Multi-Workloads End-to-End QnA and Object Detection Implementation on Intel® Core™ Ultra Processors

Reference Implementation

July 2024

Document Number: 827068-1.0

Contents

Figures

Tables

Revision History

Date	Revision	Description
July 2024	1.0	Initial release.

§

1.0 Introduction

With the increase of processing capability year over year as well as additional compute accelerators like the Neural Processing Unit (NPU), a single system is now capable of running multiple end-to-end workloads like QnA (Question and Answer), video surveillance analytics, and retail applications. Intel® Core™ Ultra Processors have up to 14 CPU cores with 20 threads as well as additional 2 low-power efficient cores that can be allocated to each of the workloads using containerization. The CPU is suitable for low latency workloads, the integrated GPU (iGPU) is for high throughput tasks, and the NPU is targeted for low power sustain workloads.

Other than for agile resource allocation, the containerization of the workloads is useful to improve security by protecting the business-critical workloads. Customers expect the workloads to have adequate resources to run smoothly while one workload cannot access the information of the other workloads.

This paper introduces the concept of workload consolidation using containerized solutions and the reference implementation for retail use cases that include QnA and video surveillance analytics applications. The QnA workload includes Automatic Speech Recognition (ASR), Large Language Model (LLM), and Text-to-speech (TTS) modules. Whereas video surveillance analytics utilizes object detection as one of the components.

1.1 Terminology

Table 1. Terminology

Term	Description
ASR	Automatic Speech Recognition
BKM	Best Known Method
Cmd	Command
CPU	Central Processing Unit
DP	DisplayPort
E-core	Efficient-core
fps	Frames per Second
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface

Term	Description
iGPU	Integrated Graphics Processing Unit
INT4	Integer 4-bit
IP	Internet Protocol
LLM	Large Language Model
LPE-core	Low Power Efficient-core
NPU	Neural Processing Unit
OpenVINO™	Open Visual Inference and Neural network Optimization
OS	Operating System
P-core	Performance-core
PoE	Power over Ethernet
POS	Points of Sale
QnA	Question and Answer
TTS	Text to Speech
USB	Universal Serial Bus

§

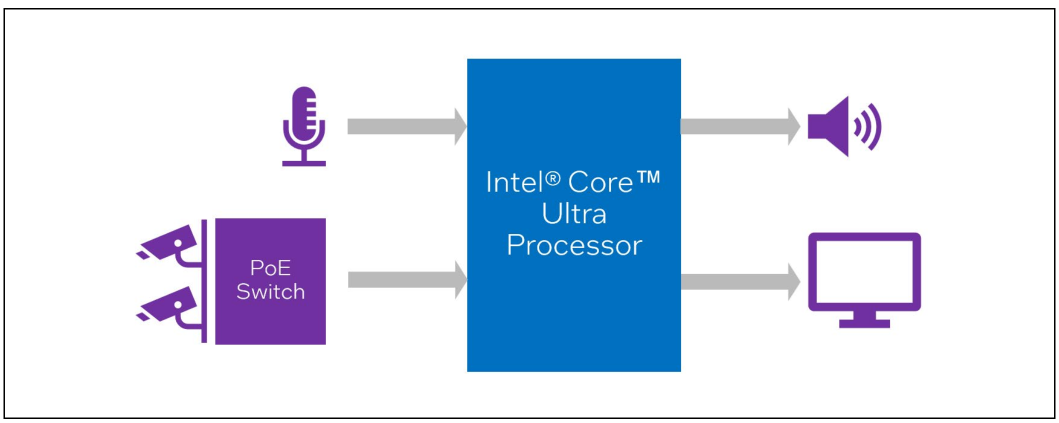
2.0 *Architecture*

This section describes both the hardware and software architecture of both QnA and video surveillance analytics applications running on a single Intel® Core™ Ultra Processor. This includes the hardware components diagram as well as the software deployment and sequence diagrams.

2.1 Hardware Architecture

Figure 1 shows the hardware architecture of both QnA and video surveillance analysis implementation. The QnA application allows the users to ask questions through a microphone and receive the audible answer from the speaker. Whereas for the video surveillance, it takes input from one or multiple cameras and output the detected objects on the monitor for further analysis. A system with Intel® Core™ Ultra Processor can provide all of the required interfaces such as 3.5mm line-out + microphone, Ethernet, USB, HDMI, and DP connectors. The cameras can be directly connected using USB or through PoE switch using Ethernet connector.

Figure 1. Hardware Components Diagram



2.2 Software Architecture

For the deployment using containerization as shown in Figure 2, the workloads can be run on top of Docker* using Ubuntu* as the Operating System (OS). In the retail scenario where Points of Sale (POS) is an essential component, the POS can be run directly on top of the host OS. Using such approach, we can limit the resources for the workloads that run on the containers in favor of the critical applications like POS.

The first containerized application is QnA encapsulated inside a OpenVINO™ Toolkit container. This application has LLM component that runs best on GPU. In this case, the Docker* run command can pass the iGPU access inside the container. The other components like ASR and TTS will run on CPU. Other than the iGPU, Docker* run command also needs to pass the USB or Sound device to the application inside the container to access the microphone and speaker.

The second containerized application is video surveillance analysis that is also encapsulated inside a OpenVINO™ Toolkit container. This application performs sustain real-time object detection that is best to run on NPU to get the low-power benefit of the compute unit. Other than NPU, the Docker* run command also needs to pass the USB or Ethernet device to the containerized application to access the cameras.

If both of the containerized workloads are web-based applications, the Docker* run command can expose the port numbers. Therefore, the web browser running directly on the host OS can manage the applications, and in the case of video surveillance application, the object detection result can be displayed on to the monitor.

Figure 2. Software Deployment Diagram

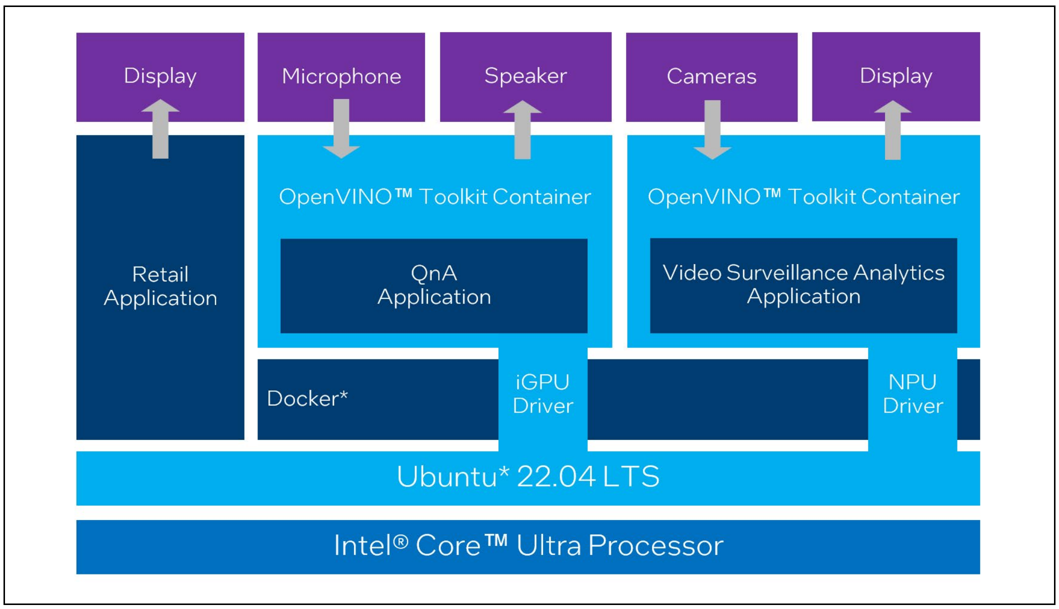
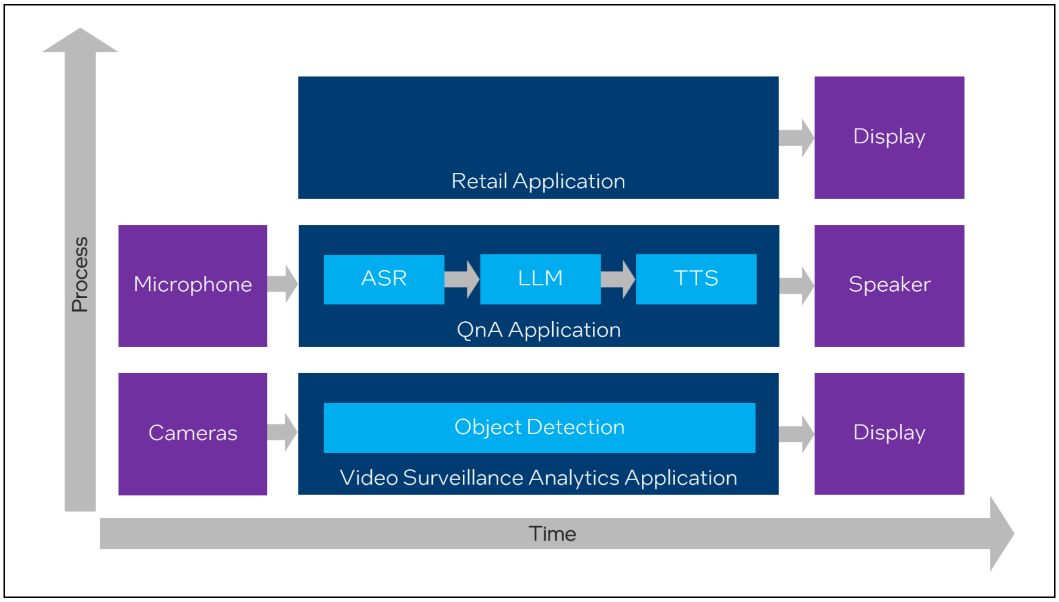


Figure 3 This shows the three-application sequence diagram. Retail, QnA, and video surveillance applications are running in parallel, whereas the ASR, LLM, and TTS components in the QnA application are running sequentially. Therefore, the three components are not utilizing the compute resources at the same time.

As for the memory consumption, it is recommended to load and warm up all of the components to speed up the execution/inference time. This becomes critically important as the same OpenVINO™ model can run on different compute units like CPU, GPU, and NPU. Therefore, it requires initial model compilation to the target device, which consumes time and compute resources.

To save memory for object detection with multiple camera implementation, it is recommended to use batching inference instead of creating multiple models. For four cameras, for example, a model can comply using [4,3,640,640] input shape instead of creating four models with [1,3,640,640] input shape, where 4 is the batch size, 3 is the number of channels, and 640x640 is the image resolution.

Figure 3. Software Sequence Diagram



§

3.0 Implementation

This section provides the steps to install operating system and required drivers as well as to build and run Docker* containers.

3.1 Install Operating System and Docker*

Download Ubuntu* Desktop 22.04 LTS and perform normal installation (not minimal installation). Then, update the OS with the latest kernel version 6.5 and install Docker* using Cmd 1.

Cmd 1. Command to Install Docker*

```
$ sudo snap install docker
```

3.2 Install GPU and NPU Drivers

Follow these links to install GPU and NPU version 1.5.0 drivers.

Run Cmd 2 to activate the GPU. Use this link to check your GPU device identification. The example uses 7d55. Update accordingly then reboot the OS.

Cmd 2. Command to Activate GPU

```
$ sudo vim /etc/default/grub
GRUB_CMDLINE_LINUX="i915.force_probe=7d55"

$ sudo update-grub
```

3.3 Build xPU Docker* Image

xPU Docker* image consists of packages to access different processing units (iGPU and NPU). Cmd 3 shows the Dockerfile and Cmd 4 shows the steps to build the image. Use Cmd 5 to detect different compute units in the system.

Cmd 3. xPU Dockerfile

```
$ vim Dockerfile_xpu
FROM openvino/ubuntu22_dev:2023.3.0

# Install Basics
USER root
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update -y \
 && apt-get upgrade -y
```

```
RUN apt-get update -y \
 && apt-get install -y apt-utils git curl ca-certificates bzip2
unzip wget cmake pkg-config tree htop iotop g++ gcc libc6-dev
make sudo \
 && apt-get install -y libglib2.0-0 libsm6 libxext6 libxrender-
dev python3-dev python3-opencv ffmpeg \
 && apt-get install -y libtbb12 libtbbmalloc2 \
 && apt-get install -y gpg

# Install GPU Packages
RUN wget -qO - https://repositories.intel.com/gpu/intel-
graphics.key | \
 gpg --dearmor --output /usr/share/keyrings/intel-graphics.gpg
RUN echo "deb [arch=amd64,i386 signed-
by=/usr/share/keyrings/intel-graphics.gpg]
https://repositories.intel.com/gpu/ubuntu jammy client" | \
 tee /etc/apt/sources.list.d/intel-gpu-jammy.list
RUN apt-get update -y \
 && apt-get install -y \
  intel-opencl-icd intel-level-zero-gpu level-zero \
  intel-media-va-driver-non-free libmfx1 libmfxgen1 libvpl2 \
  libegl-mesa0 libegl1-mesa libegl1-mesa-dev libgbm1 libgl1-mesa-
dev libgl1-mesa-dri \
  libglapi-mesa libgles2-mesa-dev libglx-mesa0 libigdgmm12
libxatracker2 mesa-va-drivers \
  mesa-vdpau-drivers mesa-vulkan-drivers va-driver-all vainfo
hwinfo clinfo

# Install NPU Packages
RUN wget https://github.com/intel/linux-npu-
driver/releases/download/v1.5.0/intel-driver-compiler-
npu_1.5.0.20240619-9582784383_ubuntu22.04_amd64.deb
RUN dpkg -i intel-driver-compiler-npu_1.5.0.20240619-
9582784383_ubuntu22.04_amd64.deb
RUN wget https://github.com/intel/linux-npu-
driver/releases/download/v1.5.0/intel-fw-npu_1.5.0.20240619-
9582784383_ubuntu22.04_amd64.deb
RUN dpkg -i intel-fw-npu_1.5.0.20240619-
9582784383_ubuntu22.04_amd64.deb
RUN wget https://github.com/intel/linux-npu-
driver/releases/download/v1.5.0/intel-level-zero-
npu_1.5.0.20240619-9582784383_ubuntu22.04_amd64.deb
RUN dpkg -i intel-level-zero-npu_1.5.0.20240619-
9582784383_ubuntu22.04_amd64.deb

RUN apt-get remove -y level-zero
RUN wget https://github.com/oneapi-src/level-
zero/releases/download/v1.17.2/level-zero_1.17.2+u22.04_amd64.deb
RUN dpkg -i level-zero_1.17.2+u22.04_amd64.deb

# Clean Workspace
RUN rm *.deb
```

```
# Finalizing
USER openvino
WORKDIR /opt/intel/openvino
```

Cmd 4. Command to Build xPU Docker* Image

```
$ sudo docker build \
  -f Dockerfile_xpu \
  -t openvino/ubuntu22_dev_xpu:2023.3.0 .

$ sudo docker images
REPOSITORY                 TAG        ...   SIZE
openvino/ubuntu22_dev_xpu  2023.3.0   ...   5.87GB
```

Cmd 5. Command to Test xPU Docker* Image

```
$ sudo docker run -it \
  --device=/dev/dri/renderD128 \
  --device=/dev/accel/accel0 \
  --user root \
  --rm openvino/ubuntu22_dev_xpu:2023.3.0 \

"/opt/intel/openvino/samples/cpp/samples_bin/samples_bin/benchmar
k_app" "--help"

...
Available target devices:  CPU  GNA  GPU  NPU
```

3.4 Build OpenVINO™ Notebooks Docker* Image

Use the latest OpenVINO™ notebooks version 2024.2 to get more updated examples but when running the notebooks, remove `pip` installation of the OpenVINO™ package. Cmd 6 shows the Dockerfile and Cmd 7 shows the steps to build the Docker* image.

Cmd 6. OpenVINO™ Notebooks Dockerfile

```
$ sudo vim Dockerfile_xpu_nb
FROM openvino/ubuntu22_dev_xpu:2023.3.0

# Install Basics
USER root
RUN pip3 install tensorflow==2.12
RUN pip3 install keras==2.12

# Install OpenVINO Open Model Zoo
RUN wget
https://github.com/openvinotoolkit/open_model_zoo/archive/refs/ta
gs/2023.3.0.zip \
 && unzip 2023.3.0.zip \
 && mv open_model_zoo-2023.3.0 open_model_zoo
RUN pip3 install -r open_model_zoo/demos/requirements.txt
```

```
# Install OpenVINO Notebooks
RUN git clone -b 2024.2
https://github.com/openvinotoolkit/openvino_notebooks.git
WORKDIR /opt/intel/openvino
RUN python3 -m pip install --upgrade pip
RUN pip3 install wheel
RUN pip3 install jupyterlab ipywidgets "ipykernel>=5.0"
"ipython>=7.16.3" "setuptools<70"
RUN rm 2023.3.0.zip

# Finalizing
USER openvino
WORKDIR /opt/intel/openvino
```

Cmd 7. Command to Build OpenVINO™ Notebooks Docker* Image

```
$ sudo docker build \
  -f Dockerfile_xpu_nb \
  -t openvino/ubuntu22_dev_xpu_nb:2023.3.0 .

$ sudo docker images
REPOSITORY                     TAG          ...    SIZE
openvino/ubuntu22_dev_xpu_nb   2023.3.0     ...    11.5GB
openvino/ubuntu22_dev_xpu      2023.3.0     ...    5.87GB
```

Since a container is stateless, create a directory to store the files permanently. In this case, it is `results` directory. Any changes made in the container outside this directory will be gone after restarting the container. Use Cmd 8 to run the notebooks container. Then, open a web browser to access the notebooks using the provided IP address, port, as well as the token. All notebooks are located inside `openvino_notebooks` directory.

Cmd 8. Command to Run OpenVINO™ Notebooks Docker* Image

```
$ mkdir results

$ sudo docker run -it \
  -p 8888:8888 \
  -v ./results:/opt/intel/openvino/results \
  --device=/dev/dri/renderD128 \
  --device=/dev/accel/accel0 \
  --user root \
  --rm openvino/ubuntu22_dev_xpu_nb:2023.3.0 \
  bash -c "jupyter lab --no-browser --allow-root --ip 0.0.0.0 --
port=8888"
```

3.5 Run QnA Notebooks

Quantizing LLM with 7 to 8 billion parameters to INT4 requires up to 64 GiB of memory. Use Cmd 9 to increase the system memory using Swap file. Please note that the example increases the memory to 128 GiB but the current Intel® Core™ Ultra Processors support up to 96 GiB only.

Cmd 9. Command to Increase Memory Size using Swap File

```
$ free -m
               total        ...
Mem:           63932        ...
Swap:              0        ...

$ sudo fallocate -l 64G /swapfile
$ sudo chmod 600 /swapfile
$ sudo mkswap /swapfile
Setting up swapspace version 1, size = 64 GiB (68719472640 bytes)
no label, UUID=4c2fcc31-5b73-4a34-97e3-d1b8056f66b6
$ sudo swapon /swapfile

$ free -m
               total        ...
Mem:           63932        ...
Swap:          65535        ...
```

Use Cmd 10 to run QnA application using 12 CPUs and iGPU (`/dev/dri/renderD128`). For microphone and speaker, please add USB (`/dev/bus/usb`) or sound (`/dev/snd`) device accordingly.

Cmd 10. Command to Run QnA Notebooks

```
$ mkdir results

$ sudo docker run -it \
  --cpus="12.0" \
  -p 8887:8887 \
  -v ./results:/opt/intel/openvino/results \
  --device=/dev/dri/renderD128 \
  --device=/dev/accel/accel0 \
  --user root \
  --rm openvino/ubuntu22_dev_xpu_nb:2023.3.0 \
  bash -c "jupyter lab --no-browser --allow-root --ip 0.0.0.0 --
port=8887"
```

Followings are the sample notebooks for each of the components in QnA application. As mentioned earlier, when running the notebooks, please remove OpenVINO™ from the `pip` installation to have the consistent OpenVINO™ version as the one installed in the Docker* image.

- ASR
    - Automatic speech recognition using Distil-Whisper and OpenVINO™
    - Video Subtitle Generation using Whisper and OpenVINO™
- LLM
    - LLM Instruction-following pipeline with OpenVINO™

- TTS
  - [Voice tone cloning with OpenVoice and OpenVINO™]
  - [Text-to-speech Python* Demo]

3.6 Run Video Surveillance Analytics Notebooks

For 4 cameras implementation, it is recommended to utilize 4 CPUs. One CPU can be used for each camera pre and post-processing. Cmd 11 shows the command to run video surveillance analytics application using 4 CPUs and NPU (`/dev/accel/accel0`).

Cmd 11. Command to Run Video Surveillance Analytics Notebooks

```
$ mkdir results

$ sudo docker run -it \
  --cpus="4.0" \
  -p 8886:8886 \
  -v ./results:/opt/intel/openvino/results \
  --device=/dev/dri/renderD128 \
  --device=/dev/accel/accel0 \
  --user root \
  --rm openvino/ubuntu22_dev_xpu_nb:2023.3.0 \
  bash -c "jupyter lab --no-browser --allow-root --ip 0.0.0.0 --
port=8886"
```

Following notebook can be used for video surveillance analytics application using YOLOv8 model from Ultralytics*. For OpenVINO™ version 2023.3 LTS, please `pip` install Ultralytics* package version 8.1.42 instead of 8.2.24.

- Object Detection
  - [Convert and Optimize YOLOv8 real-time object detection with OpenVINO™]

§

4.0 BOMs

4.1 Test Setup

Table 2 lists both hardware and software BOMs for the implementation.

Table 2. Hardware and Software BOMs

Component	Information
Processor	Intel® Core™ Ultra 7 Processor 155H 6P+8E+2LPE-cores, 22 Threads 24 MB Intel® Smart Cache, up to 4.80 GHz
Memory	32 GiB LPDDR5 96 GiB Swap File
Storage	500 GiB SSD
Operating System	Ubuntu* version 22.04 LTS, Kernel 6.5
Graphics Driver	Version 24.13.29138.29
NPU Driver	Version 1.5.0
Docker*	Version 24.0.5
OpenVINO™ Toolkit	Version 2023.3 LTS
OpenVINO™ Notebooks	Version 2024.2
Ultralytics*	Version 8.1.42

4.2 Results

This implementation is targeted for edge use cases. The QnA application can be run in real time. The ASR and TSS can produce the results in one-third of the time whereas the LLM (8 billion quantized parameters) takes less than one second to the first token with subsequent 18 tokens/second. For English, one word comprises two to three tokens. So, the LLM can produce between 6 and 9 English words per second.

Object detection using the YOLOv8n model requires about 100 milliseconds of latency to infer 4 batch images from 4 input cameras. By utilizing asynchronous pre- and post-processing together with model inference, the video surveillance analytics application can handle 10 fps input from 4 cameras.

§