The science behind the report:

Workstations powered by Intel can play a vital role in CPU-intensive AI developer tasks

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Workstations powered by Intel can play a vital role in CPU-intensive AI developer tasks.

We concluded our hands-on testing on April 23, 2024. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on April 20, 2024 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: The amount of time, in seconds, each phase of Workflow 1 took using Intel Python and libraries (average of three workstations).

Workflow 1	Load	Split	Chunk	Load embed model	DB upload	Total
Time to complete on tower workstations (seconds)	12.9922	5.17E-06	0.0263	1.4284	35.4489	**49.8958**
Time to complete on mobile workstations (seconds)	11.0388	6.12E-06	0.0223	1.1806	87.9027	**100.1444**

Table 2: The amount of time, in seconds, each phase of Workflow 2 took using Intel Python and libraries (average of three workstations).

Workflow 2	Pre-processing	Processing	Post-processing	Total
Time to complete on tower workstations (seconds)	0.6214	136.5530	0.3064	**137.4808**
Time to complete on mobile workstations (seconds)	1.3099	265.7100	0.51851	**267.5384**

Table 3: The amount of time, in seconds, each phase of Workflow 3 took using Intel Python and libraries (average of three tower workstations).

Workflow 3: Time	ModelCheck	ModelOpt	Inputs	Outputs	Total
Time to complete using 32-bit FP precision on tower workstations (seconds)	0.2788	4.3338	0.3512	47.1547	**52.1185**
Time to complete using 16-bit FP precision on tower workstations (seconds)	0.3048	4.2573	0.3532	33.2730	**38.1882**

Table 4: The amount of memory each phase of Workflow 3 consumed using Intel Python and libraries (average of three tower workstations).

Workflow 3: Memory usage	ModelCheck	ModelOpt	Inputs	Outputs	Total
Average memory usage when using 32-bit FP precision on tower workstations (GB)	1.3866	30.7443	30.8535	32.2801	**95.2644**
Average memory usage when using 16-bit FP precision on tower workstations (GB)	1.3869	16.2995	16.4211	18.9202	**53.0277**

Table 5: The amount of time, in seconds, each phase of Workflow 3 took using Intel Python and libraries (average of two mobile workstations).

Workflow 3: Time	ModelCheck	ModelOpt	Inputs	Outputs	Total
Time to complete using 32-bit FP precision on mobile workstations (seconds)	0.3504	5.7059	0.3770	217.3690	**223.8023**

System configuration information

Table 6: Detailed configuration information for the mobile workstations we tested.

System configuration information	HP ZBook Fury 16 G10	Lenovo® ThinkPad® P16 Gen 2	Dell™ Precision™ 7780
BIOS name and version	V96 Ver. 01.04.00	N3TET51W (1.51)	1.10.0
Non-default BIOS settings	None	None	None
Operating system, version / build number	Windows 11 23H2 / 22631.34457	Windows 11 23H2 / 22631.34457	Windows 11 23H2 / 22631.34457
Date of last OS updates / patches applied	4/11/2024	4/11/2024	4/11/2024
Power management policy	OS: best performance	OS: best performance	OS: best performance
Processor			
Number of processors	1	1	1
Vendor and model	Intel® Core™ i7-13850HX	Intel Core i9-13980HX	Intel Core i7-13980HX
Performance / Efficient cores (per processor)	8 / 12	8 / 16	8 / 12
Performance / Efficient logical cores (per processor)	16 / 12	16 / 16	16 / 12
Stepping	1	1	1
Memory module(s)			
Total memory in system (GB)	32	64	64
Number of memory modules	2	2	1
Vendor and model	Samsung® M425R2GA3BB0-CWMOD	SK Hynix® HMCG88AGBSA095N	Dell KP7TJK-HYA-I
Size (GB)	16	32	64
Type	PC5-44800	PC5-44800	PC5-44800
Speed (MHz)	5,600	5,600	5,600
Speed running in the system (MHz)	4,000	4,000	5,200
Integrated Graphics			
Vendor and model	Intel UHD Graphics for 13th Gen Intel Processors	Intel UHD Graphics for 13th Gen Intel Processors	Intel UHD Graphics for 13th Gen Intel Processors
Driver version	31.0.101.5186	31.0.101.5186	31.0.101.4575
Discrete graphics			
Vendor and model	NVIDIA® RTX 2000 Ada Generation	NVIDIA RTX 5000 Ada Generation	NVIDIA RTX 4000 Ada Generation
Driver version	31.0.15.3818	31.0.15.3808	31.0.15.3827
Number of cards	1	1	1
Local storage (type A)			
Number of drives	1	1	1
Drive vendor and model	Micron® MTFDKBA512TFH-1BC1AABHA	Kioxia® KXG8AZNV1T02	Samsung PM9A1
Drive size (GB)	512	1,024	1,024
Drive information (speed, interface, type)	PCIe NVMe® SSD	PCIe NVMe SSD	PCIe NVMe SSD

Table 7: Detailed configuration information for the tower workstations we tested.

System configuration information	HP Z8 Fury G5	Lenovo ThinkStation® P7	Dell Precision 7960
BIOS name and version	U61 Ver. 01.01.27	S0DKT14A	1.1.14
Non-default BIOS settings	None	None	None
Operating system, version / build number	Ubuntu 22.04.4 / 5.15.0-105-generic	Ubuntu 22.04.4 / 5.15.0-105-generic	Ubuntu 22.04.4 / 5.15.0-105-generic
Date of last OS updates / patches applied	4/20/2024	4/20/2024	4/20/2024
Power management policy	OS: energy performance preference set to performance	OS: energy performance preference set to performance	OS: energy performance preference set to performance
Processor			
Number of processors	1	1	1
Vendor and model	Intel Xeon® w7-3455	Intel Xeon w9-3495X	Intel Xeon w7-3455
Cores (per processor)	24	56	24
Logical cores (per processor)	48	112	48
Core frequency (GHz)	2.5	1.9	2.5
Stepping	6	8	8
Memory module(s)			
Total memory in system (GB)	128	128	128
Number of memory modules	8	8	4
Vendor and model	SK Hynix HMCG78MEBRA113N	Micron MTC10F1084S1RC48BA1	SK Hynix HMCG88AEBRA107N
Size (GB)	16	16	32
Type	PC5-38400	PC5-38400	PC5-38400
Speed (MHz)	4,800	4,800	4,800
Speed running in the system (MHz)	4,800	4,800	4,800
Discrete graphics			
Vendor and model	NVIDIA RTX A4000	N/A	NVIDIA RTX A4000 (GA104GL)
Driver version	Not installed	Not installed	Not installed
Number of cards	1	0	1
Local storage (type A)			
Number of drives	1	1	1
Drive vendor and model	Samsung MZVL21T0HCLR-00BH1	Samsung MZVL21T0HCLR-00BL7	SK Hynix PC801
Drive size (GB)	1,024	1,2024	1,024
Drive information (speed, interface, type)	PCIe NVMe SSD	PCIe NVMe SSD	PCIe NVMe SSD

How we tested

Installing the operating system on the tower workstations

1. Install Ubuntu 22.04 LTS on the bare-metal server with the server minimal packages plus OpenSSH server and client.
2. Log into the system.
3. Update the system software:

```
sudo apt update
sudo apt upgrade
sudo shutdown -r now
```

4. Log into the system using ssh.
5. Install the Redis vector database:

```
o0="/usr/share/keyrings/redis-archive-keyring.gpg"
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o $o0
sudo chmod 644 $o0
echo "deb [signed-by=$o0] https://packages.redis.io/deb jammy main" | \
  sudo tee /etc/apt/sources.list.d/redis.list
unset o0
sudo apt-get update
sudo apt-get install redis-stack-server
sudo systemctl enable redis-stack-server.service
```

6. Reboot the workstation:

```
sudo shutdown -r now
```

Installing the operating system on the mobile workstations

1. Log into the Windows 11 system.
2. Use Windows Update to update the system.
3. Reboot the laptop.
4. Log into the system.
5. Install WSL2:
    i. Open an Admin terminal.
    ii. Install WSL and the Ubuntu 22.04 LTS as the Linux default:

```
wsl --install --no-launch --distribution Ubuntu-22.04
```

6. Configure WSL to use more memory.
    i. Open a non-admin terminal.
    ii. Create and edit the WSL configuration file:

```
notepad c:\Users\<ACCOUNT>\.wslconfig
```

    iii. Add the following to the file. Uncomment the memory specification line that corresponds to the laptop manufacturer.

```
[wsl2]
# Uncomment the next line for the HP mobile workstation
#memory=24GB

# Uncomment the next line for the Dell or Lenovo mobile workstation
#memory=48GB
```

iv. Save the file, and start the Ubuntu instance.

```
wsl
```

7. Open a terminal to the WSL Ubuntu instance.
8. Update the Ubuntu system software:

```
sudo apt update
sudo apt upgrade
sudo apt install openssh-client openssh-server
```

9. Install the Redis vector database:

```
o0="/usr/share/keyrings/redis-archive-keyring.gpg"
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o $o0
sudo chmod 644 $o0
echo "deb [signed-by=$o0] https://packages.redis.io/deb jammy main" | \
  sudo tee /etc/apt/sources.list.d/redis.list
unset o0
sudo apt-get update
sudo apt-get install redis-stack-server
sudo systemctl enable redis-stack-server.service
```

10. Reboot the laptop:

```
sudo shutdown -r now
```

11. Log into the Windows system, and open an admin terminal.
12. Configure the Windows firewall and network to allow incoming SSH connections:

```
New-NetFirewallRule -Name sshd -DisplayName 'OpenSSH Server (sshd) for WSL' -Enabled True -Direction
Inbound -Protocol TCP -Action Allow -LocalPort 22
# get the IP address for the Ubuntu instance
wsl hostname -I
# use this IP addres in the following command
netsh.exe interface portproxy add v4tov4 listenport=22 listenaddress=0.0.0.0 connectport=22
connectaddress=<THE IP ADDRESS FROM THE PREVIOUS COMMAND>
```

13. Close the admin terminal.
14. Open a non-admin terminal, and start WSL:

```
wsl
```

15. Configure the SSH server to start when the Ubuntu instance starts:

```
sudo systemctl enable sshd.service
sudo systemctl start sshd.service
```

16. Leave this terminal window open.

Preparing the Python environment for the workflows

1. Install Anaconda to manage the Python environment. We used its default installation location and configuration.

```
curl -O https://repo.anaconda.com/archive/Anaconda3-2024.02-1-Linux-x86_64.sh
bash Anaconda3-2024.02-1-Linux-x86_64.sh
conda update conda
conda config --add channels intel
```

2. Create three Python environments, one for each workflow, and install the optimized Intel Python distribution. We installed additional CPU-only packages from either the Intel optimized repository or from the default repository.

   - For Workflow 1:

```
conda create -n wf1 intelpython3_core python=3.10
conda activate wf1
conda install -c intel -c conda-forge --override-channels \
  intel-extension-for-tensorflow=2.15=*cpu* \
  intel/label/oneapi::intel-extension-for-pytorch=2.2.0 intel/label/oneapi::pytorch=2.2.0 \
  intel/label/oneapi::oneccl_bind_pt=2.2.0 intel/label/oneapi::torchvision=0.17.0 \
  intel/label/oneapi::torchaudio=2.2.0 conda-forge::deepspeed=0.14.0 python=3.10
pip install langchain==0.1.13 pydantic lxml pypdf accelerate langchain-community \
  redis sentence-transformers psutil
o0="/usr/share/keyrings/redis-archive-keyring.gpg"
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o $o0
sudo chmod 644 $o0
echo "deb [signed-by=$o0] https://packages.redis.io/deb jammy main" | \
  sudo tee /etc/apt/sources.list.d/redis.list
unset o0
sudo apt-get update
sudo apt-get install redis-stack-server
sudo systemctl enable redis-stack-server.service
sudo systemctl start redis-stack-server.service
```

   - For Workflow 2:

```
conda create -n wf2 intelpython3_core python=3.10
conda activate wf2
pip install opencv-python
```

   - For Workflow 3:

```
conda create -n wf3 intelpython3_core python=3.10
conda activate wf3
conda install -c intel -c conda-forge --override-channels \
  intel-extension-for-tensorflow=2.15=*cpu* \
  intel/label/oneapi::intel-extension-for-pytorch=2.2.0 intel/label/oneapi::pytorch=2.2.0 \
  intel/label/oneapi::oneccl_bind_pt=2.2.0 intel/label/oneapi::torchvision=0.17.0 \
  intel/label/oneapi::torchaudio=2.2.0 conda-forge::deepspeed=0.14.0 python=3.10

pip install git+https://github.com/huggingface/accelerate.git
pip install git+https://github.com/huggingface/transformers.git
conda install pillow
```

Running Workflow 1

Note: Run this workload (step 5) once before testing to download the necessary data from the internet.

1. Start the Redis database:

```
sudo systemctl enable redis-stack-server.service
```

2. Activate the Python environment:

```
conda activate wf1
```

3. Download the input documents by running the script docs_download.sh:

```
cd ~
mkdir docs
cd docs
bash ~/docs_download.sh
cd ~
```

4. Clean the operating system's file cache:

```
echo 3 | sudo tee /proc/sys/vm/drop_caches
```

5. Run the workload. Find the Python script below as wf1.py:

```
time python wf1.py |& tee wf1-out.txt
```

6. The system reports how much time each task in the workflow took to complete at the end.

Running Workflow 2

Note: Run this workload (step 5) once before testing to download the necessary data from the Internet.

1. Activate the Python environment:

```
conda activate wf2
```

2. Create the directories for the input and processed image files:

```
mkdir inputs outputs
```

3. Download the images:

```
curl -O https://raw.githubusercontent.com/UCSD-AI4H/COVID-CT/master/Images-processed/CT_COVID.zip
curl -O https://raw.githubusercontent.com/UCSD-AI4H/COVID-CT/master/Images-processed/CT_NonCOVID.zip
unzip CT_COVID.zip
unzip CT_NonCOVID.zip
mv CT_COVID/* inputs
mv CT_NonCOVID/* inputs
rm -rf CT*
```

4. Clean the operating system’s file cache:

```
echo 3 | sudo tee /proc/sys/vm/drop_caches
```

5. Run the workload. Find the Python script below as wf2.py:

```
time python wf2.py |& tee wf2-out.txt
```

6. The workflow reports the running times for each image as a list of three times in seconds. They are the time necessary to read the original image and convert it to a format that can be analyzed; the time necessary to analyze the image; and the time necessary to reformat the result, convert it to PNG format, and save it.

Running Workflow 3

Note: Run this workflow once (step 3) before testing to download the necessary data from the internet.

1. Activate the Python environment:

```
conda activate wf3
```

2. Create the directories.
3. Clean the operating system's file cache:

```
echo 3 | sudo tee /proc/sys/vm/drop_caches
```

4. Run the workload. Find the Python script below as wf3.py:

```
time python wf3.py |& tee wf3-out.txt
```

5. The system reports how much time each task in the workflow took to complete at the end.

Scripts

File wf1.py: Python script for Workflow 1

```
#!/bin/env python3
import time

from langchain.vectorstores.redis import Redis
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import PyPDFDirectoryLoader
from langchain_community.embeddings import HuggingFaceEmbeddings

# grab PDFs in direfctory docs_dir
start_time = time.time()
docs_dir = "docs/"
loader = PyPDFDirectoryLoader(docs_dir)
data = loader.load()
load_time = time.time() - start_time

# chunk the docs
start_time = time.time()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=10000, chunk_overlap=20)
split_time = time.time() - start_time

start_time = time.time()
text_chunks = text_splitter.split_documents(data)
chunk_time = time.time() - start_time

# select embedding model
start_time = time.time()
embedder = HuggingFaceEmbeddings(model_name="msmarco-distilbert-base-v4")
em_time = time.time() - start_time

# upload chunks and create the index
start_time = time.time()
```

```
docs_vectorstore = Redis.from_documents(
    text_chunks,
    embedding=embedder,  # an Embeddings object
    redis_url="redis://localhost:6379",
    index_name="idx:faa_vss",
)

upload_time = time.time() - start_time

print("PDF_load___time {} s".format(load_time))
print("Text_Split_time {} s".format(split_time))
print("Text_Chunk_time {} s".format(chunk_time))
print("Load_Embed_time {} s".format(em_time))
print("DB_Upload__time {} s".format(upload_time))
```

File docs_download.sh: shell script to download input documents for Workflow 1

```
#!/bin/bash
declare -A dir file
dir[NCT02593864]=64; dir[NCT01625013]=13; dir[NCT01878253]=53
dir[NCT02034409]=09; dir[NCT02155257]=57; dir[NCT02155257a]=57
dir[NCT02560922]=22; dir[NCT02683785]=85; dir[NCT02683785a]=85
dir[NCT02688400]=00; dir[NCT02713542]=42; dir[NCT02888119]=19
dir[NCT02905240]=40dir[NCT02905240a]=40; dir[NCT03491397]=97
dir[NCT03780400]=00; dir[NCT04456686]=86; dir[NCT04456686a]=86
dir[CT04627038]=38; dir[NCT04627038a]=38
file[NCT02593864]=Prot_SAP_000.pdf; file[NCT01625013]=Prot_SAP_001.pdf
file[NCT01878253]=Prot_000.pdf; file[NCT02034409]=Prot_SAP_000.pdf
file[NCT02155257]=Prot_000.pdf; file[NCT02155257a]=SAP_001.pdf; file[NCT02560922]=Prot_SAP_000.pdf;
file[NCT02683785]=Prot_000.pdf
file[NCT02683785a]=SAP_001.pdf; file[NCT02688400]=Prot_SAP_000.pdf
file[NCT02713542]=Prot_SAP_000.pdf; file[NCT02888119]=Prot_SAP_000.pdf
file[NCT02905240]=Prot_000.pdf; file[NCT02905240a]=SAP_001.pdf
file[NCT03491397]=Prot_SAP_000.pdf; file[NCT03780400]=Prot_SAP_000.pdf
file[NCT04456686]=Prot_000.pdf; file[NCT04456686a]=SAP_001.pdf
file[NCT04627038]=Prot_000.pdf; file[NCT04627038a]=SAP_001.pdf
studies=(NCT02593864 NCT01625013 NCT01878253 NCT02034409 NCT02155257 \
 NCT02155257a NCT02560922 NCT02683785 NCT02683785a NCT02688400 NCT02713542 \
 NCT02888119 NCT02905240 NCT02905240a NCT03491397 NCT03780400 NCT04456686 \
 NCT04456686a NCT04627038 NCT04627038)
url="https://cdn.clinicaltrials.gov/large-docs"
for study in "${studies[@]}"; do
 curl "$url/${dir[$study]}/$study/${file[$study]}" -o "${study}_${file[$study]}"
 sleep 1
done
```

File wf2.py: Python script for Workflow 2

```
#!/bin/env python
import concurrent.futures
import numpy as np
import cv2
import os
import time

input_dir = "inputs"
output_dir = "outputs"

def segment_image(name):
    timings = [0] * 3
    start_time = time.time()
    # Read in the image
    image = cv2.imread(os.path.join(input_dir, name))
    # Change color to RGB (from BGR)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Reshaping the image into a 2D array of pixels and 3 color values (RGB)
    pixel_vals = image.reshape((-1, 3))
```

```
    # Convert to float type
    pixel_vals = np.float32(pixel_vals)
    timings[0] += time.time() - start_time
    # kmeans stopping criteria
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.95)
    start_time = time.time()
    # then perform k-means clustering with number of clusters defined as 3
    # perform kmeans with 3 clusters and 1a00 iterations
    k = 3
    retval, labels, centers = cv2.kmeans(
        pixel_vals, k, None, criteria, 100, cv2.KMEANS_RANDOM_CENTERS
    )

    timings[1] += time.time() - start_time
    start_time = time.time()
    # convert data into 8-bit values
    centers = np.uint8(centers)
    segmented_data = centers[labels.flatten()]
    # reshape data into the original image dimensions
    segmented_image = segmented_data.reshape((image.shape))
    cv2.imwrite(os.path.join(output_dir, name), segmented_image)
    timings[2] += time.time() - start_time
    return timings

names = os.listdir(input_dir)
with concurrent.futures.ProcessPoolExecutor() as executor:
    for name, timing in zip(names, executor.map(segment_image, names)):
        print("Times for {}: {}".format(name, timing))
```

File wf3.py: Python script for Workflow 3

```
#!/bin/env python
import time
import os
import psutil
from transformers import LlavaNextProcessor, LlavaNextForConditionalGeneration
import torch
from PIL import Image
import requests

pid = os.getpid()
python_process = psutil.Process(pid)
gig = 1 / 2.0**30

memoryUse = python_process.memory_info()[0]
print('Inital memory use: {} GB'.format(memoryUse * gig))
start_time = time.time()
processor = LlavaNextProcessor.from_pretrained("llava-hf/llava-v1.6-mistral-7b-hf")
model_check = time.time() - start_time
memory_check = python_process.memory_info()[0]
start_time = time.time()
model = LlavaNextForConditionalGeneration.from_pretrained("llava-hf/llava-v1.6-mistral-7b-hf", torch_
dtype=torch.bfloat16, low_cpu_mem_usage=True)
model.to("cpu")
model_opt = time.time() - start_time
memory_opt = python_process.memory_info()[0]
# prepare image and text prompt, using the appropriate prompt template
url= "https://media.nga.gov/iiif/8f29e3c9-a289-4d53-abf0-31a66e9e98fa/full/full/0/default.
jpg?attachment_filename=ginevra_de%27_benci__obverse__1967.6.1.a.jpg"
image = Image.open(requests.get(url, stream=True).raw)
prompt = "[INST] <image>\nWhat is shown in this image? [/INST]"
start_time = time.time()
inputs = processor(prompt, image, return_tensors="pt").to("cpu")
inputs_time = time.time() - start_time
memory_inputs = python_process.memory_info()[0]
# autoregressively complete prompt
start_time = time.time()
output = model.generate(**inputs, max_new_tokens=100)
output_time = time.time() - start_time
memory_outputs = python_process.memory_info()[0]
```

```
print(processor.decode(output[0], skip_special_tokens=True))
print("Model_Check_time {} s".format(model_check))
print("Model_Opt___time {} s".format(model_opt))
print("Inputs______time {} s".format(inputs_time))
print("Outputs_____time {} s".format(output_time))
print("Model_Check__mem {}{ GB".format(memory_check * gig))
print("Model_Opt____mem {} GB".format(memory_opt * gig))
print("Inputs______mem {} GB".format(memory_inputs * gig))
print("Outputs_____mem {} GB".format(memory_outputs * gig))
```

Read the report at https://facts.pt/8xoaOpQ ▶

This project was commissioned by Intel.

Principled Technologies®

Facts matter.®