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

Reference Architecture

Revision 1.0
October 2024

Authors
Abhijit Sinha
Yuan Kuok Nee
Ai Bee Lim

Key Contributors
Timothy Miskell
Shin Wei Lim
Jonathan Tsai
Jessie Ritchey
Edel Curley

Contents

Figures

Tables

Revision History

Document Number	Revision Number	Description	Revision Date
834789	1.0	Initial release	October 2024

§

1 Introduction

Intel® Enterprise AI systems are a range of optimized commercial AI systems that are delivered and sold through OEM/ODM in the Intel® ecosystem. They are commercial platforms that are 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 offer 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 using the dual socket 5th Gen Intel® Xeon® Scalable processor family and Intel® Data Center Flex GPUs.

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 – Large for Computer Vision, and GEN AI is based on dual socket 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", "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 – Large for Computer Vision, and GEN AI covering a base and plus configuration:

- Intel® AI System for Edge Verified Reference Blueprint – Large for Computer Vision, and GEN AI Plus configuration for the Node is defined with at least a 64-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 capable to support multiple Intel® Data Center Flex GPUs.

- Intel® AI System for Edge Verified Reference Blueprint – Large for Computer Vision, and GEN AI Base configuration for the Node 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 capable to support multiple Intel® Data Center Flex GPUs.

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 Blueprint 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 – Large for Computer Vision, and GEN AI.

2.1 Platform Requirements

The checklists in this chapter are a guide for assessing the platform's conformance to Intel® AI System for Edge Verified Reference Blueprint – Large 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 – Large for Computer Vision, and GEN AI - Plus Configuration

Ingredient	Requirement	Required/ Recommended	Quantity
Processor	Intel® Xeon® Platinum 8592+ Processor at 1.9GHz, 64C/128T, 350W or higher number SKU	Required	2
Memory	16x 32 GB DDR5, 5600 MHz (512 GB total)	Required	16 (8 per NUMA)
Storage (Boot Drive)	480 GB or equivalent boot drive	Required	1
Storage (Capacity)	1 TB or equivalent drive (recommended Non-Uniform Memory Access (NUMA) aligned)	Recommended	4 (2 per NUMA)
GPU	Intel® Data Center GPU Flex 140/170	Required	2 (1 per NUMA)
Network	Intel® Ethernet Network Adapter E810-2CQDA2	Required	1
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for Pre-boot Execution Environment (PXE) and Operation, Administration and Management (OAM)	Required	2 (1 per NUMA)
	1/10 Gbps port for Management Network Interface Controller (NIC)	Required	1

Table 2. Intel® AI System for Edge Verified Reference Blueprint – Large for Computer Vision, and GEN AI – 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	2
Memory	32 GB DDR5, 5200 MHz (512 GB total)	Required	16 (8 per NUMA)
Storage (Boot Drive)	480 GB or equivalent boot drive	Required	1

Ingredient	Requirement	Required/ Recommended	Quantity
Storage (Capacity)	1 TB or equivalent drive (recommended NUMA aligned)	Recommended	2 (1 per NUMA)
GPU	Intel® Data Center GPU Flex 140/170	Required	2 (1 per NUMA)
Network	Intel® Ethernet Network Adapter E810-CQDA2	Recommended	1
LAN on Motherboard (LOM)	10 Gbps or 25 Gbps port for PXE/OAM	Required	2
	1/10 Gbps port for Management Network Interface Controller (NIC)	Required	1

2.2 BIOS Settings

To meet the performance requirements for an Intel® AI System for Edge Verified Reference Blueprint – Large for Computer Vision, and GEN AI, Intel® recommends using the BIOS settings for enabling 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).*

Table 3. Recommended BIOS Settings

Setting	Value
Hardware Prefetcher	Enabled
Intel® (VMX) Virtualization Technology	Enabled
Hyper-Threading	Enabled
Intel® Speed Shift Technology (P-States)	Enabled
Turbo Mode	Enabled
C-States	Enabled
Enhanced C-States	Enabled

Additionally, for this specific solution please make the corresponding addition BIOS changes on top of NFVI BIOS Profile to yield optimize performance for the solution configurations.

Table 4. Additional BIOS Settings on top of NFVI BIOS Profile

Setting	Value
Sub NUMA Clustering	SNC2
AVX P1	Level 2
AVX ICCP Pre-grant License	Enabled

Setting	Value
AVC ICCP Pre-Grant Level	512 Heavy
Memory Page Policy	Adaptive

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

2.3 Solution Architecture

Figure 1 shows the architecture diagram of the Intel® AI System for Edge Verified Reference Blueprint. The software stack consists of three categories of AI software:

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

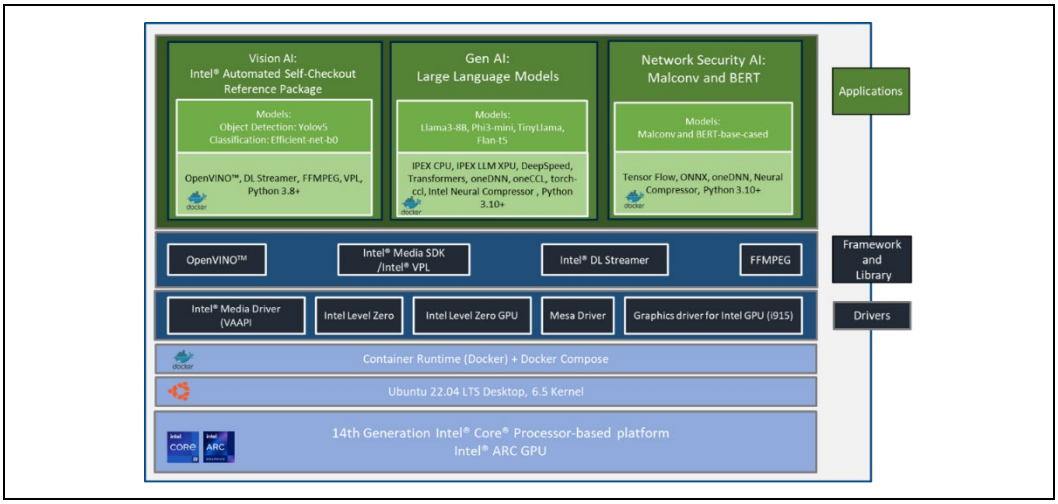
All three applications are containerized using docker.

For the Vision AI use case, we are using the Intel® Automated Self-Checkout application which measures the stream density. The video data is ingested and pre-processed before each inferencing step. The inference is performed using two models - YOLOv5 and EfficientNet, where the YOLOv5 model does the object detection and the EfficientNet model performs the Object Classification.

For the Generative AI use case, we are using Large Language Models (LLMs) using Intel® Extension of PyTorch (IPEX) framework for performing LLM inference on Intel® Xeon® Scalable Processor based CPUs and Intel® Data Center Flex GPUs.

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



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 5. 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
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 Intel® AI System for Edge Verified Reference Blueprint requires Intel® Virtualization Technology (VT) to be enabled to reap the benefits of hardware virtualization. Either Intel® Boot Guard or Intel® Trusted Execution Technology establishes the firmware verification, allowing for platform static root of trust.

Table 6. Platform Technology Requirements

Platform Technologies		Enable/Disable	Required/Recommended
Intel® VT	Intel® CPU Virtual Machine Extension (VMX) Support	Enable	Required
	Intel® I/O Virtualization	Enable	Required
Intel® AMX	Intel® Advanced Matrix Extenstions	Enable	Required
Intel® Boot Guard	Intel® Boot Guard	Enable	Required

Platform Technologies		Enable/Disable	Required/Recommended
Intel® TXT	Intel® Trusted Execution Technology	Enable	Recommended

2.5 Platform Security

For Intel® AI System for the Edge, it is recommended that Intel® Boot Guard Technology 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 for Worker Node

3.1 Boot Parameter Setup

For the workload testing, note that it is not necessary to enable hugepage support nor is it necessary to enable isolcpu support. If SR-IOV will be utilized, then in the "/etc/default/grub" file update the line "GRUB_CMDLINE_LINUX" to include the following parameters:

"intel_iommu=on iommu=pt"

After modifying the 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 intel_iommu=on iommu=pt quiet splash vt.handoff=7
```

3.2 Installing the i915 Driver

Follow the instructions provided below to install the Intel® i915 GPU DKMS Driver:

Install the prerequisites to add the necessary repository access.

```
# sudo apt update

# sudo apt install -y gpg-agent wget
```

Add the online network package repository.

```
. /etc/os-release

if [[ ! " jammy " =~ " ${VERSION_CODENAME} " ]]; then

    echo "Ubuntu version ${VERSION_CODENAME} not supported"

else

    wget -qO - https://repositories.intel.com/gpu/intel-graphics.key | \

    sudo gpg --yes --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
${VERSION_CODENAME}/lts/2350 unified" | \

    sudo tee /etc/apt/sources.list.d/intel-gpu-${VERSION_CODENAME}.list
```

```
    sudo apt update

fi
```

Install the kernel and Intel® XPU System Management Interface (XPU-SMI) packages on a bare metal system. Installation on the host is sufficient for hardware management and support of the runtimes in containers and bare metal.

```
# sudo apt install -y \

    linux-headers-$(uname -r) \

    linux-modules-extra-$(uname -r) \

    flex bison \

    intel-fw-gpu intel-i915-dkms xpu-smi

# sudo reboot
```

Install packages responsible for computing and media runtimes.

```
# sudo apt install -y \

    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 hwinfo
clinfo
```

Install the development packages.

```
# sudo apt install -y \

    libigc-dev intel-igc-cm libigdfcl-dev libigfxcmrt-dev level-zero-dev
```

List the group assigned ownership of the render nodes and the groups you are a member of. There are specific groups that users must be a part of to access certain functionalities of the GPU. The render group specifically allows access to GPU resources for the rendering tasks without giving full access to display management or other potentially more sensitive operations.

```
# stat -c "%G" /dev/dri/render*

# groups ${USER}
```

If you are not a member of the same group used by the DRM render nodes, add your user to the render node group.

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

Change the group ID of the current shell.

```
# newgrp render
```

3.3 Kubernetes Installation

3.3.1 Install Docker and cri-dockerd

Follow the instructions at https://docs.docker.com/engine/install/ubuntu/ to install Docker Engine on Ubuntu, and follow the instructions at https://www.mirantis.com/blog/how-to-install-cri-dockerd-and-migrate-nodes-from-dockershim/ to install cri-dockerd. Download the cri-dockerd binary package for version 0.3.4.

3.3.2 Install Kubernetes

Follow the instructions at https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/ to install Kubernetes including the kubelet, kubeadm, and kubectl packages. To continue to initialize the Kubernetes cluster, follow the steps below:

Note that setup does not use swap memory so it must be disabled

```
# swapoff -a

# systemctl enable --now kubelet

# systemctl start kubelet



# cat <<EOF > /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

EOF

# sysctl --system
```

In the below command, update the Kubernetes version being used and the host-ip to that of the system being used

```
# kubeadm init --kubernetes-version=v1.28.0 --pod-network-
cidr=10.244.0.0/16 --apiserver-advertise-address=<host-ip> --token-ttl 0
--ignore-preflight-errors=SystemVerification --cri-
socket=unix:///var/run/cri-dockerd.sock
```

3.3.3 Install Calico

Follow the instructions at https://docs.tigera.io/calico/latest/getting-started/kubernetes/quickstart to install Calico. In the second step of the “Install Calico”

section, the cidr address of the file needs to be modified, so run the following steps instead of step 2 listed in the instructions:

Update the URL if necessary

```
# wget
https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/
custom-resources.yaml
```

Update the cidr address in the "custom-resources.yaml" file to 10.244.0.0/16

```
# kubectl create -f custom-resources.yaml
```

Once completed, wait for the Calico pods to be running before starting to use the cluster.

§

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 Large Base and Plus solution was tested on August 30, 2024, with the following hardware and software configurations.

Table 7. Hardware and Software Configurations for Large Base and Plus

Hardware Config	Large Base	Large Plus
CPU	2x Intel® Xeon® Gold 6538N Processor	2x Intel® Xeon® Platinum EMR 8592+ Processor
Sockets	2	2
Cores per Socket	32	64
LLC Cache	60MB Cache	320MB Cache
TDP per CPU	205W	350W
Simultaneous Multithreading (SMT)	Intel® Hyper-Threading Technology Enabled	Intel® Hyper-Threading Technology Enabled
CPUs	128	256
CPU Frequency	2.1 GHz base clock speed 4.1 GHz max turbo frequency	1.9 GHz base clock speed, 2.9 GHz all-core turbo frequency, 3.9 GHz max turbo frequency,
NUMA Nodes	2	2
Hyperthreading	Enable	Enable
Turbo	Enable	Enable
C-State	Enable	Enable
Total Memory	16x32GB 512GB, DDR5-5200 MT/s, 1DPC, 8 channels	16x32GB 512GB, DDR5-5600 MT/s, 1DPC, 8 channels
Hard Drive/ Disk	1x 447.1G INTEL SSDSC2KB48	2x 447.1G INTEL SSDSC2KB48
GPU	Intel® Data Center GPU Flex 140/170	Intel® Data Center GPU 140/170
Network Interface Card/AIC	1x Dual port Intel® Ethernet Network Adapter E810-2CQDA2 2x Ethernet Connection X722 for 10GBASE-T	1x Dual port Intel® Ethernet Network Adapter E810-2CQDA2 2x Ethernet Connection X722 for 1
Network speed	1 GbE	1 GbE
Microcode	0x21000240	0x21000240
OS/Software	Ubuntu 22.04.4 (kernel 6.5.0-18-generic)	Ubuntu 22.04.4 (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/intelr-memory-latency-checker.html. Download the latest version, unzip the tarball package, go into the Linux* folder, and execute `./mlc`. Table 8 and Table 9 below should be used as a reference for verifying the validity of the system setup.

Table 8. Memory Latency Checker

Key Performance Metric	Local Socket (Base)	Local Socket (Plus)
Idle Latency (ns)	100	90
Memory Bandwidths between nodes within the system (using read-only traffic type) (MB/s)	128991	624033

Table 9. Peak Injection Memory Bandwidth (1 MB/sec) Using All Threads

Peak Injection Memory Bandwidth (1 MB/sec) using all threads	Base Solution	Plus Solution
All Reads	518515	624252
3:1 Reads-Writes	432482	549163
2:1 Reads-Writes	420184	542341
1:1 Reads-Writes	395431	533820
STREAM-Triad	401704	545087
Loaded Latencies using Read-only traffic type with Delay=0 (ns)	203	327
L2-L2 HIT latency (ns)	55	61
L2-L2 HITM latency (ns)	56	62

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

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 components. The Intel® Automated Self-Checkout serves as a proxy workload for Vision AI applications and leverages the YOLOv5 model for performing detection along with the efficientnet-b0 model for performing classification.

Table 10. Intel® Automated Self-Checkout 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+
OS	Ubuntu Desktop LTS, Kernel 6.5 (gcc 11.4.0)

4.2.1 Intel® Automated Self-Checkout on Dual Socket Intel® Xeon® Scalable Processor

The Intel® AI System for Edge Verified Reference Blueprint – Large Plus platform with Dual Socket Intel® Xeon® Scalable Processor 8592+ should be able to service up to **73** IP camera streams at 14.95 FPS per stream, for an aggregate of up to 1098 FPS.

The Intel® AI System for Edge Verified Reference Blueprint– Large Base platform with Dual Socket Intel® Xeon® Scalable Processor 6538N should be able to service up to **38** IP camera streams at 14.95 FPS per stream, for an aggregate of up to 570 FPS.

Figure 2. Intel® Automated Self-Checkout Workload Large-Plus Configuration Performance

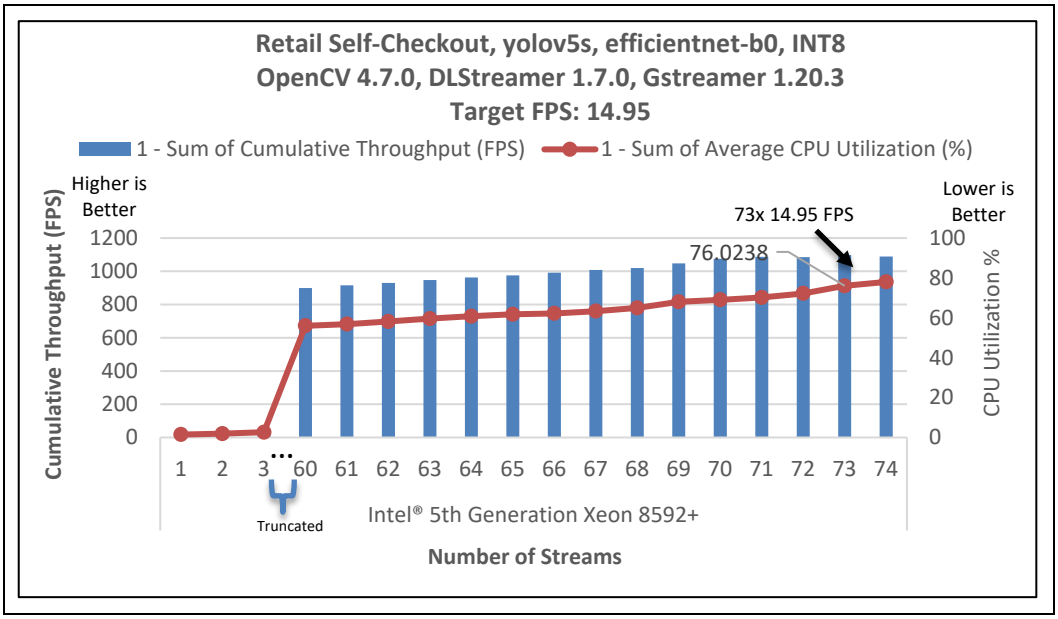
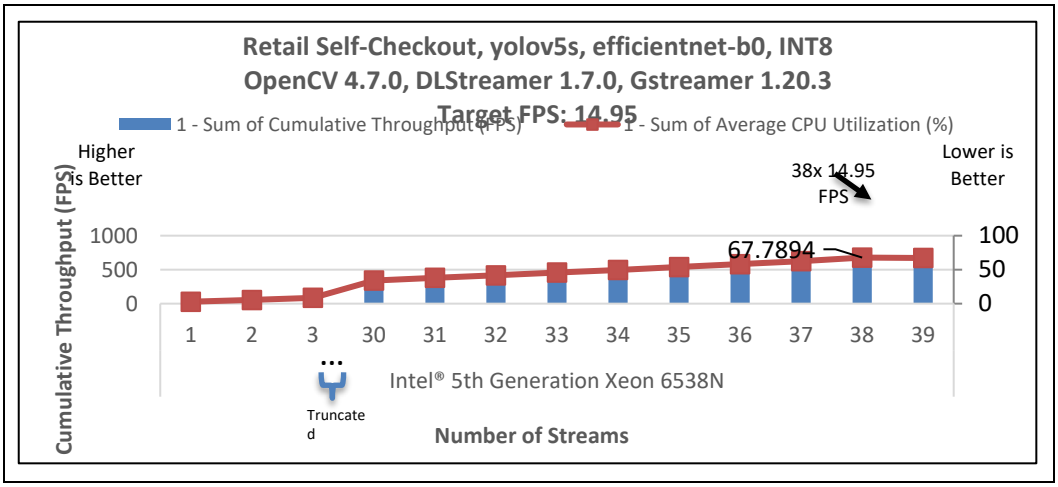


Figure 3. Intel® Automated Self-Checkout Workload Large-Base Configuration Performance



4.2.2 Intel® Automated Self-Checkout on Intel® Data Center Flex GPU

The Intel® AI System for Edge Verified Reference Blueprint – Large Plus platform with 2x Intel® Data Center Flex140 should be able to service up to **32** IP camera streams at 14.95 FPS per stream, for an aggregate of up to 495 FPS. The Intel® AI System for Edge Verified Reference Blueprint– Large Plus platform with Intel® Data Center Flex170 should be able to service up to **52** IP camera streams at 14.95 FPS per stream, for an aggregate of up to 788 FPS.

Refer to Table 10 for the software version details.

Figure 4. Intel® Automated Self-Checkout Workload Plus Configuration Performance on 2x Flex 140

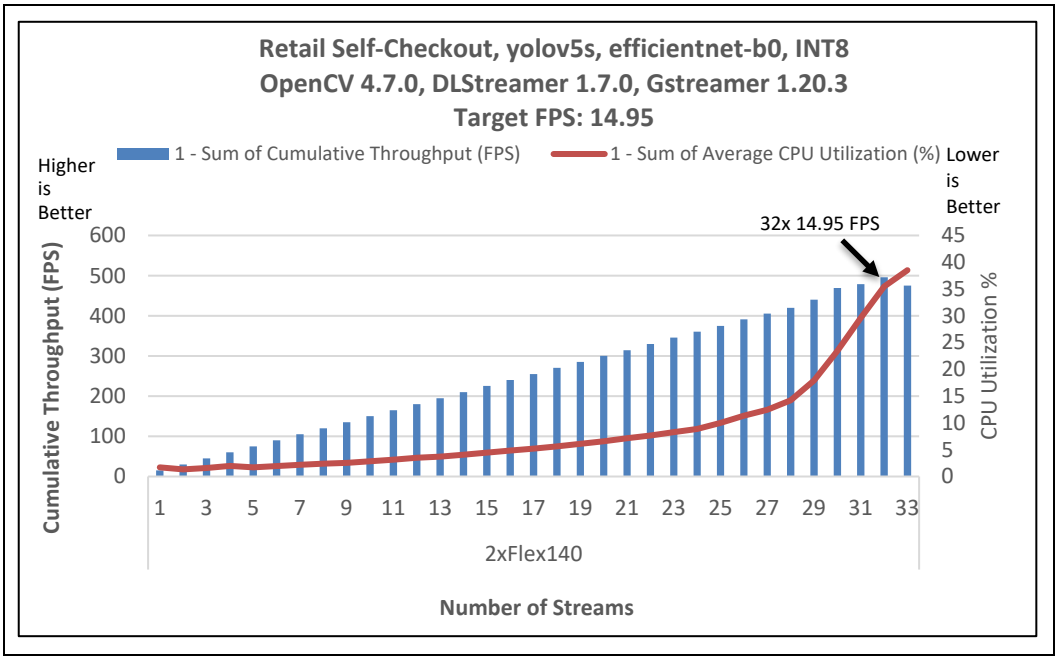
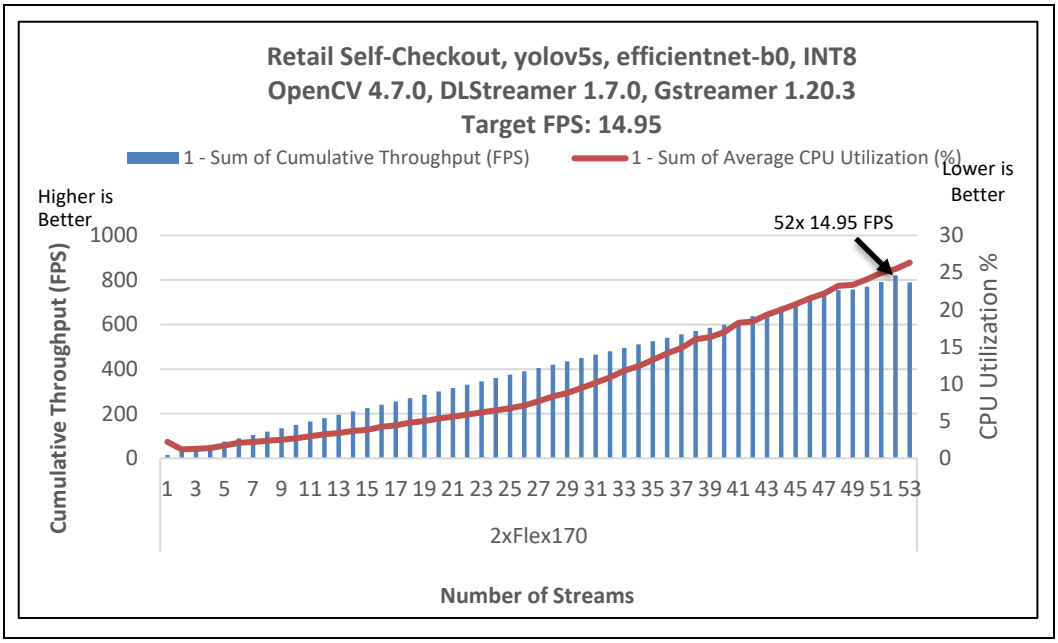


Figure 5. Intel® Automated Self-Checkout Workload Plus Configuration Performance on 2x Flex 170



4.3 Generative AI

In the current technological landscape, Generative AI (GenAI) workloads and models have gained widespread attention and popularity. Large Language Models (LLMs) have emerged as the dominant models driving these GenAI applications. The generation task is memory bound due to iterative decode and kv_cache which needs special management to reduce memory overheads.

Intel® Extension for PyTorch* provide a lot of specific optimizations for these LLMs with platform features optimizations for performance boost on Intel® hardware. The optimizations take advantage of Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Vector Neural Network Instructions (VNNI) and Intel® Advanced Matrix Extensions (Intel® AMX) on Intel® CPUs as well as Intel® Xe Matrix Extensions (XMX) AI engines on Intel® discrete GPUs. Moreover, Intel® Extension for PyTorch* provides easy GPU acceleration for Intel® discrete GPUs through the PyTorch* xpu device.

To better trade-off the performance and accuracy, different low-precision solutions like weight-only-quantization is also enabled. Additionally, tensor parallel and pipeline parallelism mechanism is also adopted for distributed inference to get lower latency for LLMs.

4.3.1 Generative AI on Dual Socket 5th Gen Intel® Xeon® Scalable Processor

The Large Language Model (LLM) proxy workload highlights the Generative AI processing capabilities of the Intel® AI System for Edge Verified Reference Blueprint– Large platform, specifically with the 8B to 40B parameter model is supported directly on 5th Gen Intel® Xeon®

Scalable processors. Specifically, we have tested Llama3-8B, GPT-Neox20B and Falcon40B model with bfloat16, INT8 and INT4 precision.

The weight only quantization method was used for model quantization for converting model from bfloat16 to INT8 and INT4. For faster inference on dual socket CPUs with multiple NUMA regions, we have used auto tensor-parallelism (TP) using DeepSpeed optimization with Sub-NUMA Clustering (SNC) setting.

Table 11. Generative AI Workload Configuration

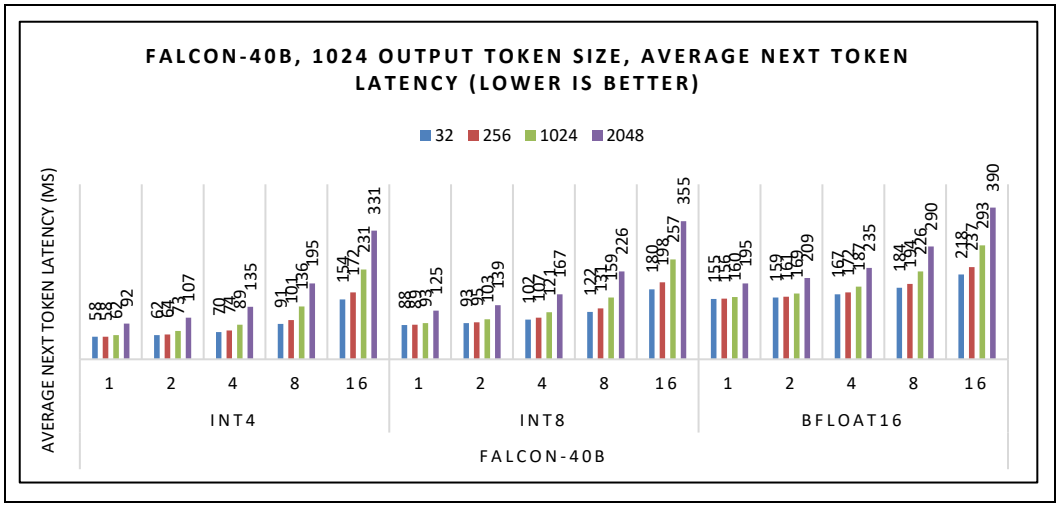
Ingredient	Software Version Details	
Framework /Toolkit	CPU: PyTorch v2.3.100+cpu Deepspeed v0.14.0 Transformers v4.38.1 IPEX-LLM	GPU: PyTorch v2.1.0a0+cxx11.abi Deepspeed @ed8aed5 Transformers v4.37.0 IPEX-LLM
Topology or ML Algorithm	tiiuae/falcon-40b EleutherAI/gpt-neox-20b meta-llama/Llama-3-8b-hf microsoft/Phi-3-mini-4k-instruct TinyLlama/TinyLlama-1.1B-Chat-v1.0	
Libraries	oneDNN v3.4.1 oneCCL v2021.11 torch-ccl v2.3.0+cpu Intel® Neural Compressor v2.4.1	
Model Precision	BF16, INT8, INT4	
Quantization methods	weight-only-quantization	
Warmup steps	1	
Number of Iterations	4	
Batch Size	1, 2, 4, 8, 16, 32	
Beam Width	1 (greedy search)	
Input Token Size	32, 256, 1024, 2048	
Output Token Size	1024	
Compiler	GCC version 12.3.0	
Python	3.10.12	
OS	Ubuntu Desktop LTS, Kernel 6.5	

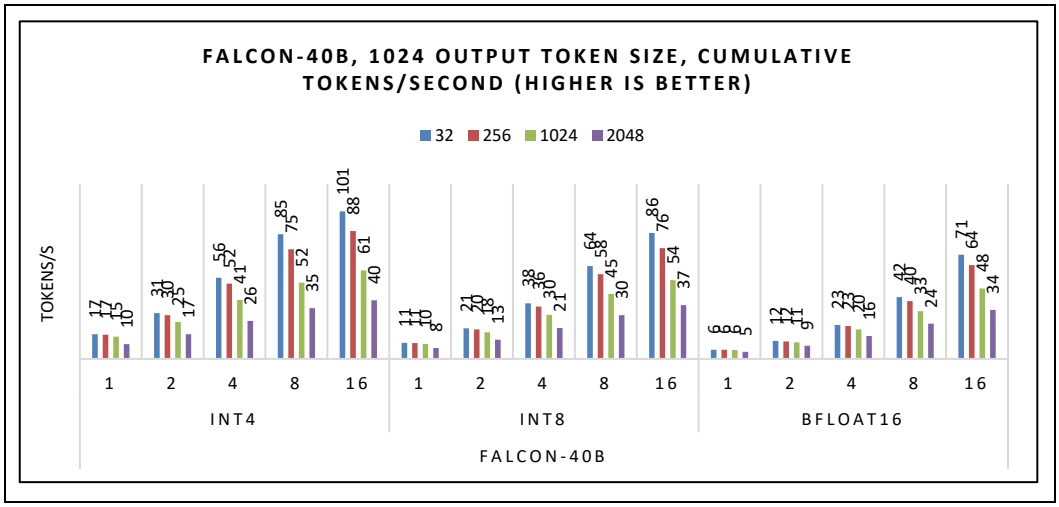
Intel® AI System for Edge Verified Reference Blueprint– Large-Base platform and Large-Plus platform ensure that the results of the system follow the expected results as shown below in order to baseline the performance of the platform. The results shown include performance values for the next token latency, the achievable number of tokens per second, and the inference latency.

Table 12. Generative AI Workload Performance on Large-Plus

Models	Precision	Input Tokens	Output Tokens	Batch Size	Average Next Token latency (ms)	Inference time
Falcon 40B	INT4	1024	1024	1	66	<60s
GPT-NEOX-20B	INT8	1024	1024	1	55	< 60s
	INT4				36	
Llama3-8B	INT4 INT8 BF16	1024	1024	1	14 20 32	< 60s

Figure 6. Falcon-40B Model Performance on Large-Plus CPU Configuration





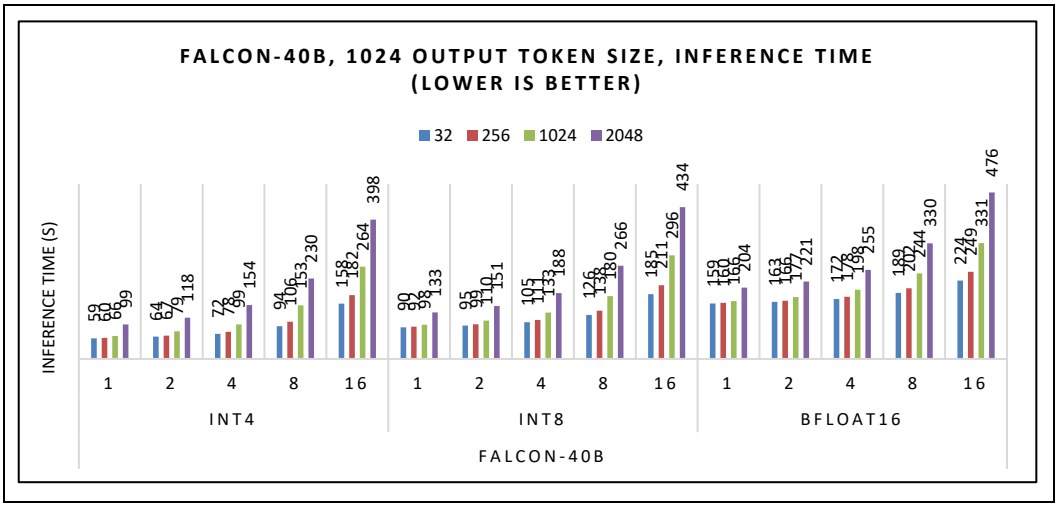
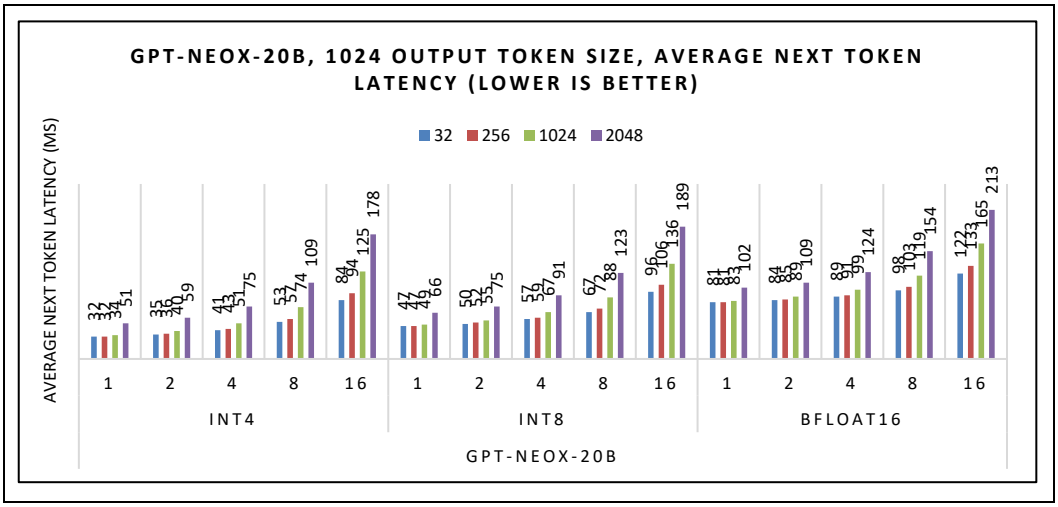
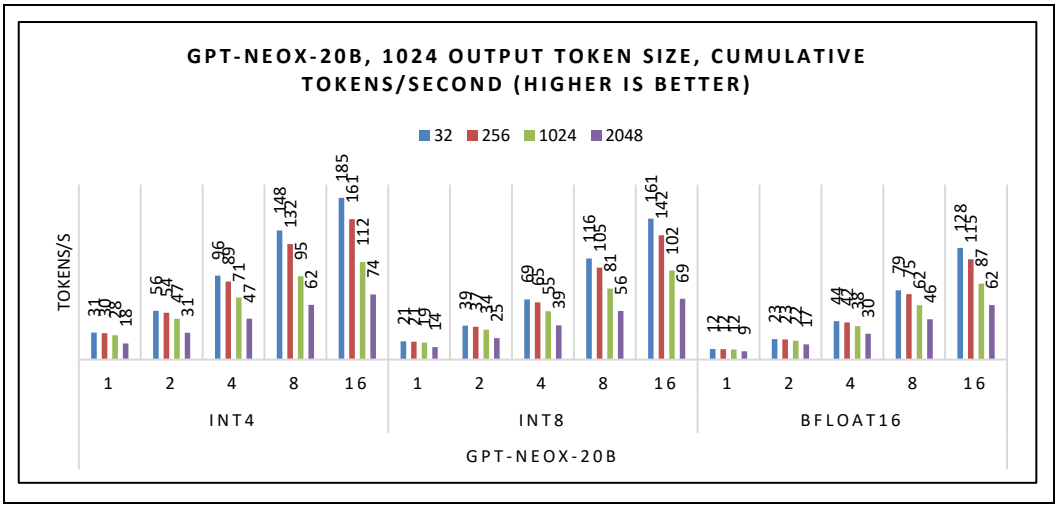


Figure 7. GPT-NEOX-20B Model Performance on Large-Plus CPU Configuration





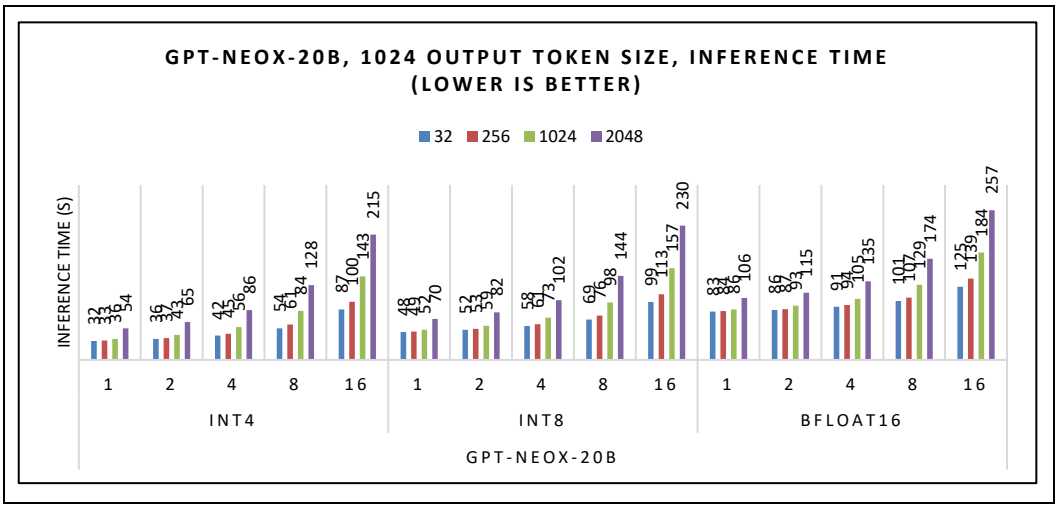
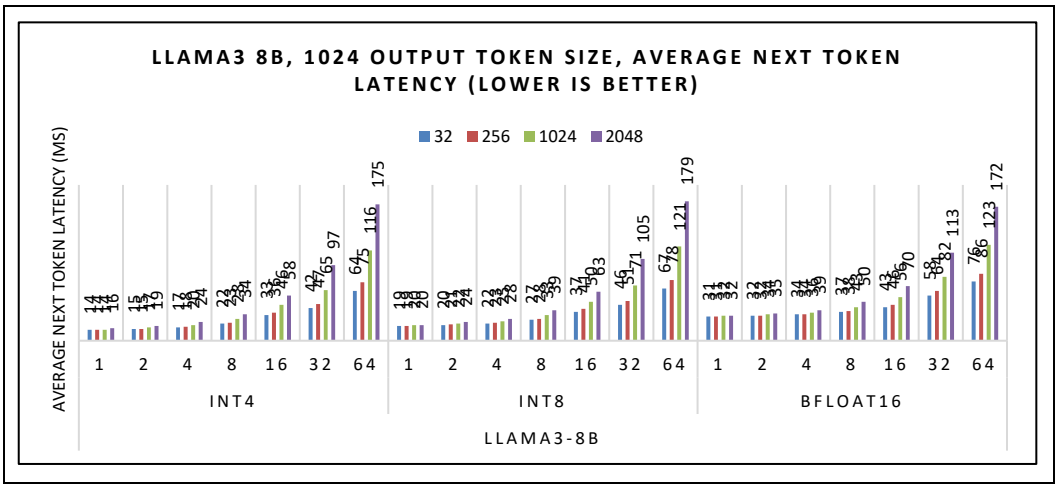
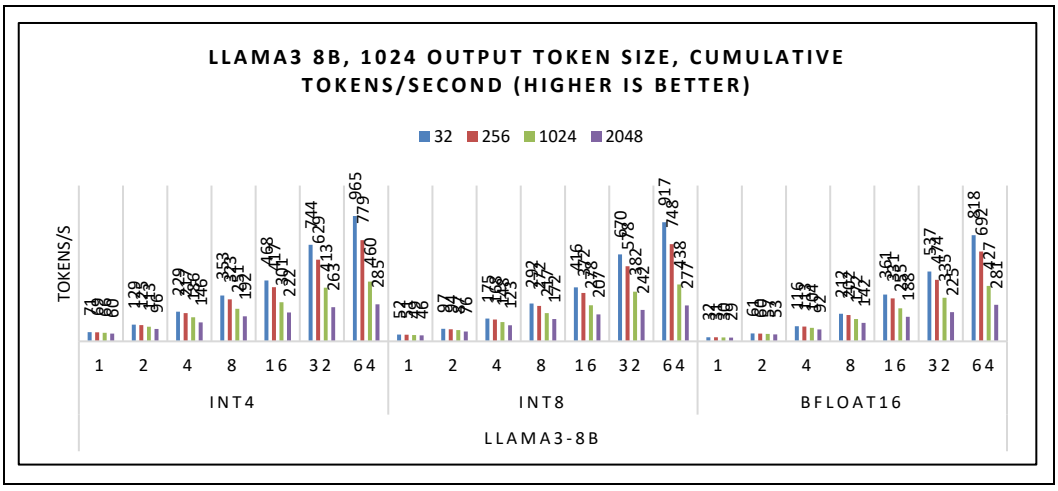


Figure 8. Llama3-8B Model Performance on Large-Plus CPU Configuration





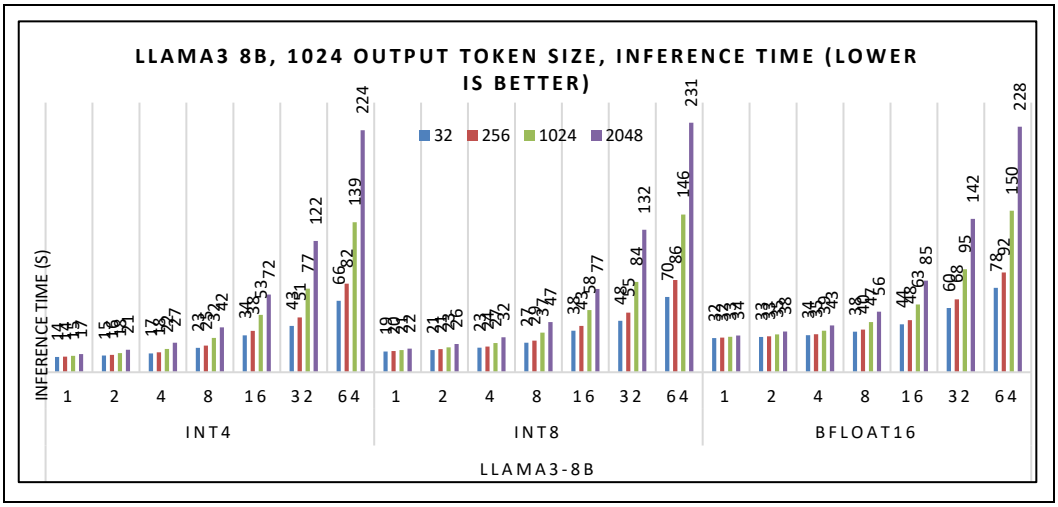
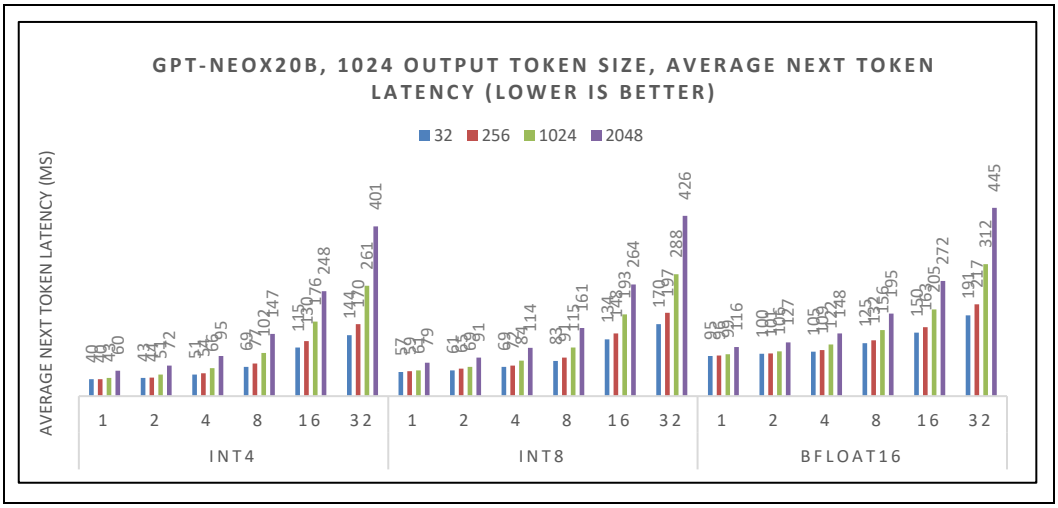
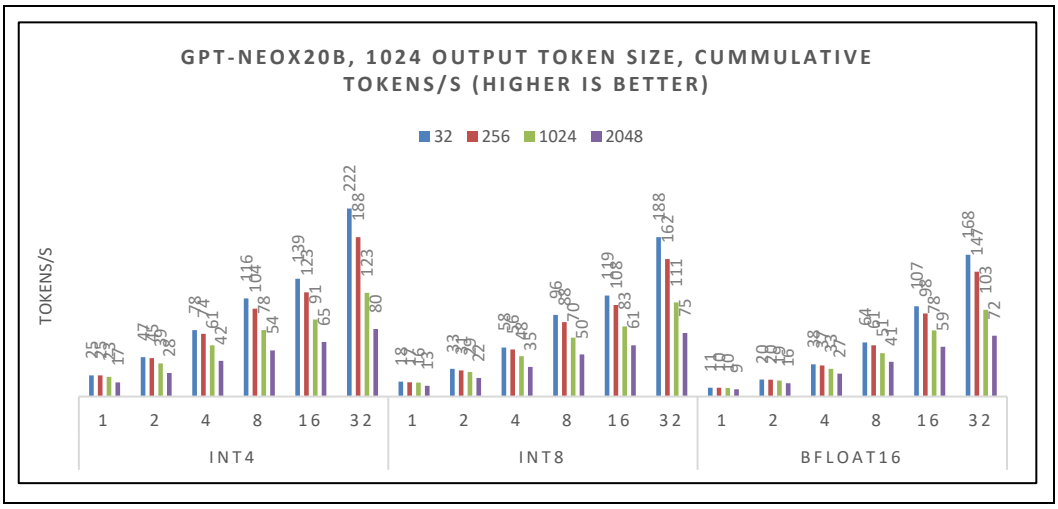


Table 13. Generative AI Workload Performance on Large-Base

Models	Precision	Input Tokens	Output Tokens	Batch Size	Average next token latency (ms)	Inference time
GPT-NEOX-20B	INT4	1024	1024	1	43	< 60s
Llama-3-8B	BF16 INT8 INT4	1024	1024	1	38 24 18	< 60s
Phi3-mini-4k-instruct	BF16 INT8 INT4	1024	1024	1	22 16 13	< 60s

Figure 9. GPT-NEOX-20B Model Performance on Large-Base CPU Configuration




GPT-NEOX20B, 1024 OUTPUT TOKEN SIZE, CUMMULATIVE TOKENS/S (HIGHER IS BETTER)
32
256
1024
2048
TOKENS/S
1 2 4 8 16 32 1 2 4 8 16 32 1 2 4 8 16 32
INT4
INT8
BFLOAT16

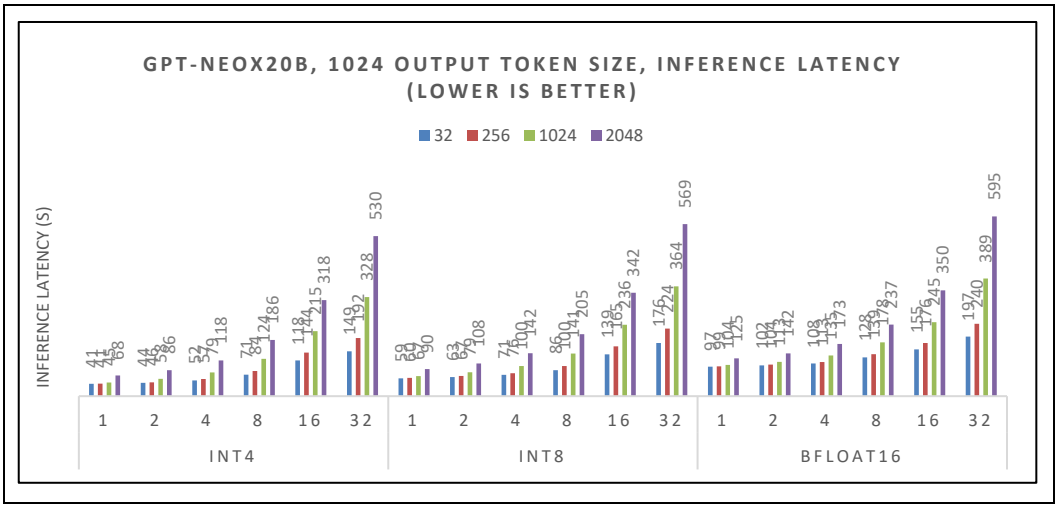
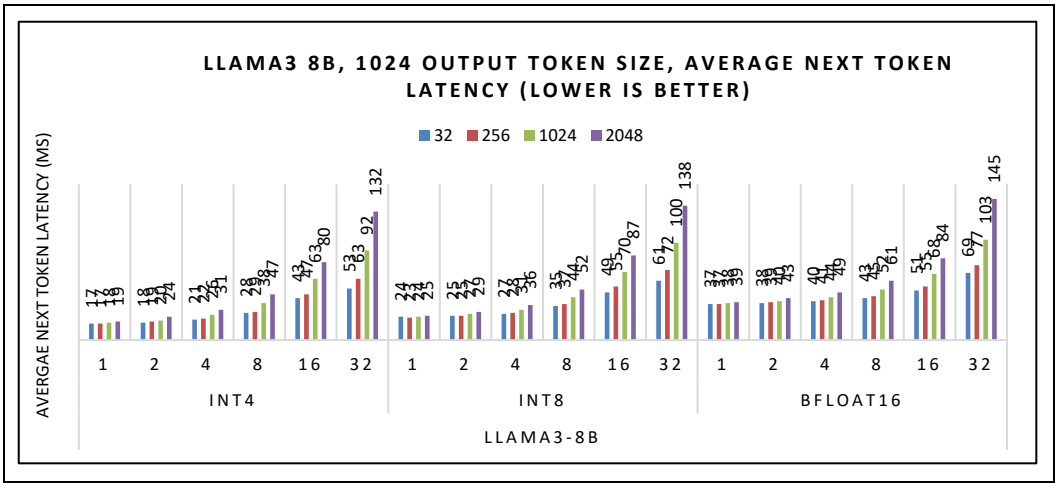
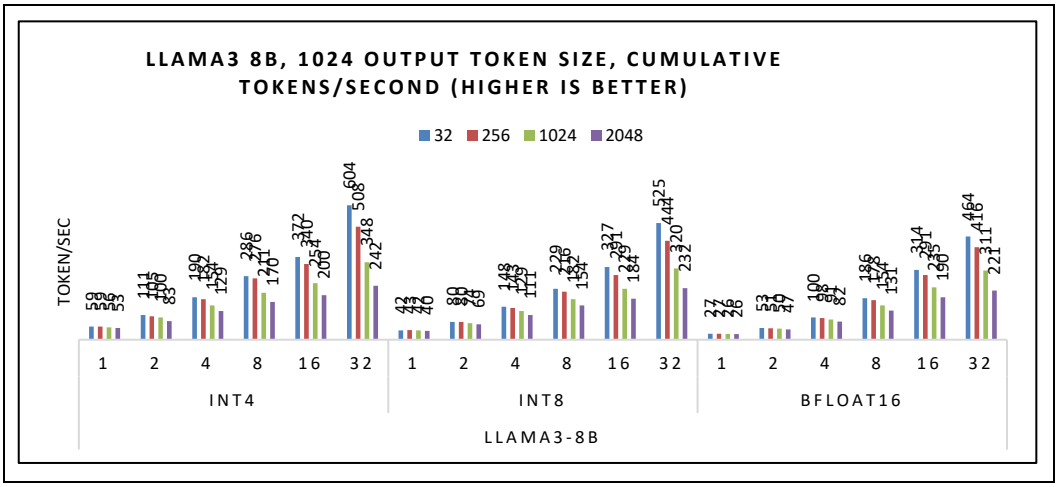

GPT-NEOX20B, 1024 OUTPUT TOKEN SIZE, INFERENCE LATENCY (LOWER IS BETTER)
32
256
1024
2048
INFERENCE LATENCY (S)
1 2 4 8 16 32 1 2 4 8 16 32 1 2 4 8 16 32
INT4
INT8
BFLOAT16

Figure 10. Llama3-8B Model Performance on Large-Base CPU Configuration





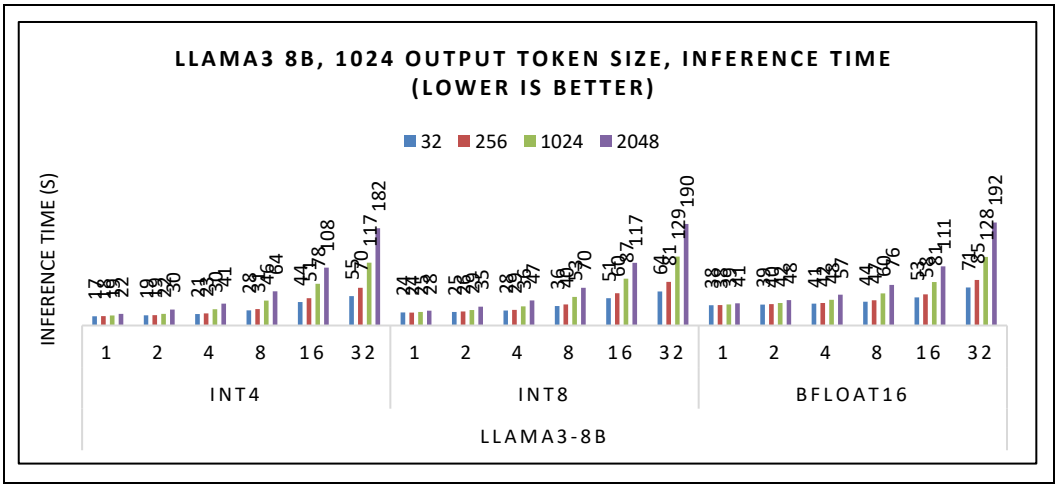
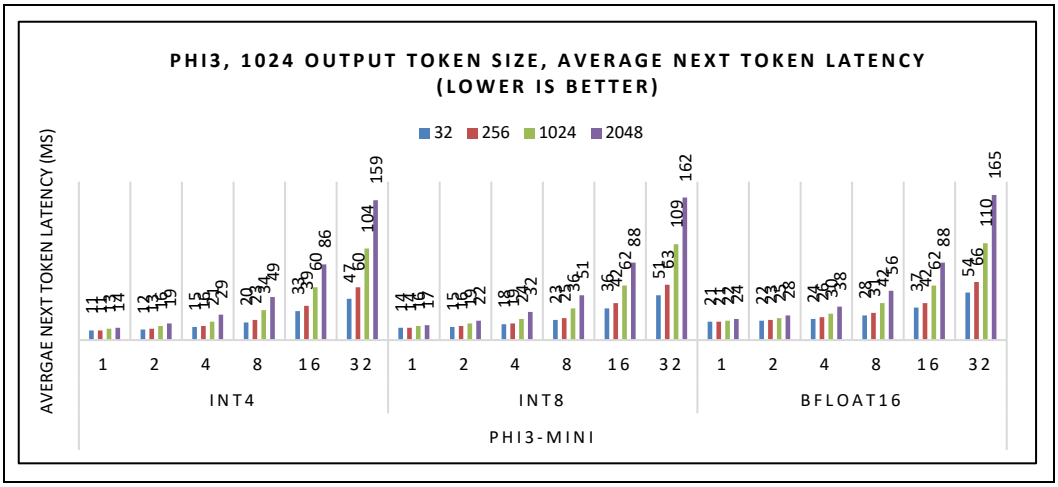
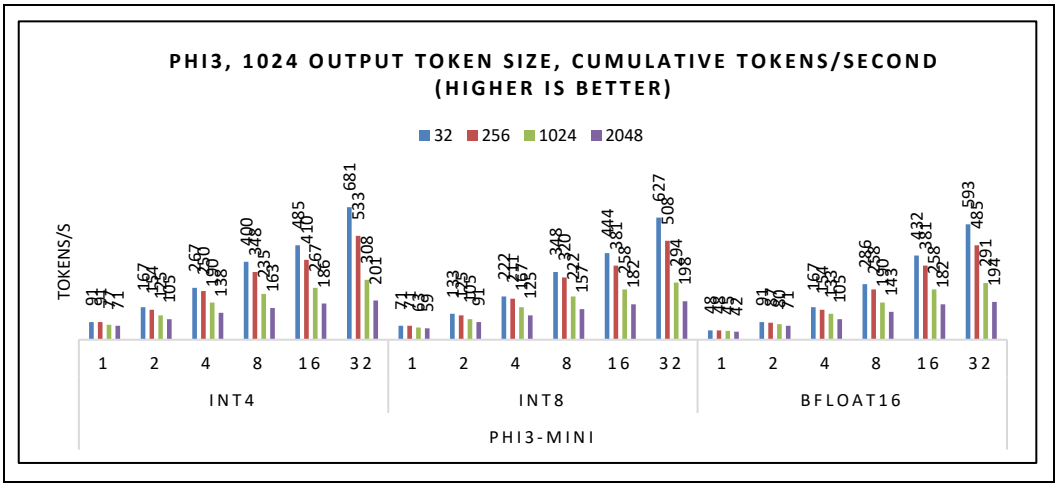
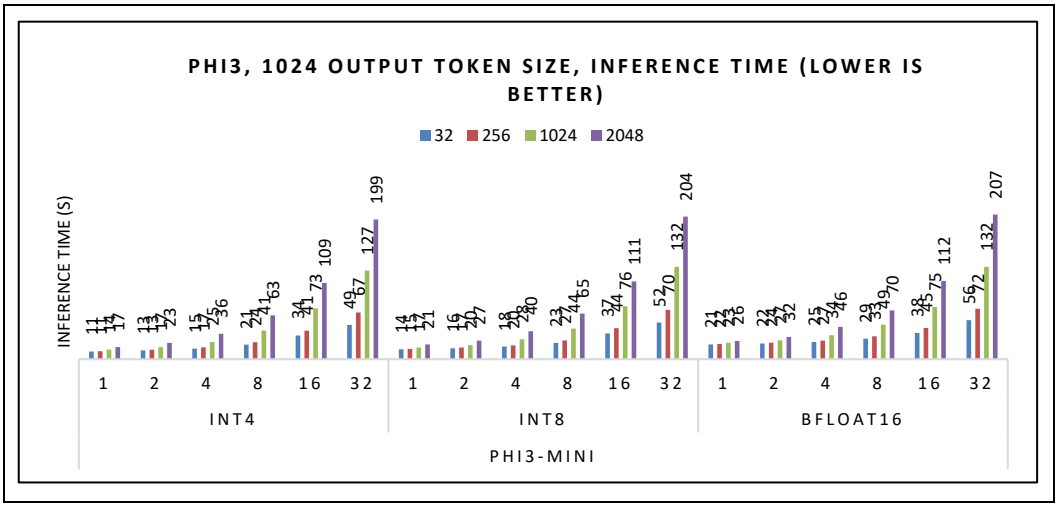


Figure 11. Phi3-min-4K-instruct Model Performance on Large-Base CPU Configuration







4.3.2 Generative AI on Intel® Data Center GPU Flex 170

The Large Language Model (LLM) proxy workload highlights the Generative AI processing capabilities of the Intel® AI System for Edge Verified Reference Blueprint– Large platform. A range of LLM models from 1B to 8B model parameters are supported directly on single Intel® Data Center Flex 170 GPUs (Large-Base) and multiple Intel® Data Center Flex 170 GPUs (Large-Plus) configurations. Specifically, we have tested TinyLlama, Phi3-mini-4k-instruct, and Llama3-8B models with bfloat16, INT8, and INT4 precision.

The weight-only quantization method was used for model quantization to convert the model from bfloat16 to INT8 and INT4. We used a pipeline-parallelism approach for faster inference on multiple GPUs.

Intel® AI System for Edge Verified Reference Blueprint– Large-Base platform ensures that the results of the system follow the expected results as shown below in order to baseline the performance of the platform. The results shown include performance values for the next token latency, the achievable number of tokens per second, along the inference latency.

Table 14. Generative AI Workload Performance on Large-Base, 1x Flex170

Models	Precision	Input Tokens	Output Tokens	Batch Size	Throughput (tokens/s)	Inference time
Llama-3-8B	INT4 INT8	32-256	1024	8	255 188	< 60s
Phi3-mini-4k-instruct	INT4 INT8	32-256	1024	8	361 294	< 60s
TinyLlama	INT4 INT8	32-256	1024	8	830 731	< 60s

Figure 12. Llama3-8B Model Performance on Large-Base CPU Configuration (1x Flex170)

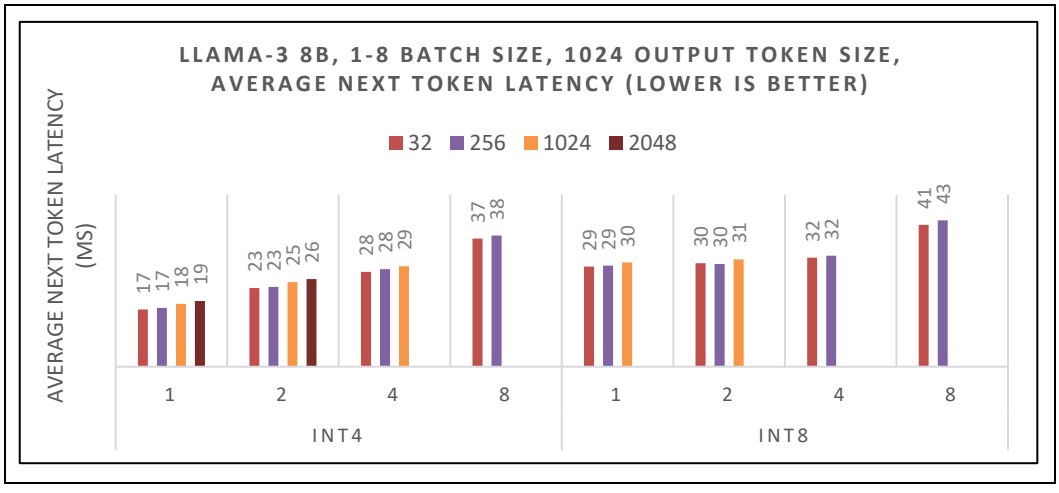
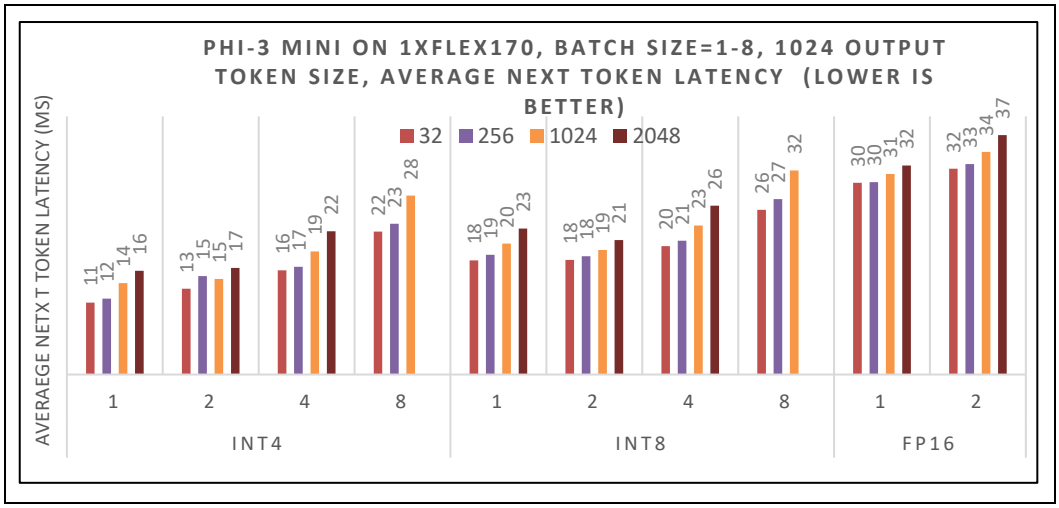
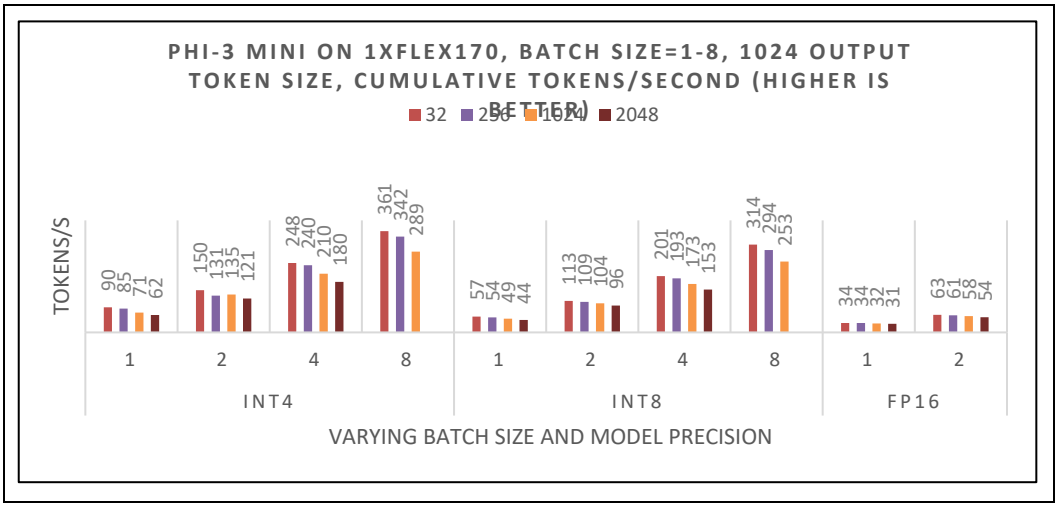


Figure 13. Phi3-min-4K-instruct Model Performance on Large-Base CPU Configuration (1x Flex170)





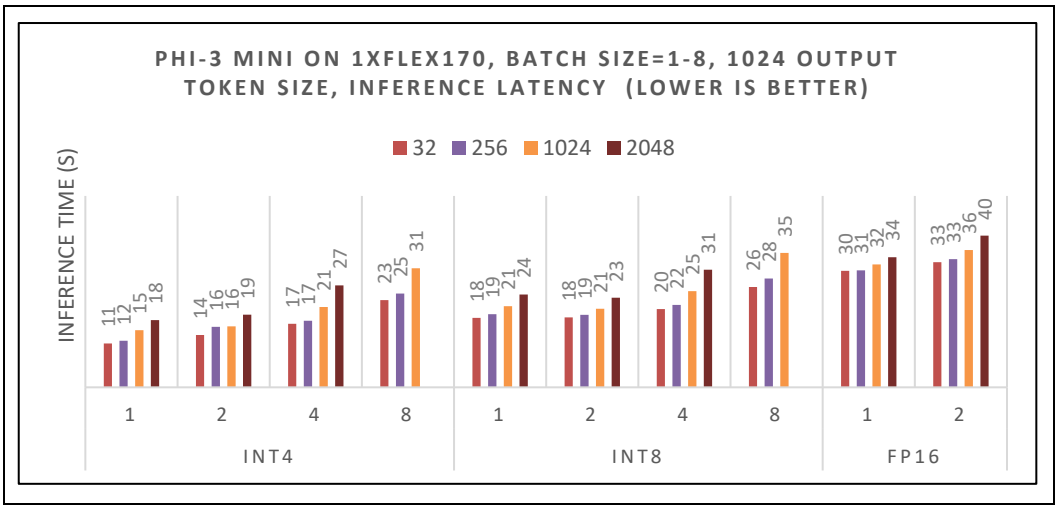
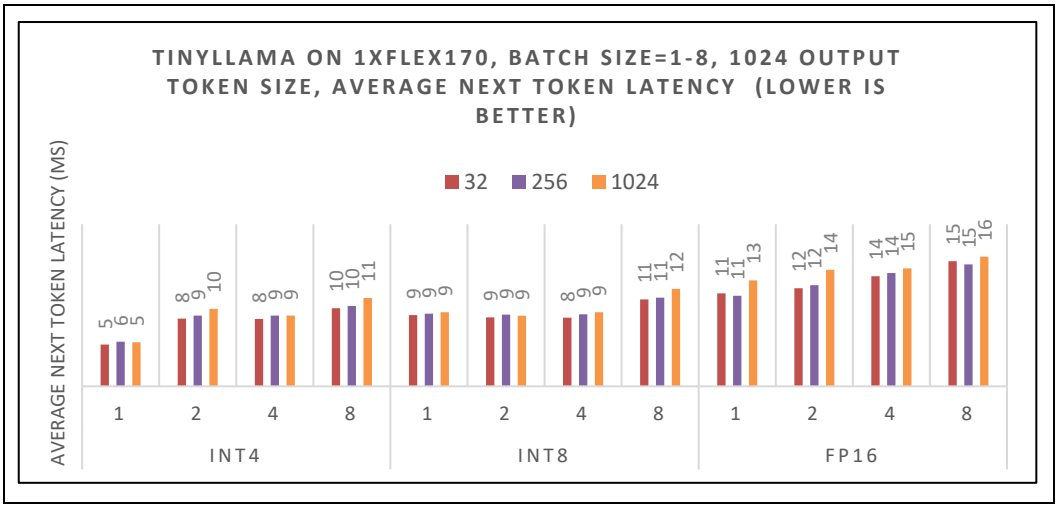
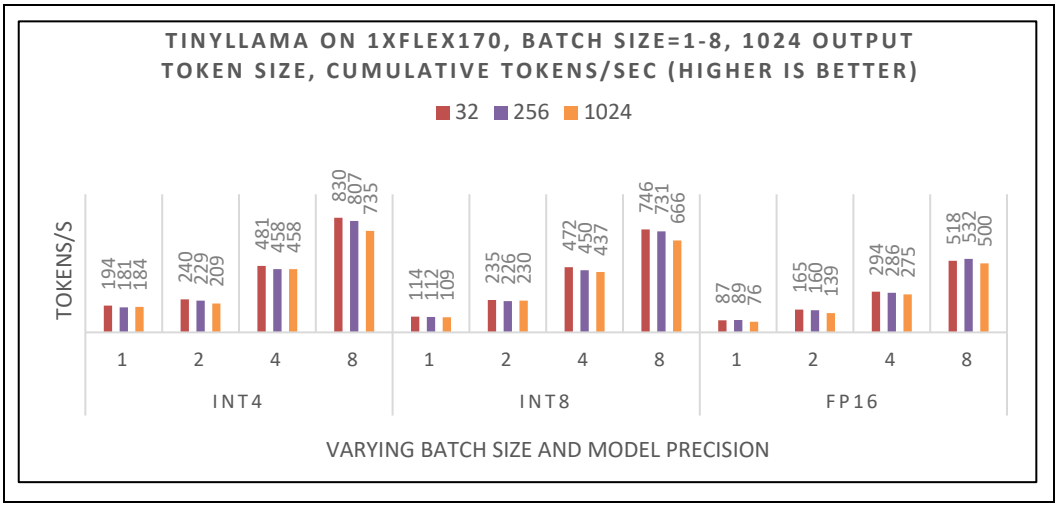


Figure 14. TinyLlama Model Performance on Large-Base CPU Configuration (1x Flex170)





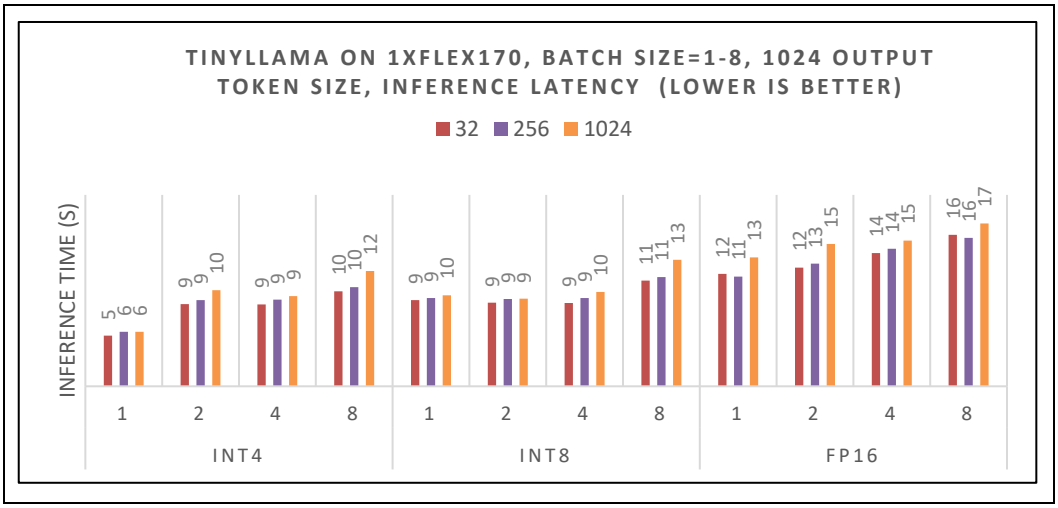
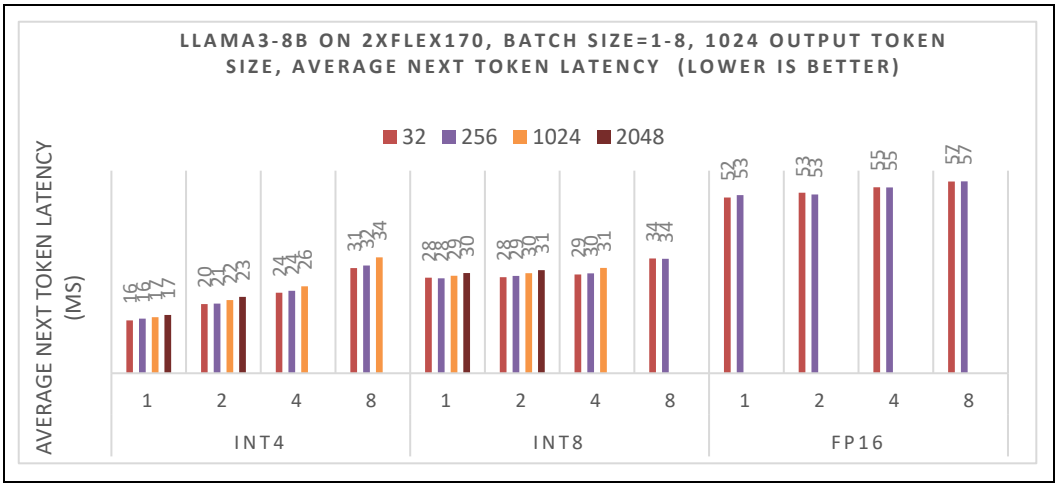


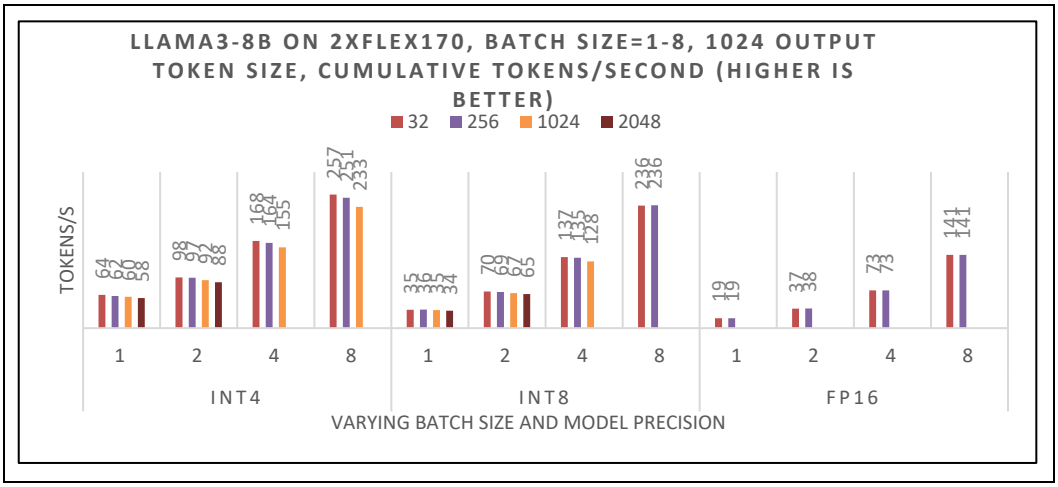
Table 15. Generative AI Workload Performance on Large-Plus, 2x Flex170*

Models	Precision	Input Tokens	Output Tokens	Batch Size	Throughput (tokens/s)	Inference time
Llama-3-8B	INT4 INT8	32-256	1024	8	257 236	< 60s
Phi3-mini-4k-instruct	INT4 INT8	32-256	1024	8	448 364	< 60s
TinyLlama	INT4 INT8	32-256	1024	8	1096 1022	< 60s

Note: *Performance scaling is not linear while performing distributed inference using pipeline parallelism due to PCIe bandwidth limitations. If the model fits one GPU, the best performance may be achieved by running 2 instances of the model on individual GPUs rather than distributing it over multiple cards.

Figure 15. Llama3-8B Model Performance on Large-Plus GPU Configuration (2x Flex170)





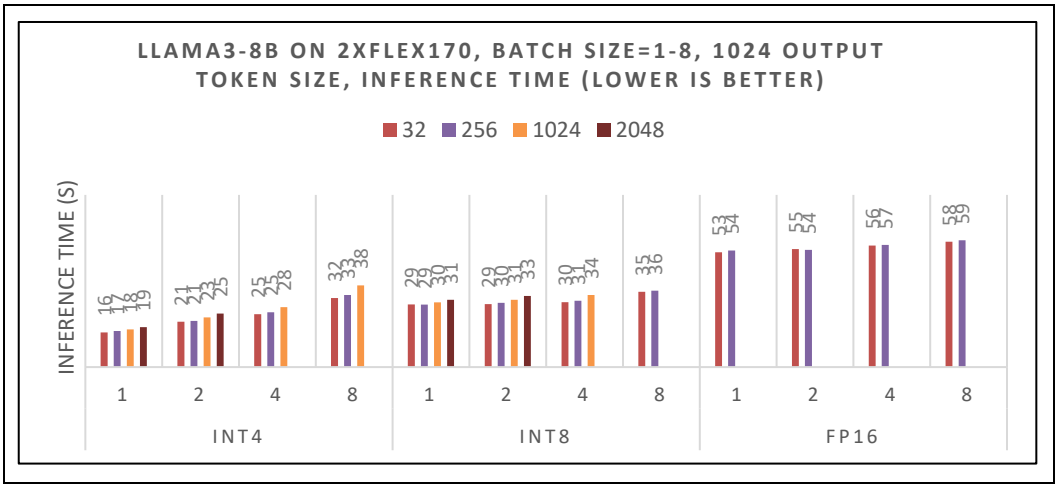
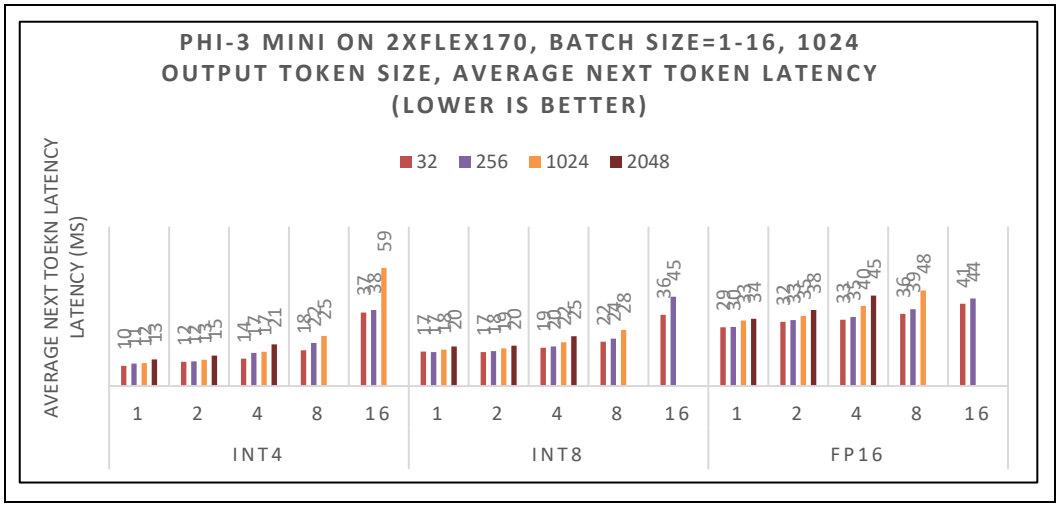
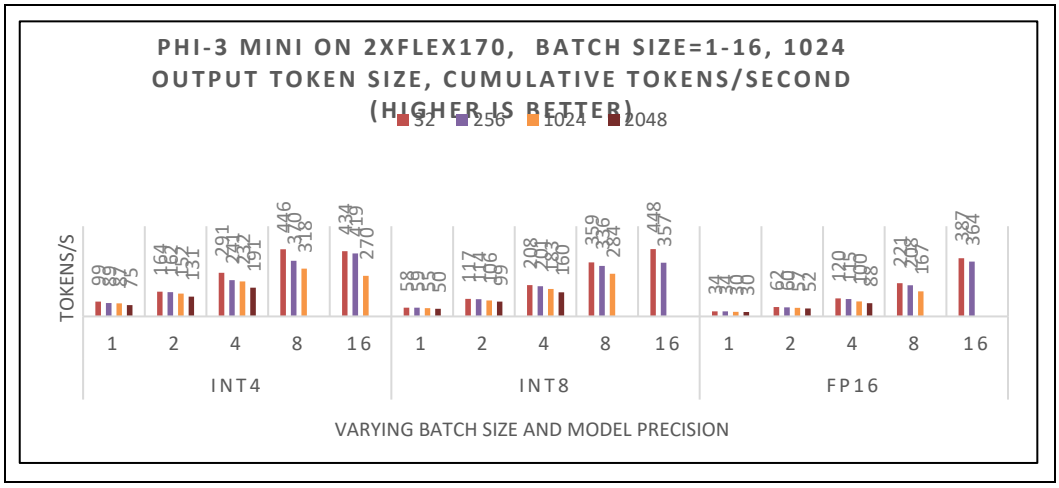


Figure 16. Phi3-min-4K-instruct Model Performance on Large-Plus GPU Configuration (2x Flex170)





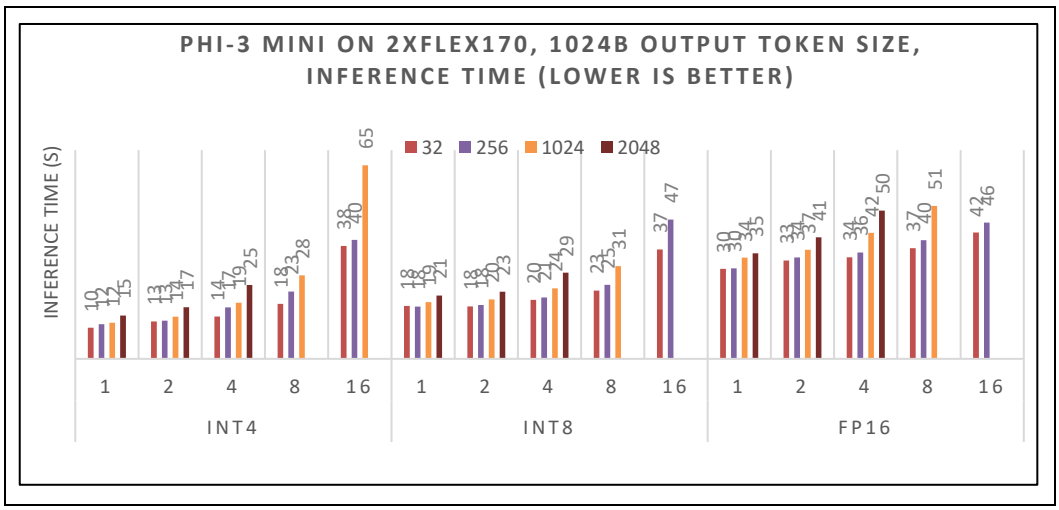
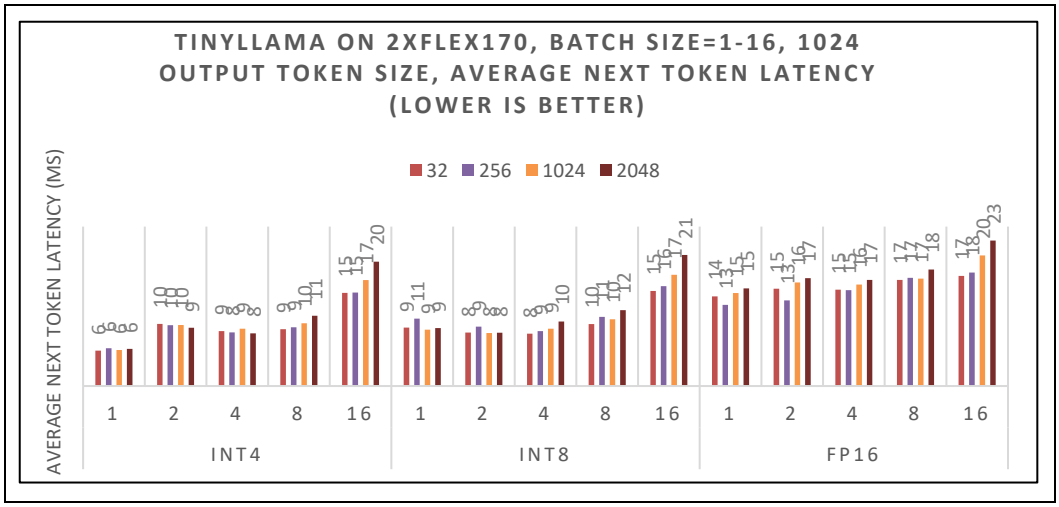
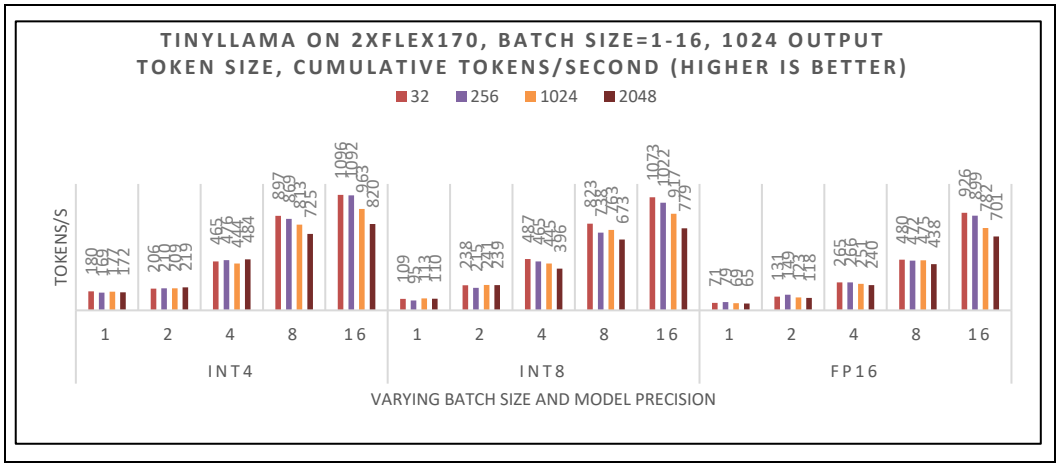
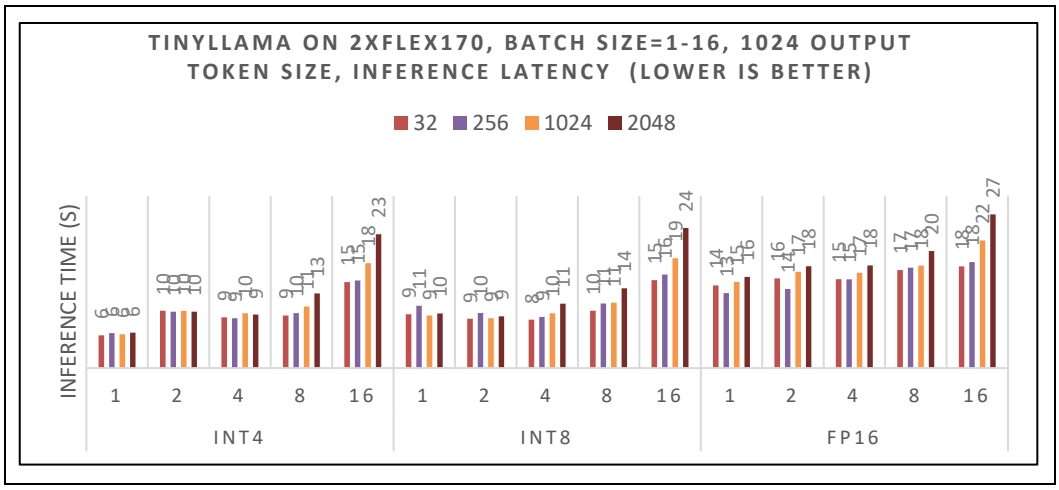


Figure 17. TinyLlama Model Performance on Large-Plus GPU Configuration (2x Flex170)







4.4 Network Security AI

For the Network Security AI performance verification, we will use Malconv and finetuned BERT-base-cased for malicious portable executable (PE) file detection and email phishing detection respectively.

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.

Intel® AI System for Edge Verified Reference Blueprint– Large-Base platform and Large-Plus platform ensure that the results of the system follow the expected results as shown below in order to baseline the performance of the platform. Table 16 shows the software used for the testing while Figures 19 and 20 show a graph of the mean inference time for each model. For the 6538N configuration, with 2 cores per instance, the INT8 model with AVX512_CORE_AMX enabled was able to reach a performance of less than 10 ms. For the 8592+ configuration, with 4 cores per instance, the INT8 model with AVX512_CORE_AMX enabled was able to reach 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 16. 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 18. MalConv AI Entry Platform Performance Graph (6538N)

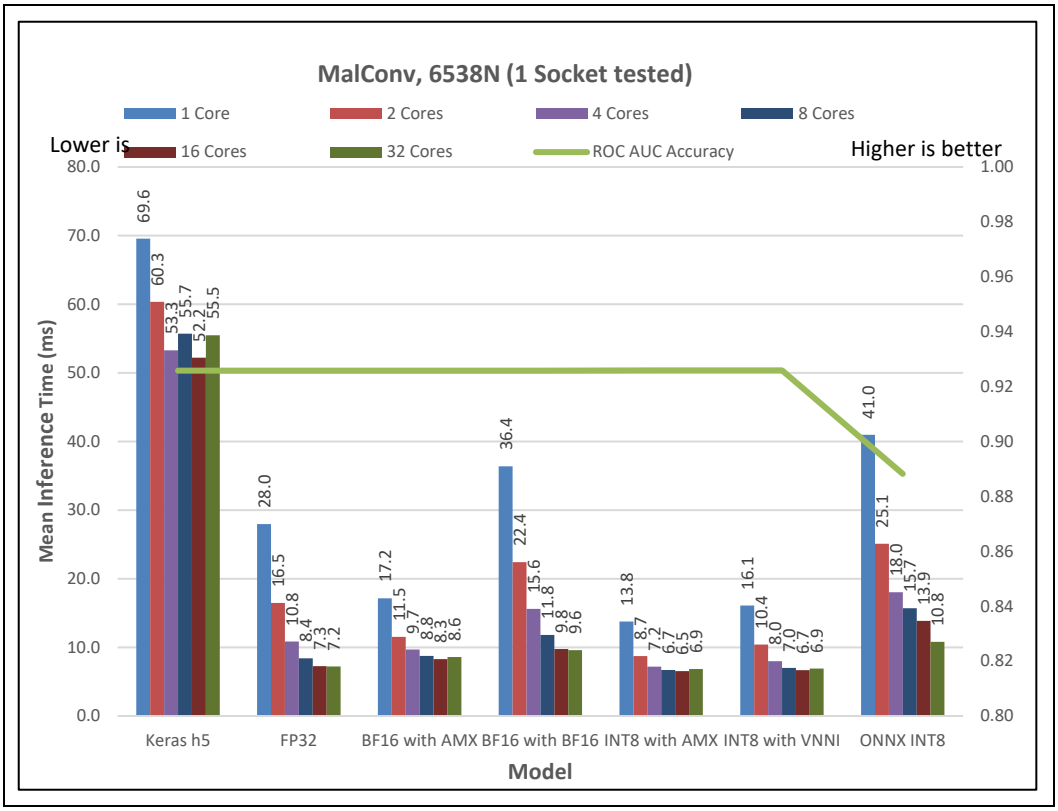
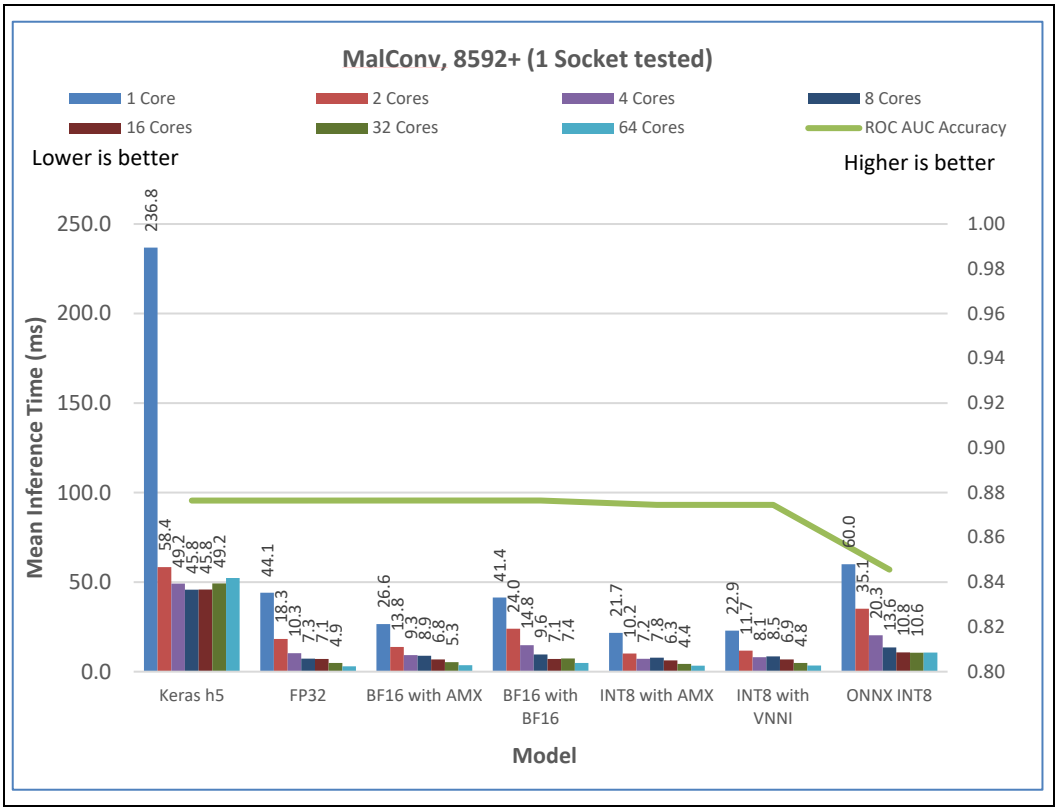


Figure 19. MalConv AI Entry Platform Performance Graph (8592+)



4.4.2 BERT for email phishing detection

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 cased 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 17 shows the software used for the testing, while Figures 21 and 22 shows a graph of the results for the INT8 and FP32 BERT models. For both the 6538N and the 8592+ configurations, 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 17. 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 20. BERT AI Entry Platform Performance Graph (6538N)

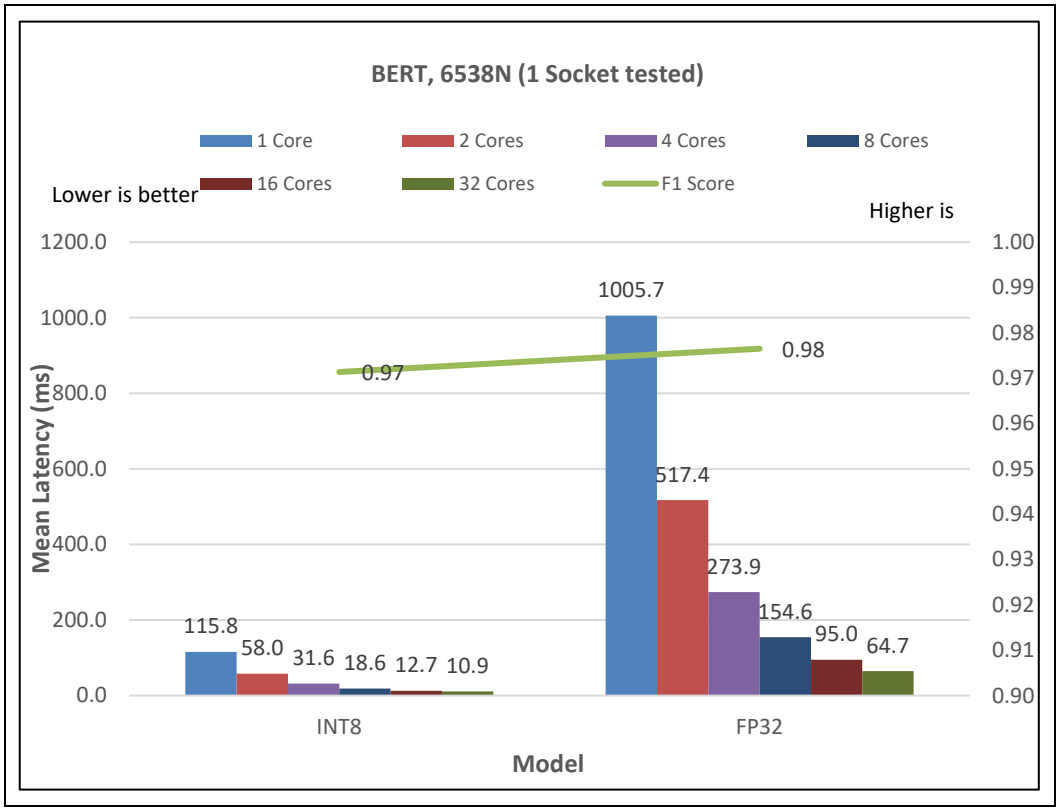
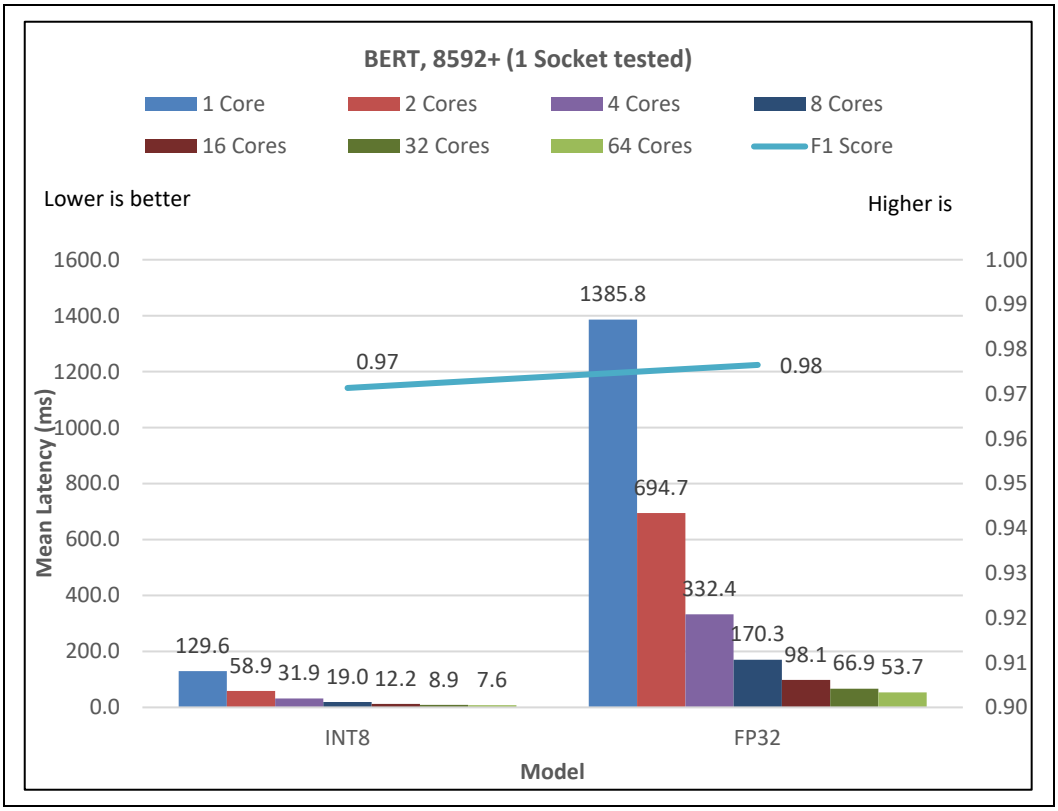


Figure 21. BERT AI Entry Platform Performance Graph (8592+)



4.5 Performance Summary

The following presents the range of performance achievable for the Intel® AI System for Edge Verified Reference Blueprint – Large 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 – Large for Computer Vision, and GEN AI defined on dual 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:

   With Processor AI acceleration only:

- Up to 73 IP camera streams of Vision AI use case with the Intel® Automated Self-Checkout application on Large Plus configurations
- Up to 38 IP camera streams of Vision AI use case with the Intel® Automated Self-Checkout application on Large Base configurations

  With GPU AI inference Offload:

- Up to 52 IP camera streams of Vision AI use case with the Intel® Automated Self-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® Automated Self-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 39 tokens/s on GPT-NEOX-20B model with INT8 precision Batch size of 2 on Large Plus CPU configuration
- Up to 17 tokens/s on Falcon40B model with INT4 precision Batch size of 1 on Large Plus CPU configuration

  With GPU AI inference Offload

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

3. For Network Security AI use case

- In Malconv testing, the INT8 model with AVX512_CORE_AMX enabled reached 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 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.

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.

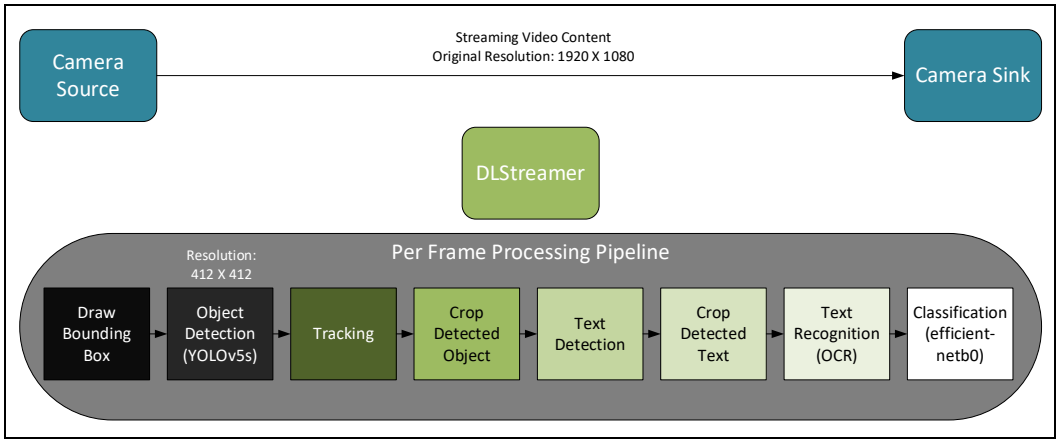
§

Appendix A Appendix

The following section provides detailed instructions for benchmarking a platform with each of the proxy workloads for Vision AI, Generative AI, along with Network Security AI. The benchmarking process leverages the tools and scripts provided as part of the Intel® AI System for Edge 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 22. Test Methodology for the Automated Self-Checkout Proxy Workload



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

- Detection Model: Yolov5s
- Classification Model: efficientnet-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.
- 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.
- Optionally, platform metrics can be collected including CPU utilization, CPU power, CPU frequency, CPU temperature, along with wall power.

To run the automated self-checkout test follow the steps below:

1. Pre-Requisites:

- Intstall Docker
- Set HTTP_PROXY and HTTPS_PROXY proxies in environment if necessary
- Python version 3.8 is recommended

2. Change to the automated self-checkout test directory and initialize the environment:

```
# cd enterprise_ai/common/retail-self-checkout/

# ./init_rsc.sh
```

Optionally, update the collect_server_power.sh script with the BMC information of the server to collect the wall power metrics during the automated self-checkout benchmark.

Note: The collect_server_power.sh script is provided for convenience to collect wall power measurements and is designed to be run within a lab environment and not within a production environment.

```
# $EDITOR collect_server_power.sh



#!/usr/bin/env bash



…

ip_address=<server-ip-address>

un=<bmc-username>

pw=<bmc-password>

…
```

Note: Start the benchmark on the 5th Gen Intel® Xeon® Scalable Processor using a batch size of By default, the benchmark will use a target FPS of 14.95 along with an initial duration of 40 seconds to allow the system to reach steady state.

```
# ./benchmark_rsc.sh 1 cpu
```

3. The results will be stored within a CSV file located under rsc_results.

```
# cat ~/rsc_results/stream-density-cpu-yolov5s-effnetb0-density-
increment_1_init-duration_40_target-fps_14_95_batch_1.csv
```

4. Optionally, if turbostat is installed on the server then CPU related metrics can be converted into a CSV file as follows:

```
python3 turbostat_log_parser_infer_streams.py \

--log-file ~/rsc_results/turbostat_gpu_batch_1.log \

--num-streams <max_stream_num> \

--csv-file-name ~/rsc_results/turbostat_gpu_batch_1.csv
```

5. Optionally, if the BMC credentials have been provided then server power related metrics can be converted into a CSV file as follows:

```
python3 server_power_log_parser.py --log-file
~/rsc_results/server_power_cpu_batch_1.log --csv-file-name
~/rsc_results/server_power_cpu_batch_1.csv
```

6. Start the benchmark against Intel® Data Center Flex GPUs using a batch size of 1.

Note: By default, the benchmark will use a target FPS of 14.95 along with an initial duration of 40 seconds to allow the system to reach a steady state.

```
# ./benchmark_rsc.sh 1 gpu
```

7. The results will be stored within a CSV file located under rsc_results.

```
# cat ~/rsc_results/stream-density-gpu-yolov5s-effnetb0-density-
increment_1_init-duration_40_target-fps_14_95_batch_1.csv
```

8. Optionally, if turbostat is installed on the server then CPU related metrics can be converted into a CSV file as follows:

```
python3 turbostat_log_parser_infer_streams.py \

--log-file ~/rsc_results/turbostat_gpu_batch_1.log \

--num-streams <max_stream_num> \

--csv-file-name ~/rsc_results/turbostat_gpu_batch_1.csv
```

9. Optionally, if the BMC credentials have been provided then server power related metrics can be converted into a CSV file as follows:

```
python3 server_power_log_parser.py --log-file
~/rsc_results/server_power_cpu_batch_1.log --csv-file-name
~/rsc_results/server_power_cpu_batch_1.csv
```

A.2 Generative AI Test Methodology

A.2.1 IPEX-LLM Testing Methodology on CPU

The Generative AI benchmark on Intel® dual socket CPU was performed using Intel® Extension of PyTorch (IPEX) for LLM. To reduce the inference latency and improve throughput, tensor parallel is also enabled in our solution. We use DeepSpeed to auto shard the model and then use Distributed Inference with DeepSpeed with AutoTP feature.

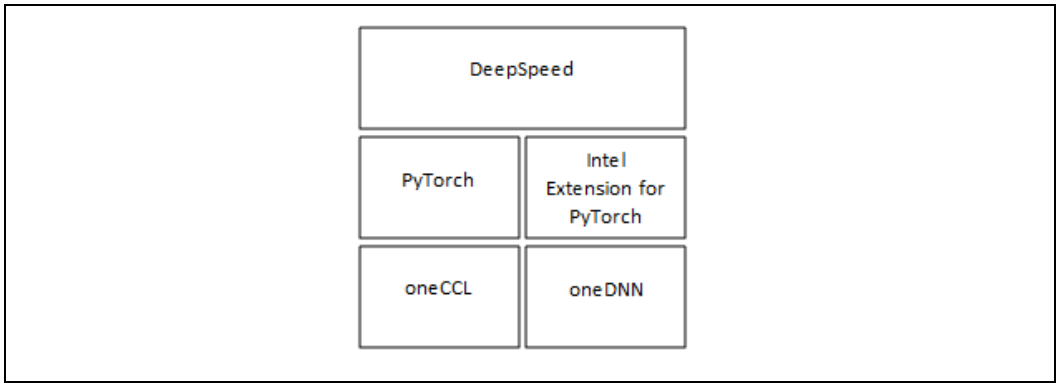
DeepSpeed builds on top of PyTorch, which has been highly optimized for CPU inference and training. Intel® Extension for PyTorch adds state-of-the-art optimizations for popular LLM architectures, including highly efficient matrix multiplication kernels to speed up linear layers and customized operators to reduce the memory footprint.

The runtime software components for DeepSpeed Inference on CPU are shown in Figure 23:

- Intel® oneAPI Deep Neural Network Library (oneDNN) uses Intel® AVX-512 VNNI and Intel® AMX optimizations.

- Intel® oneAPI Collective Communications Library (oneCCL) is a library that implements the communication patterns in deep learning.
- Intel® Neural Compressor was used to convert the LLMs from FP32 datatype to bfloat16 or int8 datatype.

Figure 23. Software components for DeepSpeed Inference on CPU



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

1. Install the baseline dependencies:

```
# sudo apt update

# sudo apt install -y make git numactl

# sudo apt install -y python3

# sudo pip install –upgrade pip
```

2. 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
```

3. 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

4. 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
```

5. Download the LLM models from HuggingFace:

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

6. 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.

7. 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>
```

8. 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/
```

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

```
# sudo chown ubuntu:ubuntu benchmark_cpu_ds.sh

# sudo chown ubuntu:ubuntu extract_kpis.py
```

10. 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"
```

11. 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
```

12. 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_ds.sh
```

13. 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
```

14. 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 IPEX-LLM Testing Methodology on 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:

1. Download the IPEX-LLM container image:

```
# export DOCKER_IMAGE=intelanalytics/ipex-llm-serving-xpu:2.1.0-SNAPSHOT

# docker pull intelanalytics/ipex-llm-serving-xpu:2.1.0-SNAPSHOT
```

2. 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
```

3. 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/
```

4. 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"

…
```

5. Login to the container and start the benchmark:

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

6. Review the benchmark results:

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

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.

§