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

Reference Architecture

*Revision 1.0*
*October 2024*

*Authors*
*Timothy Miskell*
*Abhijit Sinha*
*Ai Bee Lim*

*Key Contributors*
*Jonathan Tsai*
*Jessie Ritchey*
*Edel Curley*

Document Number: 834791

# Contents



## Figures

## Tables

# Revision History

| Document Number | Revision Number | Description | Revision Date |
|---|---|---|---|
| 834791 | 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 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 architecture for the 14th Gen Intel® Core 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 – Small 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 Blueprints 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 – Small for Computer Vision, and GEN AI covering a Base and Plus configuration:

- Intel® AI System for Edge Verified Reference Blueprint – Small for Computer Vision, and GEN AI Plus configuration for the Node is defined with at least a 32-core 14th Generation Intel® Core processor and high-performance network, with storage and integrated platform acceleration products from Intel® for maximum containerized workload density.

- Intel® AI System for Edge Verified Reference Blueprint—Small for Computer Vision and GEN AI Plus configuration for the Node are defined with a 24-core 14th Generation Intel® Core processor, with storage and add-in platform acceleration products from Intel® targeting 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 solution's value for AI Inference use cases. The solution leverages 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 on the 14th Generation Intel® Core Processor Family include:

- High core count and per-core performance
- Compact, power-efficient system-on-chip platform
- Streamlined path to cloud-native operations
- Accelerated AI inference with integrated processor capabilities
- 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 – Small 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 – Small for Computer Vision, and GEN AI. The hardware requirements for the Base Configuration and Plus Configuration are detailed below.

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

| Ingredient | Requirement | Required/ Recommended | Quantity |
|---|---|---|---|
| Processor | Intel® 14th Generation CoreTM i7-14700 Processor 8 P-Cores, 12 E-Cores, 65 W or higher number SKU | Required | 1 |
| Memory | 128 GB DDR5 4800 MT/s | Required | 1 |
| Network | Intel® Ethernet Network Adapter i226-V/LM/IT (2.5 Gbps) | Required | 1 |
| Storage (Boot/Capacity Drive) | 1 TB or equivalent boot drive | Required | 1 |
| dGPU | Intel® Arc 380 | Required | 1 |
| IP cameras | 4K video streaming with support for at least 15 FPS and RTSP | Required | 4 |
| LAN on Motherboard (LOM) | 1 Gbps I219-LM for Operation, Administration and Management (OAM) | Required | 1 |

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

| Ingredient | Requirement | Required/ Recommended | Quantity |
|---|---|---|---|
| Processor | Intel® 14th Generation CoreTM i9-14900E Processor 8 P-Cores, 16 E-Cores, 65 W or higher number SKU | Required | 1 |
| Memory | 128 GB DDR5 4800 MT/s | Required | 1 |

| Ingredient | Requirement | Required/ Recommended | Quantity |
|---|---|---|---|
| Network | Intel® Ethernet Network Adapter i226-V/LM/IT (2.5 Gbps) | Required | 1 |
| Storage (Boot/Capacity Drive) | 1 TB or equivalent boot drive | Required | 1 |
| dGPU | Intel® Arc 750 | Required | 1 |
| IP cameras | 4K video streaming with support for at least 15 FPS and RTSP | Required | 8 |
| LAN on Motherboard (LOM) | 1 Gbps I219-LM for Operation, Administration and Management (OAM) | Required | 1 |

## 2.2 BIOS Settings

To meet the performance requirements for an Intel® AI System for Edge Verified Reference Blueprint – Small 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.

Refer to the following table for the set of recommended BIOS settings.

**Table 3. Recommended BIOS Settings**

| Setting | Value |
|---|---|
| Hardware Prefetcher | Enabled |
| Intel® (VMX) Virtualization Technology | Enabled |
| Hyper-Threading | Enabled |
| Intel® Speed Shift Technology | Enabled |
| Turbo Mode | Enabled |
| C-States | Enabled |
| Enhanced C-States | Enabled |
| C-State Auto Demotion | C1 |
| C-State Un-Demotion | C1 |
| MonitorMWait | Enabled |
| Enforce DDR Memory Frequency POR | POR |
| Maximum Memory Frequency | Auto |
| Primary Display | Auto |

| Setting | Value |
|---|---|
| Internal Graphics | Auto |
| Graphics Clock Frequency | Max CdClock freq based on Reference Clk |
| VT-d | Enabled |
| Re-Size BAR Support | Enabled |
| SR-IOV Support | Enabled |

BIOS settings differ from vendor to vendor. Please contact your Intel® Representative if you do not see the exact setting in your BIOS.

## 2.3 Solution Architecture

Figure 1 shows the architecture diagram of Intel® AI System for Edge Verified Reference Blueprint – Small 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

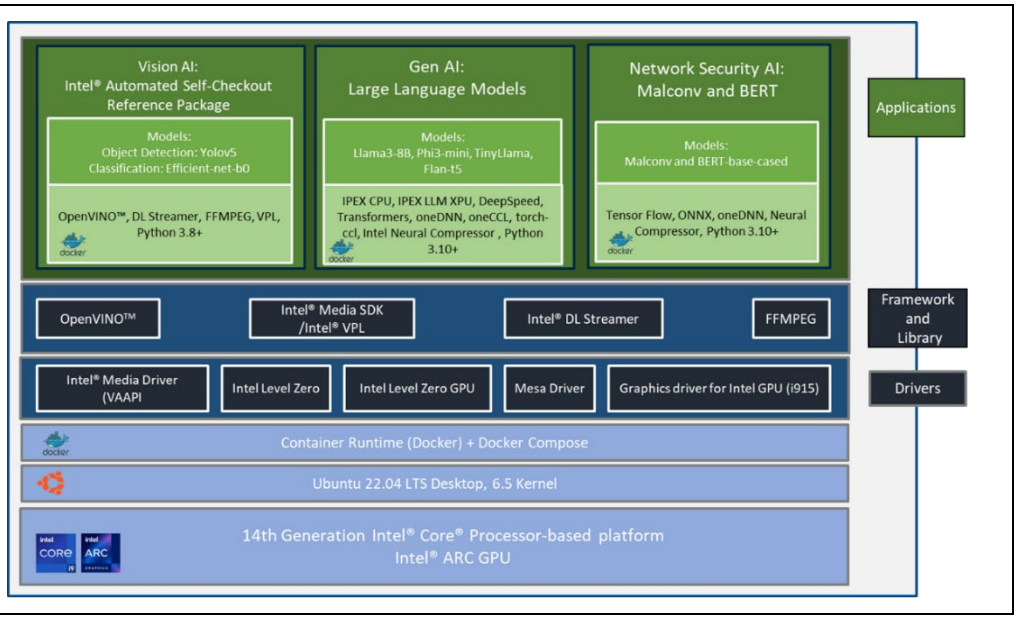
All three applications are containerized using docker.

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 detects objects, and the EfficientNet model classifies Objects.

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**




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 4. 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 Reference Blueprint recommends that the 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 5. 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® Boot Guard | Intel® Boot Guard | Enable | Required |
| 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 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 module 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 Edge AI Node

## 3.1 Boot Parameter Setup

For the workload testing, note that it is not necessary to enable hugepage support nor is 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-6.5.0-45-generic root=UUID=2f851afa-7405-4e84-8c11-
5f541adfd173 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 Driver:

1. Install the prerequisites to add the necessary repository access.

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

2. 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
```

3. 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
```

4. Install the packages responsible for computing and media.

```
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
```

5. Install the development packages.

```
sudo apt install -y \
    libigc-dev Intel®-igc-cm libigdfcl-dev libigfxcmrt-dev level-zero-dev
```

6. 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}
```

7. 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
```

8. 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 Entry solution was tested on August 31, 2024, with the following hardware and software configurations:

- 1 NUMA nodes
- 1x Intel® 14th Generation Core® i9-14900E processor
- Total Memory: 128 GB, 4 slots/32 GB/3600 MT/s DDR5
- Hyperthreading: Enabled
- Turbo: Enabled
- C-State: Enabled
- Storage: 1x 1TB Advantech SQFlash (SQF-S25V4-1TDSDC)
- Network devices: 1x Intel® Ethernet I226-LM, 1x Intel® Ethernet I219-LM
- Network speed: 1 GbE
- BIOS: American Megatrends International, LLC. 5.27
- Microcode: 0x123
- OS/Software: Ubuntu* 22.04.4 (kernel 6.5.0-45-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 (Entry) |
|---|---|
| Idle Latency (ns) | 123.8 |
| Memory Bandwidths between nodes within the system (using read-only traffic type) (MB/s) | 53577.1 |

Table 7. Peak Injection Memory Bandwidth (1 MB/sec) Using All Threads

| Peak Injection Memory Bandwidth (1 MB/sec) using all threads | Entry Solution |
|---|---|
| All Reads | 52574.6 |
| 3:1 Reads-Writes | 50359.3 |
| 2:1 Reads-Writes | 50319.6 |
| 1:1 Reads-Writes | 50145.0 |
| STREAM-Triad | 50231.9 |
| Loaded Latencies using Read-only traffic type with Delay=0 (ns) | 464.78 |
| L2-L2 HIT latency (ns) | 47.0 |
| L2-L2 HITM latency (ns) | 47.3 |

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.

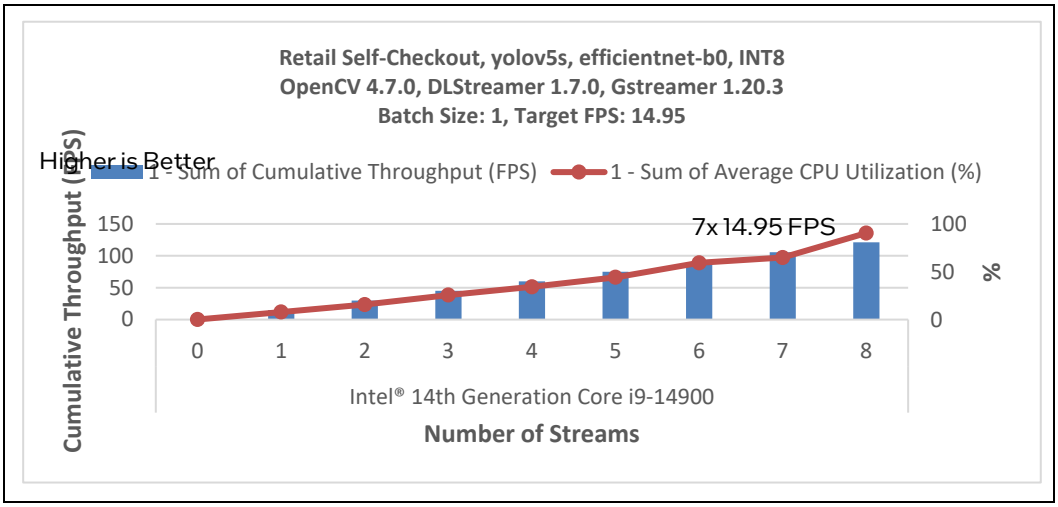
## 4.2 Intel® Automated Self-Checkout on Core

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 efficient net-b0 model for performing classification.

Table 8. 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) |

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




Intel® AI System for Edge Verified Reference Blueprint – Small for Computer Vision, and GEN AI – Entry configuration platform with Intel® 14th Generation Core i9-14900E should be able to service up to 7 IP camera streams at 14.95 FPS per stream, for an aggregate of up to 105.75 FPS.

## 4.3 Intel® Automated Self-Checkout on GPU

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 efficient net-b0 model for performing classification.

Intel® AI System for Edge Verified Reference Blueprint – Small for Computer Vision, and GEN AI – Base platform with Intel® ARC380 should be able to service up to 10 IP camera streams at 14.95 FPS per stream, for an aggregate of up to 150.31 FPS.

Intel® AI System for Edge Verified Reference Blueprint – Small for Computer Vision, and GEN AI – Plus platform with Intel® ARC750 should be able to service up to 12 IP camera streams at 14.95 FPS per stream, for an aggregate of up to 180.06 FPS.

Refer to Table 8 for the software version details.

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

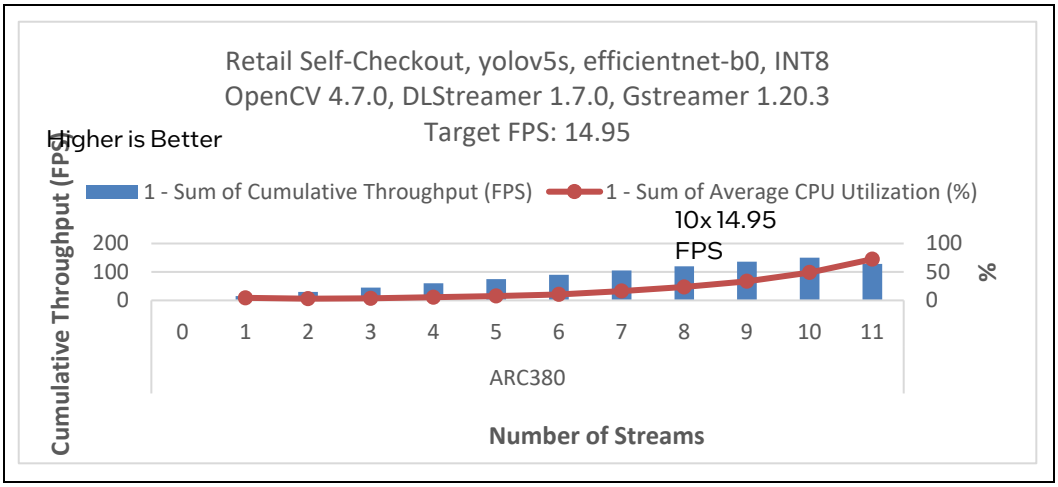
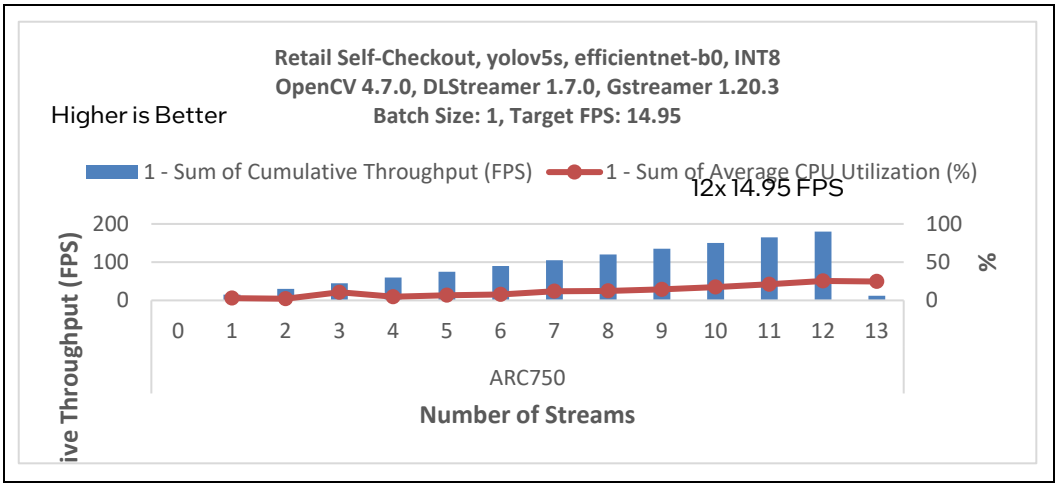



Figure 4. Intel® Automated Self-Checkout Workload Mainstream Configuration Performance




## 4.4 Generative AI on Core

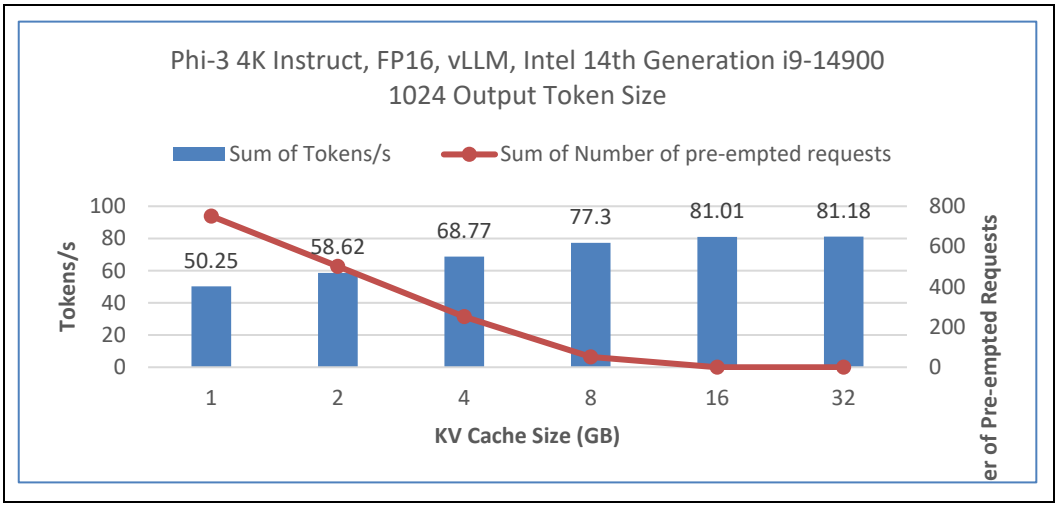
The Large Language Model (LLM) proxy workload highlights the Generative AI processing capabilities of the Intel® AI System for Edge Verified Reference Blueprint – Small for Computer Vision, and GEN AI configuration - Base platform, specifically with the Phi-3 4K Instruct model supported directly on Intel® 14th Generation Core processors.

Intel® AI System for Edge Verified Reference Blueprint – Small Base 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, along with the time per query.

Table 9. Generative AI Workload Configuration

| Ingredient | Software Version Details |
|---|---|
| Docker Engine | 27.1.0 |
| Docker Compose | 2.29 |
| OpenVino Toolkit | 20224.1.0 |
| OS | Ubuntu* 22.04 LTS Kernel 6.5 |

Figure 5. Generative AI Entry Configuration Performance Graph (Phi-3 4K Instruct with Intel® 14th Generation Core i9-14900)

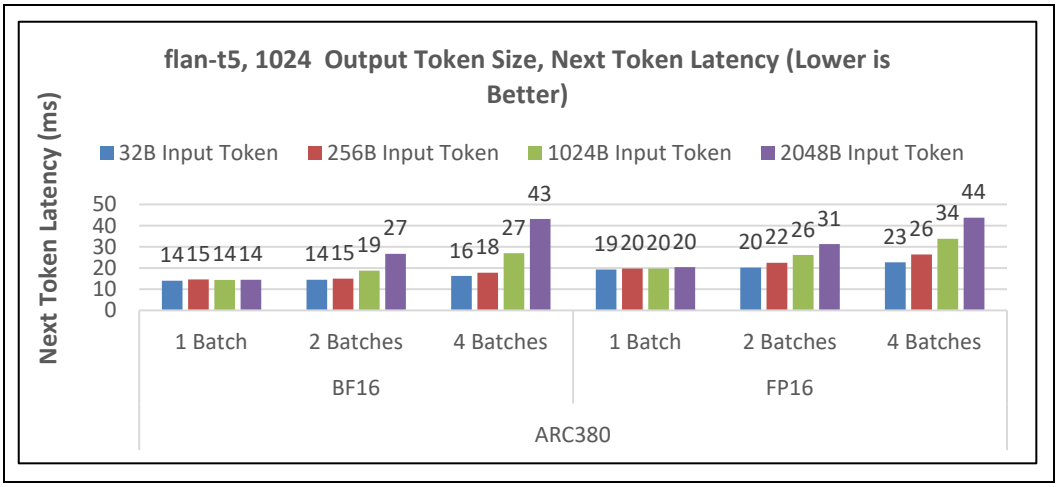


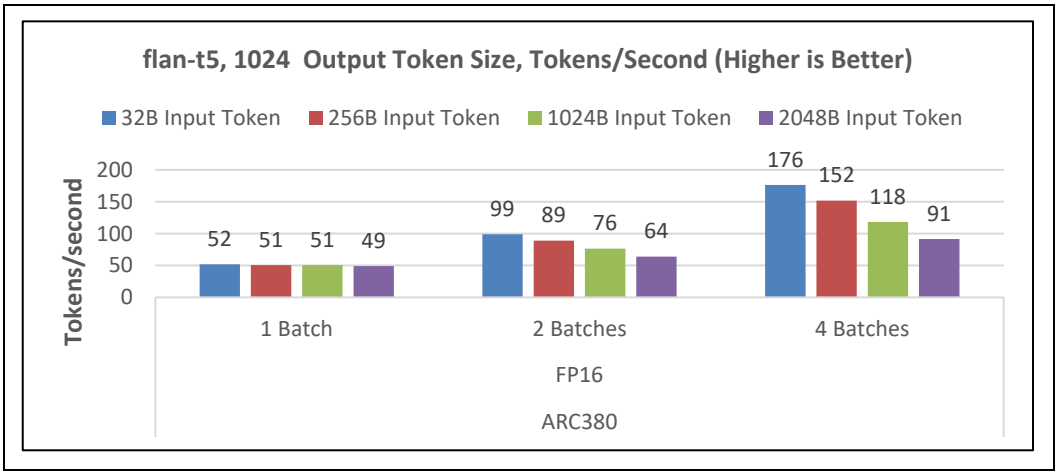

## 4.5 Generative AI on GPU

The large language model (LLM) proxy workload highlights the Generative AI processing capabilities of the Intel® AI System for Edge Verified Reference Blueprint—Small Base and Plus platform. It includes multiple models, including the Flan-t5, TinyLLama 1B, Phi-3 4K Instruct, and Llama3 8B models with Intel® ARC380 and Intel® ARC750 GPUs.

Intel® AI System for Edge Verified Reference Blueprint – Small Base platform and 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, along the time per query.

Figure 6. Generative AI Entry Configuration Performance Graph (flan-t5 with Intel® ARC380)



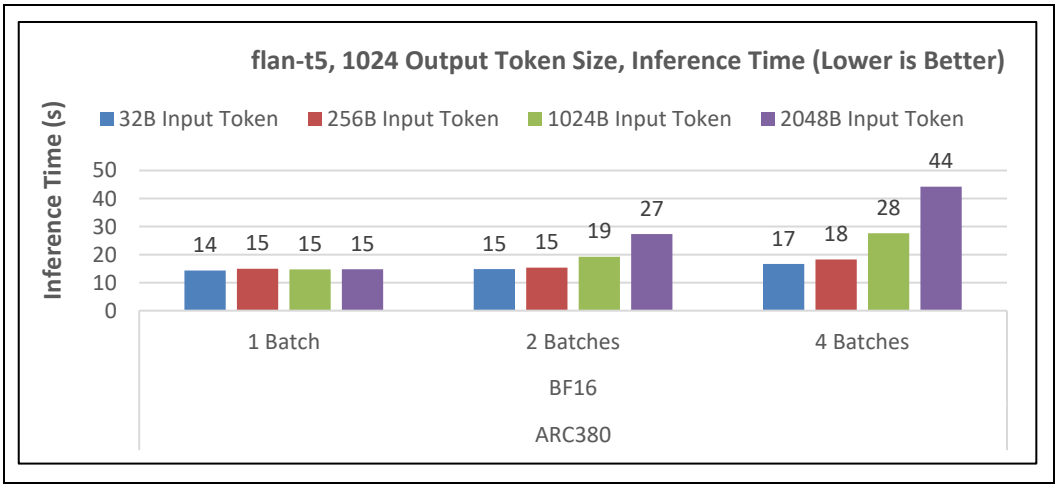
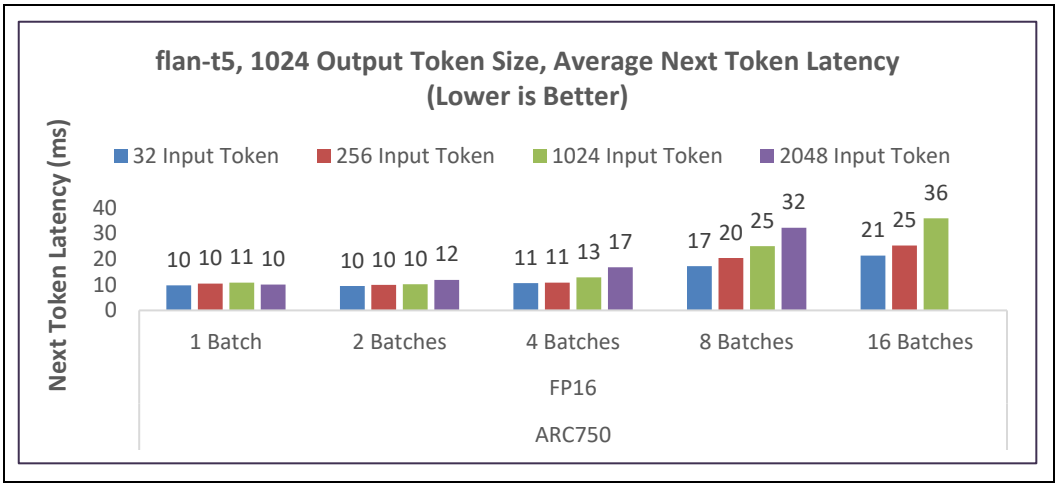
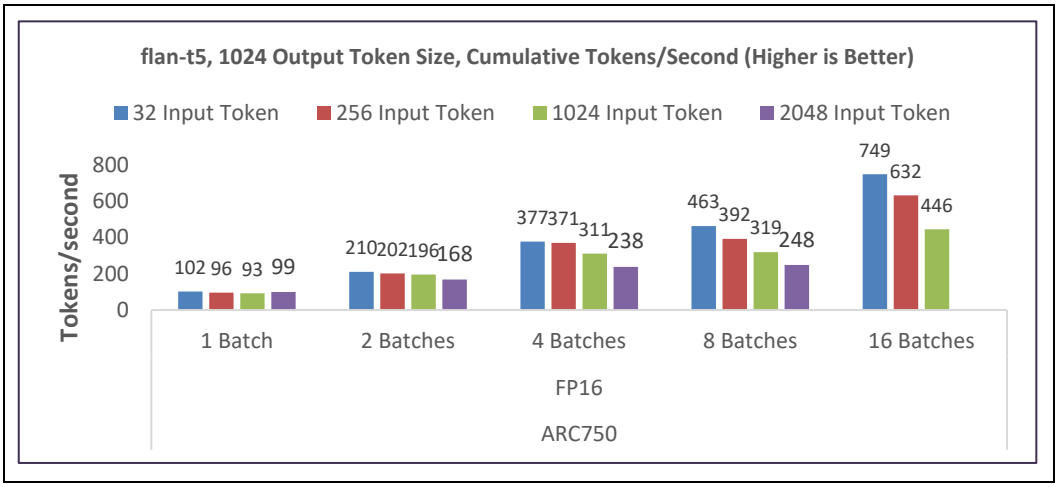



Figure 7. Generative AI Entry Configuration Performance Graph (flan-t5 with Intel® ARC750)



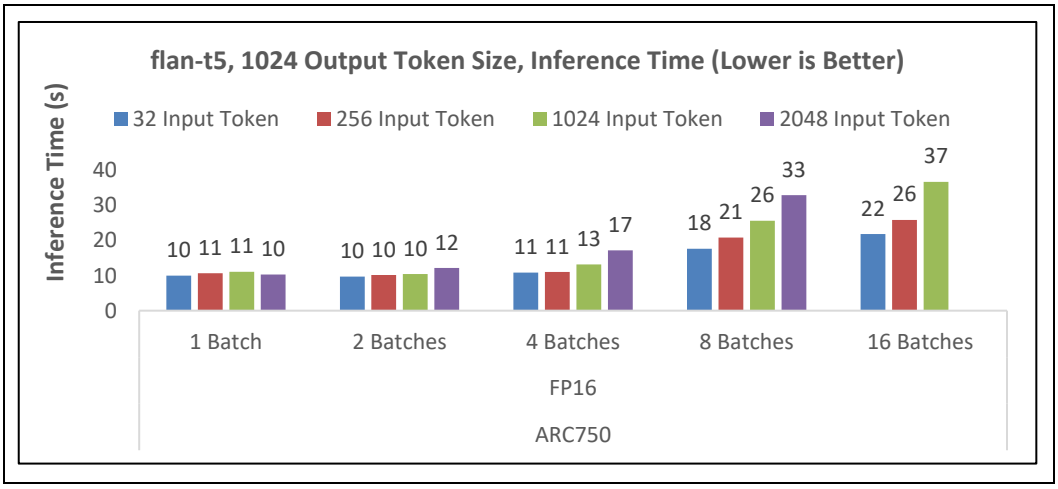
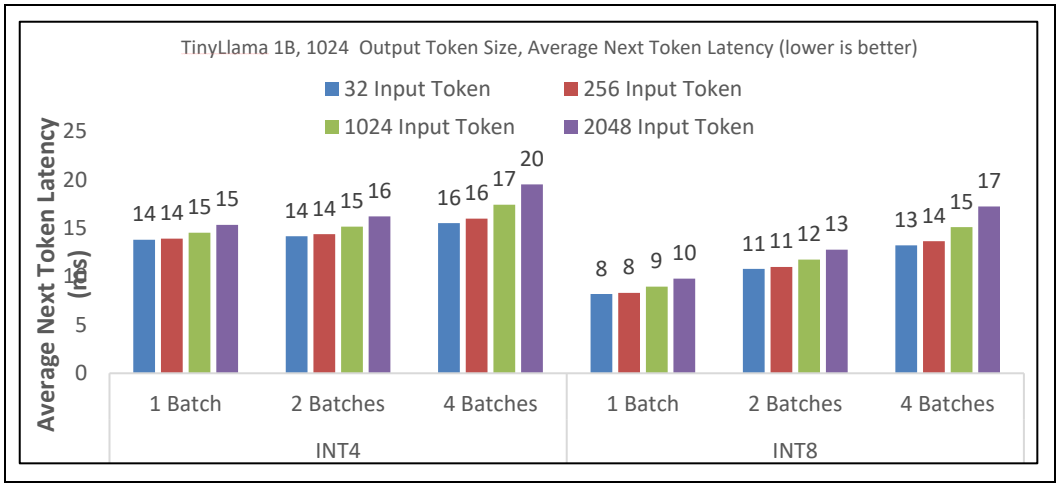
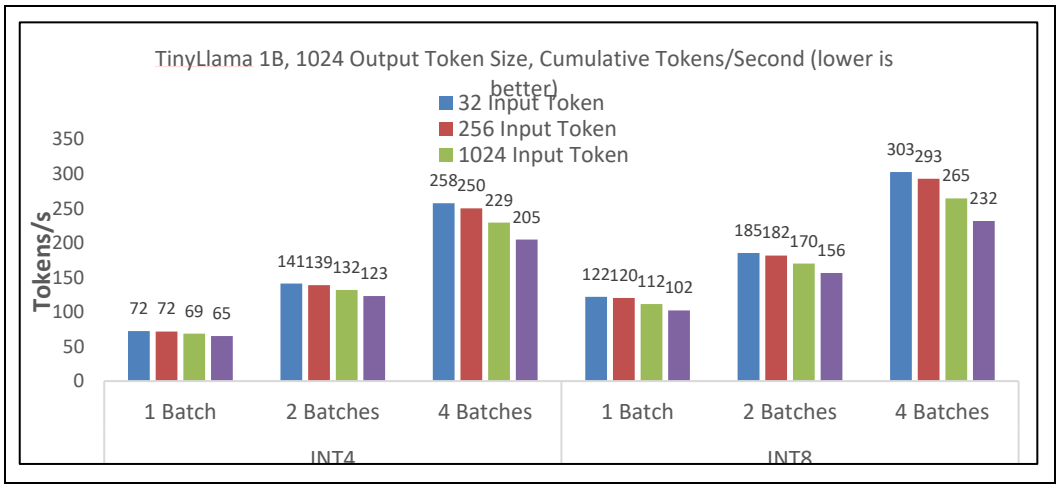



**Figure 8. Generative AI Entry Configuration Performance Graph (TinyLlama with Intel® ARC380)**



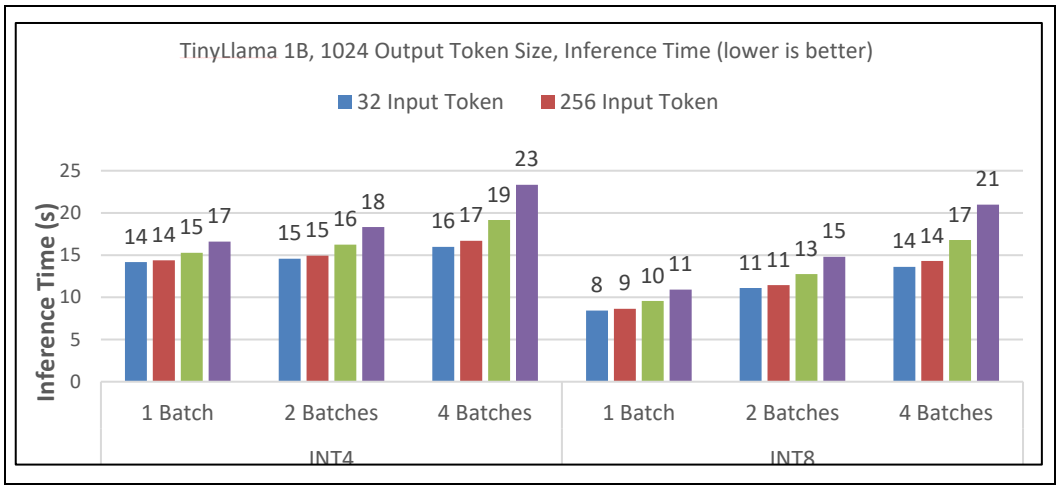
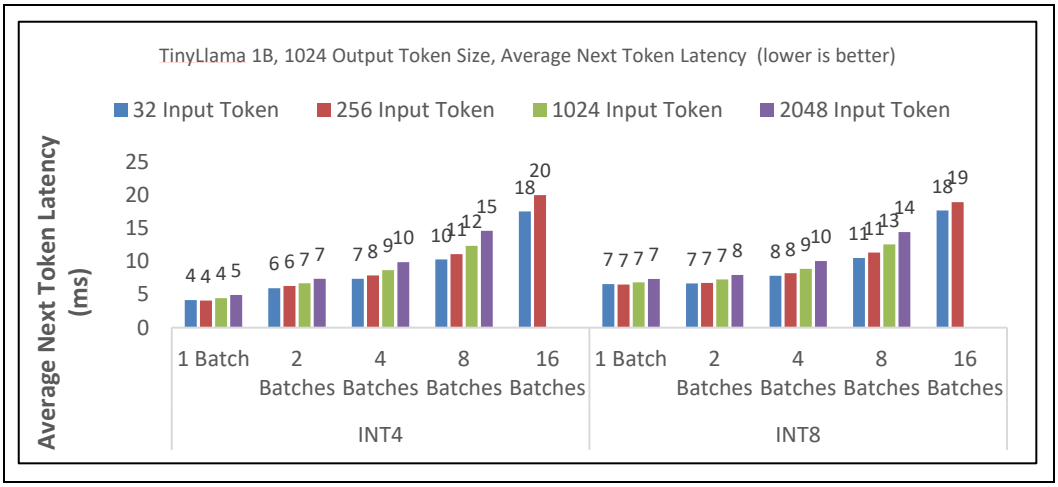
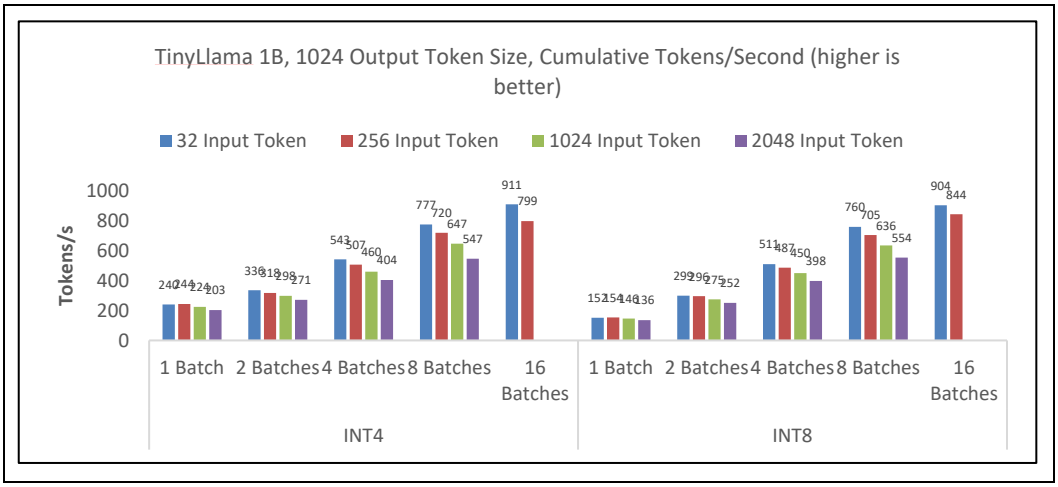



**Figure 9. Generative AI Entry Configuration Performance Graph (TinyLlama with Intel® ARC750)**



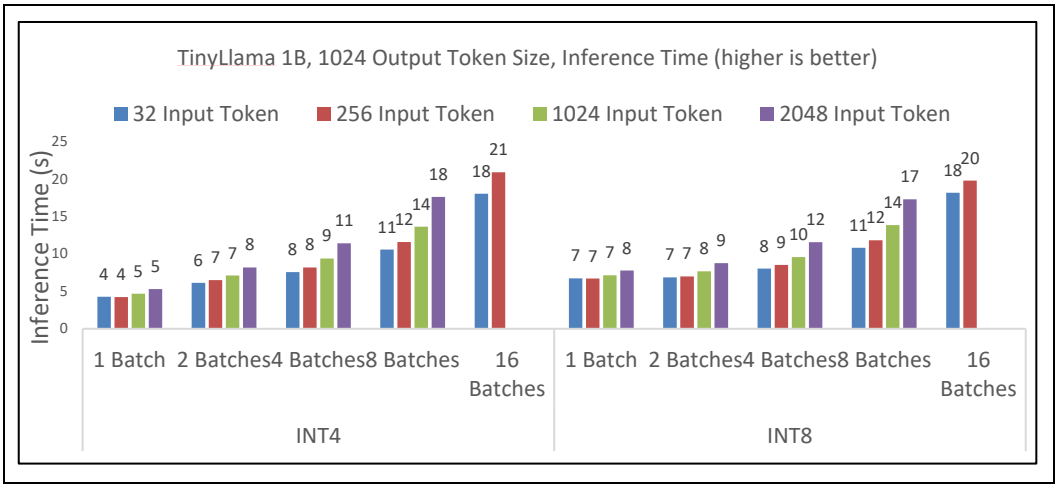
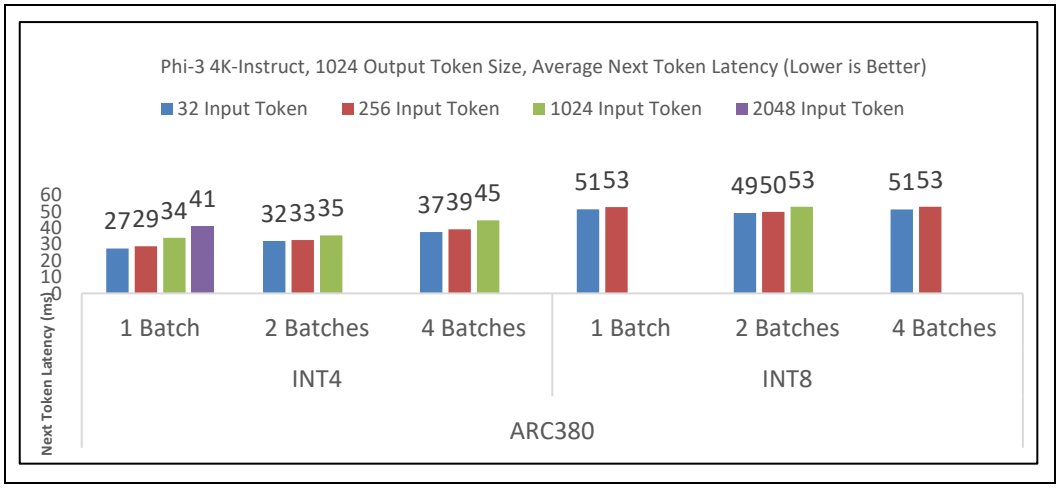
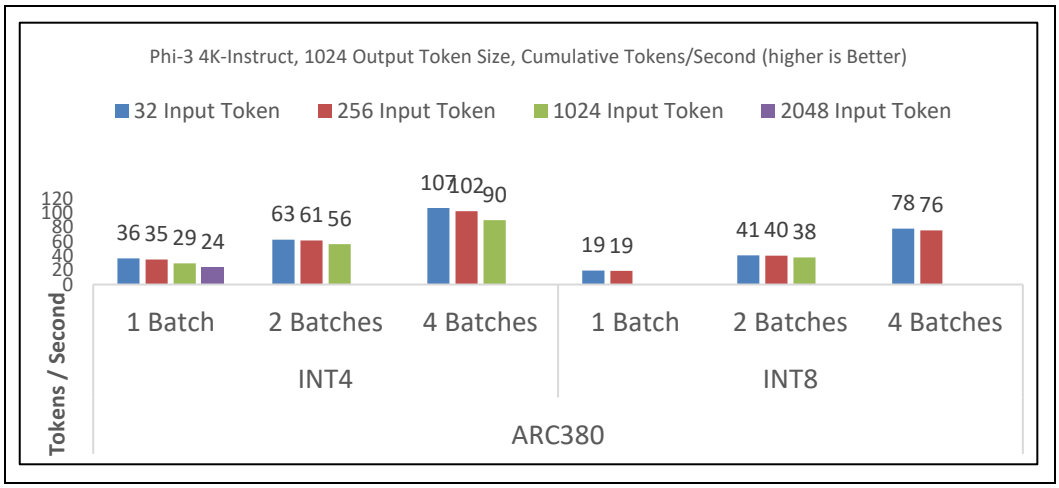



**Figure 10. Generative AI Entry Configuration Performance Graph (Phi-3 4K-Instruct with Intel® ARC380)**



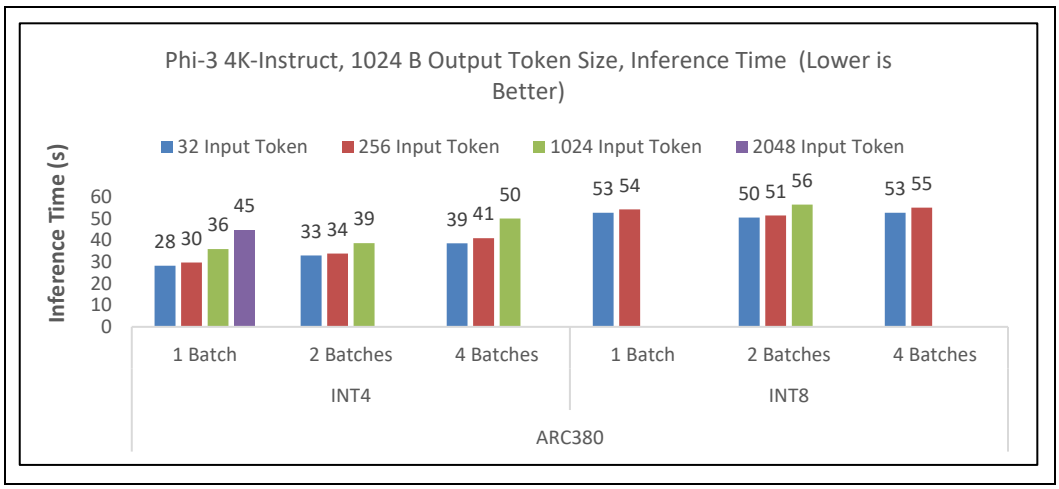
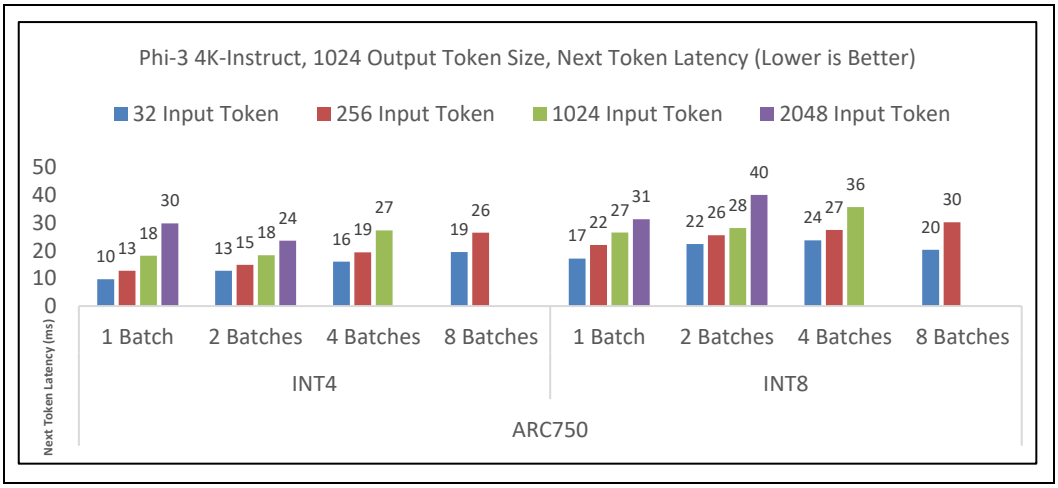
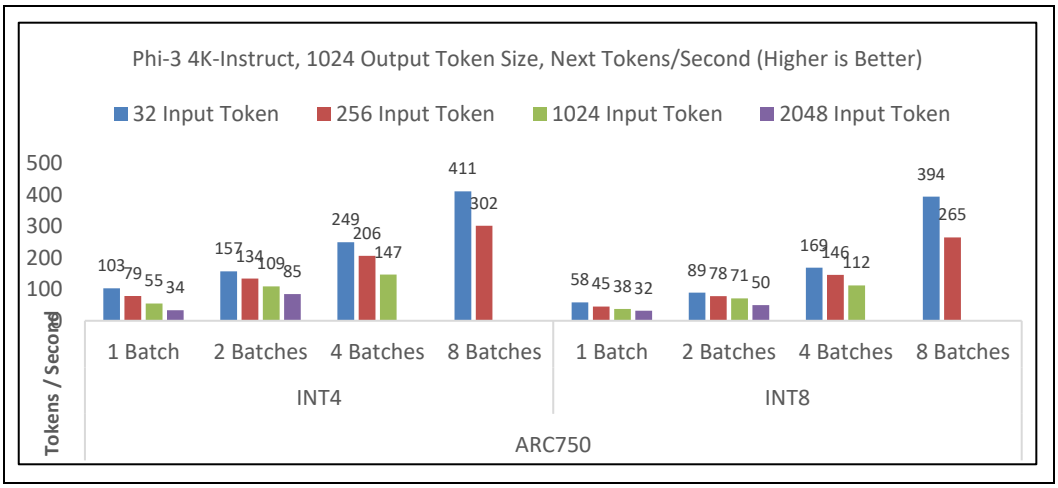
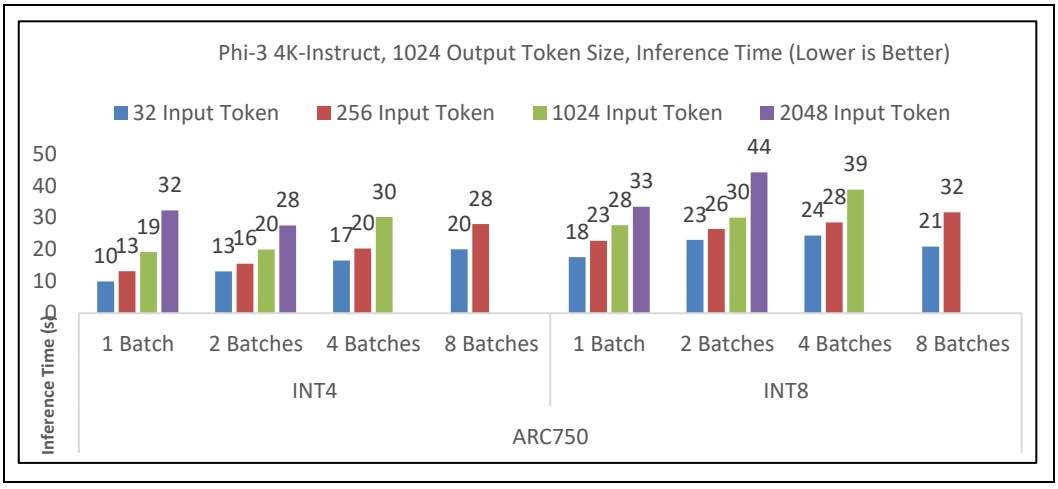



**Figure 11. Generative AI Entry Configuration Performance Graph (Phi-3 4K-Instruct with Intel® ARC750)**



## 4.6 Malconv and BERT

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.

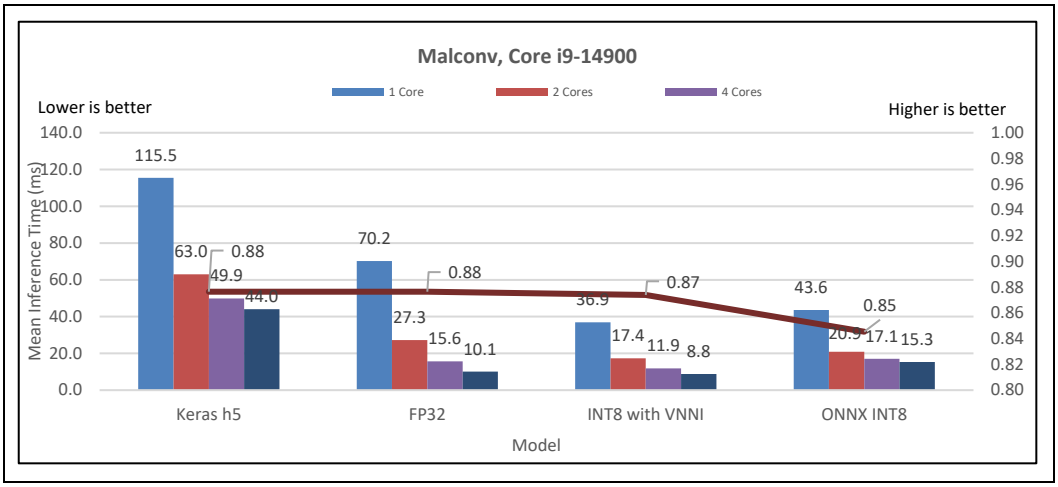
Ensure that the test results follow the expected results, as shown in the following tables, to establish a baseline for the platform's performance. Table 10 shows the software used for the testing while Figure 12 shows a graph of the mean inference time for each model. With 8 cores per instance, the INT8 model with AVX512_CORE_VNNI enabled was able to reach a performance of less than 10 ms.

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

**Table 10. 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 12. 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 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 test results follow the expected results, as shown in the following graph, to establish a baseline for the platform's performance. Table 11 shows the software used for the testing, while Figure 13 shows a graph of the results for the FP32 BERT model. With 8 cores per instance, the mean latency of the model reaches below 150ms.

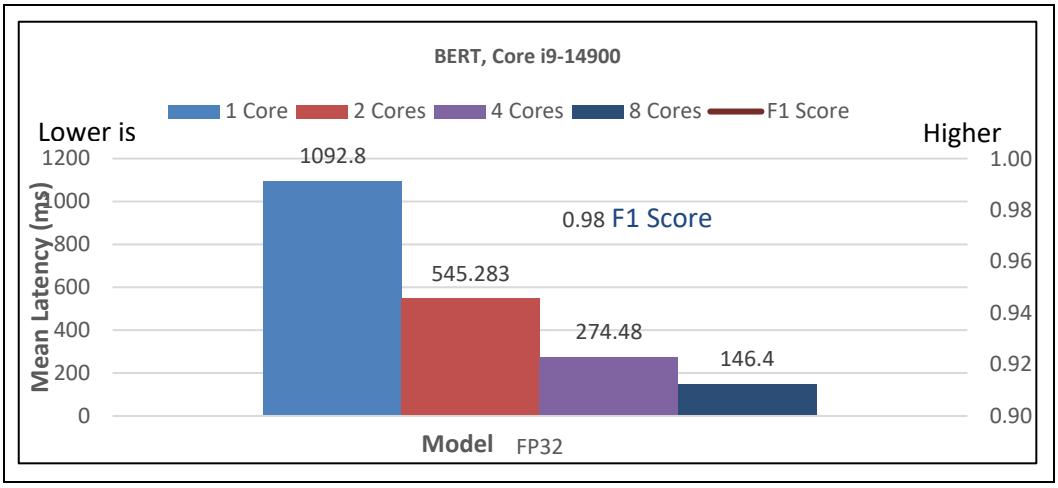
***Note:*** Refer to https://huggingface.co/bert-base-cased for the original Hugging Face BERT base model.

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

**Table 11. 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 13. BERT AI Entry Platform Performance Graph




## 4.7 Performance Summary

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

Figure 14. Performance Summary for the Vision AI Workload

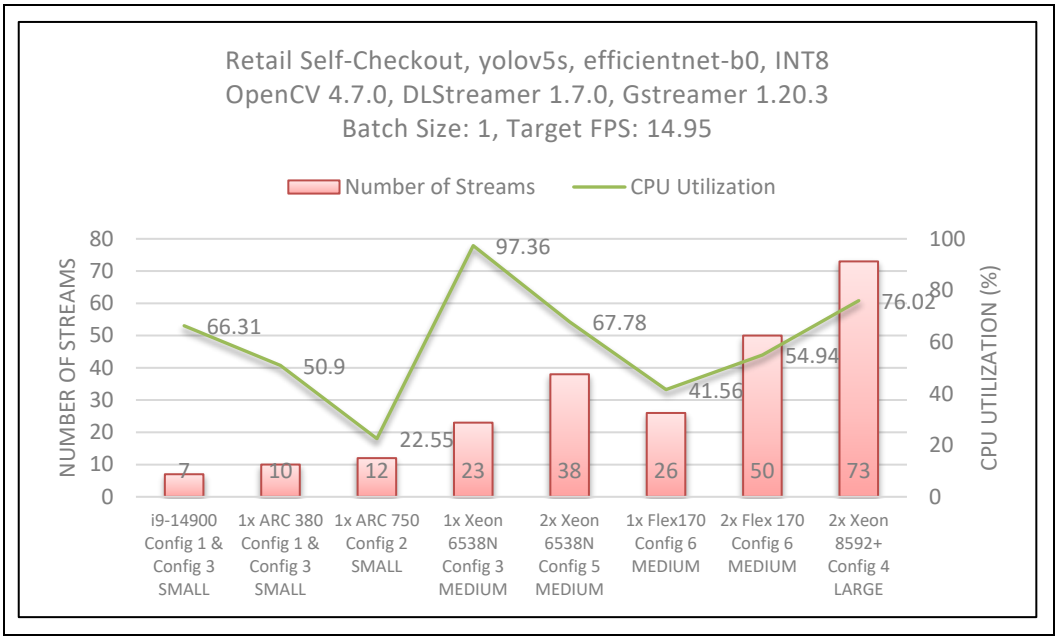


**Figure 15. Performance Summary for the Generative AI Workload**

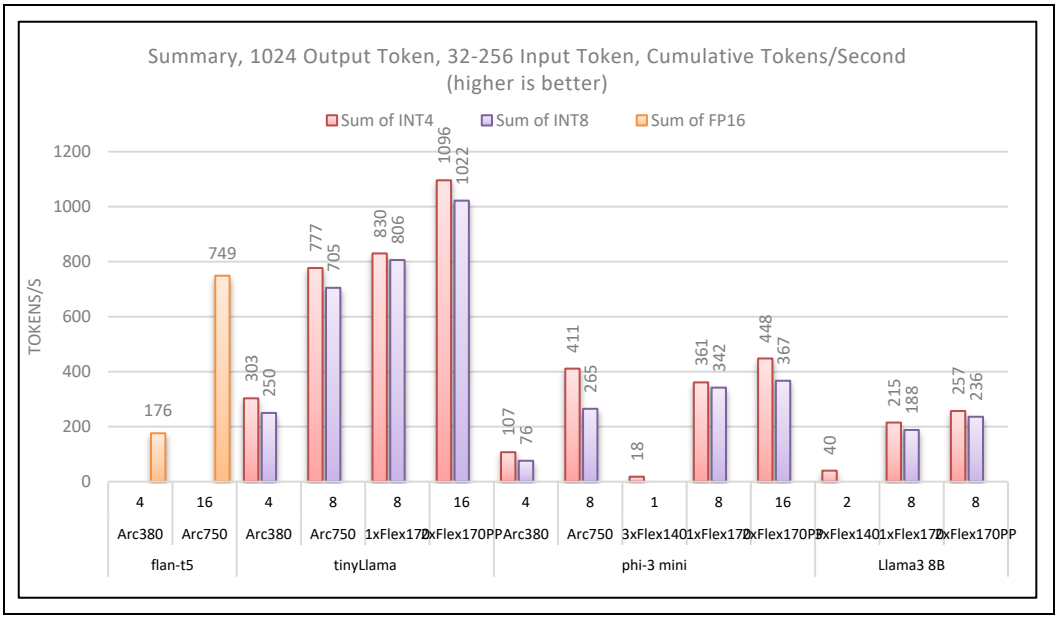



**Figure 16. Performance Summary for the Malconv Network Security AI Workload**

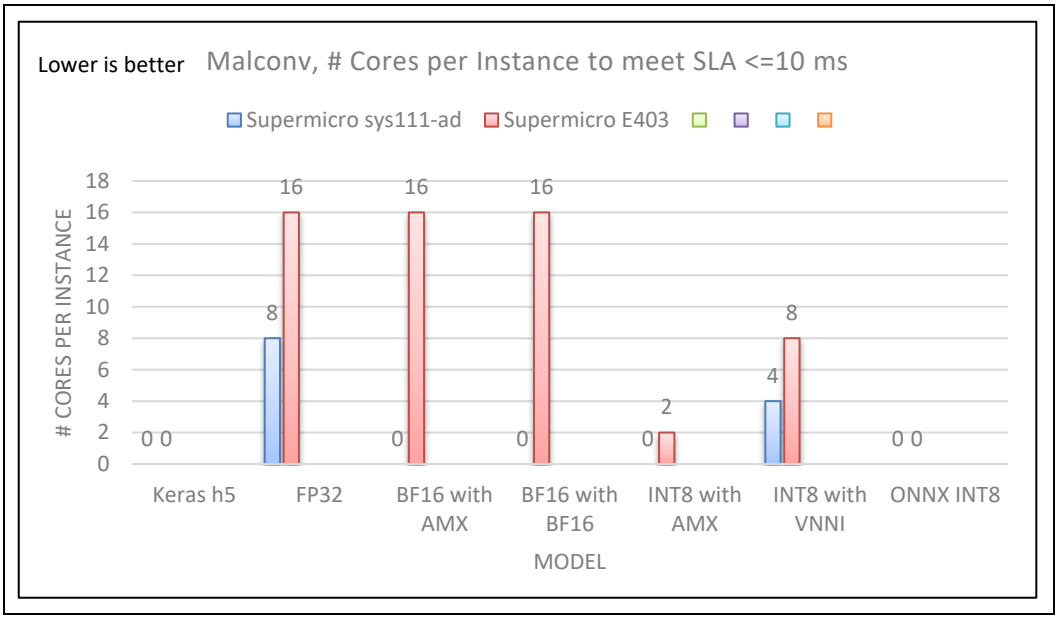
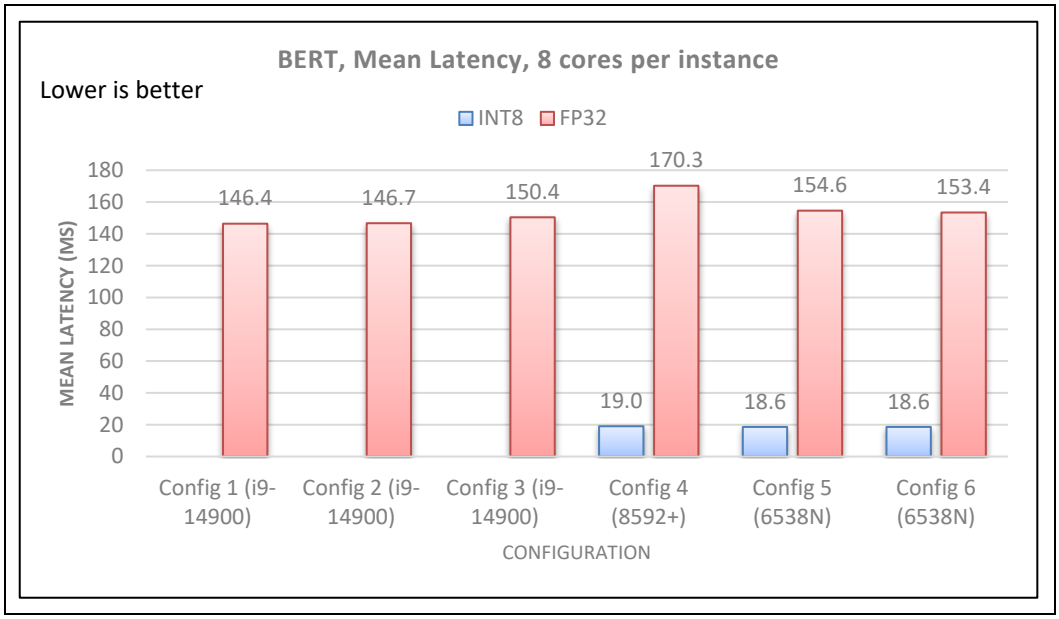


Figure 17. Performance Summary for the Bert Network Security AI Workload




§

# 5 Summary

The Intel® AI System for Edge Verified Reference Blueprint – Small for Computer Vision, and GEN AI defined on 14th Gen Intel® Core processors with Intel® Arc GPUs addresses the capabilities for AI inference by offering the following value proposition:

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

- Up to 7 IP camera streams of Vision AI use case with the Intel® Automated Self-Checkout Reference Package on Small Base CPU configuration
- Up to 10 IP camera streams of Vision AI use case with the Intel® Automated Self-Checkout Reference Package on Small Plus configuration on 1x ARC380 GPU
- Up to 12 IP camera streams of Vision AI use case with the Intel® Automated Self-Checkout Reference Package on Small Plus configuration on 1x ARC 750 GPU

2. For Generative AI use case

With Processor AI inference offload

- Up to 68 tokens/s on Phi-3 mini 4K Instruct model with FP32 precision KV Cache Size of 4GB on Small Base CPU configuration

With GPU AI inference Offload

- Up to 157-176 tokens/s on Flan-t5 model with FP16 precision Batch size of 4 on Small Plus on 1x ARC380 GPU configuration
- Up to 632-740 tokens/s on Flan-t5 model with FP16 precision Batch size of 16 on Small Plus on 1x ARC750 GPU configuration
- Up to 250-303 tokens/s on TinyLlama model with INT4 or INT8 precision Batch size of 4 on Small Plus on 1x ARC 380 GPU configuration
- Up to 705-777 tokens/s on TinyLlama model with INT4 or INT8 precision Batch size of 8 on Small Plus on 1x ARC 750 GPU configuration
- Up to 76-107 tokens/s on Phi-3 mini 4k Instruct model with INT4 or INT8 precision Batch size of 4 on Small Plus on 1x ARC 380 GPU configuration
- Up to 265-411 tokens/s on Phi-3 mini 4k Instruct model with INT4 or INT8 precision Batch size of 8 on Small Plus on 1x ARC 750 GPU configuration
- Up to 275-285 tokens/s on Llama3 8B model with INT4 precision Batch size of 8 on Small Plus configuration on 1x ARC 750 GPU configuration

3. For Network Security AI use case

- Malconv testing, the INT8 model with AVX512_VNNI enabled was able to reach a performance of less than 10 ms. with 8 cores per instance.
- Bert testing, the mean latency of the FP32 model reaches below 150ms 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 of 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® Arc GPUs and OpenVINO.

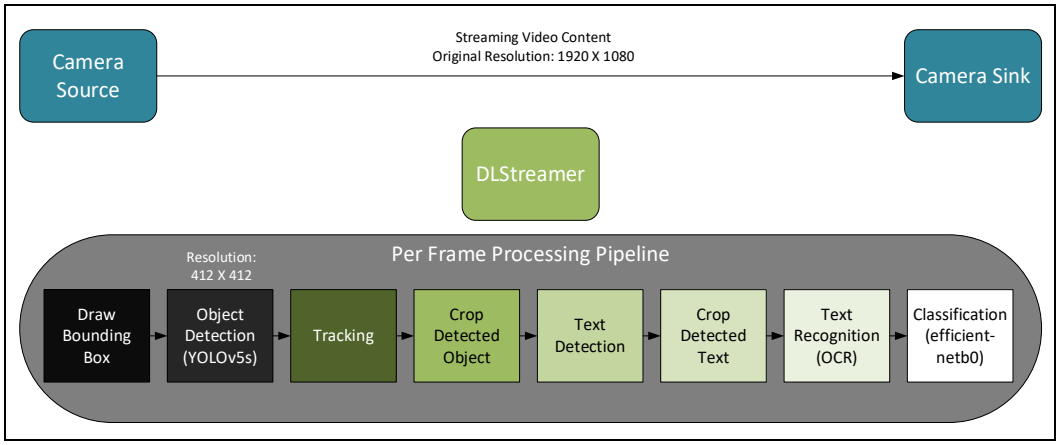
§

# 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 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 18. 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: efficient net-b0
- OpenVino 2024.0.1.
- DLStreamer 2024.0.1
- FFmpeg 2023.3.0
- VPL 2023.4.0.0-799
- The test measures the number of streams that the server can sustain at the target FPS. For each test iteration, the number of camera streams is monotonically increased until the currently measured FPS value falls below the target FPS value.
- 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 wall power.

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

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

```
# cd enterprise_ai/common/retail-self-checkout/
# ./init_rsc.sh
```

2. 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>
…
```

3. Start the benchmark against 14th Generation Intel® Core 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 steady state.

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

4. 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
```

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

```
./turbostat_log_parser.py --log-file
~/rsc_results/turbostat_cpu_batch_1.log --csv-file-name
~/rsc_results/turbostat_cpu_batch_1.csv
```

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

```
./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
```

7. Start the benchmark against Intel® ARC 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 steady state.

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

8. 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
```

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

```
./turbostat_log_parser.py --log-file
~/rsc_results/turbostat_gpu_batch_1.log --csv-file-name
~/rsc_results/turbostat_gpu_batch_1.csv
```

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

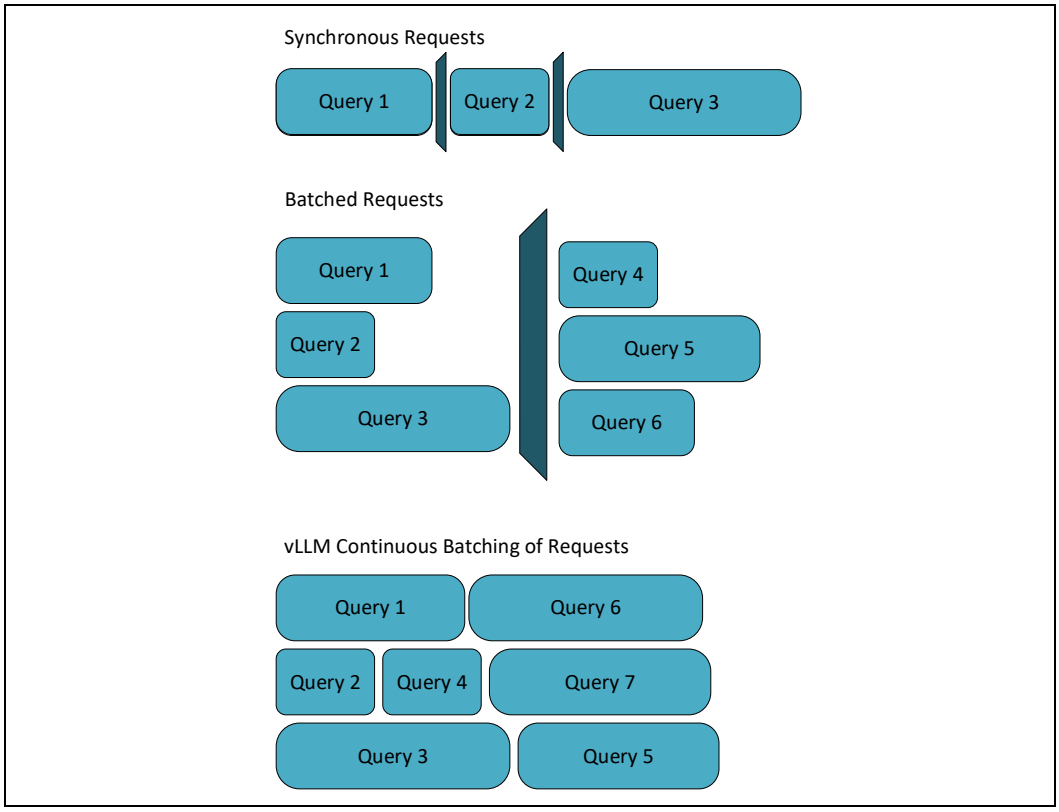
```
./server_power_log_parser.py --log-file
~/rsc_results/server_power_cpu_batch_1.log --csv-file-name
~/rsc_results/server_power_cpu_batch_1.csv
```

# A.2 Generative AI Test Methodology

## A.2.1 vLLM Testing Methodology on Core

The Generative AI benchmark on the 14th Generation Intel® Core leverages vLLM, which, as shown in the figure below, performs continuous batching of requests to the LLM.

Figure 19. vLLM Continuous Batching




To set the vLLM test and benchmark on core:

1. Clone the vLLM project and install the baseline dependencies:

```
# git clone https://github.com/vllm-project/vllm.git
```

```
# cd vllm
# pip install -e .
# apt-get update
# apt-get install python3
# pip install -upgrade pip
# pip install -r requirements-build.txt -extra-index-url
https://download.pytorch.org/whl/cpu
```

2. Install vLLM with the OpenVino backend:

```
# PIP_EXTRA_INDEX_URL="https://download.pytorch.org/whl/cpu"
VLLM_TARGET_DEVICE=openvino python -m pip install -v .
```

3. Download the Phi-3 4K Instruct model from HuggingFace:

```
# huggingface-cli download microsoft/Phi-3-mini-4k-instruct --local-dir
~/Phi-3-mini-4k-instruct
```

4. Download the dataset:

```
#
wget https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered/resolve/main/ShareGPT_V3_unfiltered_cleaned_split.json
```

5. Set the vLLM environment variables. For example, to use a KV cache size of 1GB:

```
# export VLLM_OPENVINO_KVCACHE_SPACE=1
# export VLLM_OPENVINO_CPU_KV_CACHE_PRECISION=u8
# export VLLM_OPENVINO_ENABLE_QUANTIZED_WEIGHTS=ON
# export TOKENIZERS_PARALLELISM=false
```

6. Start the vLLM benchmark:

```
# python3 ./benchmark_throughput.py --model ~/Phi-3-mini-4k-instruct --
dataset ./ShareGPT_V3_unfiltered_cleaned_split.json --enable-chunked-
prefill --max-num-batched-tokens 256
```

### A.2.2 IPEX-LLM Testing Methodology on GPU

The Generative AI benchmark on Intel® ARC GPUs leverages the IPEX-LLM framework and is deployed in a containerized manner.

To run the Generative AI benchmark on Intel® ARC GPUs:

1. 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
```

2. Launch the IPEX-LLM container. For example to benchmark with the Phi-3 4K Instruct model:

```
# export CONTAINER_NAME=ipex-llm-serving-xpu
# export MODEL_PATH=~/Phi-3-mini-4k-instruct
# docker run -itd Let me know if there is anything else I can help you
with.
              --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 Phi-3 4K Instruct model:

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

…
current_model_name="Phi-3-mini-4k-instruct"
…
```

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 4.6.

§