export.py --weights yolov7.pt --grid --end2end --
simplify --topk-all 100 --iou-thres 0.65 --conf-thres 0.35
--img-size 384 384 --max-wh 384
```

Using the *end2end* parameter in the above exports the full model to ONNX including post-processing to achieve more performant results. The input image size (*img-size*) is changed from 640 (default) to 384 as the latter is found to satisfy the target performance metrics of the ISV. The resulting ONNX model is saved as yolov7.onnx in the current working directory.

b. Convert the ONNX model into IR format with FP32 precision using Model Optimizer

```
mo --input_model yolov7.onnx --data_type FP32 --output_dir
./FP32
```

c. Verify the object detection results of the optimized YOLOv7 model in IR format (yolov7.xml) using the customized script *main.py* as shown below

```
import sys
import numpy as np
import random
import cv2
from openvino.runtime import Core
import time

names = ['person', 'bicycle', 'car', 'motorcycle',
'airplane', 'bus', 'train', 'truck', 'boat', 'traffic
light',
'fire hydrant', 'stop sign', 'parking meter', 'bench',
'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
'elephant', 'bear', 'zebra', 'giraffe', 'backpack',
'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat',
'baseball glove', 'skateboard', 'surfboard',
'tennis racket', 'bottle', 'wine glass', 'cup', 'fork',
'knife', 'spoon', 'bowl', 'banana', 'apple',
'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog',
'pizza', 'donut', 'cake', 'chair', 'couch',
'potted plant', 'bed', 'dining table', 'toilet', 'tv',
'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
'microwave', 'oven', 'toaster', 'sink', 'refrigerator',
'book', 'clock', 'vase', 'scissors', 'teddy bear',
'hair drier', 'toothbrush']
```

```
def letterbox(im, new_shape=(384, 384), color=(114, 114, 114),
              auto=True, scaleup=True, stride=32):

    # Resize and pad image while meeting stride-multiple
    # constraints
    shape = im.shape[:2]  # current shape [height, width]
    if isinstance(new_shape, int):
        new_shape = (new_shape, new_shape)

    # Scale ratio (new / old)
    r = min(new_shape[0] / shape[0], new_shape[1] / shape[1])
    # only scale down, do not scale up (for better val mAP)
    if not scaleup:
        r = min(r, 1.0)

    # Compute padding
    new_unpad = int(round(shape[1] * r)), int(round(shape[0] * r))
    # wh padding
    dw, dh = new_shape[1] - new_unpad[0], new_shape[0] -
         new_unpad[1]

    if auto:  # minimum rectangle
        # wh padding
        dw, dh = np.mod(dw, stride), np.mod(dh, stride)

    dw /= 2  # divide padding into 2 sides
    dh /= 2
    if shape[::-1] != new_unpad:  # resize
        im = cv2.resize(im, new_unpad, interpolation=
              cv2.INTER_LINEAR)
    top, bottom = int(round(dh - 0.1)), int(round(dh + 0.1))
    left, right = int(round(dw - 0.1)), int(round(dw + 0.1))
    im = cv2.copyMakeBorder(im, top, bottom, left, right,
          cv2.BORDER_CONSTANT, value=color)  # add border
    return im, r, (dw, dh)
```

```
def detect_with_ir_model():
    core = Core()
    colors = {name: [random.randint(0, 255) for _ in range(3)] for i,
     name in enumerate(names)}

    model_path = str(sys.argv[1])
    compiled_model = core.compile_model(model_path, 'CPU')
    input_layer_ir = compiled_model.input(0)
    N, C, H, W = input_layer_ir.shape
    iname = input_layer_ir.any_name

    img_path = str(sys.argv[2])
    img = cv2.imread(img_path)
    image = img.copy()
    image, ratio, dwdh = letterbox(image, (H,W), auto=False)
    image = image.transpose((2, 0, 1))
    image = np.expand_dims(image, 0)
    image = np.ascontiguousarray(image)
    im = image.astype(np.float32)
    im /= 255
    inp = {iname: im}

    start = round(time.time() * 1000)
    ov_outputs = compiled_model(inp)[compiled_model.output(0)]
    end = round(time.time() * 1000)

    print(f' Inference time = {end - start} ms')

    ori_images = [img.copy()]

    for i, (batch_id, x0, y0, x1, y1, cls_id, score) in
     enumerate(ov_outputs):
        if (score != 0):
            image = ori_images[int(batch_id)]
            box = np.array([x0, y0, x1, y1])
            box -= np.array(dwdh * 2)
            box /= ratio
            box = box.round().astype(np.int32).tolist()
            print(box)
            cls_id = int(cls_id)
            score = round(float(score), 3)
            name = names[cls_id]
            color = colors[name]
            name += ' ' + str(score)
            cv2.rectangle(image, box[:2], box[2:], color, 2)
            cv2.putText(image, name, (box[0], box[1] - 2),
             cv2.FONT_HERSHEY_SIMPLEX, 0.75, [225, 255, 255],
             thickness=2)
            print(name)

    cv2.imshow("out", ori_images[0])
    cv2.waitKey(0)
if __name__ == '__main__':
    detect_with_ir_model()
```

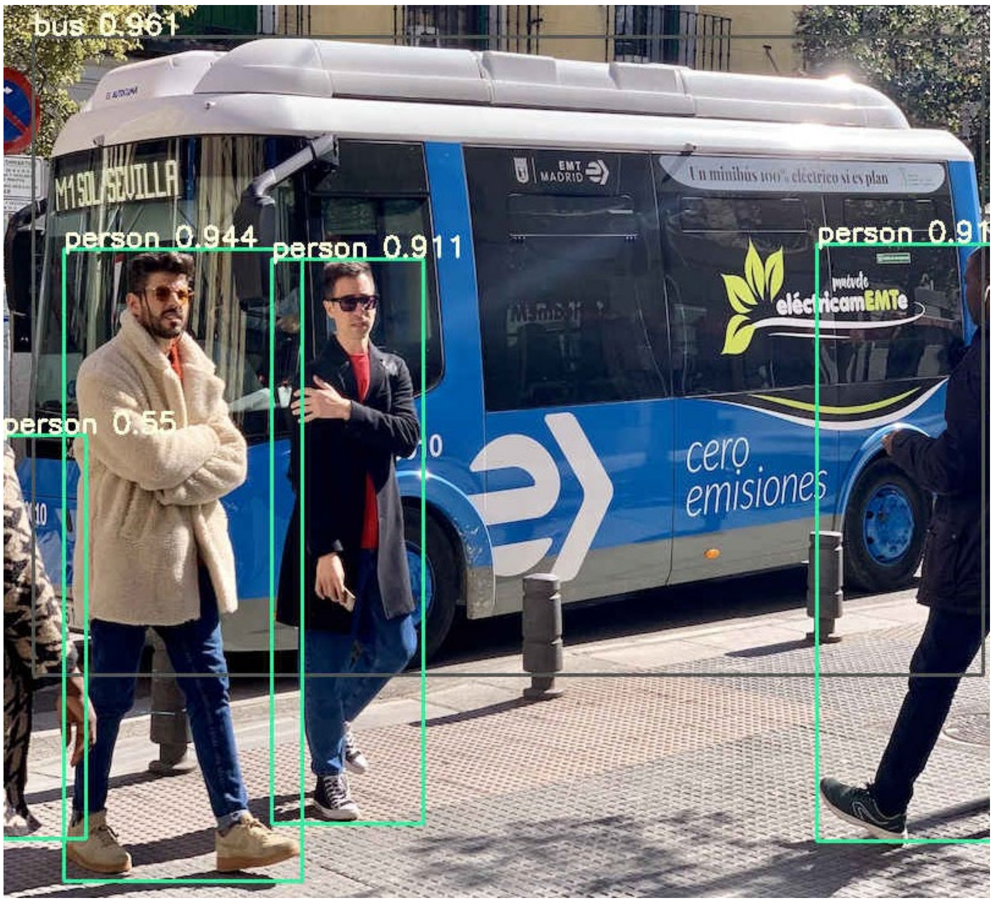
Run the script *main.py* to verify the detection results of the optimized model

```
python3 main.py FP32/yolov7.xml inference/images/bus.jpg
```

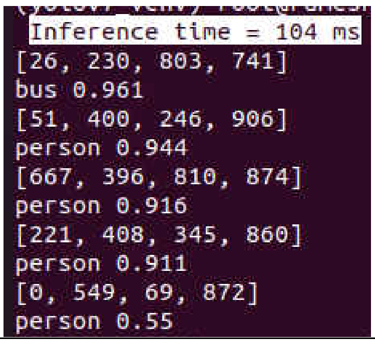
The inference time of the detection results shown in Figure 3 is **104 ms** on CPU in DUT-1. Thus, the optimized model with FP32 precision reduces the inference time of the PyTorch model (211 ms) by **51%**.

Figure 3. Detection Results of the Optimized YOLOv7 Model with FP32 Precision

A



B



```
Inference time = 104 ms
[26, 230, 803, 741]
bus 0.961
[51, 400, 246, 906]
person 0.944
[667, 396, 810, 874]
person 0.916
[221, 408, 345, 860]
person 0.911
[0, 549, 69, 872]
person 0.55
```

2.3 Model Quantization

NNCF provides a suite of advanced algorithms for inference optimization in OpenVINO™ with minimal accuracy drop. Post-training Quantization is a quantization algorithm that doesn't demand retraining of a quantized model. It utilizes a small subset of the initial dataset to calibrate quantization constants. The post-training quantization is integrated into NNCF in OpenVINO™ 2022.2.0 and later versions. The model quantization workflow involves the following steps:

i. Create a *Dataset* for quantization.

ii. Run *nncf.quantize* for getting a quantized model with INT8 precision.

iii. Serialize an OpenVINO™ IR model, using the *openvino.runtime.serialize* function

a. Move to the *yolov7* directory containing the *utils* subdirectory to execute the following code snippets to create the validation *dataloader* and *transform* function.

```
import nncf
import numpy as np
from collections import namedtuple
import yaml
from utils.datasets import create_dataloader
from utils.general import check_dataset, box_iou, xywh2xyxy,
colorstr

# read dataset config
DATA_CONFIG = 'data/coco128.yaml'
with open(DATA_CONFIG) as f:
    data = yaml.load(f, Loader=yaml.SafeLoader)

# Dataloader
TASK = 'val'  # path to train/val/test images
Option = namedtuple('Options', ['single_cls'])
opt = Option(False)
dataloader = create_dataloader(
    data[TASK], 384, 1, 32, opt, pad=0.5,
    prefix=colorstr(f'{TASK}: ')
)[0]

def prepare_input_tensor(image: np.ndarray):
    input_tensor = image.astype(np.float32)  # uint8 to fp16/32
    input_tensor /= 255.0  # 0 - 255 to 0.0 - 1.0

    if input_tensor.ndim == 3:
        input_tensor = np.expand_dims(input_tensor, 0)
    return input_tensor

def transform_fn(data_item):
    img = data_item[0].numpy()
    input_tensor = prepare_input_tensor(img)
    return input_tensor

quantization_dataset = nncf.Dataset(dataloader, transform_fn)
```

Create the config file *coco128.yaml* with the COCO validation dataset path to create dataloader using *create_dataloader* method with the size of **384** instead of its default value (640).

b. Run the following snippet to create and save the quantized model.

```
from openvino.runtime import Core
from openvino.runtime import serialize

core = Core()
model = core.read_model('FP32/yolov7.xml')

quantized_model = nncf.quantize(model,
quantization_dataset, preset=nncf.QuantizationPreset.MIXED)

serialize(quantized_model, 'INT8_test/yolov7_int8.xml')
```

Note: The above mentioned quantization steps are adapted from the OpenVINO notebook that also includes the validation results of the quantized model.

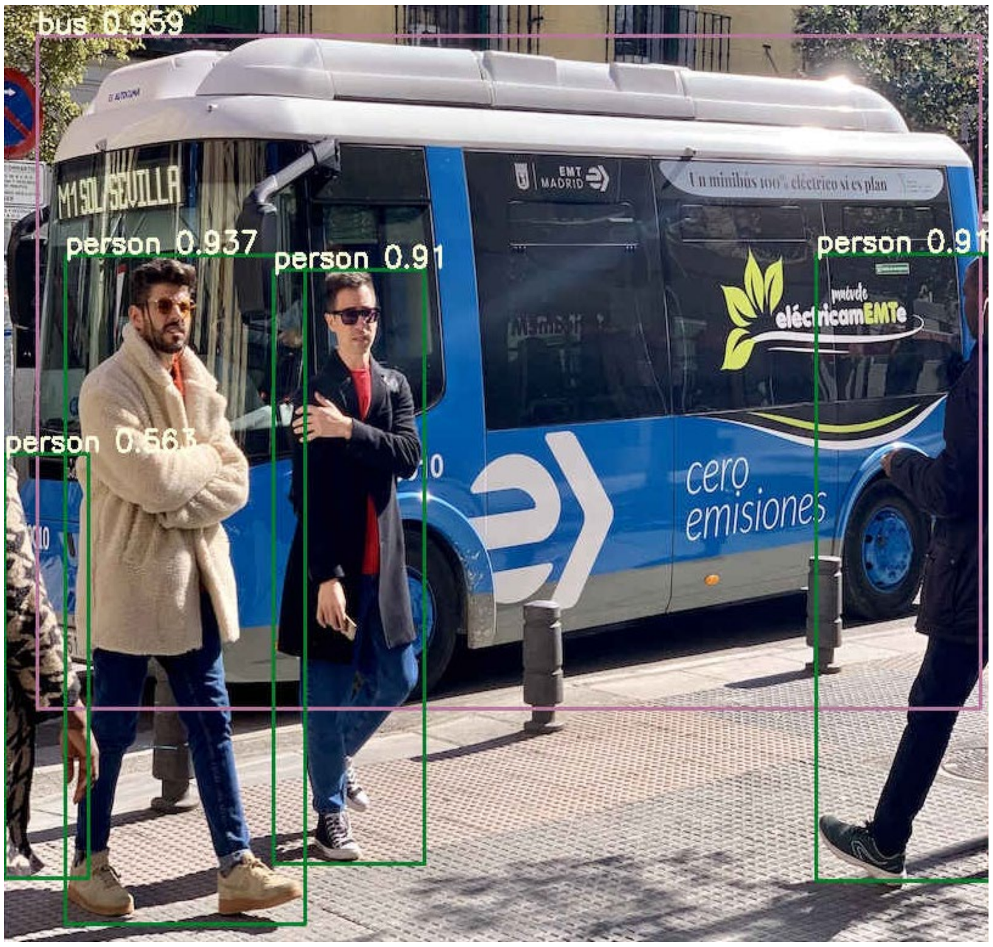
c. Run the script *main.py* to verify the detection results of the quantized model

```
python3 main.py INT8/yolov7.xml inference/images/bus.jpg
```

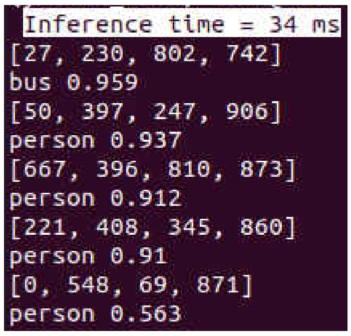
The inference time of the detection results shown in Figure 4 is **34 ms** on CPU in DUT-1. Thus, the quantized model with INT8 precision reduces the inference time of the raw PyTorch (211 ms) and the optimized FP32 (104 ms) models by **84%** and **67%**, respectively.

Figure 4. Detection Results of the Quantized YOLOv7 Model with INT8 Precision

A



B



```
Inference time = 34 ms
[27, 230, 802, 742]
bus 0.959
[50, 397, 247, 906]
person 0.937
[667, 396, 810, 873]
person 0.912
[221, 408, 345, 860]
person 0.91
[0, 548, 69, 871]
person 0.563
```

§

3.0 *Accelerating Throughput of Real-Time Use Case with Quantized YOLOv7 Model*

3.1 Use Case

Smart fleet management involves the use of AI-based algorithms for monitoring the vehicle dashboard camera to ensure driver safety by detecting the use of mobile phone, spectacle, and seat belt while driving. An ISV is currently running this object detection use case with the unoptimized YOLOv7 model at **25 FPS** on AWS SageMaker (Nvidia Jetson Nano GPU) at the price of **0.75 USD$/hour**. The objective of the ISV is to achieve the throughput of ≥25 FPS at lower price on Intel platforms by optimizing the YOLOv7 model using OpenVINO^{TM}. Follow the steps in Section 2.0 to quantize the pre-trained YOLOv7 model provided by the ISV. To verify the inference results with *main.py* in Section 2.2c, replace the value of *names* variable with the class names used by the ISV.

3.2 Deploying Object Detection on AWS EC2

Refer to the steps in **Appendix A** to establish the SSH connection and file transfer from the Windows notebook in Intel network to the remote AWS EC2 instance. The deployment of object detection based on the optimized YOLOv7 model is simple as it involves the installation of a very few modules as shown below.

a. Install the required dependencies

```
sudo apt update
sudo apt install -y python3-pip python3-venv python3-tk
libgl1
```

b. Create the Python* virtual environment to install OpenVINO™ and OpenCV

```
python -m venv deploy_venv
source deploy_venv/bin/activate
pip install --upgrade pip
pip install opencv-python openvino==2022.2.0
```

c. Verify the detection results of the optimized YOLOv7 model

```
python3 main.py INT8/yolov7.xml test.jpg
```

The step b is used to install OpenVINO™ on the Xeon-based edge device (DUT-3).

3.3 Deploying Object Detection on iBOX-6305E

The OpenVINO™ is installed using docker as it eases the method of executing inference on iGPU in the Celeron-based edge device (DUT-2).

a. Install Docker

```
sudo apt install docker.io
```

b. Create the OpenVINO™ container using this docker image

```
docker run -it --rm --name openvino_test -v /tmp/.X11-
unix:/tmp/.X11-unix -e DISPLAY="$DISPLAY" -v
/home/ubuntu:/home/openvino --device /dev/dri:/dev/dri --
group-add="$(stat -c "%g" /dev/dri/render*)"
openvino/ubuntu20_dev:2022.2.0
```

c. Verify the detection results of the optimized YOLOv7 model

```
apt update
python3 main.py INT8/yolov7.xml test.jpg
```

3.4 Performance Evaluation

The performance of object detection is evaluated on DUT 1-3 with the Python script *main_video.py* using the 15-second-long input video containing 394 frames. The optimized YOLOv7 model with INT8 precision and the input shape of 384x384 is used in the evaluation. The execution time (in seconds) and throughput (in FPS) are presented for the three DUTs in Table 4. For the Xeon-based DUT, the performance is evaluated only on the selected number of CPUs and the execution time is verified using the following three methods: a) numactl, b) taskset and c) setting the inference threads in OpenVINO CPU plugin properties. The corresponding commands used in these methods are shown below:

a) Using numactl

The CPUs (0-3 for 4 CPUs and 0-7 for 8 CPUs) to be used for executing the object detection are set using the parameter C in numactl.

```
numactl -C 0-3 python3 main_video.py
numactl -C 0-7 python3 main video.py
```

b) Using taskset

The CPUs (0-3 for 4 CPUs and 0-7 for 8 CPUs) to be used for executing the object detection are set using the parameter *c* in taskset.

```
taskset -c 0-3 python3 main_video.py
taskset -c 0-7 python3 main video.py
```

c) Setting the inference threads in OpenVINO CPU plugin properties

The number of inference threads is set as 4 (replace 4 with 8 for executing the inference script on 8 CPUs) in the OpenVNIO CPU plugin properties using the following steps before compiling the model.

```
from openvino.runtime import Core
core = Core()
core.set_properties({‘INFERENCE_NUM_THREADS’:4})
```

In this method, the Python script is executed in the conventional way as follows:

```
python3 main_video.py
```

The values of execution time for the third (Xeon (4 CPUs)) and fourth (Xeon (8 CPUs)) rows in Table 4 are verified using the above three methods.

Table 4. Performance of object detection with the optimized YOLOv7 model in DUT

DUT	Inference Device	Execution Time* (s)	Throughput (FPS)
NUC11TNHv5	CPU	15.6	25.3
iBOX-6305E	GPU	15.5	25.4
Xeon (4 CPUs)	CPU	15.6	25.3
Xeon (8 CPUs)	CPU	9.6	41

*The execution time is the time taken to process and execute the inference on the 15-second-long input video. Its value may vary depending on the software/hardware changes in the test device.

Table 5. Estimated cost savings with the recommended AWS EC2 Instances based on 3rd Gen Xeon CPU

AWS EC2 Instance	CPUs	Memory (GB)	Price (USD$/hour)	Cost Savings* (%)
m6i.xlarge	4	16	0.192	74
m6i.2xlarge	8	32	0.384	49

*Cost savings (%) is computed relative to the price of AWS SageMaker

From Table 4, it is evident that NUC11TNHv5 and iBOX-6305E are suitable for edge deployment. On the other hand, the 3rd Gen Xeon CPU is suitable for the cloud deployment and the recommended AWS EC2 instances are m6i.xlarge and m6i.2xlarge. According to the AWS pricing list, the price of m6i.xlarge and m6i.2xlarge is 0.192 USD$/hour and 0.384 USD$/hour, respectively. Refer to Table 5 for the specifications of the recommended AWS EC2 instances and the corresponding cost savings. Therefore, the recommended AWS EC2 instances could result in **49-74%** cost savings compared to AWS SageMaker, while achieving the target throughput of **≥25 FPS**.

§

4.0 Conclusion

This white paper presents the methods to optimize and quantize the YOLOv7 model using OpenVINO™ toolkit. This also covers the steps to install OpenVINO™ and verify the inference execution on the AWS EC2 instance. The real-time object detection use case of an ISV is demonstrated to achieve the target throughput (≥25 FPS) with the quantized YOLOv7 model on Intel platforms. The performance evaluation results further reveal that the recommended AWS EC2 instances could help this ISV to reduce the cloud cost by **49-74%** compared to their existing Nvidia GPU-based AWS SageMaker instance.

§

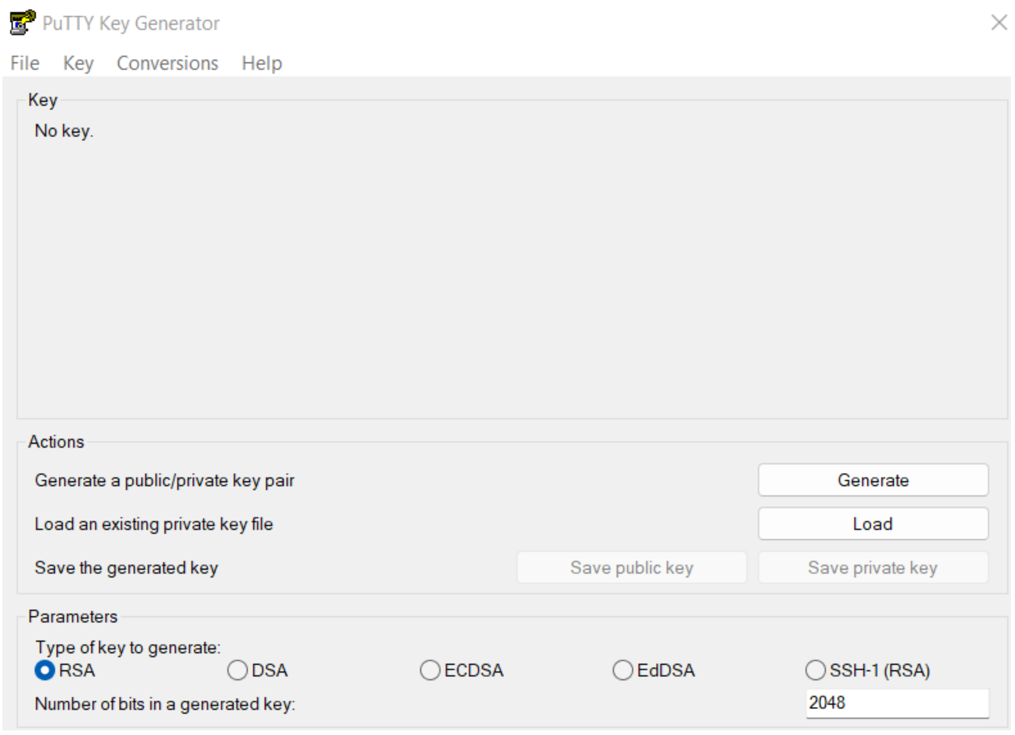
5.0 Appendix A

5.1 Converting the Private Key from PEM into PPK Format

In this case, we provided the Intel proxy hostname/port to the ISV to allow us access to their AWC EC2 instance with the private key in PEM format. We used PuTTY to connect from the Windows notebook in Intel network to the remote cloud instance. As PuTTY does not natively support the PEM format, the following steps are used to convert the private key from PEM into PPK format:

1. From the **Start** menu, choose **All Programs** -> **PuTTYgen**.

2. Under **Type of key to generate**, choose **RSA**. If your version of PuTTYgen does not include this option, choose **SSH-2 RSA**.

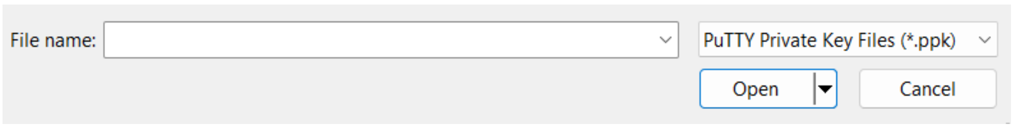


Key
No key.

Actions
Generate a public/private key pair
Load an existing private key file
Save the generated key

Parameters
Type of key to generate:
[✓ RSA] DSA ECDSA EdDSA SSH-1 (RSA)
Number of bits in a generated key: 2048

3. Choose **Load**. By default, PuTTYgen displays only files with the extension *.ppk*. To locate your *.pem* file, choose the option to display files of all types.



File name:
PuTTY Private Key Files (*.ppk)

4. Select your .pem file for the key pair that you specified when you launched your instance and choose **Open**. PuTTYgen displays a notice that the .pem file was successfully imported. Choose OK.

5. To save the key in the format that PuTTY can use, choose **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Choose **Yes**.

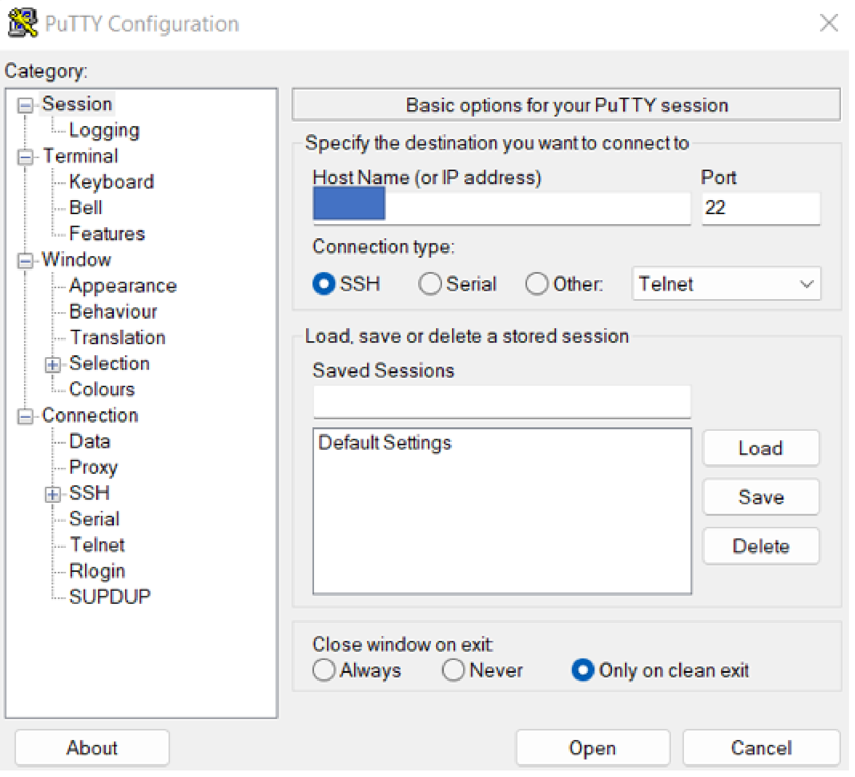
Note: passphrase on a private key is an extra layer of protection. Even if your private key is discovered, it can't be used without the passphrase. The downside to using a passphrase is that it makes automation harder because human intervention is needed to log on to an instance, or to copy files to an instance.

6. Specify the same name for the key that you used for the key pair (for example, key-pair name) and choose **Save**. PuTTY automatically adds the .ppk file extension.

Now, the private key is in the correct format for use with PuTTY to connect to the cloud instance using PuTTY's SSH client.

5.2 Connecting from Intel Network to AWS EC2 Instance

1. From the **Start** menu, choose **All Programs** -> **PuTTY.** Enter the IP address (provided by ISV) and Port of the AWS EC2 instance in PuTTY.


PuTTY Configuration

Category:

Session, Logging, Terminal, Keyboard, Bell, Features, Window, Appearance, Behaviour, Translation, Selection, Colours, Connection, Data, Proxy, SSH, Serial, Telnet, Rlogin, SUPDUP

Basic options for your PuTTY session

Specify the destination you want to connect to

Host Name (or IP address): [redacted]

Port: 22

Connection type: [✓ SSH] Serial Other: Telnet

Load, save or delete a stored session

Saved Sessions

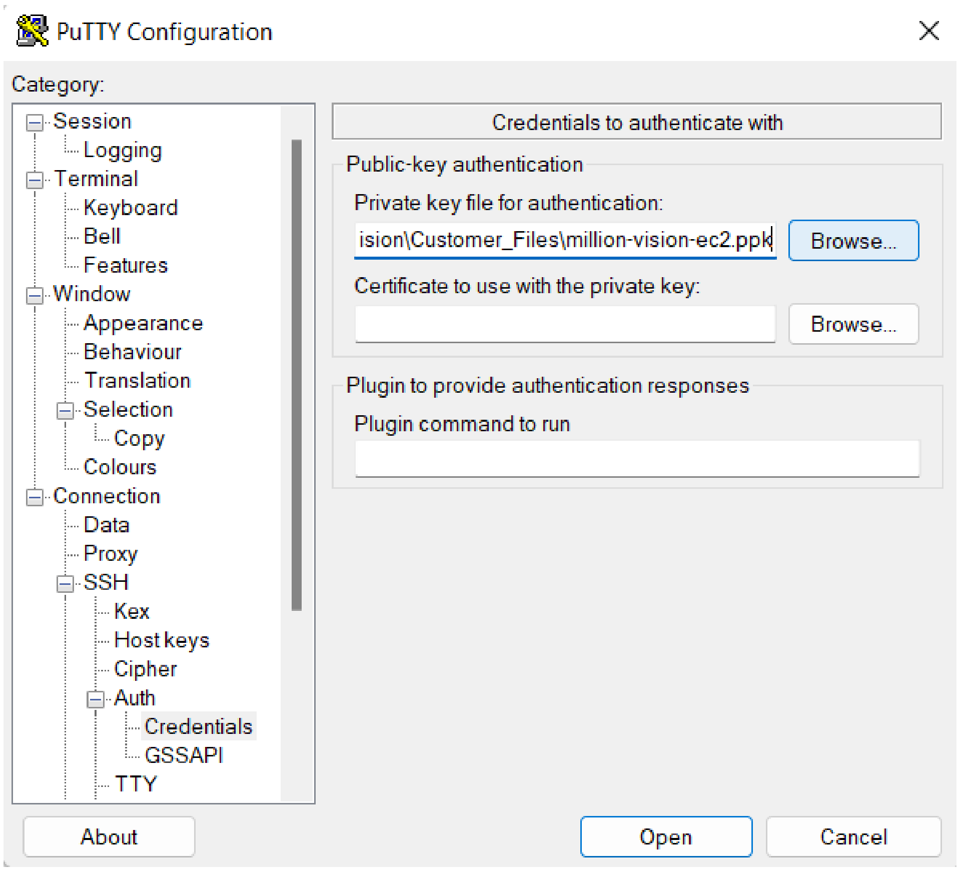
Default Settings

Load Save Delete

Close window on exit: Always Never [✓ Only on clean exit]

About Open Cancel

2. Select your private key file (.ppk) for authentication in **Connection** -> **SSH** -> **Auth** -> **Credentials**



Credentials to authenticate with

Public-key authentication

Private key file for authentication:

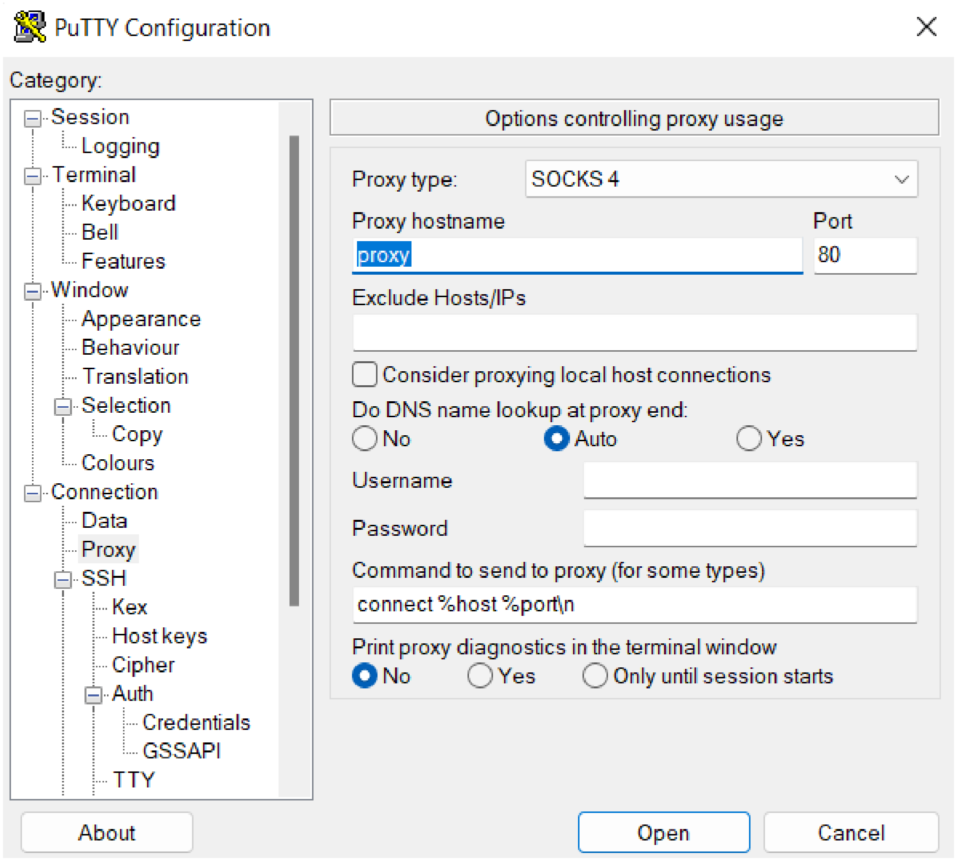
ision\Customer_Files\million-vision-ec2.ppk

Certificate to use with the private key:

Plugin to provide authentication responses

Plugin command to run

3. Select the proxy type as **SOCKS 4** in **Connection** -> **Proxy**. Enter the preferred Intel proxy hostname/port (Contact the Intel IT team to get this information and share it with ISV to allow access to their cloud instance) and click **Open** to establish the SSH connection to the remote AWS EC2 instance.


PuTTY Configuration

Options controlling proxy usage

Proxy type: SOCKS 4

Proxy hostname: proxy

Port: 80

Exclude Hosts/IPs:

☐ Consider proxying local host connections

Do DNS name lookup at proxy end: No [✓ Auto] Yes

Username:

Password:

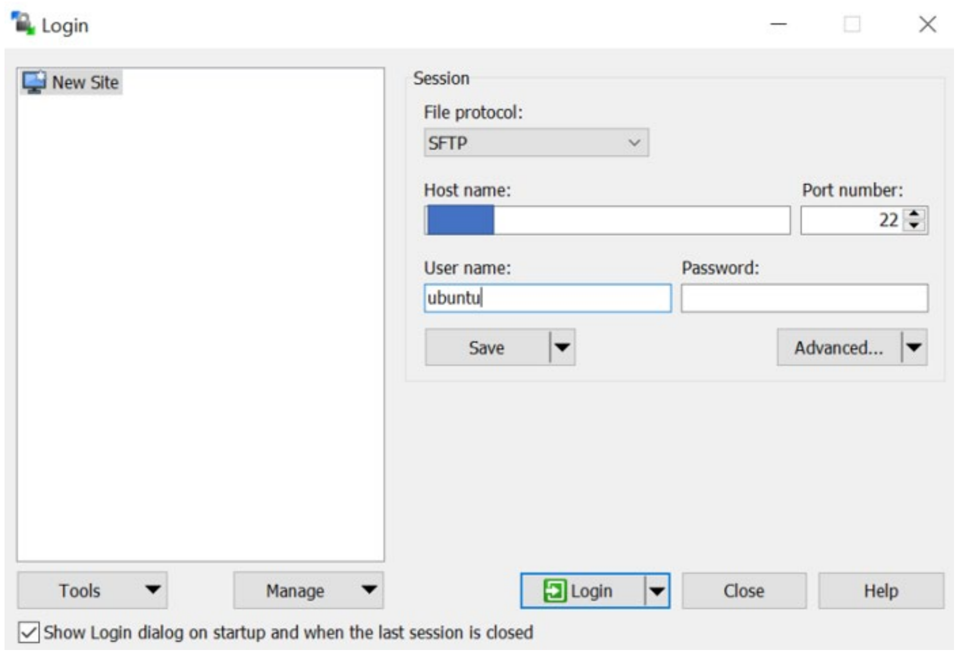
Command to send to proxy (for some types): connect %host %port\n

Print proxy diagnostics in the terminal window: [✓ No] Yes Only until session starts

5.3 Transferring Files from Intel Network to AWS EC2 Instance

1. Download WinSCP from https://winscp.net

2. Enter the IP address and Port of the cloud instance on the login screen and click **Advanced**.



Login

New Site

Session

File protocol: SFTP

Host name: [redacted]

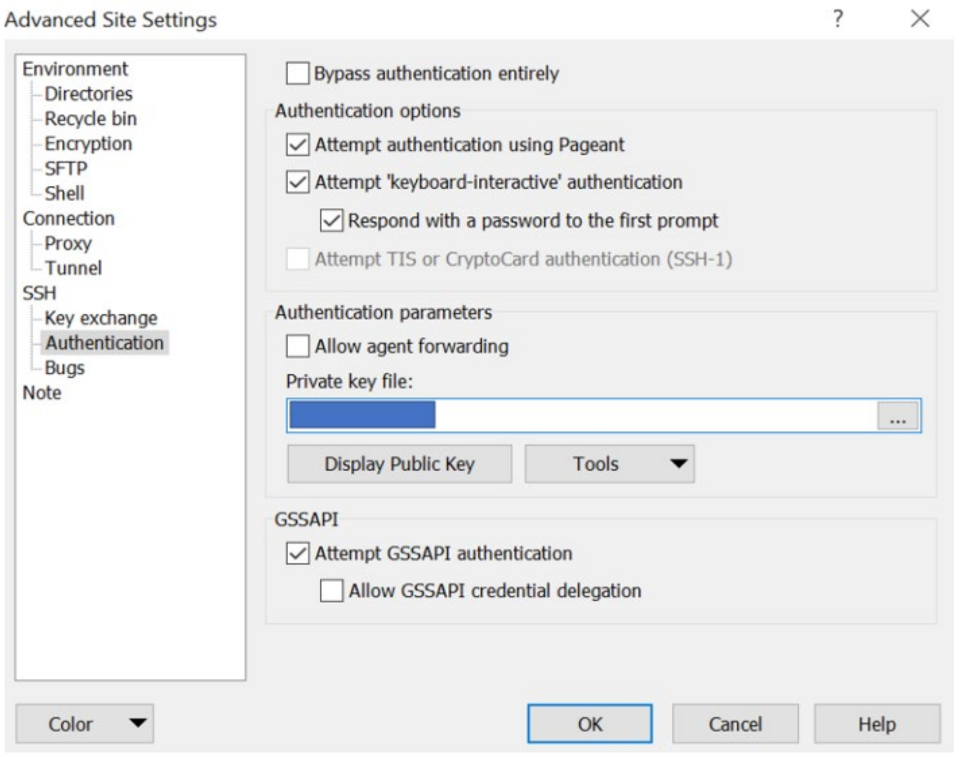
Port number: 22

User name: ubuntu

Password:

☑ Show Login dialog on startup and when the last session is closed

3. Select your private key file (PPK) for authentication in **SSH** -> **Authentication**



Advanced Site Settings

☐ Bypass authentication entirely

Authentication options

☑ Attempt authentication using Pageant

☑ Attempt 'keyboard-interactive' authentication

☑ Respond with a password to the first prompt

☐ Attempt TIS or CryptoCard authentication (SSH-1)

Authentication parameters

☐ Allow agent forwarding

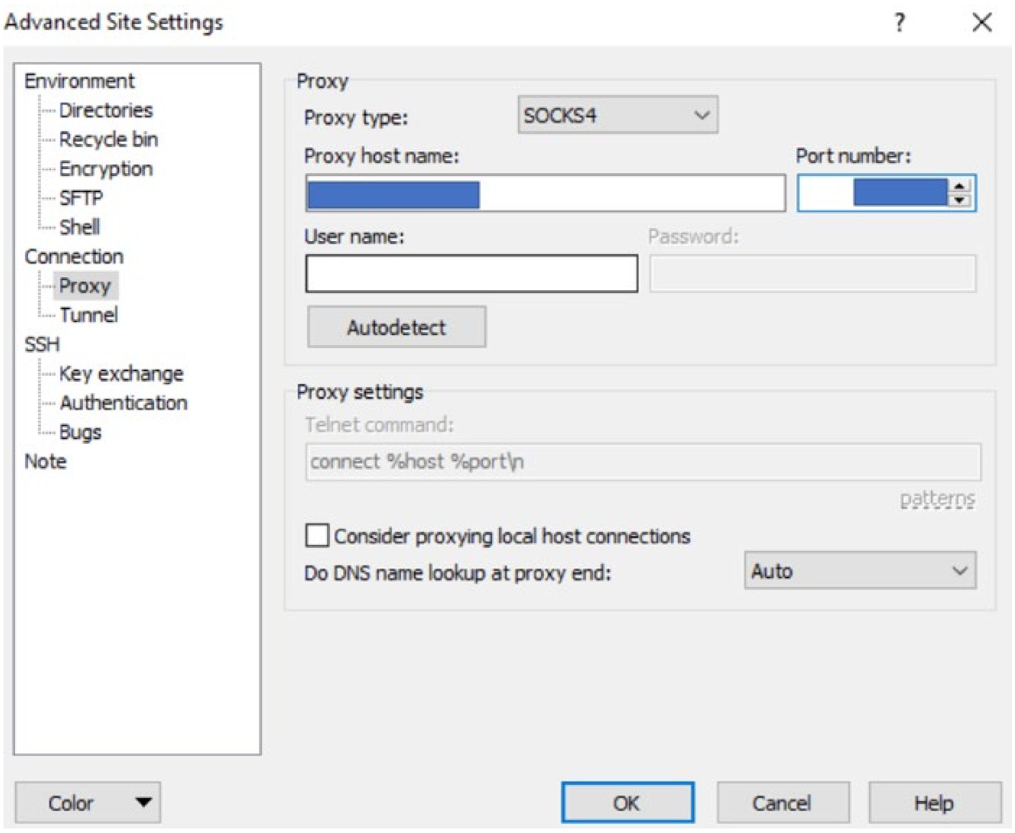
Private key file: [redacted]

GSSAPI

☑ Attempt GSSAPI authentication

☐ Allow GSSAPI credential delegation

4. Select the proxy type as **SOCKS4** in **Connection** -> **Proxy** and enter the Intel proxy hostname/port. Then, click **OK.**


Advanced Site Settings

Proxy

Proxy type: SOCKS4

Proxy host name: [redacted]

Port number: [redacted]

User name:

Password:

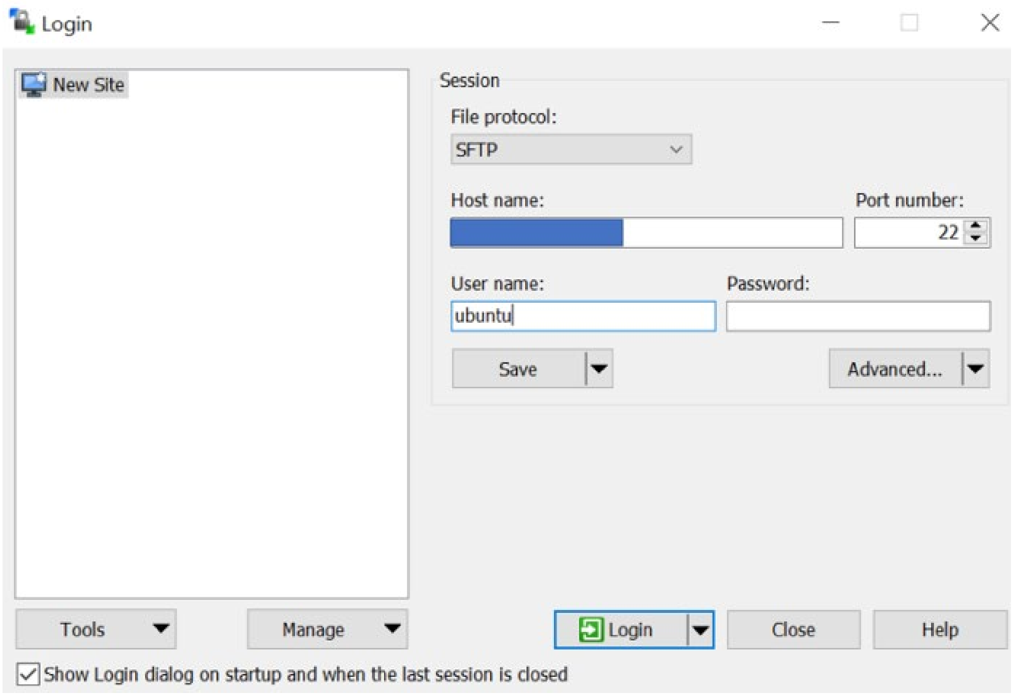
Proxy settings

Telnet command: connect %host %port\n

☐ Consider proxying local host connections

Do DNS name lookup at proxy end: Auto

5. Click **Login** to connect to the cloud instance and transfer files


Login

Session

File protocol: SFTP

Host name: [redacted]

Port number: 22

User name: ubuntu

Password:

☑ Show Login dialog on startup and when the last session is closed