Intel® AI System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI

Reference Architecture

Revision 1.0
October 2024

Authors
Yuan Kuok Nee
Shin Wei Lim
Abhijit Sinha
Ai Bee Lim

Key Contributors
Timothy Miskell,
Jonathan Tsai
Jessie Ritchey
Edel Curley

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Revision Date
834790	1.0	Initial release	October 2024

§

1 Introduction

Intel® Enterprise AI systems are a range of optimized commercial 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 Systems offers a balance between computing and AI acceleration to deliver optimal TCO, scalability, and security. AI systems enable enterprises to jumpstart development through a hardened system foundation verified by Intel®. AI 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 AI systems, Intel® is offering reference design and verified reference configuration blueprints with AI system configurations that are tuned and benchmarked for different 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 network operators, service providers, cloud service providers, or enterprise infrastructure companies choose an Intel® AI System 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 System 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 System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI is based on single-node architecture, that provides environment to execute multiple AI workloads that are common to be deployed at the edge, such as the "Intel® Automated Self-Checkout Reference Package", "Generative AI" and "Network AI based on MalConv".

All Intel® AI System for Edge 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 System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI covering a base and plus configuration:

- Intel® AI System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI Plus configuration for the Node is defined with at least a 32-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 System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI Base configuration for the Node is defined with a 24-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 System for Edge Verified Reference Blueprint is defined in collaboration with enterprise vertical users, service providers 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 System for Edge 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 using Intel® AMX and Intel® DL Boost
- Multiple discrete GPU support to accelerate for AI inference workload
- Accelerated encryption and compression
- Platform-level security enhancements

§

2 Design Compliance Requirements

This chapter focuses on the design requirements for Intel® AI System for Edge Verified Reference Blueprint – Medium 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 System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI. The hardware requirements for the Plus Configuration and Base Configuration are detailed below.

Table 1. Intel® AI System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI - Plus Configuration

Ingredient	Requirement	Required/ Recommended	Quantity
Processor	Intel® Xeon® Gold 6538N Processor at 2.1GHz, 32C/64T, 205W 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
Graphics	1 x Flex 170 or 2 x Flex 170 1 x Arc A770 or 2 x Arc A770 2 x Arc A750 3 x Flex 140	Required	1 (Required) or up to 3
Storage (Capacity)	2TB or equivalent drive	Recommended	1
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for video streaming	Recommended	1
	1/10 Gbps port for Management Network Interface Controller (NIC)	Required	1

Table 2. Intel® AI System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI - Base Configuration

Ingredient	Requirement	Required/ Recommended	Quantity
Processor	Intel® Xeon® Gold 5518N processor at 1.8 GHz, 24C/48T, 165W or higher number SKU	Required	1

Ingredient	Requirement	Required/ Recommended	Quantity
Memory1	DRAM only configuration: 256 GB (16x 16 GB DDR5, 4800 MHz)	Required	16
Graphics	1 x Flex 170 or 2 x Flex 170 1 x Arc A770 or 2 x Arc A770 2 x Arc A750 3 x Flex 140	Required	1 (Required) Up to 3 (optional)
Storage (Boot Drive)	2 TB or equivalent boot drive	Required	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 System for Edge Verified Reference Blueprint—Medium 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 System for Edge Verified Reference Blueprint – Medium 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

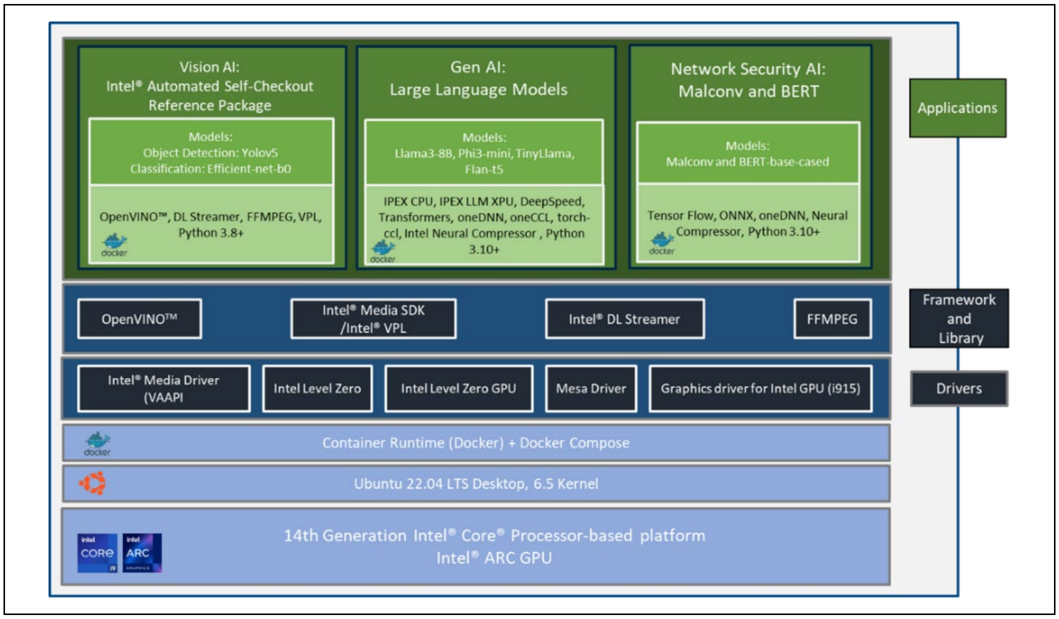
All three applications are containerized using docker.

For the Vision AI use case, we are using the Intel® Automated Self-Checkout application, which measures stream density. 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 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 and Intel® GPU.

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 System for Edge 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:
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 Level Zero
Intel 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 CORE
intel ARC
14th Generation Intel® Core® Processor-based platform
Intel® ARC GPU

The table below is a guide for assessing the conformance to the software requirements of the Intel® AI System for Edge 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 LTS
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

Ingredient	SW Version Details
DLStreamer	2024.0.1
FFmpeg	2023.3.0

2.4 Platform Technology Requirements

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

Enterprise AI requires 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® VT	Intel® CPU Virtual Machine Extension (VMX) Support	Enable	Optional
	Intel® I/O Virtualization	Enable	Optional
Intel® AMX	Intel® Advance 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.

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 and GPU Driver Setup

3.1 Boot Parameter Setup

For the workload testing, it is first necessary to set the host command line with appropriate boot parameters as well as 1GB of pages. In the “/etc/default/grub” file, update the line “GRUB_CMDLINE_LINUX” to include the following parameters:

“i915.force_probe=56c0” # Or replace the device ID with the corresponding hardware

Table 5. Boot Paramteres

Device	Intel® Flex 170	Intel® Flex 140	Intel® Arc A380	Intel® Arc A750
ID	56c0	56c1	56a5	56a1

After modifying in grub file, run “update-grub” and “reboot” to apply the changes and verify the change with “cat /proc/cmdline”:

```
cat /proc/cmdline

BOOT_IMAGE=/vmlinuz-6.5.0-44-generic root=UUID=aec107d6-c26d-4db4-a617-
117964a93819 ro i915.force_probe=56c0
```

3.2 Additional Linux Packages Installation

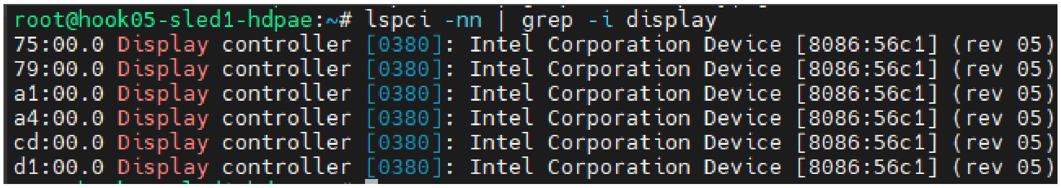
3.2.1 Install Docker

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

3.2.2 Install GPU Driver

Make sure the Intel® GPU is enumerated as a PCIe device.

Below is an example of 3 x Intel® Flex 140 GPUs (each Flex 140 card has 2 GPUs so a total of 6 is shown here)



```
root@hook05-sled1-hdpae:~# lspci -nn | grep -i display
75:00.0 Display controller [0380]: Intel Corporation Device [8086:56c1] (rev 05)
79:00.0 Display controller [0380]: Intel Corporation Device [8086:56c1] (rev 05)
a1:00.0 Display controller [0380]: Intel Corporation Device [8086:56c1] (rev 05)
a4:00.0 Display controller [0380]: Intel Corporation Device [8086:56c1] (rev 05)
cd:00.0 Display controller [0380]: Intel Corporation Device [8086:56c1] (rev 05)
d1:00.0 Display controller [0380]: Intel Corporation Device [8086:56c1] (rev 05)
```

3.2.2.1 Install Dependencies

```
$ sudo apt-get install -y gpg-agent wget

$ wget -qO - https://repositories.Intel®.com/gpu/Intel®-graphics.key | \
sudo gpg --dearmor --output /usr/share/keyrings/Intel®-graphics.gpg
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/Intel®-graphics.gpg]
https://repositories.Intel®.com/gpu/Ubuntu* jammy/lts/2350 unified" | \
sudo tee /etc/apt/sources.list.d/Intel®-gpu-jammy.list
```

3.2.2.2 Install Intel® GPU Driver

```
$ sudo apt-get update
$ sudo apt-get -y install \
    gawk \
    dkms \
    flex bison \
    linux-headers-$(uname -r) \
    linux-modules-extra-$(uname -r) \
    libc6-dev
```

3.2.2.3 Install the Intel® GPU kernel driver i915 and xpu manager

```
$ sudo apt install -y Intel®-i915-dkms Intel®-fw-gpu xpu-smi
```

Note: The Linux* kernel driver(s) provide the software connection to the Intel® GPU hardware. The kernel driver(s) are provided today as Dynamic Kernel Module Support (DKMS) drivers "out-of-tree" from the Linux kernel.

3.2.2.4 Install necessary graphics and media packages for the Intel® GPU

```
$ sudo apt-get install -y gawk libc6-dev udev\
  Intel®-opencl-icd Intel®-level-zero-gpu level-zero \
  Intel®-media-va-driver-non-free libmfx1 libmfxgen1 libvpl2 \
  libegl-mesa0 libegl1-mesa libegl1-mesa-dev libgbm1 libgl1-mesa-dev \
  libgl1-mesa-dri libglapi-mesa libgles2-mesa-dev libglx-mesa0 \
  libigdgmm12 libxatracker2 mesa-va-drivers \
  mesa-vdpau-drivers mesa-vulkan-drivers va-driver-all vainfo
```

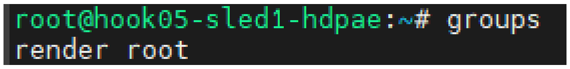
3.2.2.5 Reboot the server for the changes to take place in the OS

```
$ sudo reboot
```

3.2.3 Configure permissions on the OS groups for GPU as rendering device

```
$ sudo gpasswd -a ${USER} render
$ newgrp render
```

Verify if the render group is added as shown below:

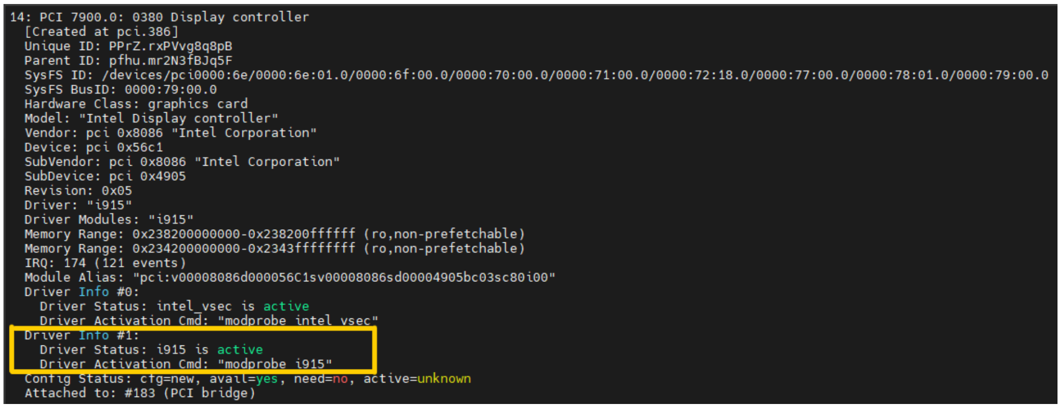


```
root@hook05-sled1-hdpae:~# groups
render root
```

3.2.4 Verify the installation to check if the GPU device is working with i915 driver

```
$ sudo apt-get install -y hwinfo
$ hwinfo --display
```

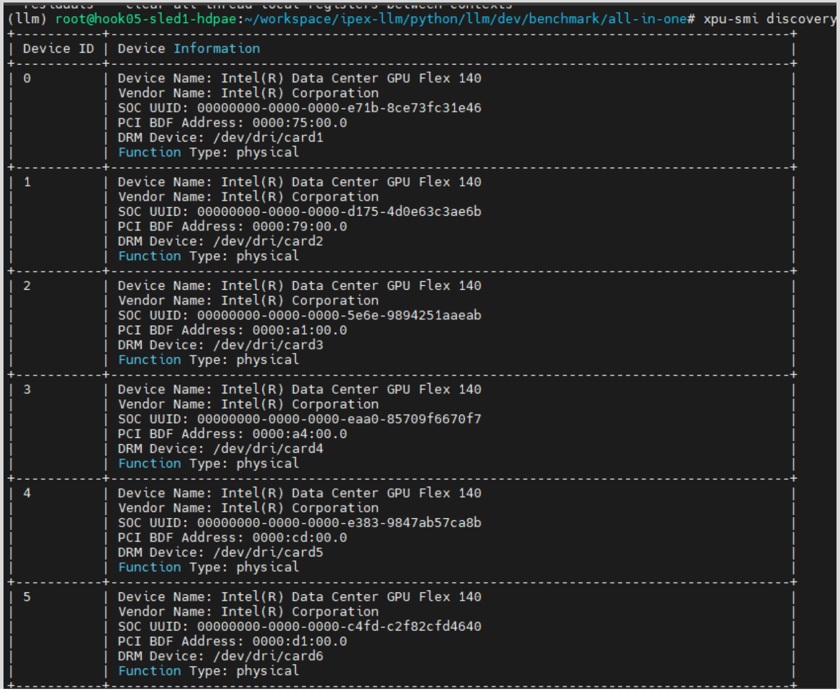
Verify if the i915 driver is active:



```
14: PCI 7900.0: 0380 Display controller
  [Created at pci.386]
  Unique ID: PPrZ.rxPVvg8q8pB
  Parent ID: pfhu.mr2N3fBJq5F
  SysFS ID: /devices/pci0000:6e/0000:6e:01.0/0000:6f:00.0/0000:70:00.0/0000:71:00.0/0000:72:18.0/0000:77:00.0/0000:78:01.0/0000:79:00.0
  SysFS BusID: 0000:79:00.0
  Hardware Class: graphics card
  Model: "Intel Display controller"
  Vendor: pci 0x8086 "Intel Corporation"
  Device: pci 0x56c1
  SubVendor: pci 0x8086 "Intel Corporation"
  SubDevice: pci 0x4905
  Revision: 0x05
  Driver: "i915"
  Driver Modules: "i915"
  Memory Range: 0x238200000000-0x238200ffffff (ro,non-prefetchable)
  Memory Range: 0x234200000000-0x2343ffffffff (ro,non-prefetchable)
  IRQ: 174 (121 events)
  Module Alias: "pci:v00008086d000056C1sv00008086sd00004905bc03sc80i00"
  Driver Info #0:
    Driver Status: intel_vsec is active
    Driver Activation Cmd: "modprobe intel_vsec"
  Driver Info #1:
    Driver Status: i915 is active
    Driver Activation Cmd: "modprobe i915"
  Config Status: cfg=new, avail=yes, need=no, active=unknown
  Attached to: #183 (PCI bridge)
```

Verify the GPU devices using xpu manager:

```
$ xpu-smi discovery
```



```
(llm) root@hook05-sled1-hdpae:~/workspace/ipex-llm/python/llm/dev/benchmark/all-in-one# xpu-smi discovery
+-----------+-------------------------------------------------------------------------------------+
| Device ID | Device Information                                                                  |
+-----------+-------------------------------------------------------------------------------------+
| 0         | Device Name: Intel(R) Data Center GPU Flex 140                                      |
|           | Vendor Name: Intel(R) Corporation                                                   |
|           | SOC UUID: 00000000-0000-0000-e71b-8ce73fc31e46                                      |
|           | PCI BDF Address: 0000:75:00.0                                                       |
|           | DRM Device: /dev/dri/card1                                                          |
|           | Function Type: physical                                                             |
+-----------+-------------------------------------------------------------------------------------+
| 1         | Device Name: Intel(R) Data Center GPU Flex 140                                      |
|           | Vendor Name: Intel(R) Corporation                                                   |
|           | SOC UUID: 00000000-0000-0000-d175-4d0e63c3ae6b                                      |
|           | PCI BDF Address: 0000:79:00.0                                                       |
|           | DRM Device: /dev/dri/card2                                                          |
|           | Function Type: physical                                                             |
+-----------+-------------------------------------------------------------------------------------+
| 2         | Device Name: Intel(R) Data Center GPU Flex 140                                      |
|           | Vendor Name: Intel(R) Corporation                                                   |
|           | SOC UUID: 00000000-0000-0000-5e6e-9894251aaeab                                      |
|           | PCI BDF Address: 0000:a1:00.0                                                       |
|           | DRM Device: /dev/dri/card3                                                          |
|           | Function Type: physical                                                             |
+-----------+-------------------------------------------------------------------------------------+
| 3         | Device Name: Intel(R) Data Center GPU Flex 140                                      |
|           | Vendor Name: Intel(R) Corporation                                                   |
|           | SOC UUID: 00000000-0000-0000-eaa0-85709f6670f7                                      |
|           | PCI BDF Address: 0000:a4:00.0                                                       |
|           | DRM Device: /dev/dri/card4                                                          |
|           | Function Type: physical                                                             |
+-----------+-------------------------------------------------------------------------------------+
| 4         | Device Name: Intel(R) Data Center GPU Flex 140                                      |
|           | Vendor Name: Intel(R) Corporation                                                   |
|           | SOC UUID: 00000000-0000-0000-e383-9847ab57ca8b                                      |
|           | PCI BDF Address: 0000:cd:00.0                                                       |
|           | DRM Device: /dev/dri/card5                                                          |
|           | Function Type: physical                                                             |
+-----------+-------------------------------------------------------------------------------------+
| 5         | Device Name: Intel(R) Data Center GPU Flex 140                                      |
|           | Vendor Name: Intel(R) Corporation                                                   |
|           | SOC UUID: 00000000-0000-0000-c4fd-c2f82cfd4640                                      |
|           | PCI BDF Address: 0000:d1:00.0                                                       |
|           | DRM Device: /dev/dri/card6                                                          |
|           | Function Type: physical                                                             |
+-----------+-------------------------------------------------------------------------------------+
```

§

4 Performance Verification

This chapter aims to verify the performance metrics for the Intel® AI System for Edge 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 Plus solution was tested on August 06, 2024, with the following hardware and software configurations:

- 1 NUMA node
- 1 x Intel® Xeon® Gold 6538N 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* 22.04.1 (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 6. 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 7. 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 Retail Self-Checkout

Retail Self-Checkout is an implementation that provides critical components to build and deploy a self-checkout use case using Intel® hardware, software, and other open-source software. 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.

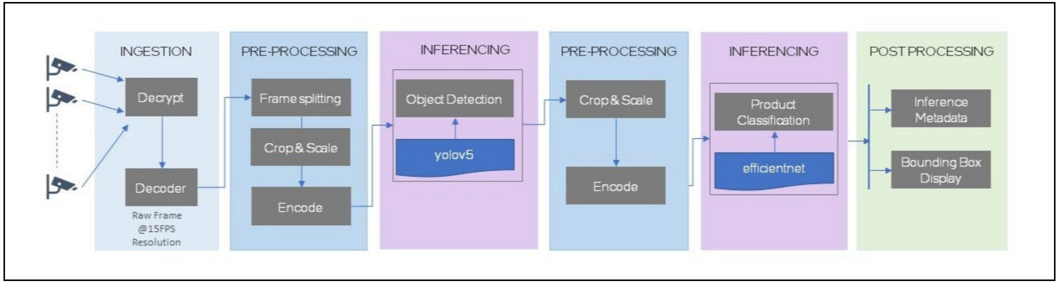


Figure 2. Retail Self-checkout Video Analytics Pipeline

Intel® AI System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI configuration, the platform CPU with AMX should be able to process up to 23 number of streams at 4K @ 14.95FPS with HEVC codec, and up to 21 number of streams when equipped with Intel® Flex 170 GPU.

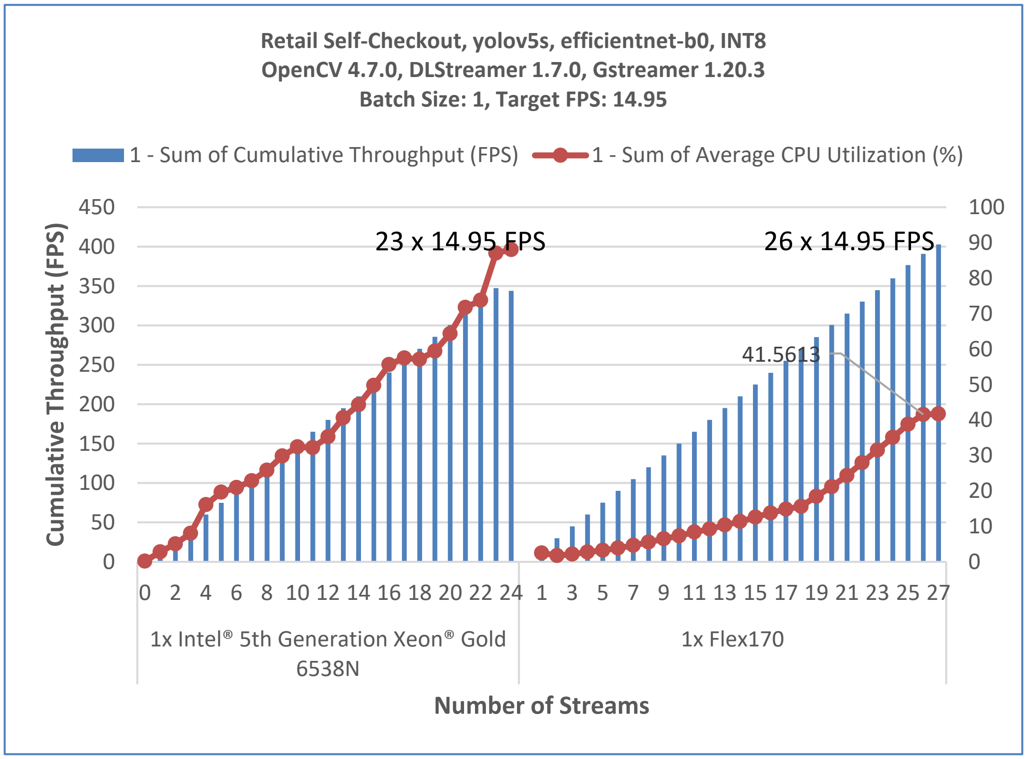
Table 8. Retail Self-checkout

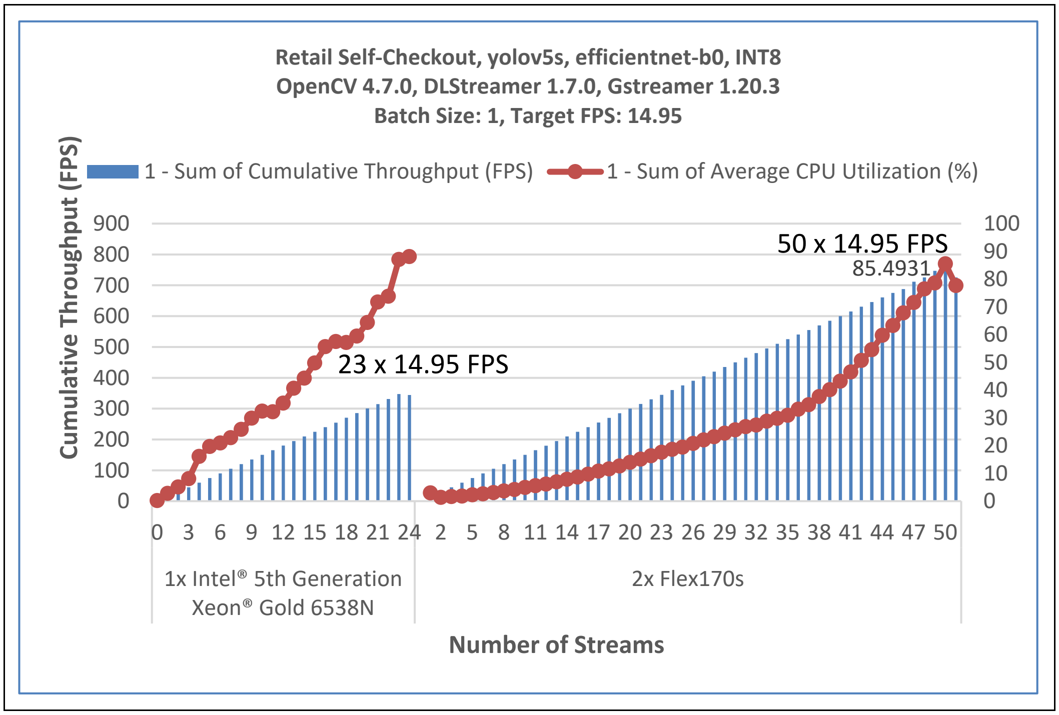
Ingredient	Software Version Details
OpenVINO	2024.0.1
DLStreamer	2024.0.1
FFMPEG	2023.3.0
VPL	2023.4.0.0-799
Python	3.8+

Table 9. Retail Self-Checkout Medium Configuration Performance

Configuration1	Intel® CPU Xeon® 6538N	Intel® Flex 170	2 x Intel® Flex 170
Plus Configuration (# of streams)	23	26	50

Figure 3. Retail Self-Checkout Medium Configuration Performance Graph on Xeon® 6538N and Intel® Flex 170





4.3 Generative AI

Intel® Generative AI solution provide 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® Enterprise AI solution with State-Of-The-Art foundational models such as GPT-NEOX-20B, Llama 3 8B, Phi3, and TinyLlama.

To ensure Generative 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.

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 System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI configuration, the system should be able to deliver results as shown in Table 10 and Table 11 as a baseline to the expected performance of this solution.

Table 10. Performance of Various Large Language Models on 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

Table 11. Performance of Various Large Language Models on Intel® Data Center Flex GPU:

Models	GPU	Precision	Input Tokens	Batch Size	Throughput (tokens/s)	Inference time
Phi-3-mini	Flex 170	INT4-8	32-256	8	257-346	< 60s
Llama 3 8B	Flex 170	INT4-8	32-256	8	188-215	< 60s
Llama 3 8B	2 x Flex 170	INT8-FP16	32-256	8	230-246	< 60s
Llama 3 8B	2 x Flex 170	FP8	Variable	Variable	748	N/A
Llama 3 8B	3 x Flex 140	INT4	32-256	1	18	< 60s
Phi-3-mini	3 x Flex 140	INT4	32-256	2	39-40	< 60s

Figure 4. Performance for GPT-NEOX-20B model on CPU

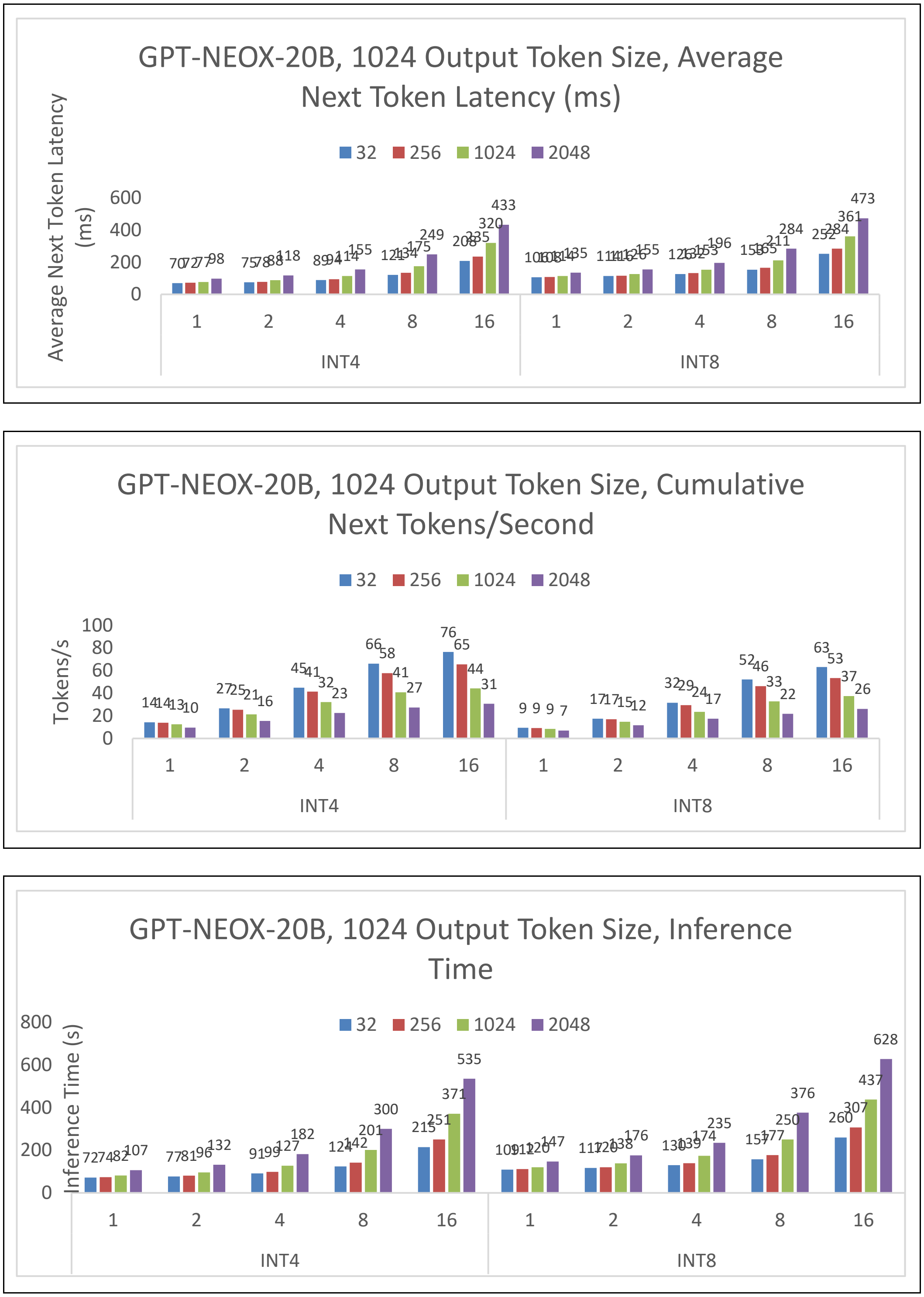


Figure 5. Performance for Llama 3 8B Model on CPU

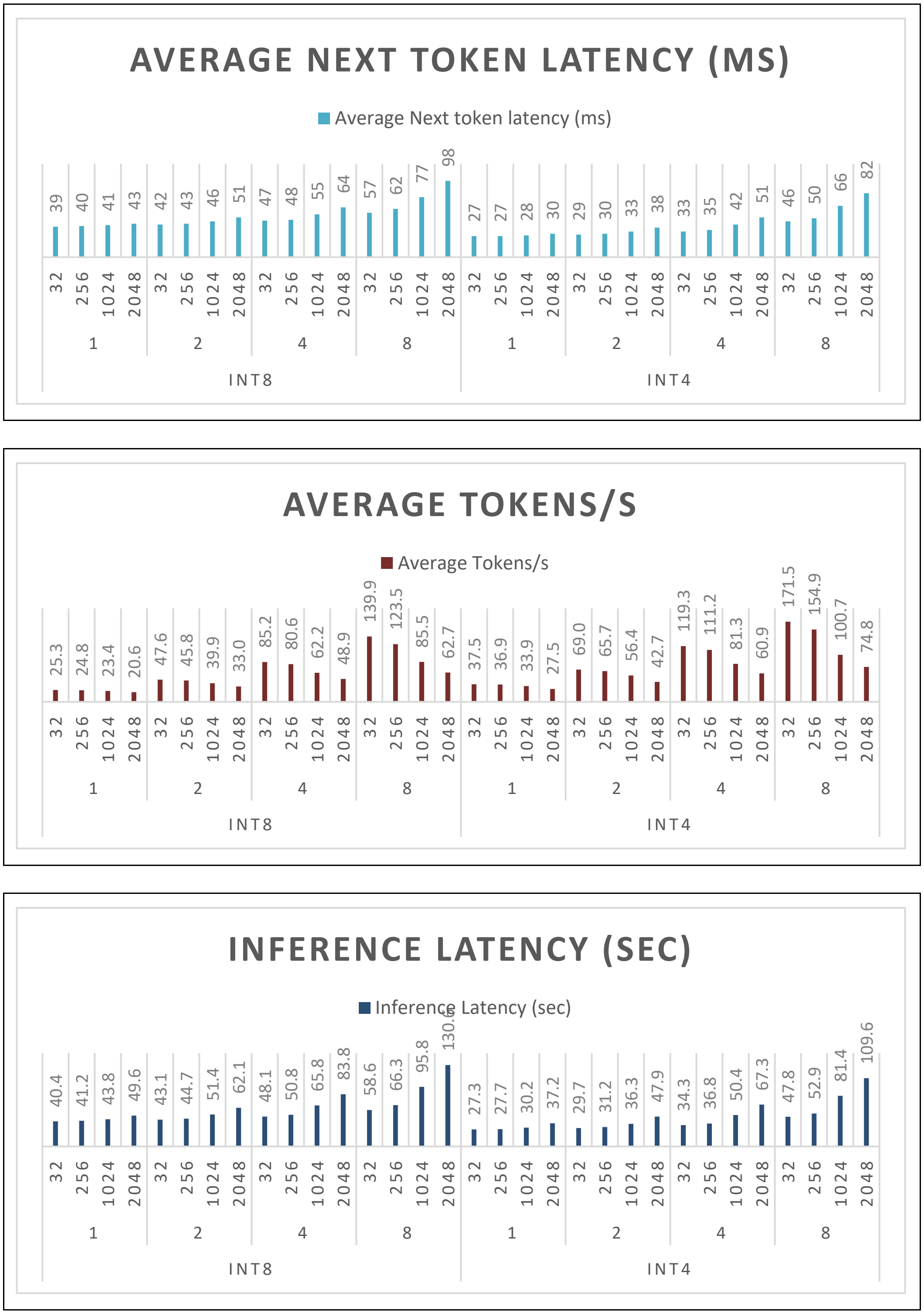


Figure 6. Performance for Llama 3 8B Model on Intel® Flex 170

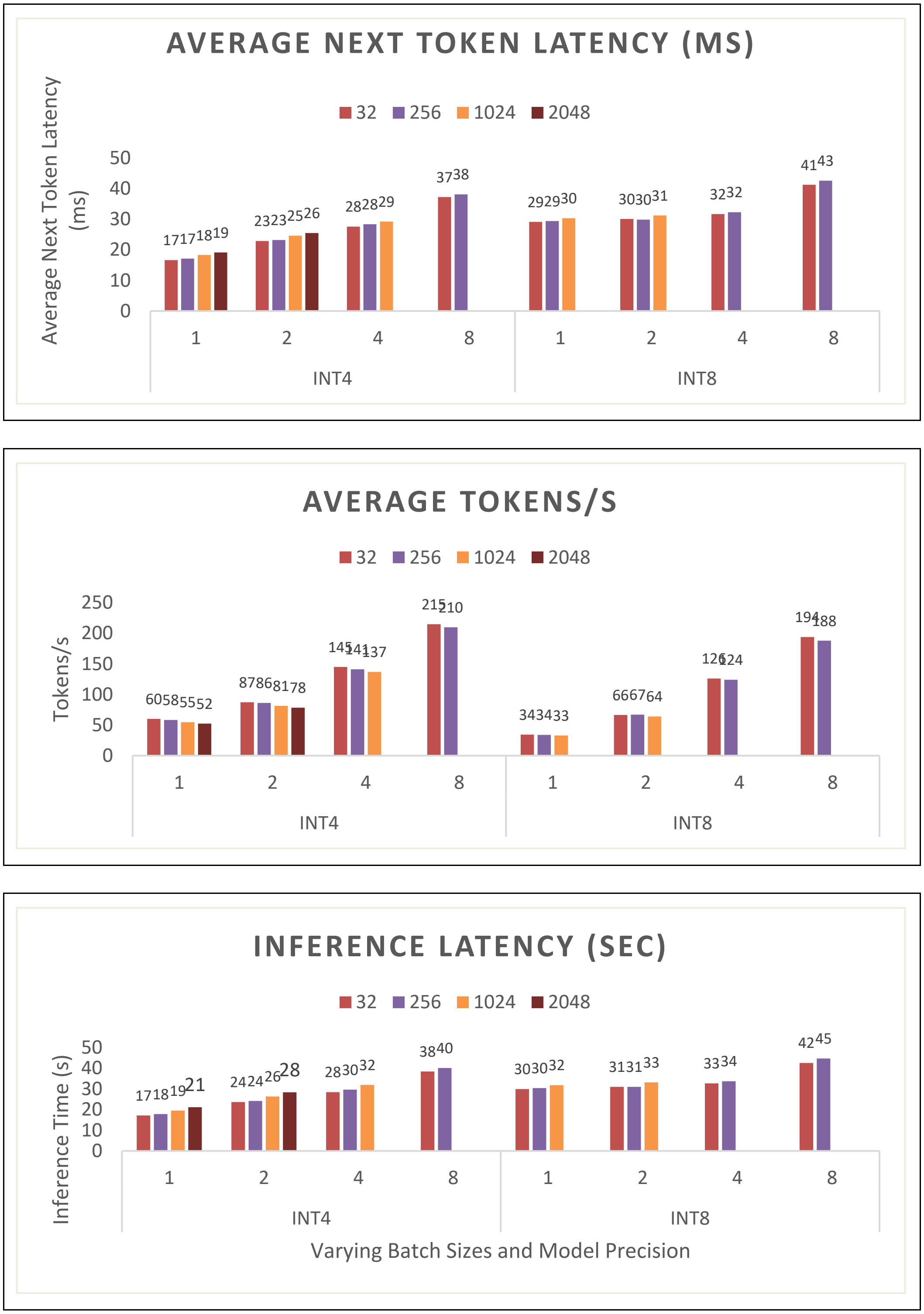


Figure 7. Performance for Phi-3-4k Mini Model on CPU

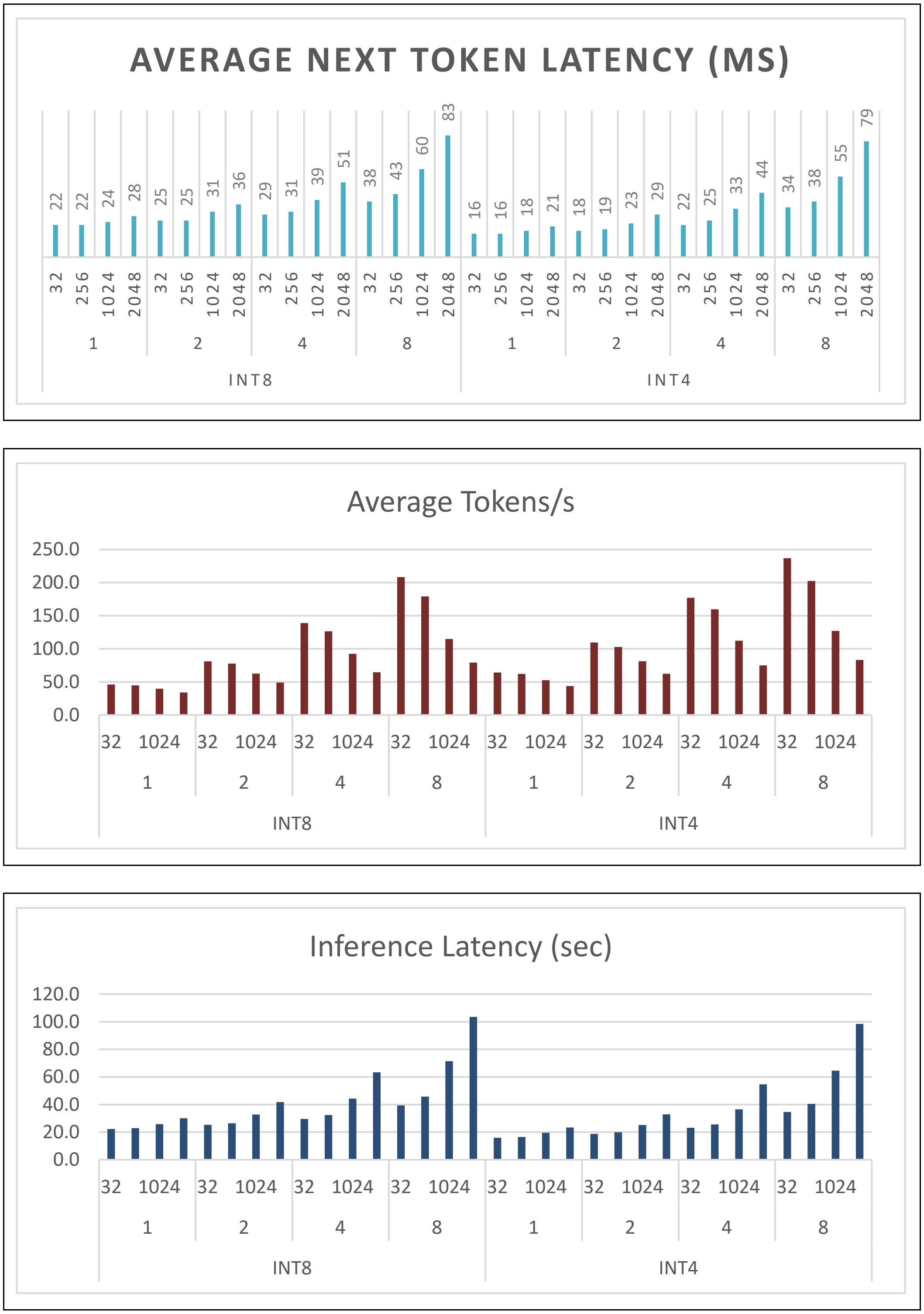


Figure 8. Performance for Phi-3-4k-Mini Model on Intel® Flex 170

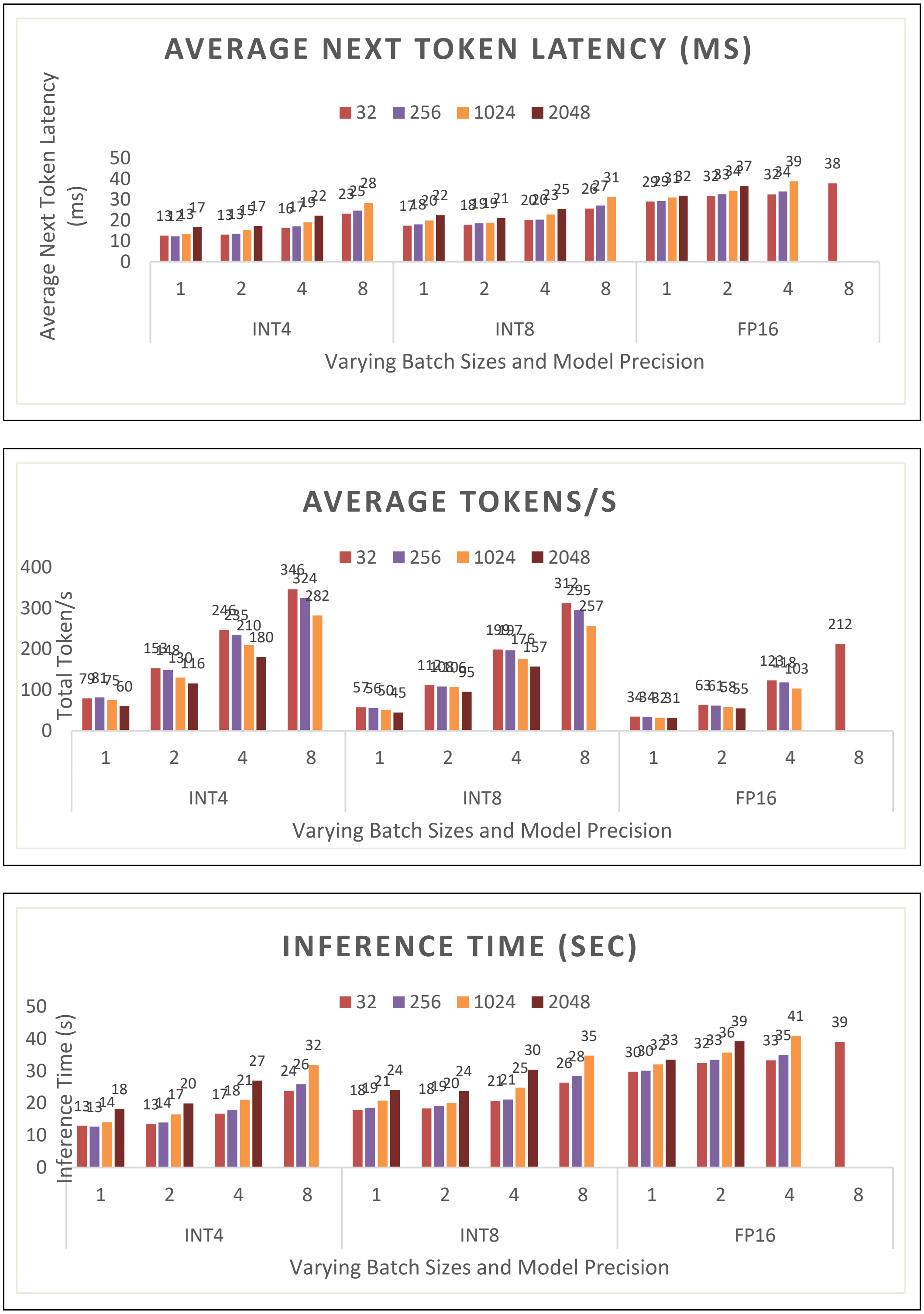
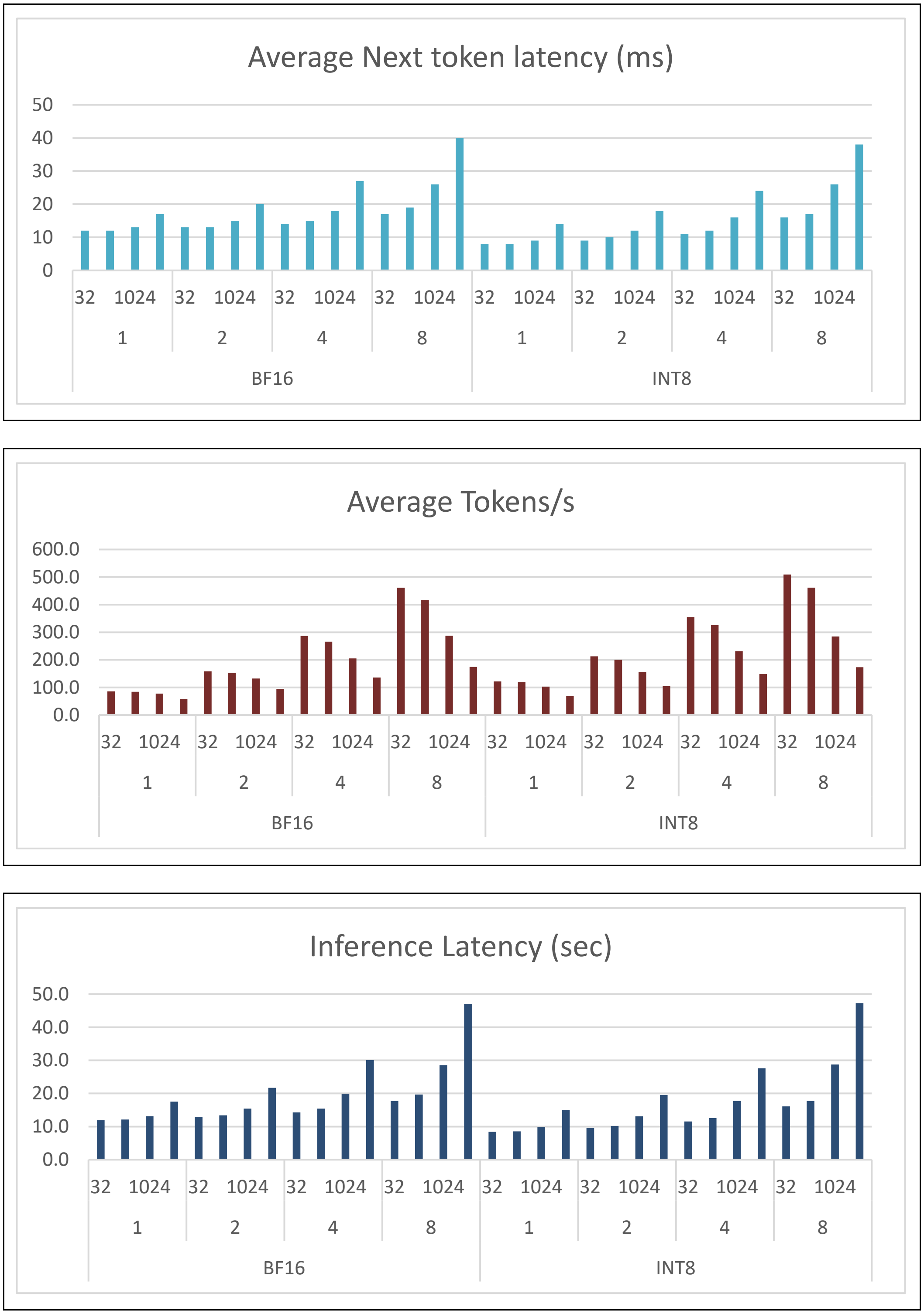


Figure 9. Performance on TinyLlama Model on CPU



Average Next token latency (ms)

Average Tokens/s

Inference Latency (sec)

Figure 10. vLLM-IPEX-LLM Performance with Llama 3 8B Model on CPU

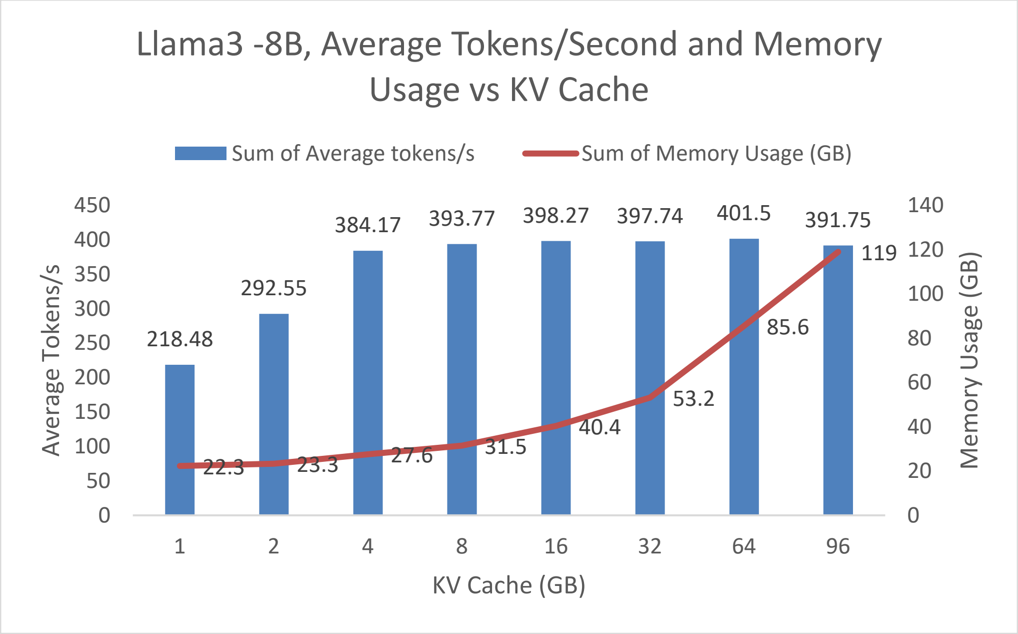
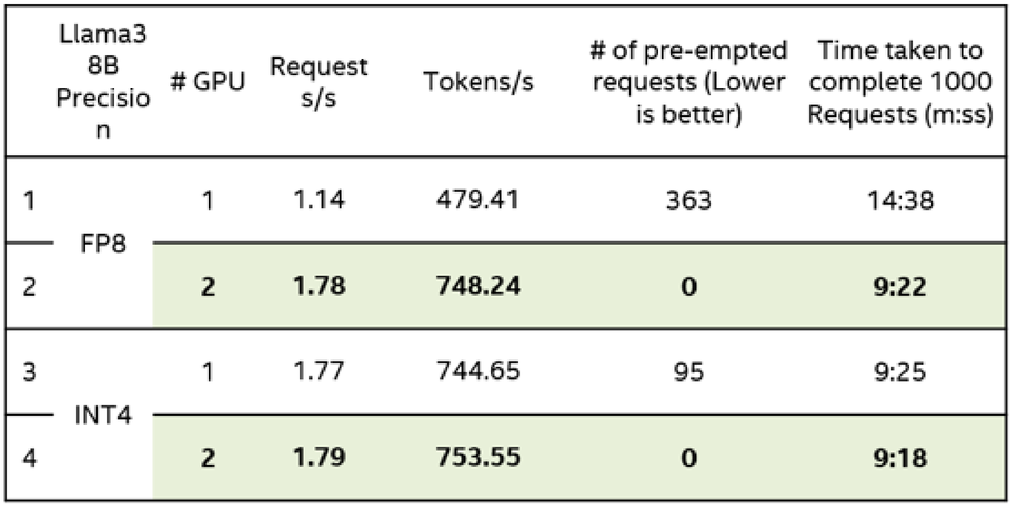
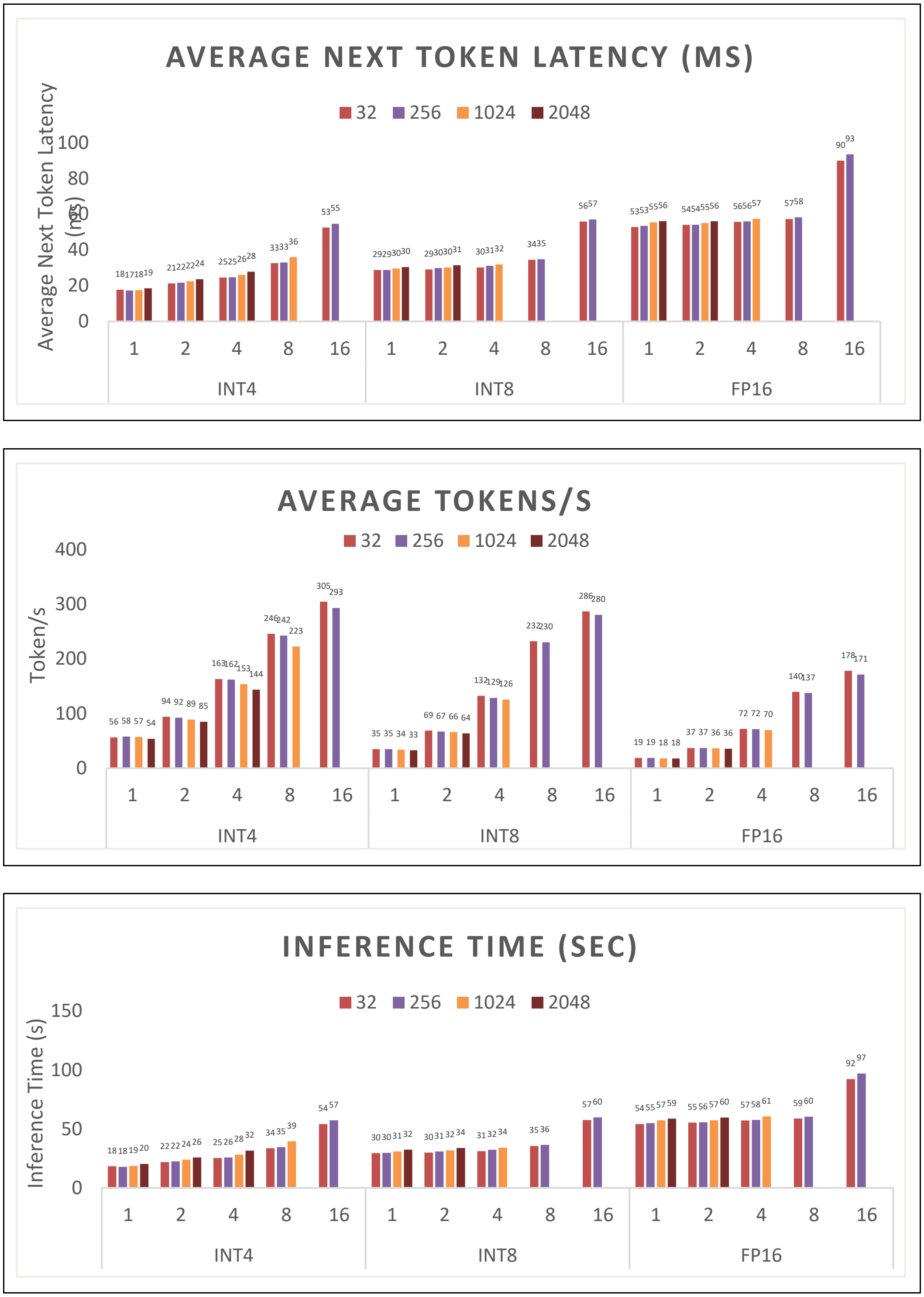


Figure 11. vLLM-IPEX-LLM Performance with Llama 3 8B Model on 2 x Flex 170 (1024 Output Token Size)



	Llama3 8B Precisio n	# GPU	Request s/s	Tokens/s	# of pre-empted requests (Lower is better)	Time taken to complete 1000 Requests (m:ss)
1	FP8	1	1.14	479.41	363	14:38
2		**2**	**1.78**	**748.24**	**0**	**9:22**
3	INT4	1	1.77	744.65	95	9:25
4		**2**	**1.79**	**753.55**	**0**	**9:18**

Figure 12. Llama 3 8B Performance on 2 x Flex 170 with Pipeline Parallel Configuration



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.

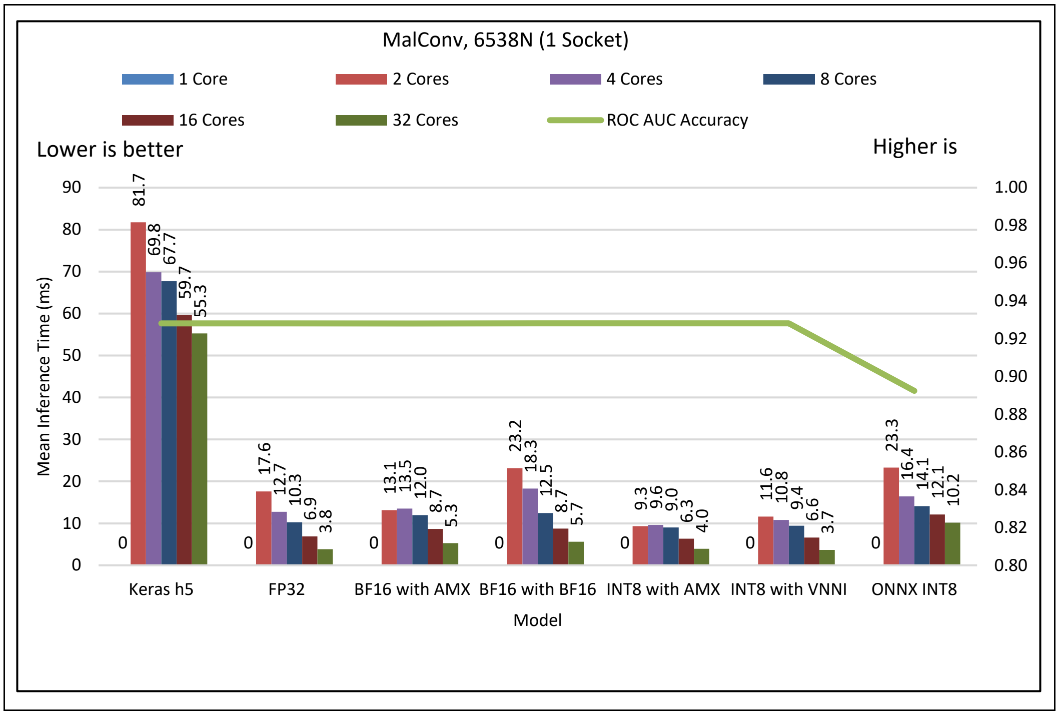
Ensure that the test results follow the expected results, as shown in the following tables, to baseline the platform's performance. Table 12 shows the software used for the testing, while Figure 14 shows a graph of the mean inference time for each model. With 2 cores per instance, the INT8 model with 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 12. 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 13. 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.

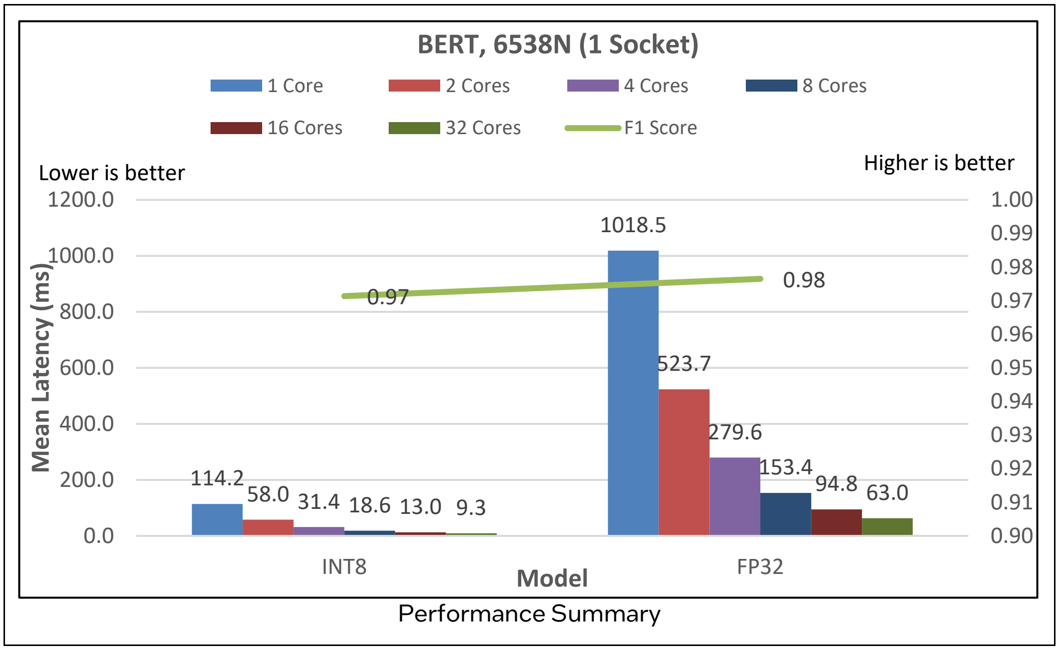
Note: The phishing email test dataset can be found at https://github.com/IBM/nlc-email-phishing/tree/master/data

Table 13. BERT AI Workload Configuration

Ingredient	Software Version Details
Torch	2.1.2

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

Figure 14. BERTAI Performance on VRC for Intel® AI System – Medium Entry Configuration Graph



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

§

5 Summary

The Intel® AI System for Edge Verified Reference Blueprint – Medium for Computer Vision, and GEN AI defined on single socket 5th Gen Intel® Xeon® Scalable processors with multiple Intel® Data Center Flex GPUs addresses the capabilities for AI Inference offering the following value proposition:

1. For Vision AI use case using Processor AI acceleration only

- Up to 23 IP camera streams of Vision AI use case with the Intel® Retail Checkout application on Large
- Up to 52 IP camera streams of Vision AI use case with the Intel® Retail Checkout application on Large Plus configuration on 2x Flex 170 GPU
- Up to 38 IP camera streams of Vision AI use case with the Intel® Retail Checkout application on Large Base configuration on 2x Flex 140 GPU

2. For Generative AI use case

With Processor AI inference offload

- Up to 670 tokens/s on Llama3 8B model with INT8 precision Batch size of 32 on Large Plus CPU configuration
- Up to 14 tokens/s on GPT-NEOX-20B model with INT4 precision Batch size of 1 on Medium Plus CPU configuration

With GPU AI inference Offload

- Up to 236-257 tokens/s on Llama3 8B model with INT4/INT8 precision Batch size of 8 on 2 Flex 170 GPU
- Up to 364-448 tokens/s on Phi3-mini-4K instruct model with INT4/INT8 precision Batch size of 8 on 2 Flex 170 GPU
- Up to 1022-1096 tokens/s on TinyLlama model with INT4/INT8 precision Batch size of 8 on 2 Flex 170 GPU
- Up to 188-255 tokens/s on Llama3 8B model with INT4/INT8 precision Batch size of 8 on 1 Flex 170 GPU
- Up to 294-361 tokens/s on Phi3-mini-4K instruct model with INT4/INT8 precision Batch size of 8 on 1 Flex 170 GPU
- Up to 731-830 tokens/s on TinyLlama model with INT4/INT8 precision Batch size of 8 on 1 Flex 170 GPU

3. For Network Security AI use case

- Malconv testing, the INT8 model with AVX512_CORE_AMX enabled was able to reach a performance of less than 10 ms. with 2 cores per instance.
- Bert testing, the mean latency of the INT8 model reaches below 20ms with 8 cores per instance.

This Configuration 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.

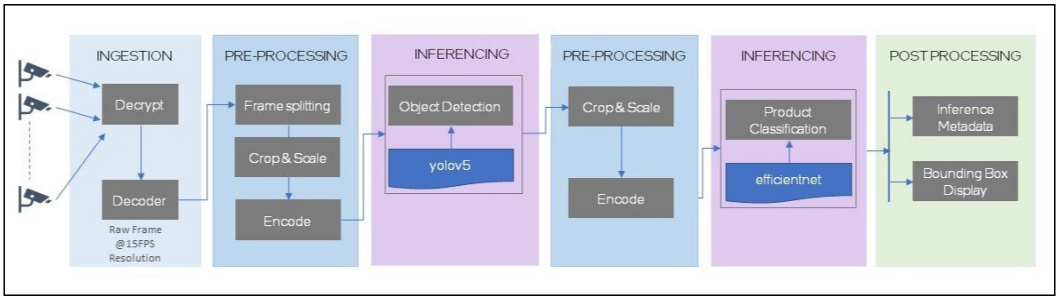
These processors are optimized for network, cloud native, wireline, and wireless core-intensive workloads, and are especially suited for AI workloads coupled with Intel® Ethernet E810-Network Controllers and Intel® Data Center Flex GPUs.

§

Appendix A Appendix

A.1 Retail Self-checkout Vision AI Test Methodology

Figure 15. Test Methodology for Retail Self-checkout Pipeline



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 below, the video data is ingested, pre-processed before each inferencing step, inferenced using two models - YOLOv5 and EfficientNet, and post-processed to generate metadata and show the bounding boxes for each frame.

Pre-Requisites

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

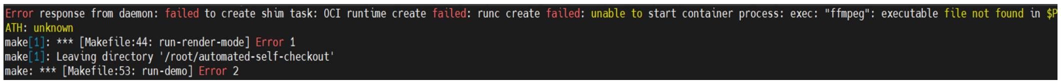
Quick Setup

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

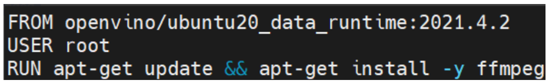
Issue #1: Binary 'ffmpeg" does not exist in OpenVINO container.



```
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 IPEX-LLM Testing Methodology on GPU

Pull and start the container.

```
$ export DOCKER_IMAGE=Intel®analytics/ipex-llm-serving-xpu:2.1.0-SNAPSHOT
$ export CONTAINER_NAME=ipex-llm-serving-xpu
$ export MODEL_PATH=<YOUR PATH TO THE MODEL WEIGHTS>
$ docker pull Intel®analytics/ipex-llm-serving-xpu:2.1.0-SNAPSHOT
$ docker run -itd \
             --net=host \
             --device=/dev/dri \
             --memory="64G" \
             --name=$CONTAINER_NAME \
             --shm-size="16g" \
             -v $MODEL_PATH:/llm/models \
             $DOCKER_IMAGE
```

Note: Ensure that you have assign enough memory via the --memory tag as the model(s) will be loaded to the container memory before moving to the GPUs.

Enter the container via bash terminal:

```
$ docker exec -it ipex-llm-serving-xpu bash
```

Enter the predefined benchmark script directory:

```
$ cd /benchmark/all-in-one
```

A.2.3 Running IPEX-LLM Benchmarking Scripts

A.2.3.1 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. Shared 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.2.3.2 Running IPEX-LLM on Single GPU

The Generative AI benchmark on Intel® Data Center GPU Flex 170 leverages the IPEX-LLM framework and is deployed in a containerized manner.

To run the Generative AI benchmark on Intel® Data Center GPU Flex 170:

18. Download the IPEX-LLM container image:

```
# export DOCKER_IMAGE=Intel®analytics/ipex-llm-serving-xpu:2.1.0-SNAPSHOT
# docker pull Intel®analytics/ipex-llm-serving-xpu:2.1.0-SNAPSHOT
```

19. Launch the IPEX-LLM container. For example, to benchmark with the Meta Llama3-8B model:

```
# export CONTAINER_NAME=ipex-llm-serving-xpu
# export MODEL_PATH=~/llama3-8b
# docker run -itd \
             --net=host \
             --device=/dev/dri/card0 \
             --device=/dev/dri/renderD128 \
             --memory="64G" \
             --name=$CONTAINER_NAME \
             --shm-size="16g" \
             -v $MODEL_PATH:/llm/models \
             $DOCKER_IMAGE bash
```

20. Copy the run-arc-sweep.sh script to the container:

```
# docker cp ~/applications.platform.Intel®-select-for-
network/enterprise_ai/common/ipex-llm-gpu/run-arc-sweep.sh ipex-llm-
serving-xpu:/benchmark/all-in-one/
```

21. Login to the container and update the run-arc-sweep.sh script to use the appropriate model. For example, to benchmark with the Meta Llama3-8B model:

```
# docker exec -it ipex-llm-serving-xpu /bin/bash
# cd /benchmark/all-in-one/
# $EDITOR run-arc-sweep.sh

…
current_model_name="llama3-8b"
…
```

22. Login to the container and start the benchmark:

```
# bash run-arc-sweep.sh
```

23. Review the benchmark results:

```
# cat optimize_model_gpu-results*.csv
```

A.2.3.3 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.

§