Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision and GEN AI

Reference Architecture

Revision 1.0
March 2025

Authors
Yuan Kuok Nee
Shin Wei Lim
Abhijit Sinha

Key Contributors
Timothy Miskell,
Jessie Ritchey
Edel Curley

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Revision Date
849086	1.0	Initial release	March 2025

§

1 Introduction

Intel® AI Edge Systems are a range of optimized commercial Edge AI systems delivered and sold through OEM/ODM in the Intel® ecosystem. They are commercial platforms verified-configured, tuned, and benchmarked using Intel's reference AI software application on Intel® hardware to deliver optimal performance for Enterprise applications.

Intel® AI Edge Systems offers a balance between computing and AI acceleration to deliver optimal TCO, scalability, and security. AI Edge systems enable our partners to jumpstart development through a hardened system foundation verified by Intel®. AI Edge systems enable the ability to add AI functionality through continuous integration into business applications for better business outcomes and streamlined implementation efforts.

To support the development of these Edge AI systems, Intel® is offering reference design and verified reference configuration blueprints with Edge AI system configurations that are tuned and benchmarked for different Edge AI System types that support Enterprise AI use cases. Verified reference blueprints (VRB) include Hardware BOM, Foundation Software configuration (OS, Firmware, Drivers) tested and verified with supported Software stack (software framework, libraries, orchestration management).

This document describes a verified reference blueprint architecture using the 5th Gen Intel® Xeon® Scalable processor family.

When end customers choose an Intel® AI Edge Systems for the edge Verified Reference Blueprint, they should be able to deploy the AI workload more securely and efficiently than ever before. End users spend less time, effort, and expense evaluating hardware and software options. Intel® AI Edge Systems Verified Reference Blueprint helps end users simplify design choices by bundling hardware and software pieces together while making the high performance more predictable.

Intel® AI Edge Systems Verified Reference Blueprints –Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision, GEN AI are based on single-node architecture, that provides an environment to execute multiple AI workloads that are common to be deployed at the edge, such as the "Intel® Automated Self-Checkout Reference Package" and "GEN AI".

All Intel® AI Edge Systems Verified Reference Blueprint Configurations feature a workload-optimized stack tuned to take full advantage of an Intel® Architecture (IA) foundation. To meet the requirements, OEM/ODM systems must meet a performance threshold that represents a premium customer experience.

There are two configurations for Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision and GEN AI covering a base and plus configuration:

- Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision, and GEN AI Plus configuration is defined with at least a 52-core 5th Generation Intel® Xeon® Scalable processor and high-performance network, with storage and integrated platform acceleration products from Intel® for maximum virtual machine density.

- Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable Base for Computer Vision, and GEN AI Base configuration is defined with a 32-core or higher 5th Generation Intel® Xeon® Scalable processor and network, with storage and add-in platform acceleration products from Intel® targeting for optimized value and performance-based solutions.

Bill of Materials (BOM) requirement details for the configurations are provided in Chapter 2 of this document.

Intel® AI Edge Systems Verified Reference Blueprint is defined in collaboration with end users and our ecosystem partners to demonstrate the value of the solution for AI Inference use cases. The solution leverages the hardened hardware, firmware, and software to allow customers to integrate on top of this known good foundation.

Intel® AI Edge Systems Verified Reference Blueprint provides numerous benefits to ensure end users have excellent performance for their AI Inference applications. Some of the key benefits of the Reference Configuration based on the 5th Generation Intel® Xeon® Scalable Processor Family processor include : :

- High core counts and per-core performance
- Compact, power-efficient system-on-chip platform
- Streamlined path to cloud-native operations
- Accelerated AI inference on CPU using Intel® AMX and Intel® DL Boost
- Accelerated encryption and compression
- Platform-level security enhancements

§

2 Design Compliance Requirements

This chapter focuses on the design requirements for Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision, and GEN AI.

2.1 Hardware Requirements

The checklists in this chapter are a guide for assessing the platform's conformance to Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision, and GEN AI. The hardware requirements for the Plus Configuration and Base Configuration are detailed below.

Table 1. Platform Plus Configuration

Ingredient	Requirement	Required/ Recommended	Quantity
Processor	Intel® Xeon® Platinum 8571N Processor at 2.4GHz, 52C/104T,300W or higher number SKU	Required	1
Memory	Option 1: DRAM only configuration: 256 GB (16x 16 GB DDR5, 4800 MHz)	Required	16
	Option 2: DRAM only configuration: 512 GB (32x 16 GB DDR5, 4800 MHz)		32
Storage (Boot Drive)	480 GB or equivalent boot drive	Required	1
Storage (Capacity)	Minimum 1 TB or equivalent drive	Recommended	1
Storage (Capacity)	2TB or equivalent drive	Recommended	1
LAN on Motherboard (LOM)	1/10 Gbps port for Management Network Interface Controller (NIC)	Required	1

Table 2. Platform Base Configuration

Ingredient	Requirement	Required/ Recommended	Quantity
Processor	Intel® Xeon® Gold 6538N processor at 2.1 GHz, 32C/64T, 205W or higher number SKU	Required	1
Memory	DRAM only configuration: 256 GB (16x 16 GB DDR5, 4800 MHz)	Required	16
Storage (Boot Drive)	480 GB or equivalent boot drive	Required	1

Ingredient	Requirement	Required/ Recommended	Quantity
Storage (Capacity)	Minimum 1 TB or equivalent drive	Recommended	1
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for PXE/OAM	Recommended	1
	1/10 Gbps port for Management NIC	Required	1

2.2 BIOS Settings

To meet the performance requirements for an Intel® AI Edge Systems Verified Reference Blueprint—Scalable Performance Edge AI on Intel® Xeon® for Computer Vision and GEN AI, Intel® recommends using the BIOS settings to enable processor p-state and c-state with Intel® Turbo Boost Technology ("turbo mode") enabled. Hyperthreading is recommended to provide higher thread density. For this solution Intel® recommends using the NFVI profile BIOS settings for on-demand Performance with power consideration.

The NFVI profile for BIOS settings is documented in Chapter 3 of BIOS Settings for Intel® Wireline, Cable, Wireless, and Converged Access Platform (#747130).

Note: BIOS settings differ from vendor to vendor. Please contact your Intel® Representative for NFVI BIOS Profile Doc# 747130 or if you have difficulty configuring the exact setting in your system BIOS.

2.3 Solution Architecture

Figure 1 shows the architecture diagram of Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision and GEN AI. The software stack consists of three categories of AI software:

1. Vision AI
2. Generative AI
3. Network Security AI

All three applications are containerized using docker.

For the Vision AI use case, we are using the Intel® Automated Self-Checkout application, which measures stream density in terms of the number of supported cameras at the target FPS, accounting for all stages within the processing pipeline. The video data is ingested and pre-processed before each inferencing step. The inference is performed using two models: YOLOv5 and EfficientNet. The YOLOv5 model does object detection, and the EfficientNet model performs Object Classification. For additional information refer to Appendix A.1 .

For the Generative AI use case, we are using large language models (LLMs) and Intel® Extension of PyTorch (IPEX) framework to perform LLM inference on Intel® CPU.

For Network Security AI, we are using Malconv and finetuned BERT-base-cased for malicious portable executable (PE) file detection and email phishing detection respectively.

Figure 1. Architecture of the Intel® AI Edge Systems Verified Reference Blueprint

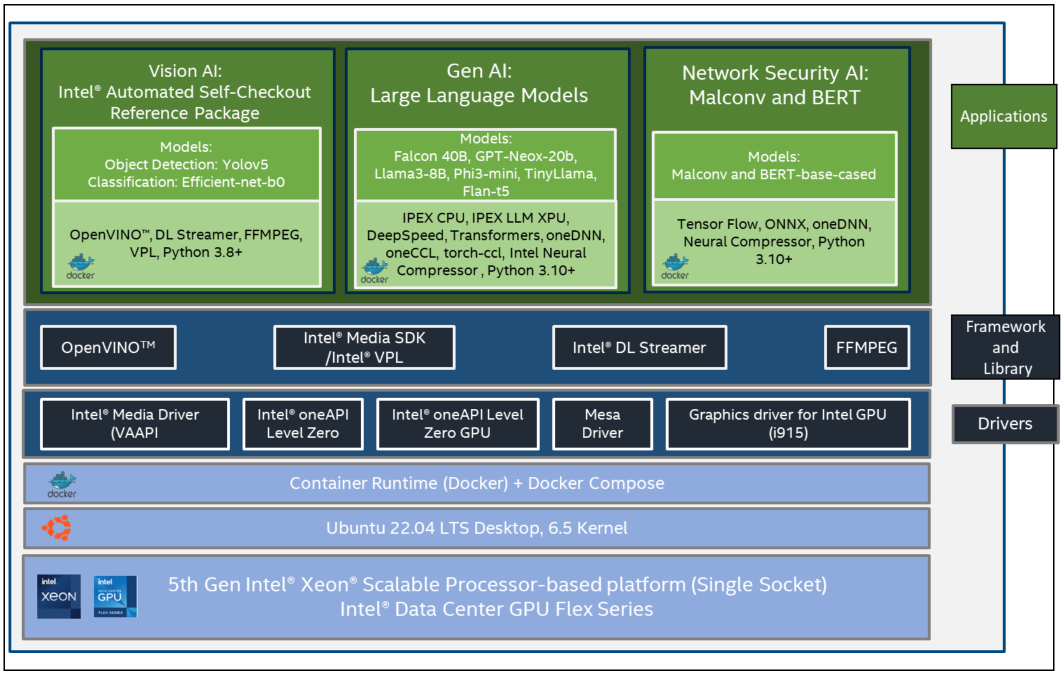
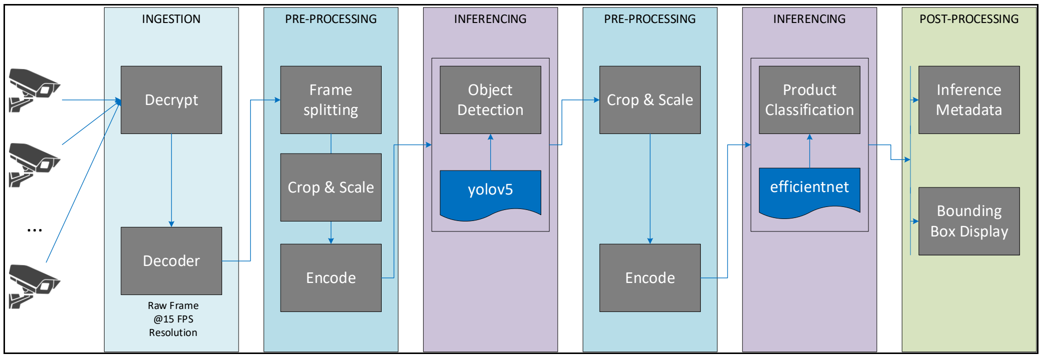

Vision AI:
Intel® Automated Self-Checkout
Reference Package
Models:
Object Detection: Yolov5
Classification: Efficient-net-b0
OpenVINO™, DL Streamer, FFMPEG,
VPL, Python 3.8+
docker
Gen AI:
Large Language Models
Models:
Falcon 40B, GPT-Neox-20b,
Llama3-8B, Phi3-mini, TinyLlama,
Flan-t5
IPEX CPU, IPEX LLM XPU,
DeepSpeed, Transformers, oneDNN,
oneCCL, torch-ccl, Intel Neural
Compressor , Python 3.10+
docker
Network Security AI:
Malconv and BERT
Models:
Malconv and BERT-base-cased
Tensor Flow, ONNX, oneDNN,
Neural Compressor, Python
3.10+
docker
Applications
OpenVINO™
Intel® Media SDK
/Intel® VPL
Intel® DL Streamer
FFMPEG
Framework
and
Library
Intel® Media Driver
(VAAPI
Intel® oneAPI
Level Zero
Intel® oneAPI Level
Zero GPU
Mesa
Driver
Graphics driver for Intel GPU
(i915)
Drivers
docker
Container Runtime (Docker) + Docker Compose
Ubuntu 22.04 LTS Desktop, 6.5 Kernel
intel
XEON
intel
GPU
5th Gen Intel® Xeon® Scalable Processor-based platform (Single Socket)
Intel® Data Center GPU Flex Series

Figure 2 shows the architecture diagram for the Intel® Automated Self-Checkout application, which in this case is deployed containerized via Docker. The Vision AI use case measures stream density in terms of the number of supported cameras at the target FPS, accounting for all stages within the processing pipeline. The video data is ingested and pre-processed before each inference stage. The inference is performed using two models: YOLOv5 and EfficientNet. The YOLOv5 model performs object detection while the EfficientNet model performs object classification. For additional information refer to the Appendix.

Figure 2. Test Methodology for the Intel® Automated Self-Checkout Pipeline



The table below is a guide for assessing the conformance to the software requirements of the Intel® AI Edge Systems Verified Reference Blueprint Ensure that the platform meets the requirements listed in the table below.

Table 3. SW Configuration

Ingredient	SW Version Details
OS	Ubuntu 22.04.4 Desktop LTS[1]
Kernel	6.5 (in-tree generic)
OpenVINO	2024.0.1
Docker Engine	27.1.0
Docker Compose	2.29
Intel® Level Zero for GPU	1.3.29735.27
Intel® Graphics Driver for GPU (i915)	24.3.23
Media Driver VAAPI	2024.1.5
Intel® OneVPL	2023.4.0.0-799
Mesa	23.2.0.20230712.1-2073
OpenCV	4.8.0
DLStreamer	2024.0.1
FFmpeg	2023.3.0

2.4 Platform Technology Requirements

This section lists the requirements for Intel's advanced platform technologies.

The Reference Blueprint recommends that Intel® AVX (Advance Vector Extensions) or AMX (Intel® Advance Matrix Extensions) to be enabled to reap the benefits of hardware-accelerated convolution.

Table 4. Platform Technology Requirements

Platform Technologies		Enable/Disable	Required/Recommended
Intel® AMX	Intel® Advanced Matrix Extension	Enable	Required
Intel® TXT	Intel® Trusted Execution Technology	Enable	Optional

2.5 Platform Security

For Intel® AI System for the Edge, it is recommended that Intel® Boot Guard Technology to be enabled so that the platform firmware is verified suitable during the boot phase.

[1] Desktop based OS selected as this kernel version includes native in-tree i915 driver support for Intel® Xe based discrete GPUs.

In addition to protecting against known attacks, all Intel® Accelerated Solutions recommend installing the Trusted Platform Module (TPM). The TPM enables administrators to secure platforms for a trusted (measured) boot with known trustworthy (measured) firmware and OS. This allows local and remote verification by third parties to advertise known safe conditions for these platforms through the implementation of Intel® Trusted Execution Technology (Intel® TXT).

2.6 Side Channel Mitigation

Intel® recommends checking your system's exposure to the "Spectre" and "Meltdown" exploits. This reference implementation has been verified with Spectre and Meltdown exposure using the latest Spectre and Meltdown Mitigation Detection Tool, which confirms the effectiveness of firmware and operating system updates against known attacks

The spectre-meltdown-checker tool is available for download at https://github.com/speed47/spectre-meltdown-checker.

§

3 *Platform Tuning*

3.1 Additional Linux Packages Installation

3.1.1 Install Docker

Follow the instructions at https://docs.docker.com/engine/install/Ubuntu*/ to install Docker Engine on Ubuntu*.

§

4 Performance Verification

This chapter aims to verify the performance metrics for the Intel® AI Edge Systems Verified Reference Blueprint to ensure that there is no anomaly seen. Refer to the information in this chapter to ensure that the performance baseline for the platform is as expected.

The solution was tested on August 06, 2024, with the following hardware and software configurations:

- 1 NUMA node
- 1 x Intel® Xeon® Gold 6538N processors or 1 x Intel® Xeon® Platinum 8571N processors
- Total Memory: 128 GB, 8 slots/16 GB/4800 MT/s DDR5 RDIMM
- Hyperthreading: Enable
- Turbo: Enable
- C-State: Enable
- Storage: 1x 1TB INTEL® SSDPE2KX010T8
- Network devices: 2x Dual port Intel® Ethernet Network Adapter E810-2CQDA2
- Network speed: 50 GbE
- BIOS: American Megatrends International, LLC. 3B05.TEL4P1
- Microcode: 0x21000161
- OS/Software: Ubuntu Desktop 22.04 LTS (kernel 6.5.0-44-generic)

4.1 Memory Latency Checker (MLC)

The Memory Latency Checker which can be downloaded from https://www.Intel®.com/content/www/us/en/developer/articles/tool/Intel®r-memory-latency-checker.html. Download the latest version, unzip the tarball package, go into the Linux* folder, and execute `./mlc`. Table 6 and Table 7 below should be used as a reference for verifying the validity of the system setup.

Table 5. Memory Latency Checker

Key Performance Metric	Local Socket (Plus)
Idle Latency (ns)	150.3
Memory Bandwidths between nodes within the system (using read-only traffic type) (MB/s)	260425

Table 6. Peak Injection Memory Bandwidth (1 MB/sec) Using All Threads

Peak Injection Memory Bandwidth (1 MB/sec) using all threads	Plus Solution
All Reads	255504

Peak Injection Memory Bandwidth (1 MB/sec) using all threads	Plus Solution
3:1 Reads-Writes	211857
2:1 Reads-Writes	202797
1:1 Reads-Writes	186870
STREAM-Triad	206753
Loaded Latencies using Read-only traffic type with Delay=0 (ns)	183.11
L2-L2 HIT latency (ns)	73.6
L2-L2 HITM latency (ns)	74.7

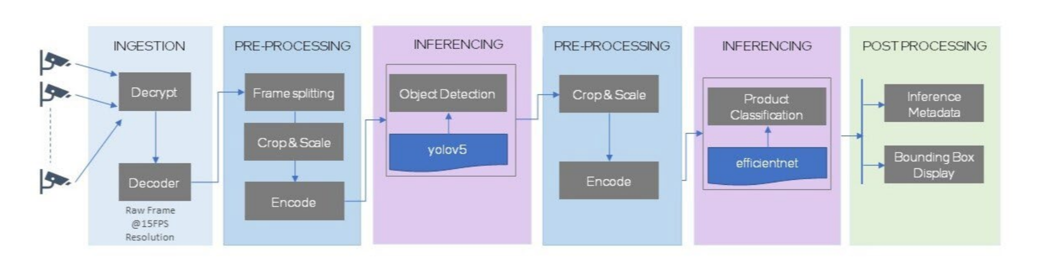
Note: If the latency performance and memory bandwidth performance are outside the range, please verify the validity of the Platform components, BIOS settings, kernel power performance profile used, and other software components.

4.2 Vision AI

Intel® Automated Self-Checkout application (previously known as Retail Checkout) provides critical components to build and deploy a self-checkout use case using Intel® hardware, software, and other open-source software such as OpenVINO™. For instance, in this case all the models in the pipeline are converted into OpenVINO format. In addition, this proxy workload makes use of both GStreamer for media processing and DLStreamer for inferencing, which includes detection and classification. This reference implementation provides a pre-configured automated self-checkout pipeline optimized for Intel® hardware.

The video stream is cropped and resized to enable the inference engine to run the associated models. The object detection and product classification features identify the SKUs during checkout. The bar code detection, text detection, and recognition feature further verify and increase the accuracy of the detected SKUs. The inference details are then aggregated and pushed to the enterprise service bus or MQTT to process the combined results further. This proxy workload supports either running directly on the CPU or fully offloading to the GPU, including encoding/decoding, along with inferencing.

Figure 3. Vision AI Video Analytics Pipeline



Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision, and GEN AI configuration, the platform CPU with AMX should be able to process up to 23 of streams at 4K @ 14.95FPS with HEVC codec, and up to 26 of streams when equipped with Intel® Flex 170 GPU.

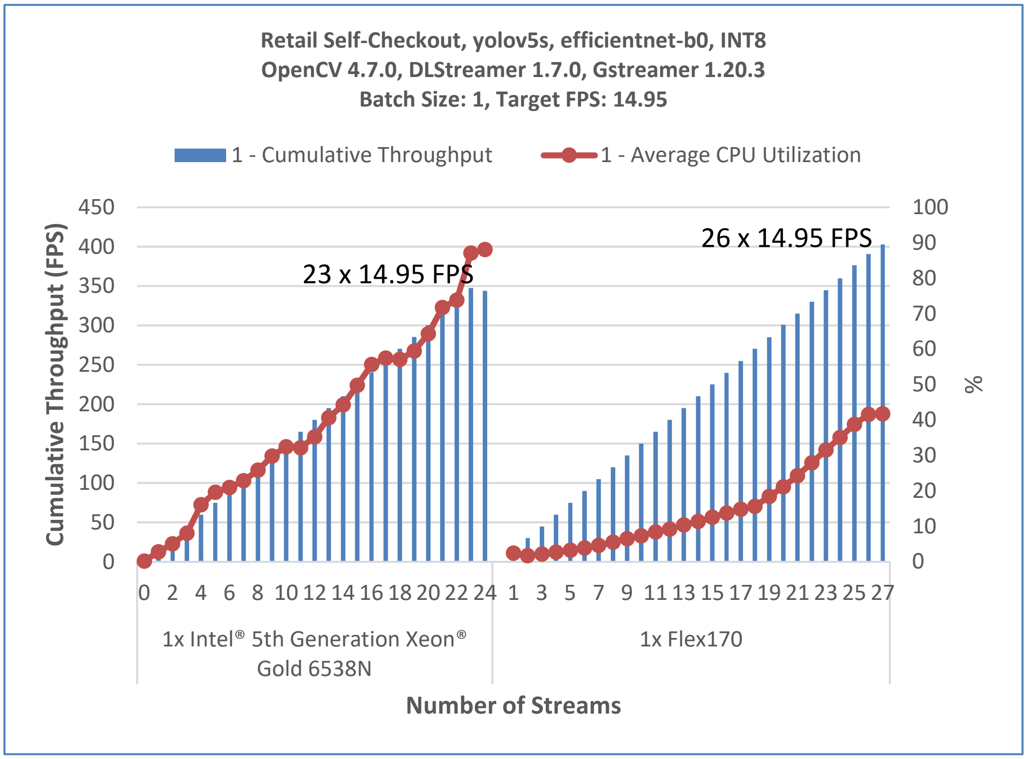
Table 7. Vision AI Workload Configuration

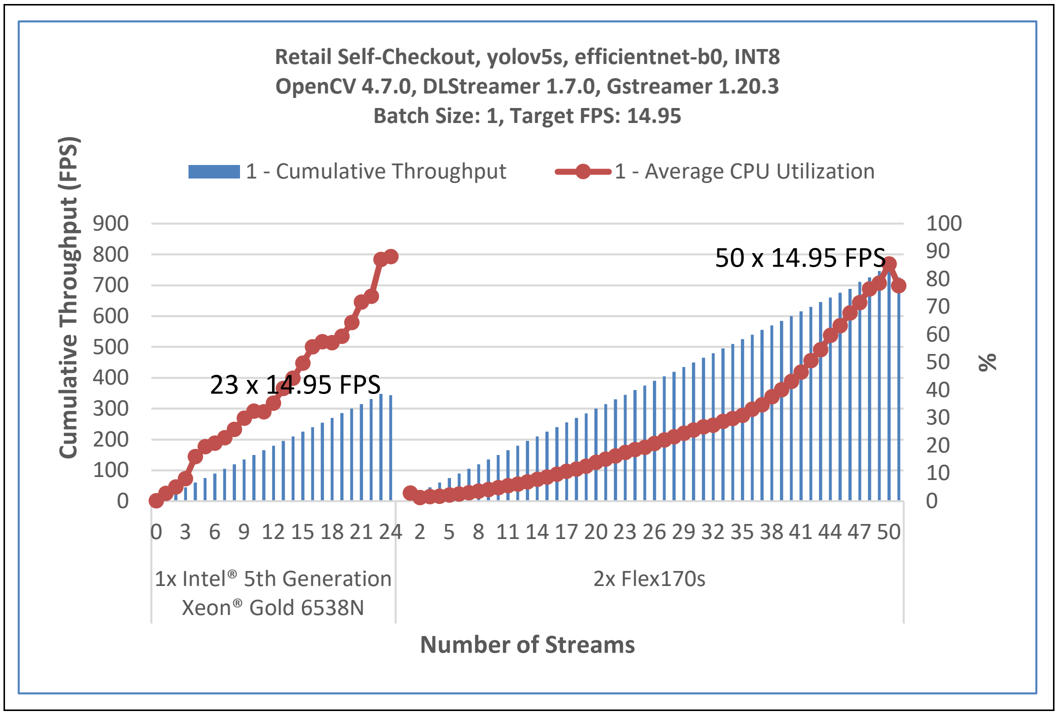
Ingredient	Software Version Details
OpenVINO	2024.0.1
DLStreamer	2024.0.1
FFMPEG	2023.3.0
VPL	2023.4.0.0-799
Python	3.8+

Table 8. Vision AI on Intel® Xeon® Scalable Configuration Performance

Intel® CPU Xeon® 6538N
23

Figure 4. Vision AI Performance Graph on Xeon® 6538N





4.3 GEN AI

Intel® GEN AI solution provides the ability to create, respond and synthesize texts to create content, summarizing text, building AI chatbots, generate images and more. In this document, we focus on inferencing performance using Intel® AI Edge Systems solution with State-Of-The-Art foundational models such as GPT-NEOX-20B, Llama 3 8B, Phi3, and TinyLlama. All of the LLM models are downloaded from the Hugging Face repository.

To ensure GEN AI is running on Intel® hardware with optimal performance, we use IPEX-LLM framework as inference workload. IPEX-LLM is optimized with Intel® AMX technology, as well as Intel® GPUs with precision from FP32 to INT4. Incrementing batch size also provides better throughput performance with latency trade-offs.

The benchmarking was conducted on CPU only AND on GPU only. The GPT-NEOX-20B was benchmarked on the Xeon® Scalable Platinum SKU (8571N). Other models were benchmarked on the Xeon® Scalable Gold SKU (6538N).

On LLM serving front, vLLM also has been integrated with IPEX-LLM, and provides excellent throughput by employing continuous batching, especially the LLM serving framework is optimized with underlying Intel® hardware enhancements such as AMX/AVX512 and AVX2.

For the Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision and GEN AI configuration, the system should be able to deliver results as shown in the graphs below as a baseline to the expected performance of this solution.

Figure 5. Performance for GPT-NEOX-20B model on CPU (Xeon® Scalable Platinum SKU (8571N))

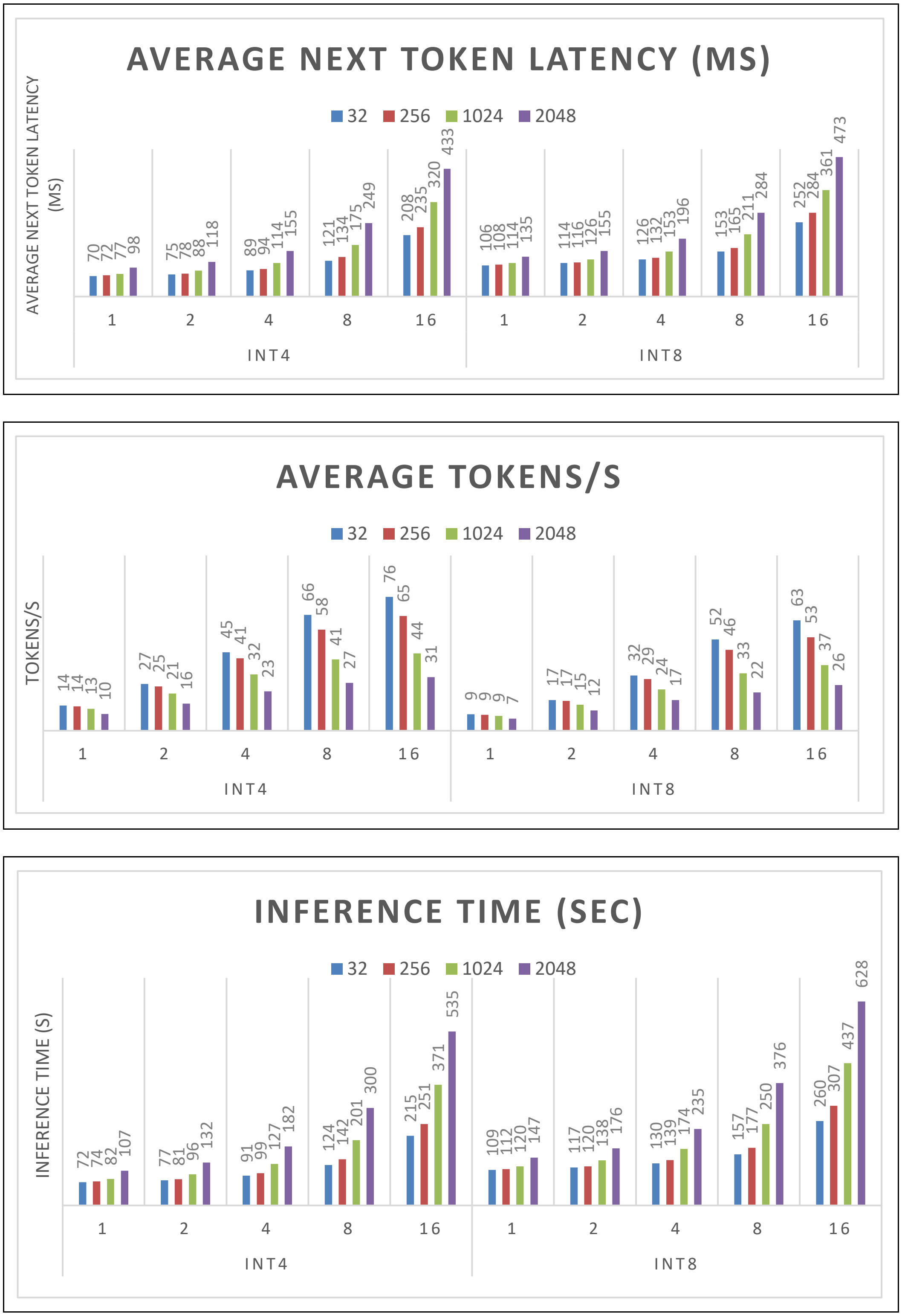


Figure 6. Performance for Llama 3 8B Model on CPU (Xeon® Scalable Gold SKU (6538N)).

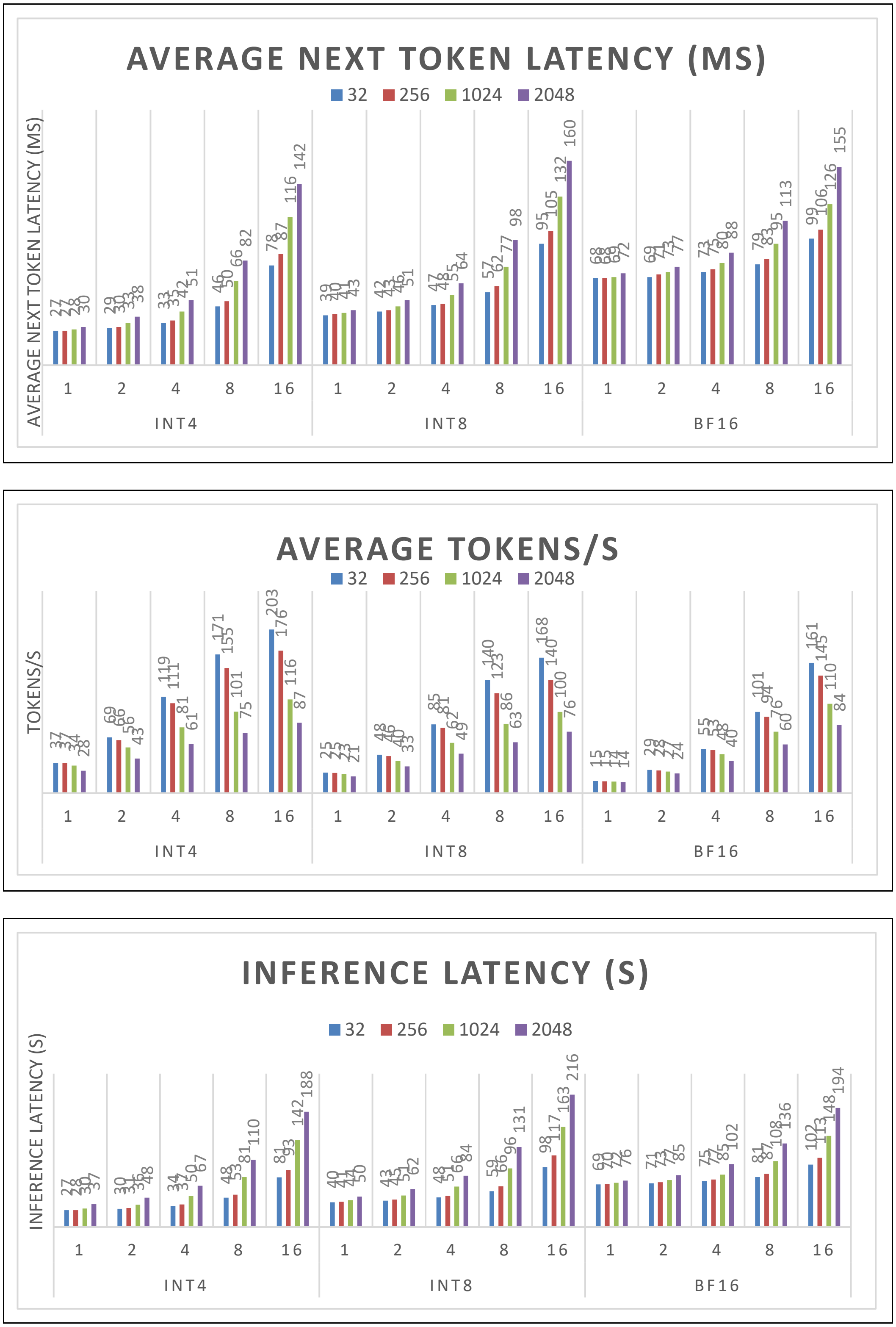


Figure 7. Performance for Phi-3-4k Mini Model on CPU (Xeon® Scalable Gold SKU (6538N))

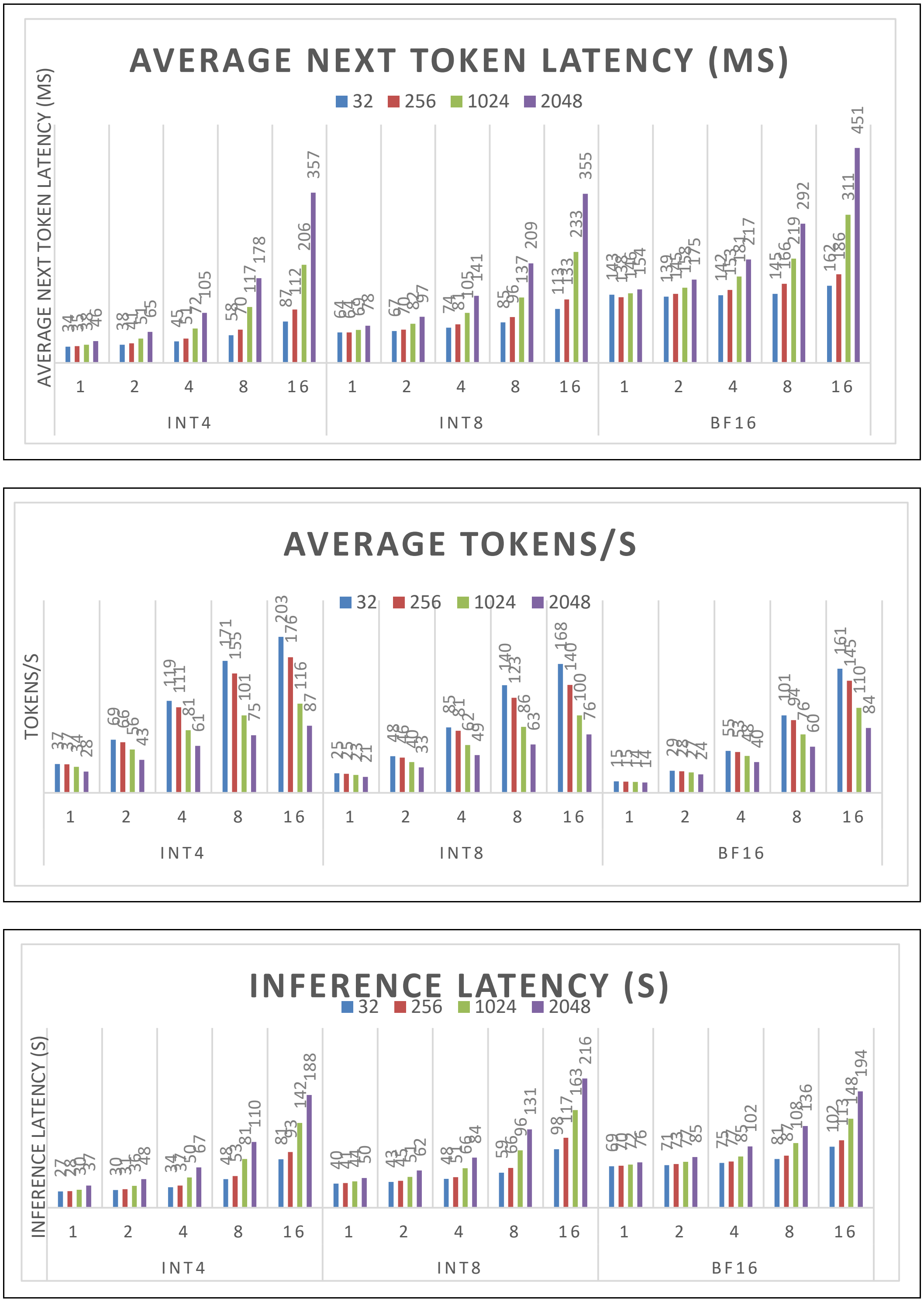


Figure 8. Performance on TinyLlama Model on CPU (Xeon® Scalable Gold SKU (6538N))

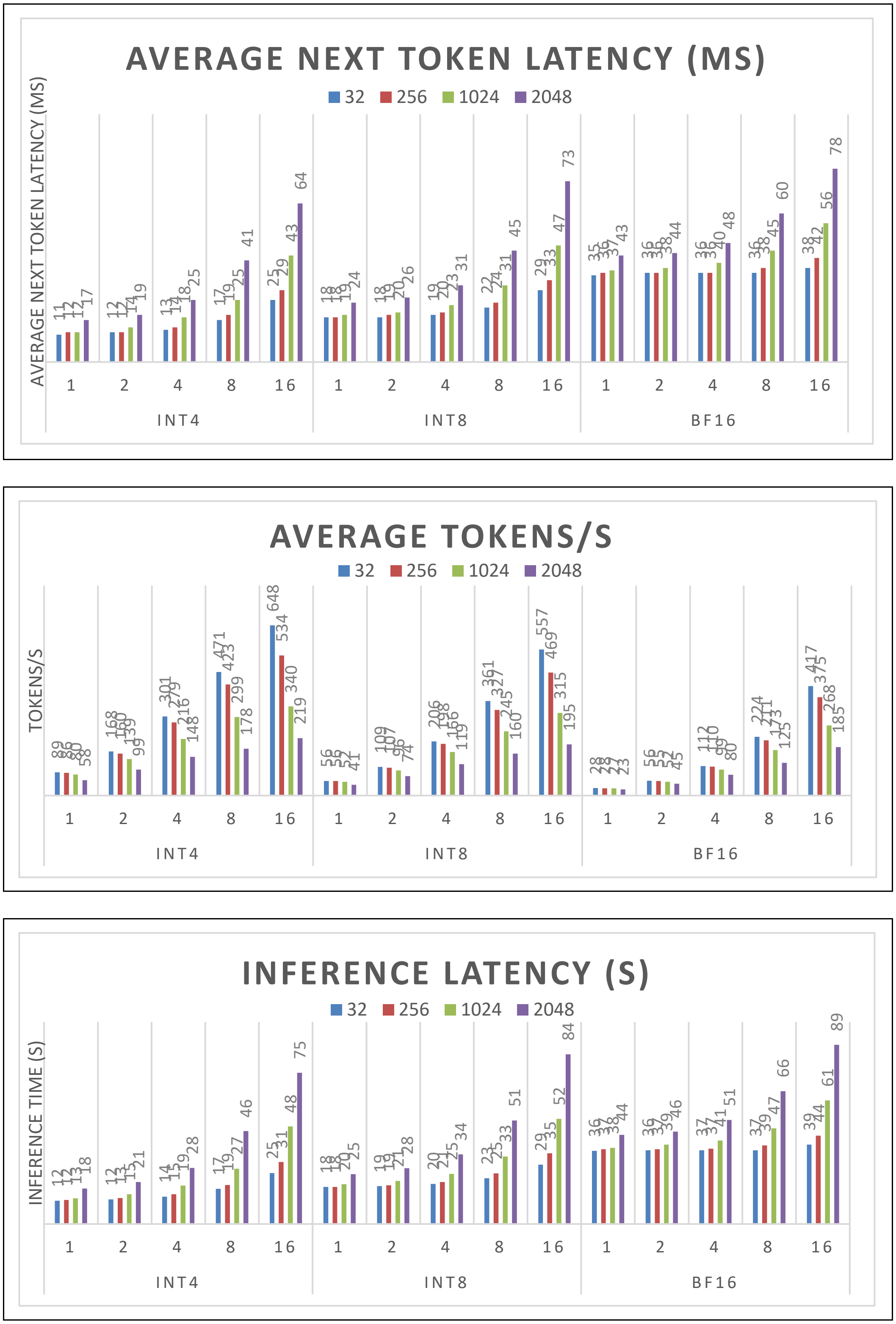
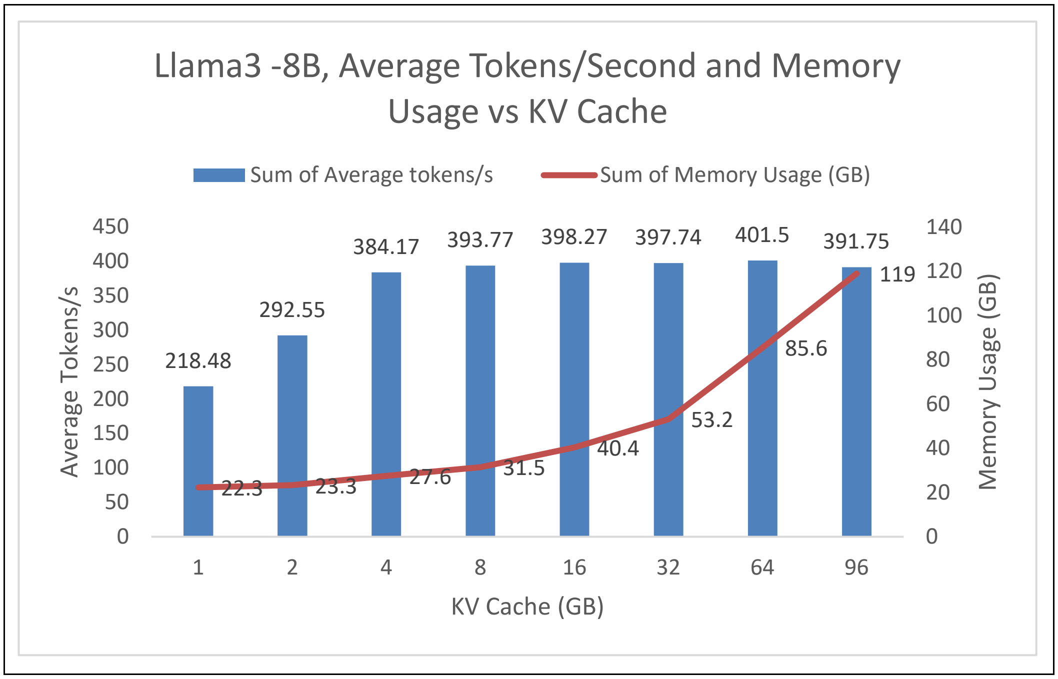


Figure 9. vLLM-IPEX-LLM Performance with Llama 3 8B Model on CPU (Xeon® Scalable Gold SKU (6538N))



4.4 Network Security AI: MalConv and BERT

4.4.1 MalConv for Malicious portable executable (PE) detection

AI inference is used in network/security to help prevent advanced cyber-attacks. To improve the latency associated with this application, the Intel® Xeon® Scalable Processor contains technologies to accelerate AI inference such as AVX-512, Advanced Matric Extensions (AMX), and Vector Neural Network Instructions. The MalConv AI workload utilizes the TensorFlow deep-learning framework, Intel® oneAPI Deep Neural Network Library (oneDNN), AMX, and Intel® Neural Compressor to improve the performance of the AI inference model.

The starting model for the MalConv AI workload is an open-source deep-learning model called MalConv which is given as a pre-trained Keras H5 format file. This model is used to detect malware by reading the raw execution bytes of files. An Intel® optimized version of this h5 model is used for this workload, and the testing dataset is about a 32GB subset of the dataset from https://github.com/sophos/SOREL-20M. The performance of the model can be improved by various procedures including conversion to a floating-point frozen model and using the Intel® Neural Compressor for post-training quantization to acquire BF16, INT8, and ONNX INT8 precision models.

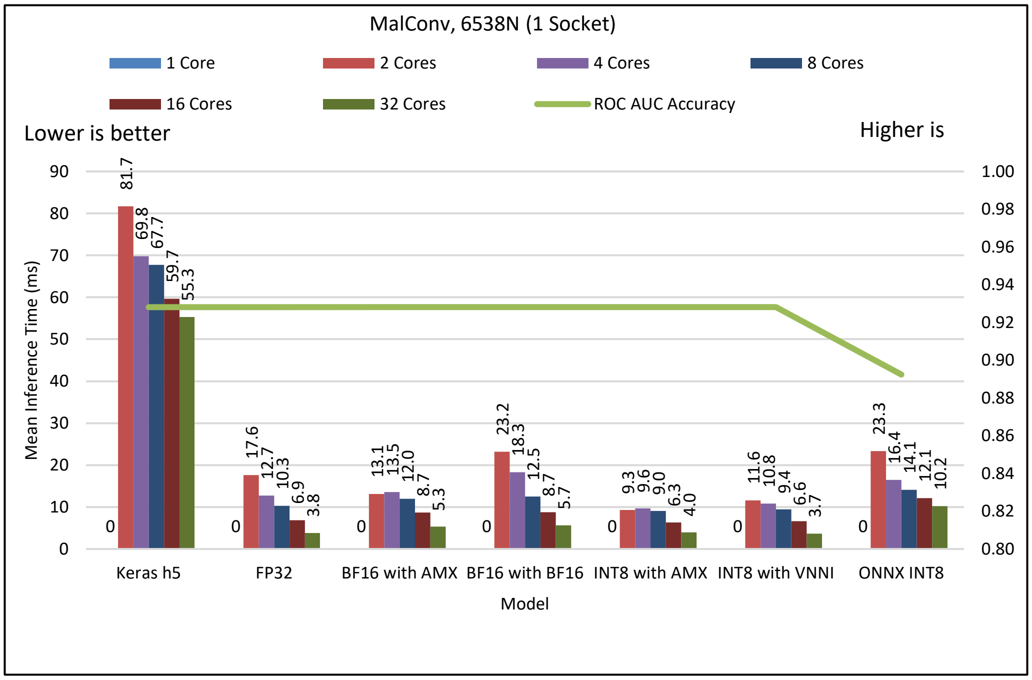
Ensure that the test results follow the expected results, as shown in the following tables, to baseline the platform's performance. Table 12 shows the software used for the testing, while Figure 14 shows a graph of the mean inference time for each model. With 2 cores per instance, the INT8 model with AVX512_CORE_AMX enabled reached a performance of less than 10 ms.

Note: Refer to https://hub.docker.com/r/Intel®/malconv-model-base for the Intel® Optimized MalConv Model.

Table 9. MalConv AI Workload Configuration

Ingredient	Software Version Details
TensorFlow	2.13.0
Intel® Extension for Tensorflow	2.13.0.1
oneDNN	2024.2.0
Python	3.11.7
Intel® Neural Compressor	2.6
ONNX	1.16.1

Figure 10. MalConv AI Entry Platform Performance Graph



BERT is a pre-trained language representation model developed by Google AI Language researchers in 2018, which consists of transformer blocks with a variable number of encoder layers and a self-attention head. The model used in the testing is a fine-tuned version of the Hugging Face BERT base model.

To detect phishing emails, the input email is first tokenized into chunks of words using the Hugging Face tokenizer, with a special CLS token was added at the beginning. The tokens are then padded to the maximum BERT input size, which by default is 512. The total input tokens are converted to integer IDs and fed to the BERT model. A dense layer is added for email classification, which takes the last hidden state for the CLS token as input.

Ensure that the results of the tests follow the expected results as shown in the following graph to baseline the performance of the platform. Table 12 shows the software used for the testing, while Figure 16 shows a graph of the results for the INT8 and FP32 BERT models. With 8 cores per instance, the mean latency of the INT8 model reaches below 20ms.

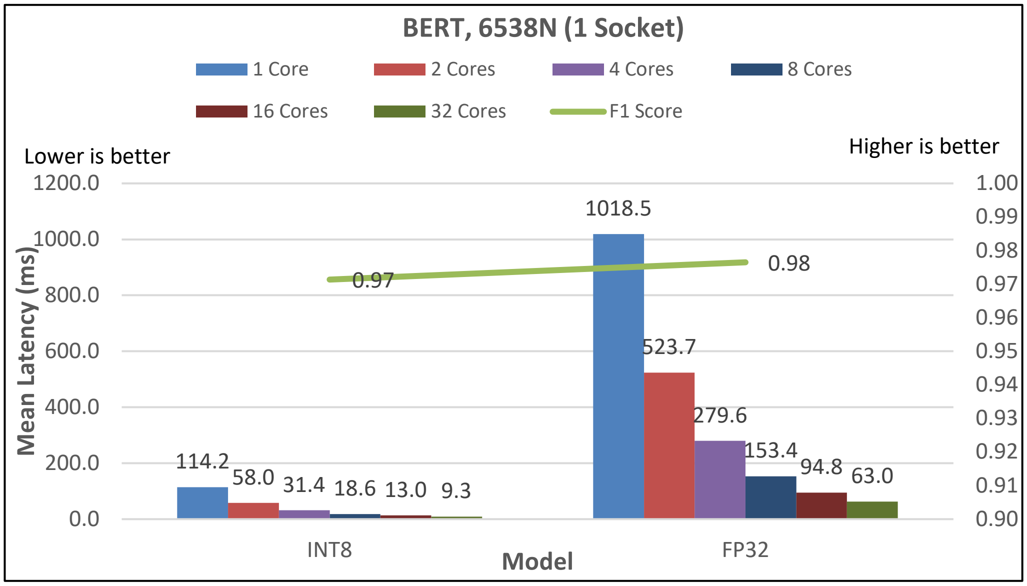
Note: Refer to https://huggingface.co/bert-base-cased for the original Hugging Face BERT base model.

Note: The phishing email test dataset can be found at https://github.com/IBM/nlc-email-phishing/tree/master/data

Table 10. BERT AI Workload Configuration

Ingredient	Software Version Details
Torch	2.1.2
Intel® Extension for PyTorch	2.1.100
oneDNN	2024.2.0
Python	3.11.7
Intel® Neural Compressor	2.6

Figure 11. BERTAI Performance on VRC for Intel® AI System –Scalable Performance Edge AI on Intel® Xeon® Scalable Entry Configuration Graph



Performance Summary

The following presents the range of performance achievable for the Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable configuration across each of the Vision AI, Generative AI, and Network Security AI workloads.

§

5 *Summary*

The Intel® AI Edge Systems Verified Reference Blueprint – Scalable Performance Edge AI on Intel® Xeon® Scalable for Computer Vision and GEN AI defined on single socket 5th Gen Intel® Xeon® Scalable processors addresses the capabilities for AI Inference offering the following value proposition:

5.1 Vision AI Performance Summary

- Up to 23 IP camera streams of Vision AI workload

5.2 GEN AI Performance Summary

A summary of the benchmarking is shown below.

Table 11. Table Performance of Various Large Language Models on Xeon® Scalable CPU:

Models	Precision	Input Tokens	Batch Size	Throughput (tokens/s)	Inference time
GPT-NEOX-20B*	INT4	32	1	14	< 72s
Llama-3-8B	INT4	32-1024	4	81-119	< 60s
Phi3-4k-mini	INT4	32-256	8	179-208	< 60s
TinyLlama	INT4	32-1024	16	321-695	< 60s
vLLM Llama 3 8B	BF16	Variable	Variable	393	N/A

*This model was benchmarked on the Intel® Platinum 8571N SKU

This blueprint combined with architectural improvements, feature enhancements, and integrated Accelerators with high memory and IO bandwidth, provides a significant performance and scalability advantage in support for today's AI workload.

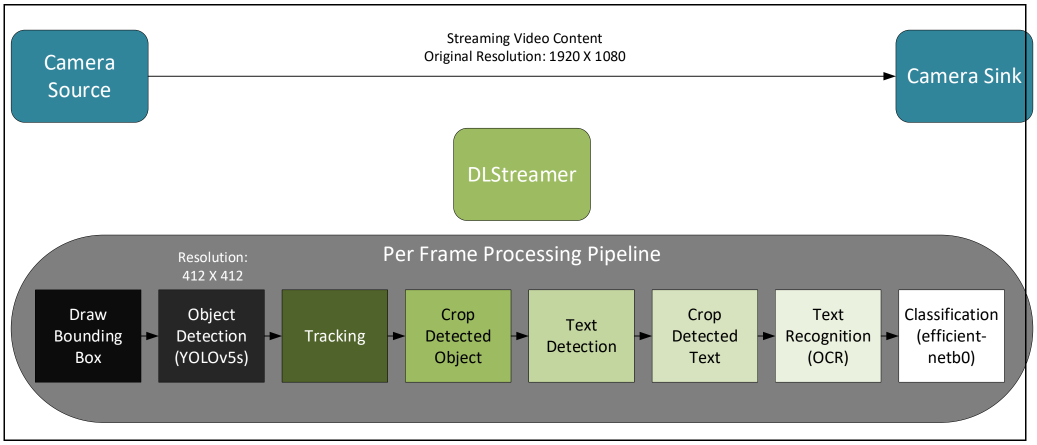
§

Appendix A Appendix

The following section provides detailed instructions for benchmarking a platform with each of the proxy workloads for Vision AI, Gen AI, along with Network Security AI. The benchmarking process leveraged the tools and scripts provided as part of the Intel® AI Edge Systems Verified Reference Blueprint will be available later, please reach out to your Intel® Field Representative for access.

A.1 Automated Self-Checkout Test Methodology

Figure 12. Test Methodology for the Automated Self-Checkout Proxy Workload



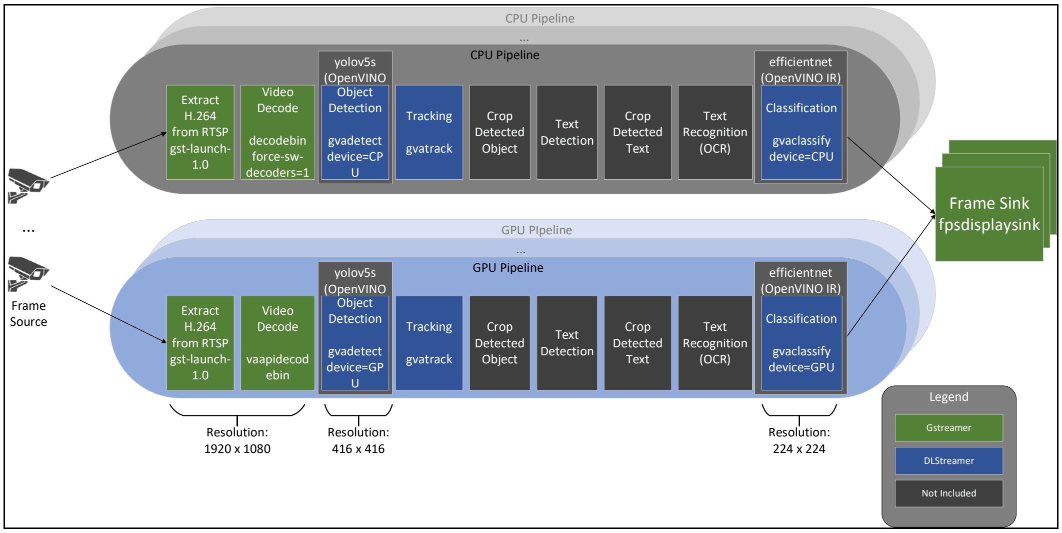
The Intel® Automated Self-Checkout Reference Package provides critical components required to build and deploy a self-checkout use case using Intel® hardware, software, and other open-source software. Vision workloads are large and complex and need to go through many stages. For instance, in the pipeline shown within the figure below, the video data is ingested, pre-processed before each inferencing stage, inferenced using two models - YOLOv5 and EfficientNet, and post-processed to generate metadata along with drawing the bounding boxes for each frame. The camera source plays back pre-recorded video content, which is then processed by the media analytics pipeline. The video stream input is decoded within the CPU pipeline using software-based decodebin API calls, while for the GPU pipeline the decoding is offloaded using vaapidecodebin API calls. The video content is freely available from https://www.pexels.com .

The Intel® Automated Self-Checkout Reference makes use of Intel® Deep Learning Streamer (Intel® DL Streamer), which leverages the open-source media framework GStreamer to provide optimized media operations along with the Deep Learning Inference Engine from the OpenVINO™ Toolkit to provide optimized inference. DLStreamer accelerates the media analytics pipeline for the Vision AI use case and allows for offloading to the underlying Intel® ARC™ and Intel® Data Center Flex GPUs.

The media analytics pipeline for Vision AI utilizes DLStreamer to performs object classification on the Region(s) of Interest (ROI) detected by gvadetect using the gvaclassify element and Intermediate Representation (IR) formatted object classification model. The models used for detection are in OpenVINO Intermediate Representation format, which is optimized for Intel® CPUs and GPUs. One advantage for the OpenVINO IR format is that the models can be used as-is without the need for retraining to leverage Intel® CPUs and GPUs. The Vision AI pipeline also uses object tracking for reducing the frequency of object detection and classification, thereby increasing the throughput, using gvatrack. The pipeline publishes the detection and classification results within a JSON file, which is then parsed, and the final results are reported in a log file.

Note: The GStreamer multi-media framework is used to stream video content by the frame source and the frame sink endpoints. The current release does not make use of the underlying media engines, offloading to the media engines is planned for future releases of the Intel® Automated Self-Checkout Reference.

Figure 13. Detailed Test Methodology for Retail Self-Checkout Pipeline


CPU Pipeline
...
CPU Pipeline
Extract H.264 from RTSP gst-launch-1.0
Video Decode decodebin force-sw-decoders=1
yolov5s (OpenVINO Object Detection gvadetect device=CPU
Tracking gvatrack
Crop Detected Object
Text Detection
Crop Detected Text
Text Recognition (OCR)
efficientnet (OpenVINO IR) Classification gvaclassify device=CPU
Frame Sink fpsdisplaysink
...
Frame Source
GPU PIpeline
...
GPU Pipeline
Extract H.264 from RTSP gst-launch-1.0
Video Decode vaapidecodebin
yolov5s (OpenVINO Object Detection gvadetect device=GPU
Tracking gvatrack
Crop Detected Object
Text Detection
Crop Detected Text
Text Recognition (OCR)
efficientnet (OpenVINO IR) Classification gvaclassify device=GPU
Resolution: 1920 x 1080
Resolution: 416 x 416
Resolution: 224 x 224
Legend
Gstreamer
DLStreamer
Not Included

The test methodology implements the following to measure the maximum number of streams that the system can sustain:

- Detection Model: Yolov5s
- Classification Model: efficient net-b0
- OpenVino 2024.0.1.
- DLStreamer 2024.0.1
- FFmpeg 2023.3.0
- VPL 2023.4.0.0-799

The test measures the number of streams that the server can sustain at the target FPS. For each test iteration, the number of camera streams is monotonically increased until the currently measured FPS value falls below the target FPS value. The number of streams is then monotonically decremented until the target FPS is met. Upon test completion the results are captured for the average FPS, the cumulative FPS, along with the peak number of streams achieved at the target FPS.

Pre-Requisites

1. Install Docker
2. Set HTTP_PROXY and HTTPS_PROXY proxies in environment if necessary.
3. Python version 3.8 is recommended.

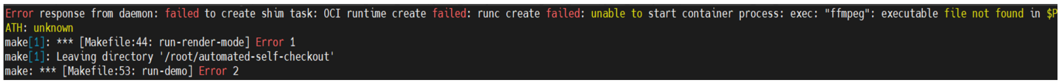
Quick Setup

Download videos, models, docker images and build containers.

```
$ git clone https://github.com/Intel®-retail/automated-self-checkout.git
$ git checkout tags/3.0.0
#make run-demo
```

Issue and Workaround

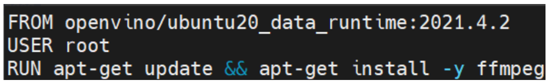
Issue #1: Binary 'ffmpeg" does not exist in OpenVINO container.



```
Error response from daemon: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "ffmpeg": executable file not found in $P
ATH: unknown
make[1]: *** [Makefile:44: run-render-mode] Error 1
make[1]: Leaving directory '/root/automated-self-checkout'
make: *** [Makefile:53: run-demo] Error 2
```

Workaround:

a. Create a Dockerfile named Dockerfile.OV.



```
FROM openvino/ubuntu20_data_runtime:2021.4.2
USER root
RUN apt-get update && apt-get install -y ffmpeg
```

b. Build the OpenVINO image.

```
$ docker build --build-arg HTTPS_PROXY=${HTTPS_PROXY} --build-arg
HTTP_PROXY=${HTTP_PROXY} -t openvino/Ubuntu*20_data_runtime:2021.4.2 -f
src/Dockerfile.OV .
```

A.2 Generative AI Test Methodology

A.2.1 IPEX-LLM Testing Methodology on CPU

The Generative AI benchmark on Intel® CPU was performed using Intel® Extension of PyTorch (IPEX) for LLM. All cores are being used and sustained at 100% CPU utilization throughout the inference process.

Please refer to the link below for more information on the configuration

https://www.Intel®.com/content/www/us/en/developer/articles/technical/accelerate-meta-llama3-with-Intel®-ai-solutions.html

A.2.2 Running IPEX-LLM Benchmarking Scripts

A.2.2.1 Running IPEX-LLM on CPU

Running IPEX-LLM on CPU Follow the steps to setup the IPEX-CPU test and benchmark on Single socket Intel® Xeon Scalable Processor. The user is expected to have privileged rights.

4. Install the baseline dependencies:

```
# sudo apt update
# sudo apt install -y make git numactl
# sudo apt install -y python3
# sudo pip install -upgrade pip
```

5. Clone the IPEX project:

```
# git clone https://github.com/Intel®/Intel®-extension-for-pytorch.git
# cd Intel®-extension-for-pytorch
# git checkout v2.3.100+cpu
# git submodule sync
# git submodule update --init --recursive
```

6. Build the IPEX docker image:

```
# DOCKER_BUILDKIT=1 docker build --build-arg HTTPS_PROXY=${HTTPS_PROXY} -
-build-arg HTTP_PROXY=${HTTP_PROXY} -f
examples/cpu/inference/python/llm/Dockerfile --build-arg COMPILE=ON -t
ipex-cpu:2.3.100 .
```

Note: The ipex-cpu container build takes approx. 30 mins

7. Verify the IPEX container is built

```
# docker images | grep ipex
REPOSITORY  TAG     IMAGE ID     CREATED     SIZE
ipex-cpu   2.3.100  d5ce81fe66f8  3 hours ago  4.61GB
```

8. Download the LLM models from HuggingFace:

```
# huggingface-cli download <model_card> --local-dir ~/<local_model_path>
--token <your_huggingface_token>
```

9. Start the ipex-cpu docker container

```
# export DOCKER_IMAGE=ipex-cpu:2.3.100
# export CONTAINER_NAME=ipex-cpu
# export MODEL_PATH=<CHANGE TO PATH TO THE MODEL DIRECTORY>

# docker run --rm -it --privileged --memory="256G" --shm-size="128G" --
name=$CONTAINER_NAME -v $MODEL_PATH:/llm/models $DOCKER_IMAGE bash
```

Note: It's recommended to use shard_model before running distributed inference to save time during model inference.

10. Shard model for Distributed inference inside the ipex-cpu docker container

```
# cd ./llm/utils
# create_shard_model.py -m /llm/models/<MODEL_ID> --save-path
/llm/models/<SHARD-MODEL-DIRECTORY>
```

11. Copy the benchmark_cpu_ds.sh and extract_kpis.py script to the container:

```
# docker cp ~/applications.platform.Intel®-select-for-
network/enterprise_ai/common/ipex-llm-cpu/benchmark_cpu_ds.sh ipex-
cpu://home/Ubuntu*/llm/

# docker cp ~/applications.platform.Intel®-select-for-
network/enterprise_ai/common/ipex-llm-cpu/extract_kpis.py ipex-
cpu://home/Ubuntu*/llm/
```

12. Change the user:group of the scripts inside the container:

```
# sudo chown Ubuntu*:Ubuntu* benchmark_cpu.sh
# sudo chown Ubuntu*:Ubuntu* extract_kpis.py
```

13. Edit the shard model path and model name in the benchmark_cpu_ds.sh script as shown

```
model_shard="/llm/models/llama3-8B/shard_model_hf"
model_name="llama3-8B"
```

14. Download the prompt json files for model tests

```
For Llama3 models download the below prompt file
# wget -O prompt.json https://Intel®-extension-for-
pytorch.s3.amazonaws.com/miscellaneous/llm/prompt-3.json

For other models, use the below prompt file
# wget https://Intel®-extension-for-
pytorch.s3.amazonaws.com/miscellaneous/llm/prompt.json
```

15. Run the benchmark script for distributed inference. This script will create a "result-model_name_mmddyyhhss" folder in the same directory and will contain text files for each test iteration

```
# ./benchmark_cpu.sh
```

16. Extract KPIs using the python script. This script generate a CSV file named llm_benchmark_results.csv with all the KPIs

```
# python extract_kpis.py --results-dir results-model_name_mmddyyhhss
```

17. Copy the llm_benchmark_results.csv file from docker to host

```
# docker cp ipex-
cpu:/home/Ubuntu*/llm/llm_benchmark_results.csv ./root/workspace
```

A.2.2.2 Running vLLM-IPEX-LLM on CPU

Create conda environment

```
$ wget https://github.com/conda-
forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.sh
$ chmod +x ./Miniforge3-Linux-x86_64.sh
$ ./Miniforge3-Linux-x86_64.sh
$ conda create -n ipex-vllm python=3.11
$ conda activate ipex-vllm
```

Install dependencies

```
pip3 install numpy
pip3 install --pre --upgrade ipex-llm[all] --extra-index-
url https://download.pytorch.org/whl/cpu
pip3 install psutil fastapi "uvicorn[standard]"
pip3 install sentencepiece # Required for LLaMA tokenizer.
pip3 install "pydantic<2" # Required for OpenAI server.
```

Install vLLM

```
git clone https://github.com/vllm-project/vllm.git andand \
cd ./vllm andand \
git checkout v0.4.2 andand \
pip install wheel packaging ninja setuptools==49.4.0 numpy andand \
pip install -v -r requirements-cpu.txt --extra-index-
url https://download.pytorch.org/whl/cpu andand \
sudo apt install build-essential
VLLM_TARGET_DEVICE=cpu python3 setup.py install
pip install ray
```

Download Dataset

```
$wget https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfi
ltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json
```

Run throughput benchmarking command line:

```
VLLM_CPU_KVCACHE_SPACE=16 # 16GB for KV_CACHE
python3 ./benchmark_throughput.py --device cpu --n 1000
```

```
--model Meta-Llama-3-8B --enable-chunked-prefill --dataset
ShareGPT_V3_unfiltered_cleaned_split.json
--trust-remote-code --max-num-batched-tokens 256
```

A.3 Network Security AI Test Methodology

A.3.1 MalConv AI Test Methodology

Follow the instructions below to run the MalConv AI testing:

1. You will need to provide your own testing dataset to use. Create the following directories:

   ```
   mkdir -p malconv/datasets/KNOWN
   mkdir -p malconv/datasets/MALICIOUS
   ```

2. Place the benign files into the “malconv/datasets/KNOWN” directory, and place the malicious files in the “malconv/datasets/MALICIOUS” directory
3. Use the “build_dockerfile.sh” script to build the Dockerfile image for the MalConv testing. If proxy variables for Internet access are needed, please set them in the Dockerfile before running the script.
4. Run the “run_malconv_test.sh” script to run the MalConv benchmarking test. The generated “malconv_results.log” file will contain five runs of the mean inference time results and ROC AUC accuracy of each model tested with different numbers of cores per instance.

A.3.2 Bert AI Test Methodology

Follow the instructions below to run the BERT testing:

1. Use the “build_dockerfile.sh” script to build the Dockerfile image for the MalConv testing. If proxy variables for Internet access are needed, please set them in the Dockerfile before running the script.
2. Run the “`run_bert_test.sh`” script to run the benchmarking test. The generated “`bert_results.log`” file will contain five runs of the testing showing multiple statistics for different numbers of cores per instance. The mean latency value is highlighted in the results shown in Section 4.4.2.

§