Google Cloud

We tested Intel's AMX CPU accelerator for AI. Here's what we learned

October 2024

Running AI/ML workloads on CPUs have gained a new advantage: stronger security

The latest Confidential VMs we've developed with Intel have a built-in CPU accelerator on by default. This makes it easier for customers to secure their inference, fine tuning, and small-to-medium sized training jobs.

Confidential computing is the protection of data in-use through hardware-based technologies. Google Cloud offers Confidential VM, a type of Compute Engine VM that uses Confidential Computing to ensure data and applications stay private and encrypted even while in use.

Confidential VM is available with a confidential computing technology called Intel Trust Domain Extensions (Intel TDX). Confidential VM with Intel TDX can help safeguard against insider attacks and software vulnerabilities and ensure data and code confidentiality and integrity.

These Confidential VMs, like all C3 VMs, have Intel Advanced Matrix Extensions (Intel AMX) enabled. Intel AMX improves the performance of deep-learning training and inference on the CPU, and is ideal for workloads including natural-language processing, recommendation systems, and image recognition.

"We are excited to partner with Google, delivering Confidential Computing and AI acceleration on C3 instances with Intel TDX and Intel AMX and software acceleration available in the popular AI libraries, to democratize access to secured, accelerated computing,"

Andres Rodriguez
Intel Fellow and datacenter AI architect

To harness the full potential of Intel AMX, existing machine learning (ML) pipelines and frameworks must be optimized to support its capabilities. Intel has undertaken extensive efforts to optimize popular ML frameworks, such as TensorFlow, PyTorch, and JAX, for Intel AMX utilization (see here for details). Intel has also provided guidance on how an AI pipeline can be optimized and how a PyTorch extension on GitHub can be used to accelerate many large language models (LLMs).

Performance testing AI/ML workloads on Confidential VMs

We conducted ML training and inference experiments to evaluate the performance of Confidential VMs with Intel TDX and Intel AMX. In general, workloads running in Confidential VMs with Intel TDX may experience some performance overhead due to the memory encryption and integrity protection provided and the additional changes required in the software stack to enable confidential computing.

Configurations and settings

We ran our experiments with an Ubuntu 22.04 LTS guest OS image. To reduce experiment noise, we disabled hyperthreading on all VMs. Performance impact can vary greatly depending on the optimization library used and software stack changes in the guest and host kernel.

For our experiments, we compared three VMs that were the largest machine types by vCPU available:

01

N2 VM

N2 machine series

VMs that do not have Intel AMX or Intel TDX enabled.

02

C3 VM

C3 machine series

VMs that have Intel AMX enabled.

03

C3+TDX VM

C3 machine series

VMs have Intel TDX and Intel AMX enabled.

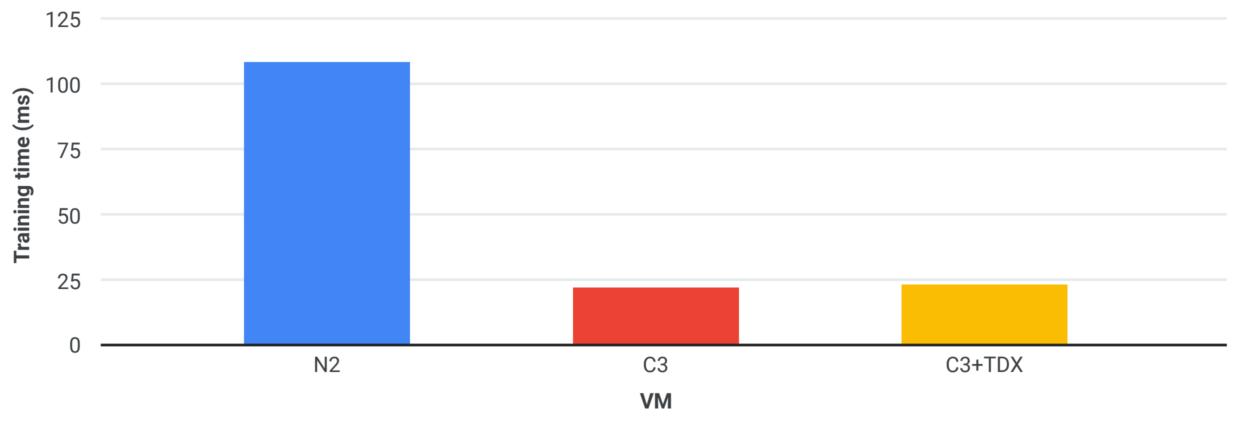
Training findings

We trained RoBERTa, a LLM that provides improvement on BERT and is widely used in natural language processing (NLP) applications. We fine-tuned RoBERTa on its ability to answer questions with the Stanford Question Answering Dataset (SQuAD) for one epoch. We used the HuggingFace Trainer and PyTorch frameworks to run the evaluation and to use their smooth integrations with Intel Extension for PyTorch (IPEX) software acceleration and Intel AMX hardware acceleration.

Note that you could use other popular ML frameworks such as TensorFlow to do this evaluation.

We compared the training time (in minutes) on the aforementioned N2, C3, and C3+TDX VMs. Compared to the N2 VM, the C3 VM provided a 4.54x speedup on training time whereas the C3+TDX VM provided a 4.14x speedup on training time for one workload, with the same hyperparameters such as global batch size and on-par quality metrics.

Training time by virtual machine in minutes (lower is better)



Considerations to optimize training

Our investigations gave us many insights on optimizing performance.
To optimize training performance on the C3 machines series, consider the following:

01

The C3 machine series has multiple cores so you can use distributed training techniques like Distributed Data Parallel (DDP) to improve CPU utilization. In DDP, you can control the number of processes for data parallelism and number of threads in each process for intra-op parallelism. The choice of these parameters trades off compute utilization and communication cost and affects training time. When the number of threads is small, some cores may be idle, so CPU utilization may not be saturated. On the other hand, a large number of processes has a higher communication overhead and is more prone to be delayed by stragglers.

02

C3 VM shapes are optimized for the underlying NUMA architecture to deliver consistent performance. For C3 VMs that have less than 44 vCPUs, there will be one NUMA node. For C3 VMs that have 44 to 88 vCPUs, there will be two NUMA nodes. For C3 VMs that have more than 88 vCPUs, there will be four NUMA nodes. Spreading threads across different NUMA nodes can impact the memory access latencies and performance. Therefore, depending on the workload needed, there should be at least one process per node that groups a set of threads and binds them to the vCPUs on the same NUMA node.

03

Intel AMX is an accelerator that resides on each physical CPU core and is placed near system memory. C3 VM shapes have Simultaneous Multithreading (SMT) enabled by default but running more threads than the number of Intel AMX accelerators may be inefficient. We recommend limiting the number of worker threads to be less than half of the number of available vCPU to avoid oversubscription and resource contention. Alternatively, SMT features can be disabled when creating a C3 VM. To learn how to disable hyperthreading, click here.

In our training experiments, we obtained good results with 8 processes, each with 10 threads on our C3 machine with 176vCPU, 4 NUMA nodes, and hyperthreading disabled. We also configured “CCL_WORKER_AFFINITY=auto”, which pins the communication worker to the last cores of the pin domains. Open MPI’s default settings largely take care of process pinning. To learn more about Open MPI process pinning, click here.

ML training parameters

Generally, a greater batch size may speed up training. When Intel TDX is enabled, a greater batch size also reduces the Intel TDX overhead. This is due to reduced memory encryption cost involved in I/O operations, given the same amount of ingested data. However, empirically, a greater batch size may compromise model quality, so this is a tradeoff you need to consider. In this experiment, we use a global batch size of 256.

Inference findings

We ran a text generation task on two LLMs, Llama-2-7B and Llama-2-13B. Using the aforementioned N2, C3, and C3+TDX VMs, we evaluated task performance based on two metrics:

Time per output token (TPOT)

Time to generate an output token for a single request. This metric corresponds to how a user would perceive the "speed" of the model. For example, a TPOT of 100 milliseconds per token would be approximately 450 words per minute (WPM), which is faster than an average person's reading speed.

Throughput (token per sec)

The number of output tokens per second an inference server can generate for batch requests.

We compared the N2 VM to C3 and C3+TDX VMs and found that Intel AMX on the C3 VM and C3+TDX VM provided a speedup of approximately a three-fold improvement in latency (TPOT) and approximately a seven-fold increase in throughput (with a batch size of six). All inference experiments were run with DeepSpeed and averaged over 50 runs. See table.

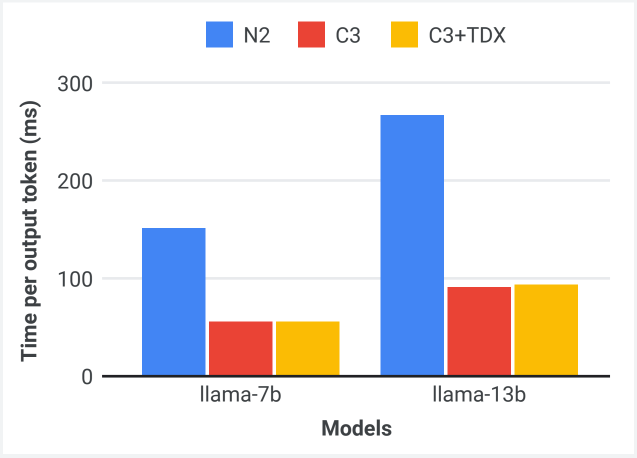
Time per output token speedup (ratio)

	C3 VM	C3+TDX
LLaMA2-7B	2.7x	2.7x
LLaMA2-13B	3.0x	2.9x

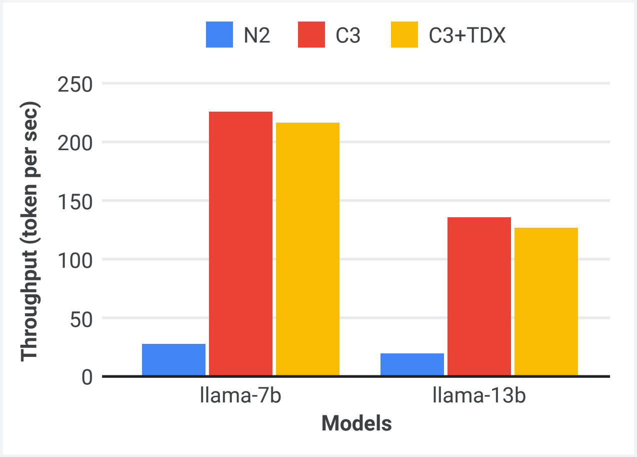
Throughput speedup (ratio)

	C3 VM	C3+TDX
LLaMA2-7B	7.7x	7.3x
LLaMA2-13B	7.5x	7.1x

Time per output token by virtual machine (lower is better)



Throughput by virtual machine (higher is better)

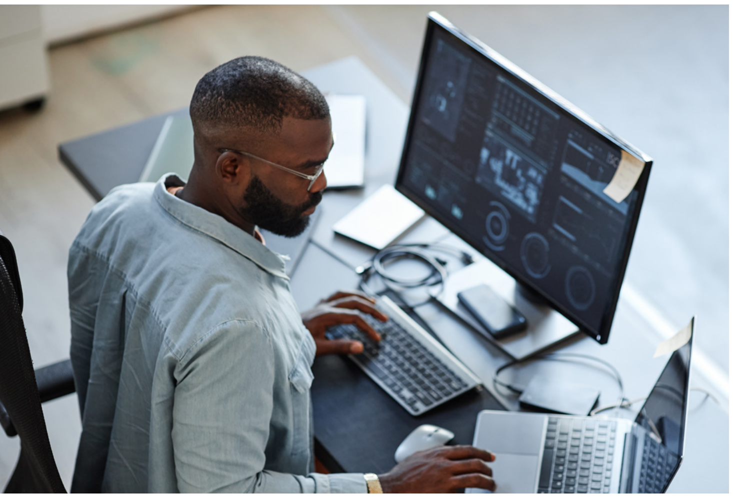


Considerations to optimize inference

We found that DeepSpeed, a deep learning optimization software suite, improves inference performance. In our inference experiment on C3 machines series, using DeepSeed reduced TPOT by down to 65.20% and improved throughput by up to 2.9x.

Try Intel AMX on Confidential VMs today

You can collect benchmarks for LLMs by following the Intel® Extension for PyTorch instructions. For training, here is a code snippet to get you started.



```
Unset

// Code snippet for creating a TDX VM (for more
information see public documentation)
gcloud beta compute instances create <instance_name> \
    --machine-type=c3-standard-176 \
    --threads-per-core=1 \
    --confidential-compute-type=TDX \
    --maintenance-policy=TERMINATE \
    --image-family=ubuntu-2204-lts \
    --image-project=tdx-guest-images
//

// Sample code snippets for training

// Create a new virtual environment.
python3 -m venv amx-exp
source amx-exp/bin/activate

// Install pytorch, ipex and oneccl.
pip3 install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cpu
// Sanity check with python -c "import torch;
print(torch.__version__)"
pip install intel-extension-for-pytorch
python -m pip install oneccl_bind_pt --extra-index-url
https://pytorch-extension.intel.com/release-whl/stable/
cpu/us/

// Install HuggingFace from source which is required to
run examples.
git clone https://github.com/huggingface/transformers.
git
cd transformers
pip install .

// Install question answering task specific
requirements.
cd examples/pytorch/question-answering
pip install -r requirements.txt

// Setup OneCCL.
oneccl_bindings_for_pytorch_path=$(python -c "from
oneccl_bindings_for_pytorch import cwd; print(cwd)")
source $oneccl_bindings_for_pytorch_path/env/setvars.sh

// Run!
export CCL_WORKER_COUNT=1 && \
export CCL_WORKER_AFFINITY=auto && \
export MASTER_ADDR=127.0.0.1 && \
mpirun -n 8 -genv OMP_NUM_THREADS=10 \
python run_qa.py \
  --model_name_or_path FacebookAI/roberta-base \
  --dataset_name squad \
  --do_train \
  --do_eval \
  --per_device_train_batch_size 32 \
  --per_device_eval_batch_size 128 \
  --learning_rate 3e-5 \
  --num_train_epochs 1 \
  --max_seq_length 384 \
  --doc_stride 128 \
  --output_dir /tmp/debug_squad/ \
  --no_cuda \
  --ddp_backend ccl \
  --bf16=True --use_ipex=True
```

Google Cloud