Heterogeneous AI Powerhouse: Unveiling the Hardware and Software Foundation of Intel® Core™ Ultra Processors for the Edge

White Paper

Authors:

Ramesh Perumal, Software Enabling and Optimization Engineer
Nikitha Chinthalapani, Product Manager
Rohit D'Souza, AI Product Manager

Contributors:

Isaac Sever, Product Line Manager
Steve Goodell, Product Line Manager
Ryan Loney, Senior Product Manager for OpenVINO™
Eka A. Kurniawan, Software Enabling and Optimization Engineer

April 2024

Document Number: 817736-1.0

Performance varies by use, configuration and other factors. Learn more at Performance Index site.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. No product or component can be absolutely secure.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or visiting the Intel Resource and Documentation Center.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel is committed to respecting human rights and avoiding complicity in human rights abuses. See Intel's Global Human Rights Principles. Intel's products and software are intended only to be used in applications that do not cause or contribute to a violation of an internationally recognized human right.

Contents

April 2024
Document Number: 817736-1.0

Figures

Tables

Revision History

Date	Revision	Description
April 2024	1.0	Initial release.

1.0 Abstract

This white paper provides an overview of the artificial intelligence (AI) capabilities of Intel® Core™ Ultra Processors for the edge, delving into hardware architecture, technical specifications, and software enablement, including a detailed case study.

The paper begins with introducing the family of Intel® Core™ Ultra Processors, highlighting the key differences between the various SKUs of the processor family and outlining the hardware capabilities in the processors that provide AI acceleration. It then provides a snapshot of the technical specifications of various SKUs in the processor family.

The paper explores the software ecosystem and how OpenVINO™ unlocks the full potential of Intel Core Ultra processors for edge AI development, providing a robust and mature toolkit for optimizing and deploying deep learning models. A section on benchmarks highlights the throughput of a few AI models, targeting image classification and object detection.

Finally, the paper concludes with a case study on product classification and person detection use cases executing on each of the three AI engines – Central Processing Unit (CPU), integrated Graphics Processing Unit (iGPU), and built-in Neural Processing Unit (NPU). The case study provides step-by-step instructions to reproduce the results to enable developers. By combining powerful hardware with a developer-friendly software environment, Intel Core Ultra processors offer a competitive solution for various demanding edge AI workloads.

2.0 Introduction

Artificial Intelligence (AI) at the Edge requires diverse workloads to run efficiently on a single platform. AI needs at the Edge are unique because processing happens on local devices or on-prem servers rather than the cloud. The edge can be any device outside a central data center, such as network video recorders, retail kiosks, industrial robots, ultrasound machines, etc. Instead of sending data to the cloud for processing, AI algorithms are run on the edge device. In this resource-constrained environment, AI platforms must provide (but are not limited to) –

- **Performance Acceleration:** Efficient hardware acceleration is critical to handle AI tasks at the edge.
- **Power Efficiency:** Low power consumption is essential to ensure power efficiency or efficient operation without dedicated sources.
- **Low Latency:** Low latency is critical for applications requiring real-time processing. Eliminating the need for data to travel back and forth to the cloud significantly reduces latency.
- **Security and Privacy:** Sensitive data must stay on device or on premises and processed locally.

Intel is dedicated to solving all these challenges at the edge with the launch of Intel® Core™ Ultra Processors (formerly known as Meteor Lake). By prioritizing computational efficiency, low power consumption, low latency, and security, developers can create intelligent edge devices that operate autonomously, make real-time decisions, and optimize resource utilization.

3.0 Introducing Intel® Core™ Ultra Processors for the Edge

Intel® Core™ Ultra Processors, with its AI focus, hold tremendous potential to address key challenges at the edge. Efficient AI processing for diverse workloads is enabled by three dedicated engines (as shown in Figure 1):

- Neural Processing Unit (NPU): The built-in NPU is a dedicated AI accelerator that offloads AI workloads from the CPU and GPU, improving the overall efficiency and power consumption.
- Graphics Processing Unit (GPU): The integrated Intel® Arc™ GPU further scales AI capabilities and graphics and media processing at the edge.
- Central Processing Unit (CPU): The CPU is the ideal AI engine when low latency is critical for AI workloads.

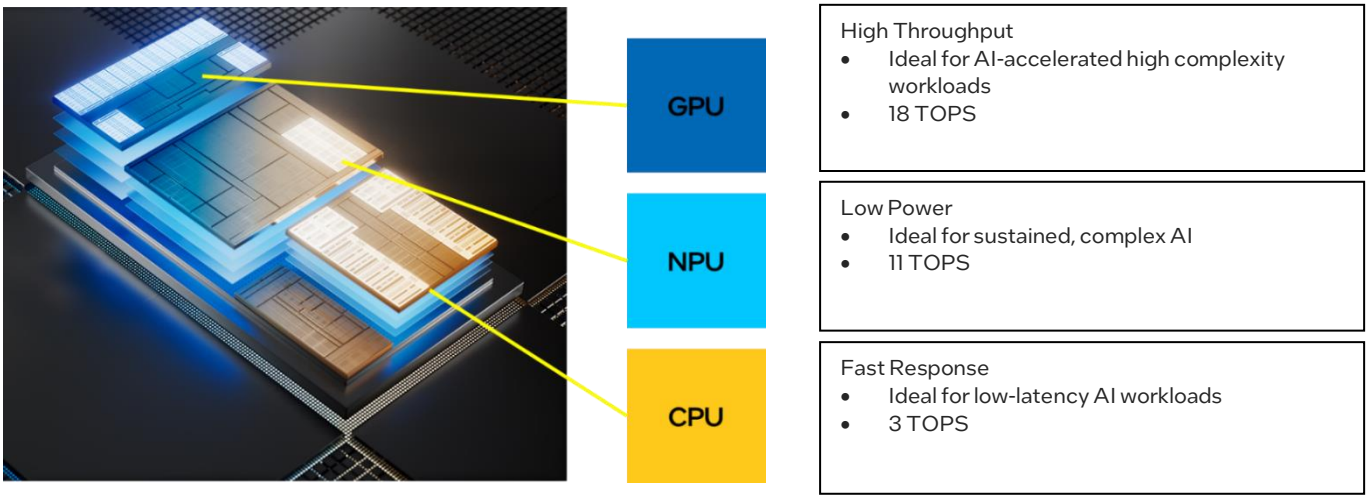

Figure 1: High-Level Architecture Diagram

3.1 Hardware Features of Intel® Core™ Ultra Processors for Edge AI

Intel® Core™ Ultra Processors is the next generation of Intel® Core™ Series for Edge platforms, a follow-up to the 12th and 13th Generation Intel® Core® Processors (formerly known as Alder Lake and Raptor Lake). This is the first processor built on Intel 4 process technology.

Intel® Core™ Ultra Processors have the following integrated compute engines that help in AI acceleration:

- Up to 16 cores/22 threads (Up to 6 P-cores, up to 8 E-cores & 2 LPE-cores)
- Intel® Arc™ graphics engine (X^e LPG) with up to 8 X^e cores (up to 128 Execution Units or EUs)[1]
- Intel® AI Boost, a built-in neural processing unit (NPU)

Intel® Core™ Ultra Processors include options to support both BGA soldered and LGA socket solutions, including an integrated PCH. The UL and HL parts come in the versatile LGA socket configuration, enabling a scalable platform across power levels and performance. The U and H parts in a BGA package provide board down solutions for ruggedized and small form factor designs.

3.1.1 CPU

Intel® Core™ Ultra Processors are the first client processors on Intel 4 process with modular architecture: They consist of P- cores (Performance), E- cores (Efficient) & LP-E cores (Low-power Efficient). Performance-cores are optimized for low latency single-threaded, compute-intensive workloads, while Efficient-cores are optimized for multi-threaded, less compute-intensive workloads. The newest core type, Low-power Efficient cores, is designed for scalable multithreaded performance and offloading background tasks. P-cores provide heavy lifting power for single & limited threading performance, whereas E- and LP-E cores provide multi-threaded throughput and power efficiency. Intel® Thread Director prioritizes and manages the distribution of workloads, sending tasks to optimized cores[2].

Intel® Deep Learning Boost (Intel® DL Boost) provides built in AI acceleration for training and inference workloads. This includes VNNI (for CPU) and DP4a (for GPU) instruction set support first introduced in 12th generation Intel® Core™ Processors brings improved INT8 inferencing performance to edge workloads.

3.1.1.1 Vector Neural Network Instructions (VNNI)

Intel® DL Boost includes VNNI, an extension to the Intel instruction set. It is specifically designed to accelerate convolutional neural networks (CNNs), widely used in tasks such as image recognition, object detection, and natural language processing. As illustrated in Figure 2, it achieves this acceleration by combining three instructions into

[1] Intel® Arc™ GPU only available on select HL-Series, Intel® Core™ Ultra processor powered systems with at least 16GB of system memory in a dual-channel configuration.

[2] For more details on hybrid architecture, please refer to Performance Hybrid Architecture

one for execution, which increases the inference performance of the INT8 model. It is accessible through industry standard frameworks and libraries.

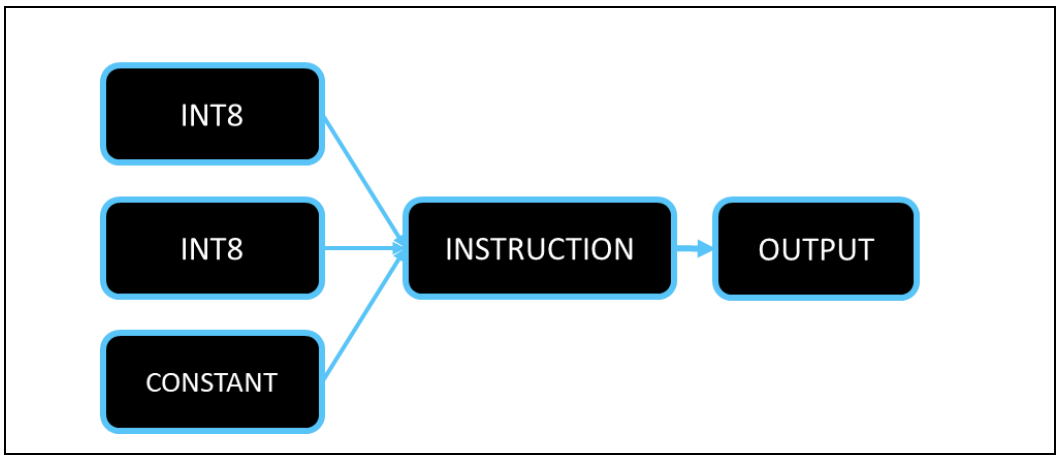

Figure 2: Vector Neural Network Instruction (VNNI)

3.1.2 Intel® Arc™ Graphics

New integrated Intel® Arc™ GPU with up to 8 Xᵉ-cores (128 EUs) offers up to 1.81x performance improvement over the 13th Gen Intel® Core™ processors[3]. It provides entry level discrete graphics performance in integrated form factor. Intel® Arc GPU capability is available on select -H/HL SKUs that support higher levels of Xᵉ-cores and the required memory configuration. Lower power -U/UL SKUs have up to 4 Xᵉ-cores (64 EUs) with improved graphics performance over the previous generation.

Figure 3 depicts the Intel Xᵉ-core, which is the fundamental building block of the GPU, with vector and matrix engines, each of which can process 256 bits and 1024 bits per engine, respectively.

[3]Performance varies by use, configuration, and other factors. Learn more at intel.com/processorclaims: Intel® Core™ Ultra processors, Edge. Results may vary.

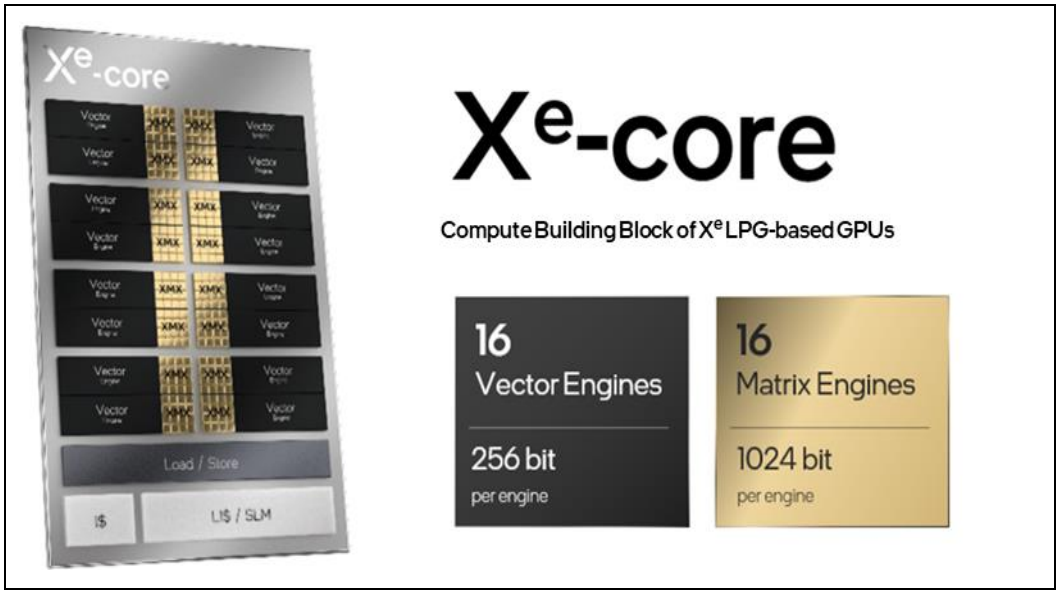

Figure 3: Xe-core: Compute Building Block of Xe LPG GPU Die

Key Features:

- New Low Power Graphics IP
- DX12 Ultimate Feature set
  - Ray Tracing
  - Mesh Shading
  - Variable Rate Shading Tier 2
  - Sampler Feedback
- Power & Area Efficiency Optimized
- Dual Render Slice scalability
- Intel® Deep Learning Boost: DP4a

3.1.2.1 DP4a (Dot-Product of 4 Elements and Accumulate)

Intel® DL Boost consists of DP4a instruction set extension designed to perform dot product operations on four pairs of 8-bit integer values simultaneously and accumulate the results, as shown in Figure 4. This instruction set is optimized with OpenVINO™ Toolkit and oneAPI to accelerate INT8 inferencing.

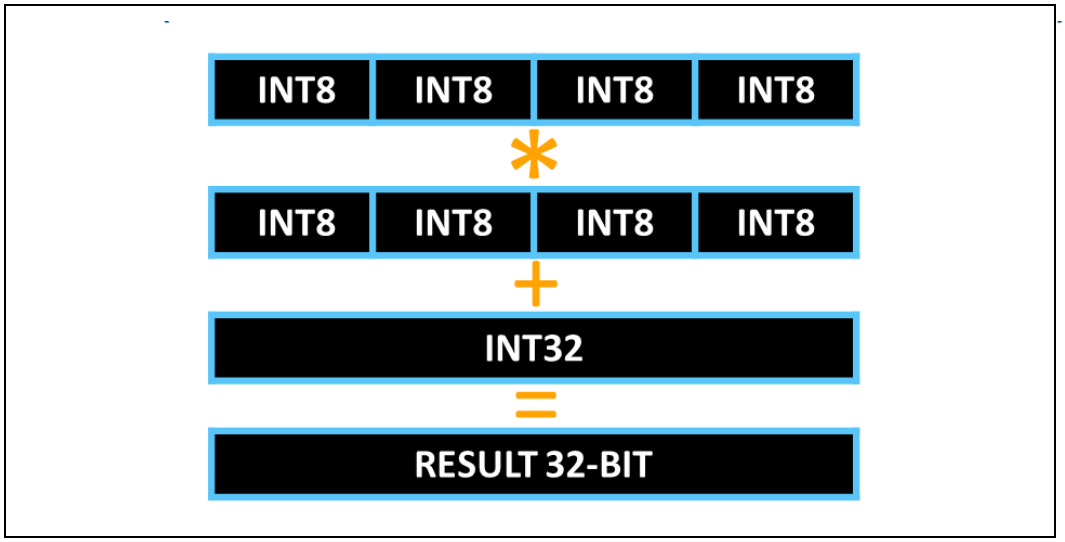

Figure 4: Dot-Product of 4 Elements and Accumulate (DP4a)

3.1.3 Neural Processing Unit

Neural Processing Unit (NPU) is a low-power integrated AI accelerator offered for the first time with Intel® Core™ Ultra Processors. It lets you offload certain neural network computation tasks from CPU and GPU, for more streamlined resource management. NPU provides up to 2.56x AI Performance/Watt vs. previous generation[4] along with improved system and application responsiveness and reduced memory I/O usage.

The NPU is enumerated to the host processor as an integrated PCIe device. NPU consists of two Neural Compute Engine (NCE) tiles. Each tile is configured with 2K Multiply Accumulate (MAC) Engines, Post Processing Engine, AI DSP Processor, and 2 MB of dedicated SRAM memory per tile as shown in Figure 5.

[4]Intel® Core™ Ultra 7 processor 165H 28W on NPU is expected to deliver up to 2.56 times the AI Performance/ Watt of previous generation P Series Intel® Core™ i7-1370P. See Intel® Core™ Ultra Processors and Intel® Core™ Ultra Processors - 1 | Performance Index

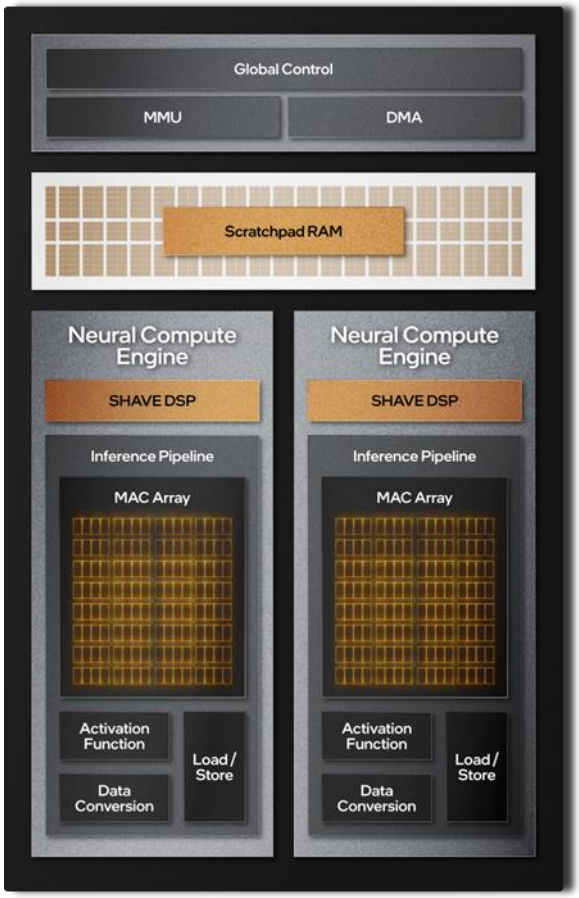

Figure 5: Neural Processing Unit (NPU)

3.1.3.1 Graphics SR-IOV

Intel® Graphics Virtualization Technology (SR-IOV) is the latest addition to its suite of virtualization technologies. Graphics virtualization allows multiple Virtual Machines (VMs) to access high-quality, high-performance graphics, with minimal software overhead. The graphics virtualization feature adds hardware and firmware to improve performance and enable VMM (Virtual Machine Monitors) to support Intel® Graphics Technology in a standard way, eliminating special requirements that could be barriers to adoption. It brings close to native GPU performance to the VM which owns the graphics.

3.1.3.2 Media

Intel® Core™ Ultra processors are the first generation of platforms with stand-alone media IP (SA media) that is disaggregated from the graphics die to enable low power

video playback and highly efficient video processing especially on non-compressed data. It has improved HEVC, AV1, VP9 and AVC HW accelerated decode support and HEVC, VP9 and AVC HW accelerated encode support from the previous generation Core platforms and in addition has -

- New AV1 HW Encode support
- AV1 HW film grain support which helps in improving AV1 compressions efficiency
- 8K30 HDR10 and Dolby Vision* support
- Improved video decode density for edge media analytics pipelines

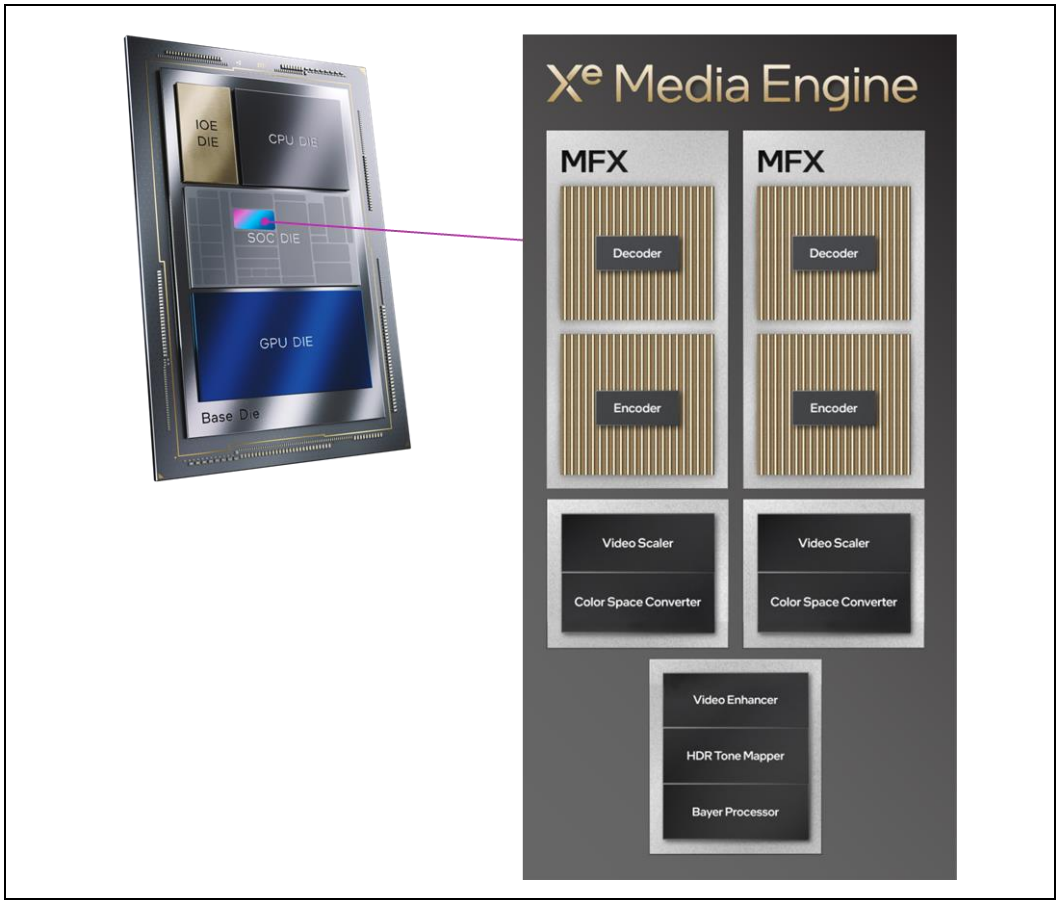

Figure 6: Xe Media Engine

3.2 Technical Specifications

Intel® Core™ Ultra processors consist of U, H, UL & HL SKUs -

- U-/UL - up to 12 cores/14 threads with an operating power range of 12-28W
- H-/HL- up to 16 cores/22 threads with an operating power range of 20-65W

The SKU stack consists of a combination of P (Performance), E (Efficiency) and LPE (Low-power Efficiency) cores. In Table 1, Processor cores listed first are the total number of cores. The number of Performance-cores, Efficient-cores, and Low-power E-cores are listed in parentheses (P+E+LPE).

	Intel® Core™ Ultra 7		Intel® Core™ Ultra 5		Intel® Core™ Ultra 3
	-H, -HL	-UL, -U	-H, -HL	-UL, -U	-UL
Cores / Threads	16C/22T (6P+8E+2LPE)	12C/14T (2P+8E+2LPE)	14C/18T (4P+8E+2LPE)	12C/14T (2P+8E+2LPE)	8C/10T (2P+4E+2LPE)
Xᵉ Cores	8	4	7-8	4	3
EUs	128	64	112-128	64	48
TDP Range	20W -65W	12W-28W	20W-65W	12W-28W	12W-28W
Max Turbo Freq[5] P/E core (GHz)	Up to 5/3.8	Up to 4.9/3.8	Up to 4.6/3.6	Up to 4.4/3.6	Up to 4.2/3.5
Graphics Max freq[4] (GHz)	Up to 2.3	Up to 2.0	Up to 2.2	Up to 1.9	Up to 1.8
OS	Windows 10[6] LTSC, Windows 11 LTSC, Linux				
Product Availability	Up to 10 years[7]				

Table 1: Intel® Core™ Ultra Processors Product Specification

[5] The frequency of cores and core types varies by workload, power consumption and other factors. Visit https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html for more information.

[6] Windows 10 LTSC not supported for NPU.

[7] Hardware product availability guidance typically extends up to 10 years. Refer to Quick Reference Guide for additional details.

Table 2 below showcases AI inference performance for specific SKUs of Intel® Core™ Ultra Processor. This is a representative list and not an exhaustive list of all the SKUs. Please refer to Intel® Core™ Ultra Processor specifications for a complete list.

System configuration:

165HL: 16C/22T, 8Xe cores, base power for performance measurement:45W, Processor Max Turbo Frequency(P-core/E-core): 5.0/3.8 GHz, Processor Base Frequency(P-core/E-core): 2.4/1.9 GHz, Graphics Max frequency: 2.3 GHz

165H: 16C/22T, 8Xe cores, base power for performance measurement:28W, Processor Max Turbo Frequency(P-core/E-core): 5.0/3.8 GHz, Processor Base Frequency(P-core/E-core): 1.4/0.9 GHz, Graphics Max frequency: 2.3 GHz

165UL & 165U: 12C/14T, 4Xe cores, base power for performance measurement below:15W, Processor Max Turbo Frequency(P-core/E-core): 4.9/3.8 GHz, Processor Base Frequency(P-core/E-core): 1.7/1.2 GHz, Graphics Max frequency: 2.0 GHz

	Intel® Core™ Ultra 7			
Processor Number	165HL	165H	165UL	165U
Total TOPS (INT 8)	32.2	31.0	19.8	19.8
GPU TOPS (INT 8)	18.8	18.8	8.2	8.2
NPU TOPS (INT 8)	10.6	10.6	10.6	10.6
CPU TOPS (INT 8)	2.82	1.54	1.05	1.05

Table 2: Intel® Core™ Ultra Processor SKU TOPS Data[8]

[8] TOPS data provided for select SKUs only. The frequency of cores and core types varies by workload, power consumption and other factors. Visit https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html for more information.

4.0 Edge AI Applications

Intel® Core™ Ultra Processors are set to enable efficient AI processing at the edge for several diverse segments, including machine vision and natural language processing. Figure 7 illustrates a few examples of AI use cases in retail, healthcare, smart cities, industrial, public sector, and education.

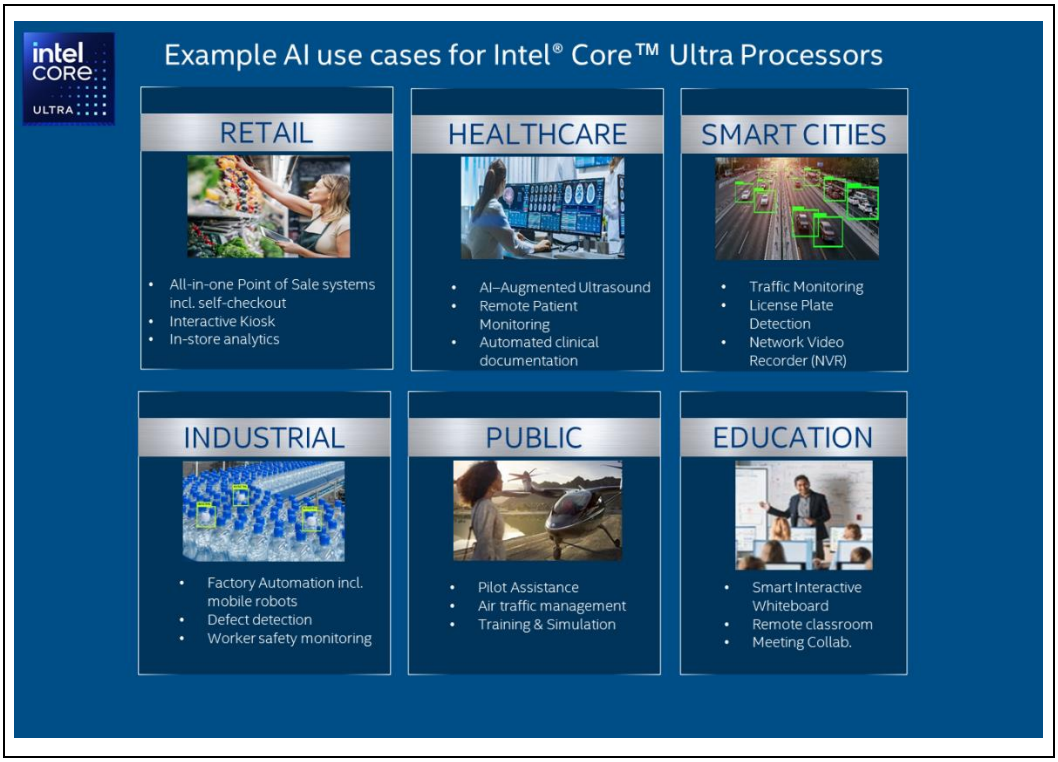

Figure 7: Example AI Use Cases for Intel® Core™ Ultra Processors

The use cases shown in Figure 7 showcase the versatility of deploying Intel Core Ultra Processors for AI workloads across various industries. Retail utilizes Intel Core Ultra processors to process high amounts of visual and customer behavior data for creating a seamless checkout experience. Healthcare uses the AI capabilities of Intel Core Ultra Processors to augment existing medical imaging technology as well as simplify patient monitoring and clinical documentation.

In Smart Cities, continuous visual data from traffic and security cameras is processed by Intel Core Ultra Processors at the edge. Industrial utilizes these enhanced processors to simplify and improve the production line in a factory. In the Public Sector, they are utilized to make air travel safer.

Education sector employs these processors to enhance conventional teaching practices and improve student engagement. The use cases and segments mentioned above are a subset of countless other applications that can leverage Intel Core Ultra Processors to enhance their existing performance and to handle unique challenges faced with edge deployments with AI.

5.0 Software Enablement

Figure 8 is a representative software stack to enable AI inference on the Intel® Core™ Ultra Processors using OpenVINO™ toolkit. OpenVINO is a versatile solution to bridge the gap between model development and efficient deployment. By providing tools to optimize trained neural networks for various hardware architectures (CPU, GPU, NPU, etc.), OpenVINO enables developers to achieve improved inference performance and reduced latency without significantly compromising accuracy. The toolkit's model optimization techniques, including quantization, distillation, pruning, and layer fusion, allow for streamlined inference execution, making it an asset in applications leveraging object detection, image classification, segmentation, real-time generation of images, and sophisticated large language model processing. The OpenVINO toolkit supports standard frameworks including TensorFlow* and PyTorch*.

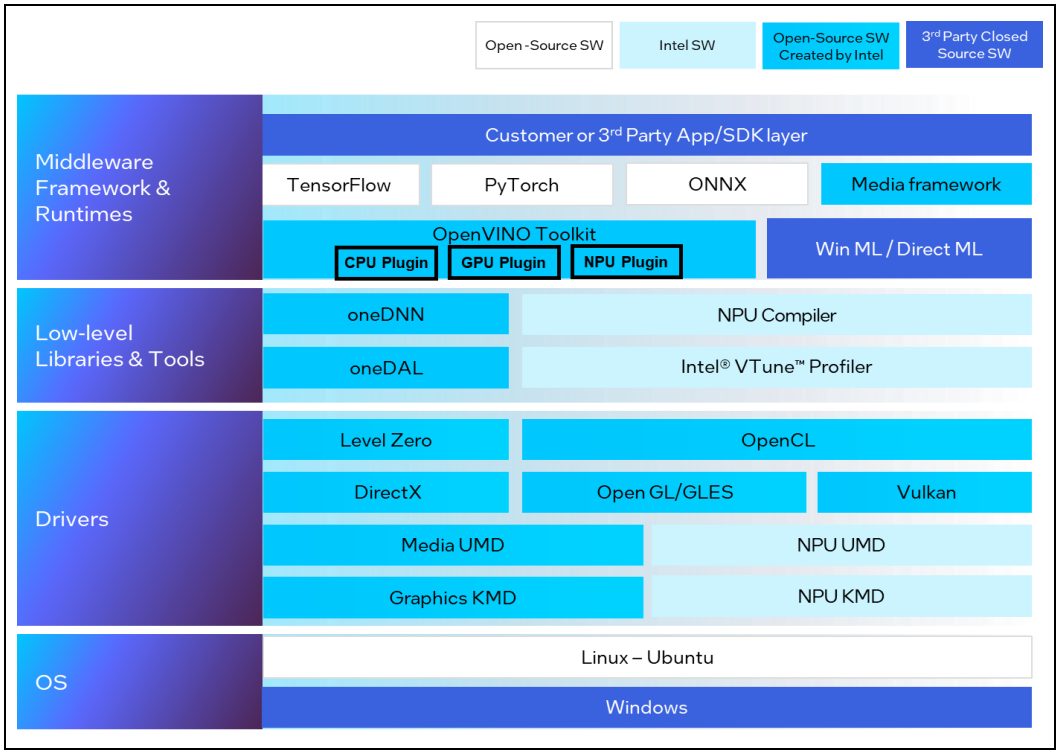

Figure 8: Representative Software Stack for Intel® Core™ Ultra Processors

5.1 Optimizing and Deploying PyTorch* Models with OpenVINO[9,10,11]

OpenVINO™ has made it easy to import PyTorch*, TensorFlow*, TensorFlow Lite* (TFLite), ONNX*, and PaddlePaddle* models and integrate them into the inference pipeline. With recent updates to OpenVINO, developers can load models and deploy them using their native format. Figure 9 and Figure 10 show an example of how to import PyTorch models into OpenVINO or use OpenVINO as a backend in PyTorch via torch.compile. Use convert_model() API to convert the model to OpenVINO intermediate representation (IR) for optimal performance, shorter model loading time, and lighter runtime package. Alternatively, use torch.compile() API to use OpenVINO in PyTorch-native applications or for quick testing after model training/fine-tuning.

```
import openvino as ov
import torch
model = torch.load("model.pt")
model.eval()
ov_model = ov.convert_model(model) # Convert model loaded from PyTorch file
core = ov.Core()
compiled_model = core.compile_model(ov_model) # Compile model from memory
```

Figure 9: PyTorch* models can be directly converted within OpenVINO™

```
import openvino.torch
import torch
# Compile PyTorch model
opts = {"device" : "CPU", "config" : {"PERFORMANCE_HINT" : "LATENCY"}}
compiled_model = torch.compile(model, backend="openvino", options=opts)
```

Figure 10: OpenVINO™ backend to torch.compile

To use torch.compile, the user needs to add an import statement and define the backend as "openvino". With this backend, Torch FX subgraphs are directly converted to OpenVINO representation without any additional PyTorch based tracing as shown in Figure 11.

[9] Importing PyTorch and TensorFlow Models Into OpenVINO | Medium

[10] PyTorch Deployment via "torch.compile" — OpenVINO™ Documentation Version (2023.3)

[11] OpenVINO Get Started Guide

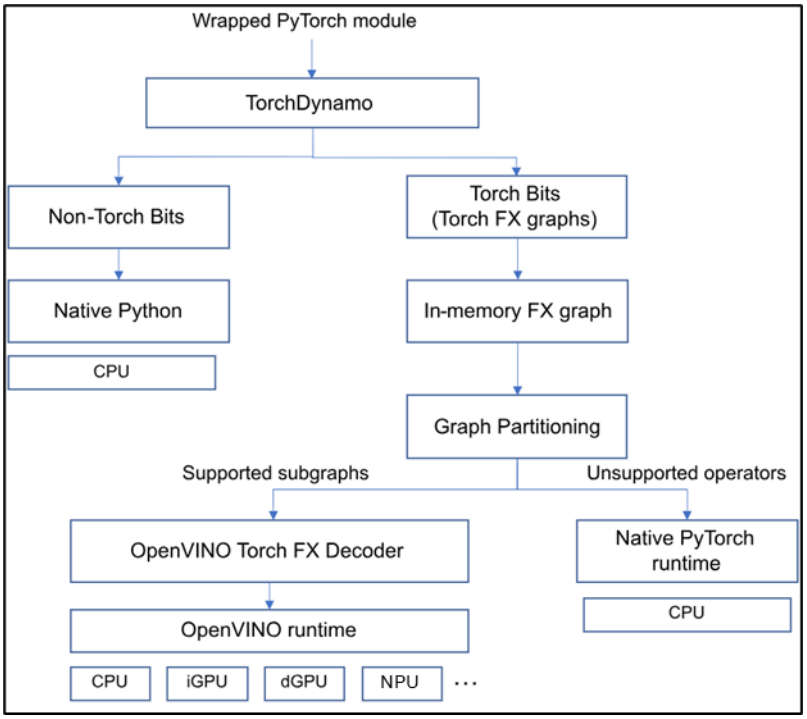

Figure 11: Flow Using PyTorch* and torch.compile with OpenVINO™

The torch.compile feature is based on TorchDynamo, which dynamically modifies Python bytecode right before it is executed and TorchInductor, which is a deep learning compiler that generates fast code for multiple accelerators and backends.

All supported operators are executed in an optimized manner using OpenVINO runtime. The unsupported operators fall back to the native PyTorch runtime on CPU.

6.0 Intel® Core™ Ultra Processor Benchmarks

Table 3 shows the inference performance of all three AI engines – CPU, GPU, and NPU, in Intel® Core™ Ultra 7 165HL processor across four models targeting image classification and object detection. The inference performance is evaluated with C++-based benchmark_app utility in Intel OpenVINO™ toolkit using throughput metric. Throughput is the amount of data an inferencing pipeline can process at once, and it is measured in frames per second (fps). In applications where large amounts of data must be inferenced simultaneously (such as multi-camera video streams), high throughput is needed.

While the performance data has been computed using Windows 11*, the Intel Core Ultra processors support Linux Ubuntu as well.

Model Category	Model Name	Precision	Batch Size	Inference Device	Throughput (fps)
Image Classification	resnet-50-tf	INT8	1	CPU	450
			1	GPU	597
			8	GPU	1046
			1	NPU	657
Image Classification	mobilenet-v2-1.0-224	INT8	1	CPU	2564
			1	GPU	1527
			8	GPU	3866
			1	NPU	1921
Object Detection	mobilenet-ssd	INT8	1	CPU	1014
			1	GPU	1386
			8	GPU	2252
			1	NPU	207
Object Detection	yolov8n	INT8	1	CPU	263
			1	GPU	462
			8	GPU	589
			1	NPU	121

Table 3: AI inference Performance of Intel® Core™ Ultra 7 165HL Processor[12,13]

[12] Performance may vary based on system configurations such as OS and GPU/NPU drivers for same target hardware.

[13] **System Configuration**: CPU: Intel Core Ultra 7 165HL (6 P-cores, 8 E-cores, 2 LPE-cores); GPU: Intel Arc Graphics Driver: 31.0.101.5333; NPU: Driver: 31.0.100.2016; Memory: 64 GB; OS: Windows 11 Enterprise; Python: 3.10.0; OpenVINO: 2023.3.0

The following step-by-step instructions enable users to reproduce the results in Table 3 and easily adopt Intel Core Ultra processor for AI inference workloads using Windows 11*.

6.1 Installation of C++ OpenVINO™ Runtime

1. Create an `Intel` folder in the `C:\Program Files (x86)\` directory. Skip this step if the folder already exists.

```
> mkdir "C:\Program Files (x86)\Intel"
```

2. Download the OpenVINO Runtime archive file for Windows to your local `Downloads` folder

3. Extract the archive file, rename the extracted folder as `openvino_2023`, and move it to the `C:\Program Files (x86)\Intel` directory

4. Configure the environment

```
> "C:\Program Files (x86)\Intel\openvino_2023\setupvars.bat"
```

Note: Activate the environment variables every time you open a new Windows command-line terminal

6.2 Building OpenVINO C++ Samples

1. Download and install CMake for the latest release of Windows* OS

2. Download Visual Studio Community 2022 and run *VisualStudioSetup.exe*. Keep the default packages, add "Desktop development with C++" in "Workloads" tab, and complete the installation

3. Open a new Windows command-line terminal and build the C++ OpenVINO samples

```
> "C:\Program Files (x86)\Intel\openvino_2023\setupvars.bat"
> cd "C:\Program Files (x86)\Intel\openvino_2023\samples\cpp\"
> .\build_samples_msvc.bat
```

Note: The compiled binary is saved in the *C:|Users|[user_name]|Documents|Intel|OpenVINO* directory.

4. Verify the execution of benchmark_app sample

```
>
C:\Users\[user_name]\Documents\Intel\OpenVINO\openvino_cpp_samples_bu
ild\intel64\Release\benchmark_app.exe --help
```

Note: The successful execution of the above command displays the available accelerators in the host.

6.3 Preparing the Models

The model optimizer in OpenVINO™ toolkit converts the raw model into intermediate representation format with floating point precision (FP32, FP16). Neural Network Compression Framework (NNCF) provides a suite of post-training and training-time algorithms for quantizing the optimized networks from FP32/FP16 to integer precision with minimal accuracy drop. Post-Training Quantization is a quantization algorithm that doesn't demand retraining of a quantized model. It utilizes a small subset of the initial dataset to calibrate quantization constants. For demonstrating the performance of INT8 models, Post-Training Quantization is used to convert the optimized models into INT8 precision. Readers are recommended to explore the training-time compression algorithms in NNCF to improve the model accuracy further. Below is the list of commands used to download, optimize, quantize, and evaluate the models using benchmark_app.

1. Create and activate a Python* virtual environment

```
> python -m venv ov_dev_2023_3

> ov_dev_2023_3\Scripts\activate
```

2. Install OpenVINO 2023.3.0 and dependencies

```
> python -m pip install pip --upgrade

> pip install openvino-dev[tensorflow2,pytorch,onnx]==2023.3.0

> pip install nncf

> pip install scipy==1.10.1
```

3. Image Classification: resnet-50-tf

```
# Activate the virtual environment ov_dev_2023_3

> omz_downloader --name resnet-50-tf

> mo --framework=tf "--output_dir=public\resnet-50-tf\FP16" --
model_name=resnet-50-tf --
input=map/TensorArrayStack/TensorArrayGatherV3 --
mean_values=[123.68,116.78,103.94] --output=softmax_tensor "--
input_model=public\resnet-50-tf/resnet_v1-50.pb" --
reverse_input_channels "--
layout=map/TensorArrayStack/TensorArrayGatherV3(NHWC->NCHW)" "--
input_shape=[1, 224, 224, 3]" --compress_to_fp16=True

> "C:\Program Files (x86)\Intel\openvino_2023\setupvars.bat"

# Evaluate the performance of CPU

>
C:\Users\[user_name]\Documents\Intel\OpenVINO\openvino_cpp_samples_bu
ild\intel64\Release\benchmark_app.exe -m <resnet-50-tf.xml> -b 1 -d
CPU -hint throughput

# Evaluate the performance of NPU

>
C:\Users\[user_name]\Documents\Intel\OpenVINO\openvino_cpp_samples_bu
ild\intel64\Release\benchmark_app.exe -m <resnet-50-tf.xml> -b 1 -d
NPU -hint throughput

# Evaluate the performance of GPU with the batch size (-b) of 8

>
C:\Users\[user_name]\Documents\Intel\OpenVINO\openvino_cpp_samples_bu
ild\intel64\Release\benchmark_app.exe -m <resnet-50-tf.xml> -b 8 -d
GPU -hint throughput
```

Note: Change the input parameter *-d* to CPU or GPU or NPU to evaluate the throughput on CPU/GPU/NPU

4. Image Classification: mobilenet-v2-1.0-224

```
# Activate the virtual environment ov_dev_2023_3

> omz_downloader --name mobilenet-v2-1.0-224

> mo --framework=tf --output_dir=public\mobilenet-v2-1.0-224\FP16 --
model_name=mobilenet-v2-1.0-224 --input=input --
reverse_input_channels "--mean_values=input[127.5,127.5,127.5]" "--
scale_values=input[127.5]" --output=MobilenetV2/Predictions/Reshape_1
--input_model=public\mobilenet-v2-1.0-
224\mobilenet_v2_1.0_224_frozen.pb "--layout=input(NHWC->NCHW)" "--
input_shape=[1, 224, 224, 3]" --compress_to_fp16=True

> "C:\Program Files (x86)\Intel\openvino_2023\setupvars.bat"

>
C:\Users\[user_name]\Documents\Intel\OpenVINO\openvino_cpp_samples_bu
ild\intel64\Release\benchmark_app.exe -m <mobilenet-v2-1.0-224.xml> -
b 1 -d NPU -hint throughput
```

It is recommended to use this script for quantizing both resnet-50-tf and mobilenet-v2-1.0-224 models by changing the model's name with a valid path to the optimized models.

5. Object Detection: mobilenet-ssd

```
> omz_downloader --name mobilenet-ssd

> omz_converter --name mobilenet-ssd --precisions FP16

# Create the datasets directory, if does not exist

> mkdir datasets

> curl -L http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtest_06-
Nov-2007.tar --output VOCtest_06-Nov-2007.tar

# Extract the VOCtest_06-Nov-2007.tar file and move the VOCdevkit
# directory to datasets

> omz_quantizer --name mobilenet-ssd --model_dir .\ --dataset_dir
.\datasets -o .\ --precisions FP16-INT8

> "C:\Program Files (x86)\Intel\openvino_2023\setupvars.bat"

>
C:\Users\[user_name]\Documents\Intel\OpenVINO\openvino_cpp_samples_bu
ild\intel64\Release\benchmark_app.exe -m <mobilenet-ssd.xml> -b 1 -d
NPU -hint throughput
```

6. Object Detection: yolov8n

Use this OpenVINO notebook with the below changes in the corresponding cells to create the quantized yolov8n models with the batch size of 1 and 8.

a. Change the input parameters to the export function in Cell [7]

```
# Export the model into OpenVINO format with the batch size of 1

det_model.export(format="openvino", batch=1, imgsz=640, half=False)

# Export the model into OpenVINO format with the batch size of 8

det_model.export(format="openvino", batch=8, imgsz=640, half=False)
```

b. Change the batch size from 1 to 8 in Cell [16]. This step is applicable only for creating the quantized model with a batch size of 8.

```
det_data_loader = det_validator.get_dataloader("datasets/coco", 8)
```

c. Replace the variable *names* in *nncf.IgnoredScope* in Cell [22] with the below

```
ignored_scope=nncf.IgnoredScope(
            types=["Multiply", "Subtract", "Sigmoid"],
            names=[
                "/model.22/dfl/conv/Conv",
                "/model.22/Add",
                "/model.22/Add_1",
                "/model.22/Add_2"
            ]
        )
```

d. Evaluate the inference performance using benchmark_app

```
> "C:\Program Files (x86)\Intel\openvino_2023\setupvars.bat"

# Evaluate the yolov8n model with the batch size of 1 on GPU

>
C:\Users\[user_name]\Documents\Intel\OpenVINO\openvino_cpp_samples_bu
ild\intel64\Release\benchmark_app.exe -m <yolov8n.xml> -b 1 -d GPU -
hint throughput

# Evaluate the yolov8n model with the batch size of 8 on GPU

>
C:\Users\[user_name]\Documents\Intel\OpenVINO\openvino_cpp_samples_bu
ild\intel64\Release\benchmark_app.exe -m <yolov8n.xml> -d GPU -hint
throughput
```

7.0 Case Study: Product Classification and Person Detection on CPU, GPU and NPU

7.1 Overview

The case study shows the end-to-end performance of the CPU, integrated GPU and the built-in NPU for the product classification and person detection use cases. Additionally, all code snippets and software dependencies are highlighted with step-by-step instructions to enable anyone looking to get started with running inference workloads on the Intel® Core™ Ultra processors.

The end-to-end pipeline involves video decoding, image pre-processing (such as color conversion or image resizing), inference execution and post-processing (optional, such as writing the inference results to the video frame or database). Video decode and media analysis can be done on dedicated hardware followed by AI inferencing on dedicated accelerators. A typical video analytics pipeline requires several functions including AI to come together seamlessly. Intel is well-positioned to provide the right IP for the right function to enable highly performant end-to-end solutions.

Figure 12 is a high-level representation of the end-to-end workflow for the product classification and person detection use cases. In this case study, we show two ways to run inference on a target accelerator – (A) Using Intel® DL Streamer pipeline framework for video decoding and pre-processing on the integrated GPU while inference can be targeted for CPU or integrated GPU, (B) Using the Open Model Zoo where video decoding and pre-processing are executed on the CPU and inference is targeted for the NPU. Note that running video decode and other media analysis on the integrated GPU takes advantage of dedicated built-in hardware, freeing up the CPU for other tasks. However, the user could employ either method[14].

[14] DL Streamer support for NPU is available on v2024.0 - Documentation, Source Code. Note that measurements in this white paper use the previous version of DL Streamer as the latest version (v2024.0) was released after the writing of this white paper.

7.2 End-to-End Workflow

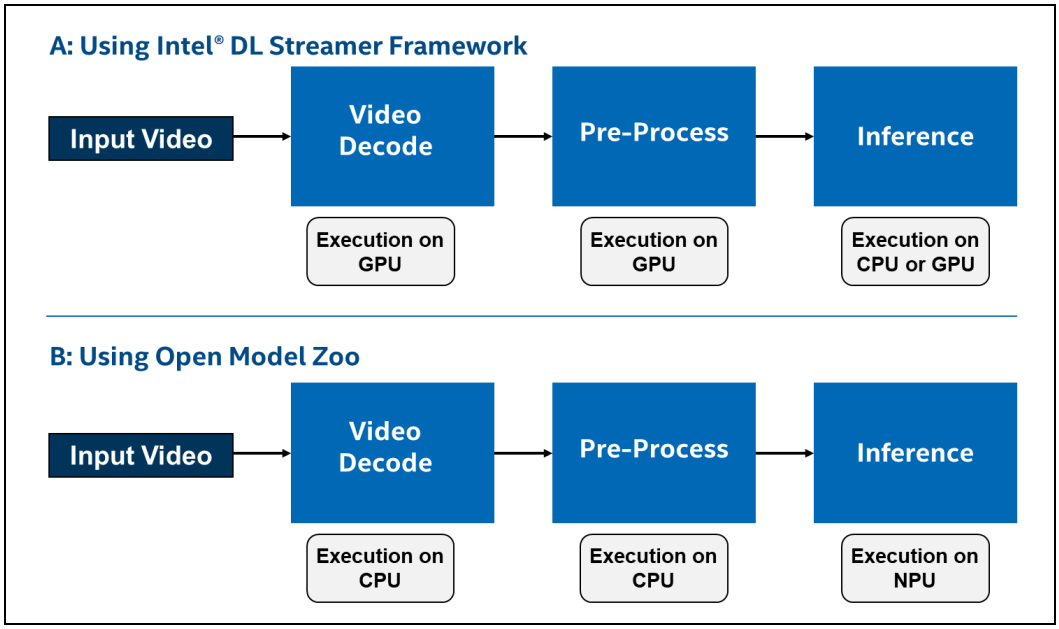

Figure 12: End-to-End Pipeline for Classification and Detection Workloads[15]

Following are the steps involved in running and evaluating the end-to-end pipeline for classification and detection workloads.

7.2.1 Installation of OpenCL™ GPU Driver

1. Install the required dependencies

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install libtbb12 libtbbmalloc2
```

2. Verify the Linux kernel version (>=6.5)

[15] **System Configuration**: CPU: Intel Core Ultra 7 processor 165HL (6 P-cores, 8 E-cores, 2 LPE-cores); GPU: Intel Arc Graphics Driver: 23.35.27191.42; NPU Driver: 1.1.0; Memory: 16 GB; OS: Ubuntu 22.04.4 LTS; Python: 3.11.8; (A) DL Streamer: 2023.0, OpenVINO: 2023.0; (B) Open Model Zoo: 2023.3, OpenVINO 2023.3

```
$ uname –a
```

3. Add the following line to /etc/default/grub

```
GRUB_CMDLINE_LINUX="i915.force_probe=7d55"
```

4. Update the grub and reboot the machine to activate the GPU driver

```
$ sudo update-grub

$ sudo reboot now
```

5. Install OpenCL Runtime

```
$ wget -qO - https://repositories.intel.com/gpu/intel-graphics.key | sudo gpg --dearmor --output /usr/share/keyrings/intel-graphics.gpg

$ echo "deb [arch=amd64,i386 signed-by=/usr/share/keyrings/intel-graphics.gpg] https://repositories.intel.com/gpu/ubuntu jammy client" | sudo tee /etc/apt/sources.list.d/intel-gpu-jammy.list

$ sudo apt update

$ 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
```

6. Verify the OpenCL GPU driver using clinfo

```
$ clinfo
```

7.2.2 Installation of NPU Driver

1. Download the NPU driver and dependencies

```
$ wget https://github.com/intel/linux-npu-
driver/releases/download/v1.1.0/intel-driver-compiler-
npu_1.1.0.20231117-6904283384_ubuntu22.04_amd64.deb

$ wget https://github.com/intel/linux-npu-
driver/releases/download/v1.1.0/intel-fw-npu_1.1.0.20231117-
6904283384_ubuntu22.04_amd64.deb

$ wget https://github.com/intel/linux-npu-
driver/releases/download/v1.1.0/intel-level-zero-npu_1.1.0.20231117-
6904283384_ubuntu22.04_amd64.deb

$ wget https://github.com/oneapi-src/level-
zero/releases/download/v1.10.0/level-zero_1.10.0+u22.04_amd64.deb
```

2. Add the following groups to the target user

```
$ sudo usermod -aG video $USER

$ sudo usermod -aG render $USER
```

3. Install the NPU driver

```
$ sudo dpkg -i *.deb
```

4. Run the following command to prevent upgrading the level-zero package to support the NPU driver

```
$ sudo apt-mark hold level-zero
```

5. Verify the NPU driver after rebooting the machine

```
$ sudo lsmod | grep vpu
```

7.2.3 Product Classification and Person Detection with Intel® DL Streamer

Intel® DL Streamer Pipeline Framework is an open-source streaming media analytics framework, based on GStreamer* multimedia framework, for creating complex media analytics pipelines. It is optimized for performance and functional interoperability between GStreamer* plugins built on various backend libraries as below:

- Inference plugins use OpenVINO™ inference engine optimized for Intel CPU, GPU and NPU platforms
- Video decode and encode plugins utilize GPU-acceleration based on VA-API
- Image processing plugins based on OpenCV and DPC++
- Other GStreamer* plugins built on various open-source libraries for media input and output, muxing and demuxing, decode and encode

gst-launch-1.0 is the command-line tool used to launch a media analytics pipeline as a series of connected elements, performing video decode, inference and overlay of detected objects. This section uses the following elements to implement the product classification and person detection use cases.

filesrc: Reads input from a local file.

decodebin: Produces raw video frames suitable for image transformation and inferencing.

capsfilter: Uses video/x-raw(memory:VASurface) for enabling hardware-accelerated video decoding on GPU using VA-API.

gvadetect: Performs object detection using SSD-like (including MobileNet-V1/V2 and ResNet), YoloV2/YoloV3/YoloV2-tiny/YoloV3-tiny and FasterRCNN-like object detection models.

gvaclassify: Performs object classification. Accepts the ROI or full frame as an input and outputs classification results with metadata.

gvawatermark: Overlays the metadata on the video frame to visualize the inference results.

videoconvert: Converts video frames between a great variety of video formats.

fpsdisplaysink: Displays the current and average framerate as a test overlay or on stdout.

gvafpscounter: Measures the throughput in frames per second across multiple streams in a single process.

1. Create the DL Streamer container using the docker image intel/dlstreamer:devel and access the running container

```
$ docker run -it --rm -u root -d --name benchmark -e DISPLAY=$DISPLAY
--device /dev/dri:/dev/dri --group-add="$(stat -c "%g"
/dev/dri/render*)" -v /dev/bus/usb:/dev/bus/usb -v
~/.Xauthority:/home/dlstreamer/.Xauthority -v /tmp/.X11-
unix/:/tmp/.X11-unix/ -v $PWD:/home/dlstreamer intel/dlstreamer:devel

$ docker exec -it benchmark /bin/bash
```

2. Download the input videos and the model-proc file for resnet-50-tf model

3. Run the product classification with resnet-50-tf (INT8) on CPU

```
$ gst-launch-1.0 filesrc location=<fruit-and-vegetable-detection.mp4>
! decodebin ! capsfilter caps="video/x-raw(memory:VASurface)" !
gvaclassify model-proc=<resnet-50-tf.json> model=<resnet-50-tf.xml>
device=CPU nireq=4 batch-size=8 inference-region=full-frame model-
instance-id=resnet50 ! queue ! gvawatermark ! videoconvert !
fpsdisplaysink video-sink=xvimagesink sync=True
```

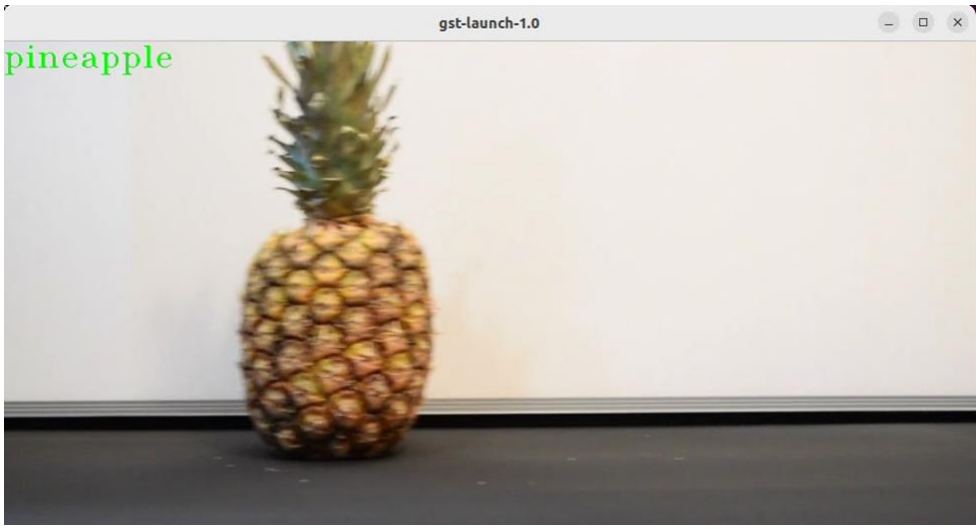

Figure 13: Sample Classification Result on CPU

4. Run image classification with resnet-50-tf (INT8) on GPU

```
$ gst-launch-1.0 filesrc location=<fruit-and-vegetable-detection.mp4>
! decodebin ! capsfilter caps="video/x-raw(memory:VASurface)" !
gvaclassify model-proc=INT8_Models/dlstreamer/resnet-50-
tf_INT8/resnet-50-tf_i8.json model=INT8_Models/dlstreamer/resnet-50-
tf_INT8/resnet-50-tf_i8.xml device=GPU nireq=4 pre-process-
backend=vaapi-surface-sharing batch-size=8 inference-region=full-
frame model-instance-id=resnet50 ! queue ! gvawatermark !
videoconvert ! fpsdisplaysink video-sink=xvimagesink sync=True
```

5. Evaluate the image classification pipeline on CPU

```
$ gst-launch-1.0 filesrc location=<fruit-and-vegetable-detection.mp4>
! decodebin ! capsfilter caps="video/x-raw(memory:VASurface)" !
gvaclassify model-proc=<resnet-50-tf_i8.json> model=<resnet-50-
tf.xml> device=CPU nireq=4 batch-size=8 inference-region=full-frame
model-instance-id=resnet50 ! queue ! gvafpscounter ! fakesink
async=False
```

6. Evaluate the image classification pipeline on GPU

```
$ gst-launch-1.0 filesrc location=<fruit-and-vegetable-detection.mp4>
! decodebin ! capsfilter caps="video/x-raw(memory:VASurface)" !
gvaclassify model-proc=<resnet-50-tf.json> model=<resnet-50-tf.xml>
device=GPU nireq=4 pre-process-backend=vaapi-surface-sharing batch-
size=8 inference-region=full-frame model-instance-id=resnet50 ! queue
! gvafpscounter ! fakesink async=False
```

7. Run the object detection with mobilenet-ssd (INT8) on CPU

```
$ gst-launch-1.0 filesrc location=<face-demographics-walking-and-
pause.mp4> ! decodebin ! capsfilter caps="video/x-
raw(memory:VASurface)" ! gvadetect model=<mobilenet-ssd.xml>
device=CPU nireq=4 batch-size=8 threshold=0.5 model-instance-
id=mobilenet-ssd ! queue ! gvawatermark ! videoconvert !
fpsdisplaysink video-sink=xvimagesink sync=True
```

8. Run the object detection with mobilenet-ssd (INT8) on GPU

```
$ gst-launch-1.0 filesrc location=<face-demographics-walking-and-
pause.mp4> ! decodebin ! capsfilter caps="video/x-
raw(memory:VASurface)" ! gvadetect model=<mobilenet-ssd.xml>
device=GPU nireq=4 pre-process-backend=vaapi-surface-sharing batch-
size=8 threshold=0.5 model-instance-id=mobilenet-ssd ! queue !
gvawatermark ! videoconvert ! fpsdisplaysink video-sink=xvimagesink
sync=True
```

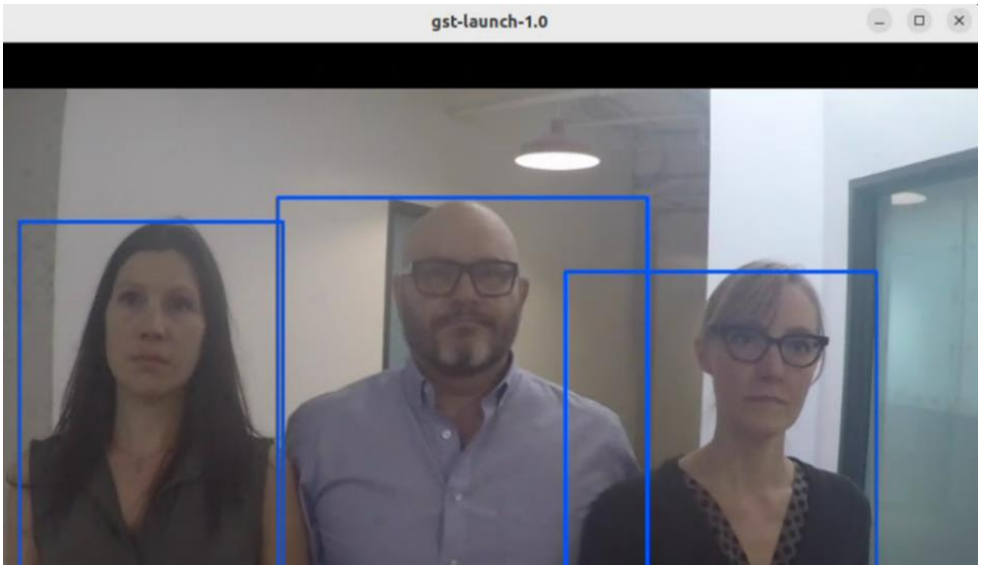

Figure 14: Sample Person Detection Results on GPU

9. Evaluate the object detection pipeline on CPU

```
$ gst-launch-1.0 filesrc location=<face-demographics-walking-and-
pause.mp4> ! decodebin ! capsfilter caps="video/x-
raw(memory:VASurface)" ! gvadetect model=<mobilenet-ssd.xml>
device=CPU nireq=4 batch-size=8 threshold=0.5 model-instance-
id=mobilenet-ssd ! queue ! gvafpscounter ! fakesink async=False
```

10. Evaluate the object detection pipeline on GPU

```
$ gst-launch-1.0 filesrc location=<face-demographics-walking-and-
pause.mp4> ! decodebin ! capsfilter caps="video/x-
raw(memory:VASurface)" ! gvadetect model=<mobilenet-ssd.xml>
device=GPU nireq=4 pre-process-backend=vaapi-surface-sharing batch-
size=8 threshold=0.5 model-instance-id=mobilenet-ssd ! queue !
gvafpscounter ! fakesink async=False
```

Note: Ensure that you provide a valid path to the input video, model-proc (.json) and model (.xml) files.

7.2.4 Product Classification and Person Detection with Open Model Zoo

Intel's Open Model Zoo repository includes optimized deep learning models and a set of demos to expedite development of high-performance deep learning inference applications. Open Model Zoo demos are console applications that provide templates to help implement specific deep learning inference scenarios. These applications show how to pre-process and post-process data for model inference and organize processing pipelines.

7.2.4.1 Installation

1. Set the OpenVINO environment variables if you are executing the below commands in a new Linux terminal

```
$ source /opt/intel/openvino_2023/setupvars.sh
```

2. Download the Open Model Zoo repository and extract the folder

3. Install the required dependencies

```
$ cd open_model_zoo/demos

$ pip install -r requirements.txt
```

7.2.4.2 Image Classification – Step by Step

1. Set the OpenVINO environment variables if you are executing the below commands in a new Linux terminal

```
$ source /opt/intel/openvino_2023/setupvars.sh
```

2. Download the input video (resolution: 960x540, frame rate: 60fps)

3. Run the image classification demo

```
$ cd open_model_zoo\demos

$ python3 classification_demo/python/classification_demo.py -i
<fruit-and-vegetable-detection.mp4> -m <resnet-50-tf.xml> -d NPU --
labels ../data/dataset_classes/imagenet_2012.txt -nstreams 1 -nireq 4
```

Note: Ensure the path to the input video file and model files is valid in the above command.

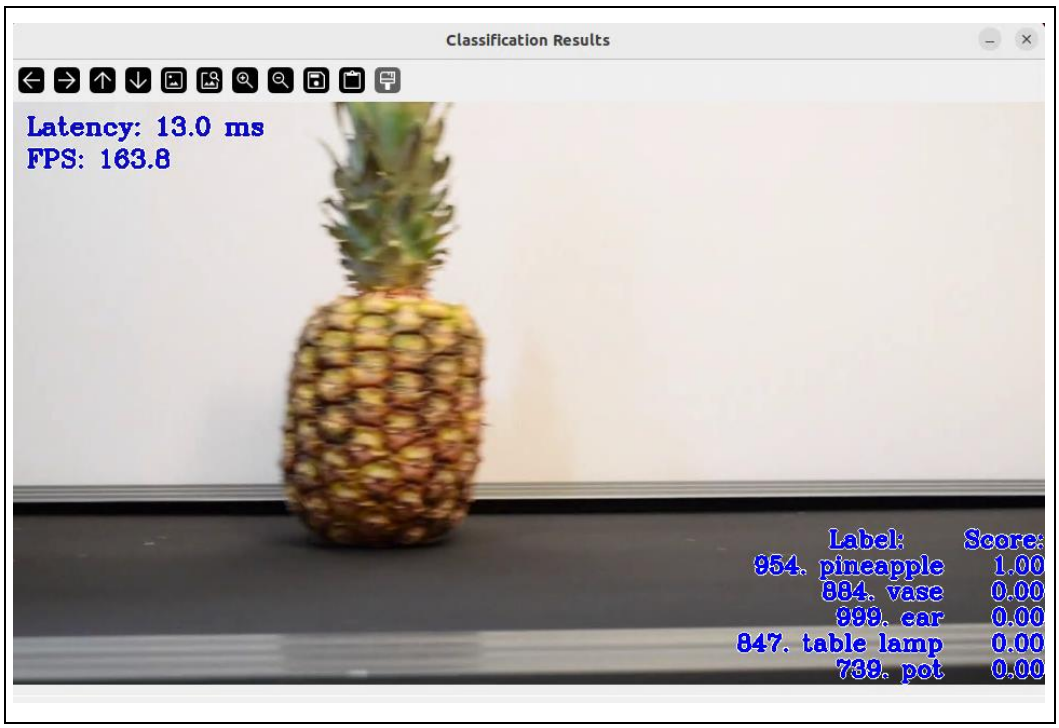

Figure 15: Sample Product Classification Output on NPU

7.2.4.3 Object Detection – Step by Step

1. Download the input video (resolution: 768x432, frame rate: 12fps)

2. Run the face detection demo

```
> cd open_model_zoo\demos

> python3 object_detection_demo/python/object_detection_demo.py -m
<mobilenet-ssd.xml> -i <face-demographics-walking-and-pause.mp4> -at
ssd -d NPU -nstreams 1 -nireq 4
```

Note: Ensure the path to the input video file and model files is valid in the above command.

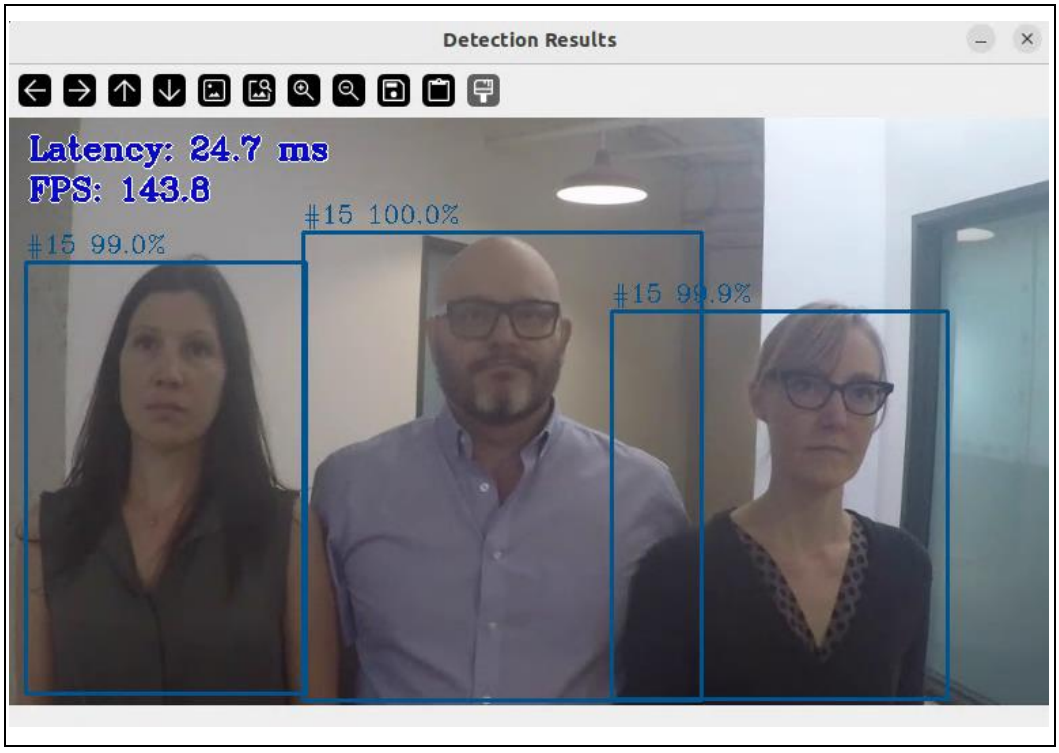

Figure 16: Sample Person Detection Output on NPU

7.3 Performance

Table 4 captures the end-to-end performance of product classification and person detection use cases on CPU, integrated GPU and NPU using resnet-50-tf and mobilenet-ssd models. The throughput of CPU and integrated GPU is computed using Intel's DL Streamer framework and the throughput on NPU is computed using Open Model Zoo. Note that the difference in inference results between Table 3 and Table 4 is that Table 3 shows the inference performance of AI models in isolation, whereas Table 4 shows the end-to-end performance of AI models including video decoding, pre-processing, and inferencing.

This case study aims to enable a user to get started with inferencing using Intel Core Ultra processors quickly. Hence, the measurements are limited to the overall throughput of each accelerator. Users can choose the optimal accelerator for their unique workloads.

Use Case	Precision	Batch Size	Inferencing Device	End-to-End Throughput (fps)
Product classification on an end-to-end pipeline with resnet-50-tf (Input video resolution: 960x540, frame rate: 60fps)	INT8	8	CPU	217
	INT8	8	GPU	752
	INT8	1	NPU	477
Person detection on an end-to-end pipeline with mobilenet-ssd (Input video resolution: 768x432, frame rate: 12fps)	INT8	8	CPU	640
	INT8	8	GPU	1063
	INT8	1	NPU	180

Table 4: End-to-End Performance of Product Classification and Person Detection Use Cases[16, 17, 18]

[16] Throughput of CPU and integrated GPU is computed using DL Streamer while the throughput of NPU is computed using Open Model Zoo demos.

[17] **System Configuration**: CPU: Intel Core Ultra 7 processor 165HL (6 P-cores, 8 E-cores, 2 LPE-cores); GPU: Intel Arc Graphics Driver: 23.35.27191.42; NPU Driver: 1.1.0; Memory: 16 GB; OS: Ubuntu 22.04.4 LTS; Python: 3.11.8; (A) DL Streamer: 2023.0, OpenVINO: 2023.0; (B) Open Model Zoo: 2023.3, OpenVINO 2023.3

[18] Performance may vary based on system configurations such as OS and GPU/NPU drivers for same target hardware.

8.0 Conclusion

The Intel® Core™ Ultra Processors rise above conventional boundaries, offering remarkable versatility for diverse edge needs. The heterogeneous architecture with the integration of CPU, GPU and NPU coupled with the OpenVINO™ software ecosystem, unleashes intelligent computing at the edge for a myriad of AI use cases. One such use case is object detection and classification which has been explored in detail with performance comparisons across all three AI engines. This case study demonstrates that Intel® Core™ Ultra processors along with OpenVINO™ empowers developers to leverage hardware capabilities effectively and with ease. The versatility extends beyond this use case, offering immense potential for various applications across diverse sections such as healthcare, retail, smart cities, industrial, public sector, and education.

While this paper discusses a handful of benchmarks, additional performance benchmarks are published on the OpenVINO site - OpenVINO Performance Benchmarks.

This white paper provides a clear foundation for the AI hardware capabilities of the Intel Core Ultra Processors as well as software support to enable use cases at the edge. As the developer community embraces this platform, expect a surge of intelligent solutions at the edge to be unleashed.

9.0 References

[9] Importing PyTorch and TensorFlow Models Into OpenVINO | Medium

[10] PyTorch Deployment via “torch.compile” — OpenVINO™ Documentation Version (2023.3)

[11] OpenVINO Get Started Guide

10.0 Additional Information

- Intel Core™ Ultra Processors
- Optimizing Large Language Models with the OpenVINO™ Toolkit
- Intel® Core™ Ultra Processors for the Edge 30-3-30
- Intel® Core™ Ultra Processor Architecture Overview
- Inference Performance Benchmarks with OpenVINO™
- Install OpenVINO Runtime on Windows from an Archive File
- Unlocking the AI Power of Intel® Arc™ GPU for the Edge: A Deep Dive into Hardware and Software Enablement

11.0 Appendix A: Intel® Core™ Ultra Processor Benchmarks on OpenVINO™

For the latest AI performance benchmarks, visit the OpenVINO™ Benchmarks site.

Results may vary. For workloads and configurations visit:

Workload: https://docs.openvino.ai/latest/openvino_docs_performance_benchmarks_faq.html#what-image-sizes-are-used-for-the-classification-network-models

System Configuration: https://docs.openvino.ai/2024/_static/benchmarks_files/OV-2024.0-platform_list.pdf

Legal: https://docs.openvinotoolkit.org/latest/openvino_docs_Legal_Information.html

Test Date: February 26, 2024

HW Platforms: Intel® Core™Ultra7-165H CPU		
Model name	Throughput (fps)	
	INT8	FP32
bert-base-cased	45.55	17.91
bert-large-uncased-whole-word-masking-squad-0001	4.47	1.51
efficientdet-d0	65.18	38.36
mask_rcnn_resnet50_atrous_coco	0.73	0.23
mobilenet-v2	1265.90	547.69
resnet-50-tf	260.29	62.00
ssd-resnet34-1200	4.07	1.16
ssd_mobilenet_v1_coco	498.41	161.67
unet-camvid-onnx-0001	1.92	1.86
yolo-v3-tiny-tf	276.05	81.29
yolo_v8n	115.88	43.03

HW Platforms: Intel® Core™Ultra7-165H iGPU		
Model name:	Throughput (fps)	
	INT8	FP16
bert-base-cased	216.20	160.16
bert-large-uncased-whole-word-masking-squad-0001	19.62	12.06
efficientdet-d0	165.10	131.30
mask_rcnn_resnet50_atrous_coco	1.88	1.32
mobilenet-v2	3141.27	1604.12
resnet-50-tf	869.54	415.99
ssd-resnet34-1200	17.19	10.32
unet-camvid-onnx-0001	18.33	18.24
yolo_v8n	345.83	252.29

HW Platforms: Intel® Core™ Ultra7-165H iGPU			
Model name:	Throughput (tokens/sec)		Latency (ms)
	INT4	INT8	FP16
Llama-2-7b-chat	8.55	5.45	3.27
Mistral-7b	6.01	4.31	2.54

HW Platforms: Intel® Core™ Ultra7-165H iGPU		
Model name:	Image Generation Latency (sec)	
	INT8	FP16
Stable-Diffusion-v2-1	21	21.1

HW Platforms: Intel® Core™Ultra7-165H NPU		
Model name:	Throughput (fps)	
	INT8	FP16
mobilenet-v2	1834.45	1140.24
resnet-50-tf	724.94	343.65
yolo-v3-tiny-tf	375.11	293.96
yolo_v8n	118.83	103.38

HW Platforms: Intel® Core™Ultra7-165H (CPU and integrated GPU combined)		
Model name:	Throughput (fps)	
	INT8	FP32
bert-base-cased	110.47	74.06
bert-large-uncased-whole-word-masking-squad-0001	10.69	5.59
efficientdet-d0	113.74	87.60
mask_rcnn_resnet50_atrous_coco	1.26	0.60
mobilenet-v2	2441.40	1349.39
resnet-50-tf	519.62	218.44
unet-camvid-onnx-0001	7.79	7.68
yolo-v3-tiny-tf	558.60	234.83