Technical Research Report

Which Toolkit Provides the Best Optimization for Large Language Models?

Prowess Consulting conducted research and testing to determine whether the Intel or Qualcomm SDK can help build the best pipeline for large language model deployment on Qualcomm® Snapdragon® Arm64–based SoCs and Intel® processors.

Executive Summary

Developers recognize the critical need for efficient AI solutions across diverse computing environments. As enterprises race to deploy AI projects, developers can gain a competitive edge by leveraging the AI and machine learning (ML) tools in software development kits (SDKs) to optimize large language models (LLMs) that power chatbots, virtual assistants, and other AI systems. Hardware-specific SDKs are designed to enable seamless integration with on-device hardware, enhancing model execution and accelerating neural network inference to improve the model's ability to apply patterns to new input.

With the transition to AI PCs, developers face important hardware-optimization choices for performance and efficiency. For example, they can build AI applications on devices powered by Intel® Core™ Ultra processors with hybrid architectures—using both Performance-cores (P-cores) and Efficient-cores (E-cores)—or on devices powered by Qualcomm® Snapdragon® Arm64 systems on a chip (SoCs), which are often used for mobile devices. In this study, commissioned by Intel, Prowess Consulting tested the Intel® OpenVINO™ toolkit and the Qualcomm® AI Engine Direct SDK on Dell™ XPS™ 13 AI PCs to determine the better choice for developers. The Intel OpenVINO toolkit earned higher scores in target hardware support, platform compatibility, features, ease of use, and other factors. Table 1 highlights our findings.

Table 1 | SDK scorecard: The Intel® OpenVINO™ toolkit versus the Qualcomm® AI Engine Direct SDK (five-star rating system: 1 [poor] to 5 [excellent])

Software Development Toolkits	Target Hardware	Platform Compatibility	Model Conversion	Inference	Community Support
Intel® Distribution of OpenVINO™ toolkit	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★★
Qualcomm® AI Engine Direct SDK	★★★☆☆	★★★☆☆	★☆☆☆☆	★★☆☆☆	★★☆☆☆

Primary drivers influencing the SDK ratings in Table 1:

- Less availability of Qualcomm® Snapdragon® X processor–based systems compared to Intel® Core™ Ultra processor–based systems, signaling slower adoption of Arm® architectures and software compatibility issues
- The Qualcomm® AI Engine Direct SDK offers enough features to run full AI pipelines, but some tools have limited functionality, particularly when working with LLMs

The AI Technology Challenge

Companies are investing in AI development with various degrees of success. One-third of generative AI (GenAI) projects fail to make it beyond the proof-of-concept stage, according to Gartner research, due to high costs, poor data quality, unclear business goals, and risk issues.[1] More industries are looking at LLMs to train smaller models and to use retrieval augmented generation (RAG) for specific tasks. As the push to integrate AI capabilities into enterprise systems fundamentally redefines line of business applications, enterprise development teams can take advantage of AI-enhanced hardware and software to automate tasks and improve their development processes.

Initially targeted at commercial markets, AI PCs are high-performing computers designed with neural processing units (NPUs), along with CPUs and GPUs, to allow machine learning and other AI functions to be performed locally on a device. The first Windows® AI PCs were previewed at the Microsoft Build 2023 conference.[2] These AI PCs featured NPUs, GPUs, and CPUs to power their AI models and the built-in Microsoft® Copilot® AI assistant. These systems began shipping in 2023 (Intel® processor–based AI PCs) and 2024 (Windows Copilot+ PCs with Qualcomm® Snapdragon® processors) from OEM partners such as Acer, ASUS, Dell Technologies, HP, Lenovo, and Samsung.[3]

By 2028, the majority of consumer PCs will support AI acceleration hardware, according to Gartner research.[4] These AI-enhanced systems provide developers with improved performance, reduced latency, and stronger data privacy and security. Additional benefits include longer battery life and lower costs compared to cloud-based AI processing.[5] This contrasts with standard PCs, which depend on cloud-based infrastructure to run AI applications due to the extensive memory and computational requirements of AI workloads, particularly those involving LLMs.

In the rapidly evolving AI landscape, most developers rely on SDKs to optimize, deploy and integrate LLMs that drive core functionality in AI-powered applications. Hardware-specific SDKs are designed to enable developers to build software that interfaces with on-device hardware, optimizing model execution and inference in neural networks. To make inference faster and more efficient, AI developers often use methods like quantization, which can improve performance and reduce memory and power consumption on the device. Quantization is a technique that converts a model's weights and activations from higher precision to lower precision (e.g., floating point to integer) during or after training. This smaller numerical footprint can speed up inference, though reduced precision may affect accuracy. With hardware-optimized tools, software developers can make use of an AI PC's hardware to enhance performance and resource efficiency.

As AI PCs become more prevalent, developers face hardware-optimization choices, such as whether to build AI applications on devices powered by Intel Core Ultra processors with hybrid architectures—using both P-cores and E-cores—or on systems powered by Qualcomm Snapdragon Arm64 SoCs. To help AI developers determine the right architecture for their needs, Prowess Consulting tested the Intel OpenVINO toolkit and the Qualcomm AI Engine Direct SDK on Dell XPS 13 AI PCs to see which tool came out on top. Specifically, we evaluated the target hardware, platform compatibility, features, ease of use, documentation, community support, open source versus proprietary tradeoffs, required skillsets, and costs.

Working with Local LLMs

Companies like Microsoft, Intel, Meta, and others are working to create open platforms for AI development. With the right hardware, AI developers can use open-source tools like Ollama or LM Studio to run models locally.

Models are getting better, faster, and in some cases smaller. The open-source Llama 3.2-3B (3 billion parameter) model from Meta AI was released in September 2024 for developers and researchers seeking pretrained models with low latency and low cost for enterprise use cases. It is a pretrained transformer[6] model for natural language processing (NLP), designed for neural networks, and it includes multilingual text and code capabilities. The 9B (9 billion parameter) and 11B (11 billion parameter) versions of the model are multimodal and can process both text and images.

Running models locally reduces dependency on cloud APIs, which often charge based on usage (for example, per 1,000 tokens). This can significantly cut costs, especially for high-volume or long-running projects.

Local customization of models for specific use cases or domains, such as healthcare, legal, finance, and customer service, can improve performance through faster iterations (with no API calls) and improved output accuracy. Organizations with high security and compliance requirements can also safeguard sensitive data and maintain privacy by running models offline.

Open-source AI tools offer flexibility and customization, but developers may need technical experience to optimize model performance and avoid vulnerabilities that could expose sensitive data. At the same time, tools like Intel OpenVINO allow developers to benefit from an active open-source community that contributes to models and frameworks—offering greater transparency into source code and project development. Companies including Amazon, Google (Alphabet), IBM, Intel, Microsoft, and Qualcomm collaborate with Hugging Face®, an open-source repository, to ensure that open-source models are accessible and safe to use in their environments.

In contrast, proprietary solutions like Qualcomm AI Engine Direct SDK may limit customization and come at a higher cost. However, they often provide access to pretrained, validated models along with dedicated vendor support. See Table 2 for more on the features and differences between the Intel OpenVINO toolkit and the Qualcomm AI Engine Direct SDK.

Table 2 | Tools face-off: The Intel® OpenVINO™ toolkit versus the Qualcomm® AI Engine Direct SDK

SDK	Intel® Distribution of OpenVINO™ toolkit	Qualcomm® AI Engine Direct (Qualcomm Neural Network [QNN])
Platform support	• Open-source tools and high-level APIs • C/C++ for low-level integrations, Python®, and Node.js®	• Suite of low-level control and optimization tools • Unified API and core-specific libraries • C/C++ for low-level integrations, Python for AI, and Java® • Development tools and environments (Qualcomm AI Hub, Qualcomm® Snapdragon® LLVM compiler, Windows®, Xcode® (DirectML), Ubuntu® (Linux®) Android™, and Anaconda®)
Hardware acceleration	• Intel® x86-64 architecture (CPUs), Intel integrated and discrete GPUs, Intel NPUs • Arm® processors (CPUs) since February 2025	• Qualcomm Snapdragon hardware, including CPUs, Qualcomm® Adreno® GPUs and Qualcomm® Hexagon™ NPUs • Snapdragon SoCs built on Arm-based architectures • Windows and Arm CPU-exclusive agreement ended in December 2024
Model handling	• Model downloader (Open Model Zoo) • Model converter (OpenVINO intermediate representation format) • Model quantizer • Neural Network Compression Framework[7] (NNCF), training and post-training algorithms for optimizing inference in OpenVINO • Optimum Intel CLI	Conversion tools for AI models and frameworks (including TensorFlow™, PyTorch®, and the ONNX® model format)
Deployment	Flexible deployment—write once, deploy anywhere, on device and with cloud AI	Deployment on device or in the cloud with AI hub
Licensing	Free for commercial use with Apache® License 2.0	Part of Qualcomm's AI Stack licensing, automatic upon SDK download, exact terms depend on use cases
Security	• Model encryption with third-party tools • OpenVINO Security Add-on (OVSA) for access control • Datumaro for encryption of computer vision datasets	Relies on hardware-level security features in the Snapdragon security platform
Other	Converged neural network (CNN), image classification, object detection, and face recognition	Doesn't have the same history of apps and tools or depth of experience for the developer community

Prowess Consulting's Research Approach

We conducted extensive research and testing to evaluate the AI development process in the Intel OpenVINO toolkit versus Qualcomm AI Engine Direct SDK. Our focus was on optimizing LLMs with Intel Core Ultra processors and Qualcomm Snapdragon X Elite processors on Windows laptops and PCs, comparing model conversion, inference, and deployment. To represent a standard GenAI workload, we selected Llama 3.2-3B to develop a chatbot. Llama is one of the most widely adopted open-source models.[8]

Our development environment for the OpenVINO toolkit is shown in Figure 1. When we configured the OpenVINO Runtime environment, the OpenVINO model server build could not be completed with the latest version of Python® (3.13), so we reverted to Python version 3.10 and no further dependency issues were triggered. We also found that the OpenVINO toolkit documentation pointed to older tutorial versions, which involved more Python code than was needed with the OpenVINO 2025.1 toolkit, which provides a streamlined approach.

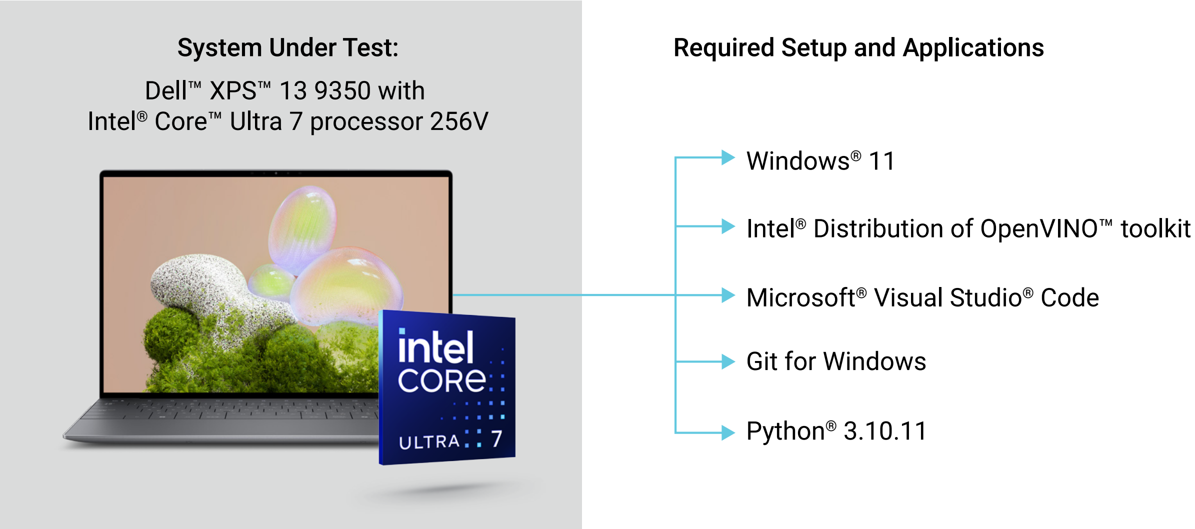
System Under Test:
Dell™ XPS™ 13 9350 with
Intel® Core™ Ultra 7 processor 256V
intel CORE ULTRA 7
Required Setup and Applications
Windows® 11
Intel® Distribution of OpenVINO™ toolkit
Microsoft® Visual Studio® Code
Git for Windows
Python® 3.10.11

Figure 1 | AI PC development environment: the Intel® OpenVINO™ toolkit

A Hugging Face account is required to create login tokens and access the Llama3.2-3B models. Hugging Face is an open-source community with more than 300,000 models and datasets; it serves a similar role to GitHub, but for the ML ecosystem. We used the Hugging Face command-line interface (CLI) to download Llama 3.2–3B from the Hugging Face model repository to the Intel OpenVINO toolkit on the Dell AI PC.

Our test environment for the Dell XPS 13 9345 laptop with a Qualcomm Snapdragon X Elite X1E-80-100 processor, shown in Figure 2, involved a similar setup as the Intel configuration, with a few notable differences. We used Qualcomm AI Engine Direct SDK—often called QNN in the software documentation and source code—in addition to the Qualcomm AI Hub package, which is required for on-device model optimization and deployment. Qualcomm AI Hub documentation is tailored for small language models (SLMs), which made working with an LLM more challenging. Small language models typically contain fewer than 1 billion parameters and require less computational power, enabling deployment on resource-constrained hardware.[9]

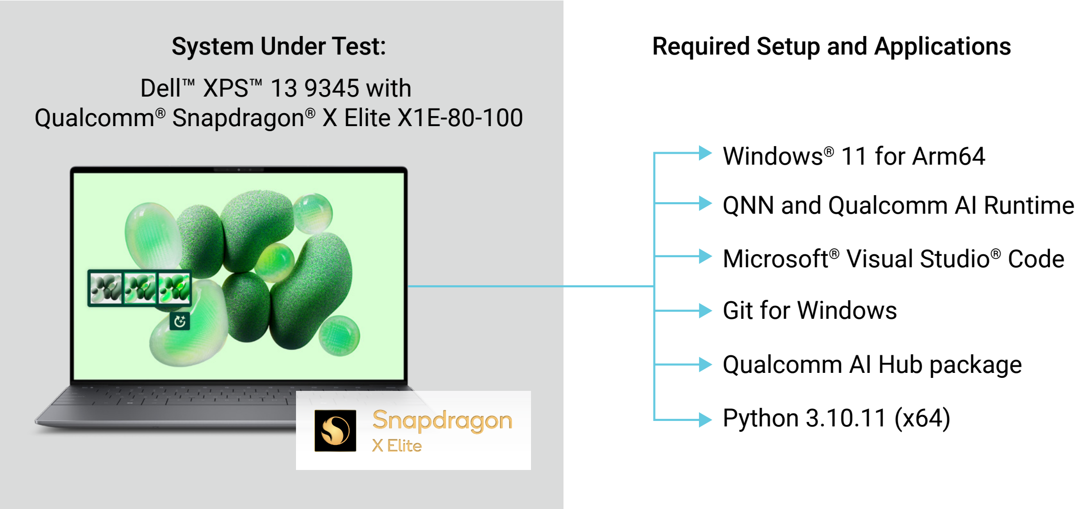

Figure 2 | AI PC development environment: Qualcomm® AI Engine Direct SDK

The Qualcomm AI Hub source-model preparation process required tracing the LLM, recording the operations it performed on representative data and converting those statistics into a static computation graph. This was necessary in order to save the model as TorchScript and to then export it to ONNX®, or Open Neural Network Exchange, an open-source format for ML models that enables models trained in one format to be used in other frameworks and environments. These steps proved difficult and required custom Python code to complete.

We found the Intel OpenVINO toolkit's runtime to be exponentially easier to work with than the Qualcomm AI Engine Direct SDK, especially for LLMs. With Intel OpenVINO toolkit, a developer can download very large models, export to the OpenVINO runtime, perform quantization to INT8 and INT4, and target the desired hardware to be run against (CPU, GPU, and NPU). Much of this process has been streamlined and can be executed via a CLI with minimal programming. While the Qualcomm AI Engine Direct SDK provides sufficient features to run full AI pipelines, some tools are still in beta, and others have limited capabilities—especially when manipulating LLMs.

Comparison of Intel® OpenVINO™ Toolkit and QNN Development Results

To highlight the differences between the AI PC development scenarios, we documented the implementation steps and development tasks to get the model operational (see the **Appendix**) and compiled and executed the code in the Intel OpenVINO toolkit and QNN runtimes. We put these tools to the test and rated the challenges of using the Qualcomm AI Engine Direct SDK compared to Intel OpenVINO toolkit.

Intel® OpenVINO™ Toolkit Pipeline Overview

Configure the OpenVINO toolkit environment.

Download the Hugging Face® model.

Convert the Hugging Face models to FP16, INT8, INT4, and NPU-specific INT4.

Run inference on each model, including the original Hugging Face PyTorch® model.

Benchmark each model by targeting the CPU (by default), GPU and NPU.

LLM Build with the Intel OpenVINO Toolkit

With Intel OpenVINO toolkit, we were able to run optimized versions of the meta-llama/Llama 3.2–3B models locally on the Dell XPS 13 9350 laptop with the Intel Core Ultra 7 processor 256V running Windows 11 without relying on cloud services. Llama 3.2–3B, developed by Meta AI, is an open-source text-in/text-out model that is free for commercial use. It's a pretrained transformer model built with data from billions of inputs from the internet, with the training data completed in 2023.

Intel has partnered with Hugging Face to advance open AI development, offering tools like Optimum Intel to optimize models for Intel hardware. We used the Optimum Intel CLI to simplify weight compression by converting the model for Intel OpenVINO toolkit from high-precision to FP16 (16-bit floating point) format, optimizing inference with minimal accuracy degradation. We applied quantization to reduce precision to fixed-point integers—8-bit (INT8) and 4-bit (INT4)—to reduce model size and increase execution efficiency. Quantization at this level might involve accuracy tradeoffs.

Once we had validated the models, we configured an Intel GenAI benchmarking tool to run against the original Hugging Face PyTorch® model and all the converted Intel OpenVINO models. We were able to benchmark each model by targeting the CPU (by default), the GPU, and the NPU. For the NPU hardware accelerator, we used NPU-specific INT4 for quantized inference.

In order to optimize the LLM running locally on the AI PC, we used the Optimum Intel CLI to convert the model to the Intel OpenVINO IR format and run it on the Intel architecture. With the Optimum Intel interface, 8-bit weight compression of models over 1 billion parameters is enabled by default. We also used it to convert the LLM to INT4 quantization for optimal performance on the NPU.

We used Intel's extension for the open-source, deep learning (DL) framework PyTorch and the OpenVINO toolkit to help optimize inference on Intel hardware (including CPUs and GPUs). We were able to run inference on each model, including the original Hugging Face PyTorch model.

Overall, we found a streamlined process using the Intel OpenVINO toolkit 2025.1 runtime. The steps for model conversion were straightforward, with minimal programming required for the Hugging Face and Optimum Intel CLI tools. Despite a few challenges with the Python code and data shapes, we were able to perform quantization and run inference. See Table 2 for more information about Prowess Consulting's evaluation of the Intel OpenVINO toolkit.

Roadblocks with Qualcomm

When we attempted to download the Llama 3.2–3B language model from the Hugging Face repository and run it locally without cloud services on the Dell XPS 13 9345 laptop with a Qualcomm Snapdragon X Elite X1E-80-100 processor, we were able to accomplish some tasks, but we ran into roadblocks on others.

We faced some challenges in our development environment setup, in part because some of the AI developer tools for AI-enhanced PCs with Copilot and Snapdragon X Elite processors are still not fully optimized for Arm architecture. At the time of our research, Python for Arm, which now runs natively in Windows on Arm, does not currently support building or installing the required Qualcomm AI Hub dependencies.

As a workaround, we were able to use an x64 version of Python, supported by x86-64 emulation on Windows 11, which is slower and less compatible. (Python 3.11.0, released in October 2022, added support for Windows on Arm.)

All Hugging Face Llama 3.2 models are based on the PyTorch format, an open-source ML framework developed by Meta AI. Most models are compatible with multiple frameworks and interoperability tools, such as the ONNX format and TensorFlow™ framework.

Qualcomm® AI Engine Direct SDK Pipeline Overview

Configure the QNN environment.

Download the Hugging Face® model.

Run Python® code to trace the model to TorchScript.

Convert the TorchScript model to an ONNX® format.

Compile the ONNX model through the Qualcomm AI Hub.

Attempt INT8 quantization to reduce model size.

We converted the PyTorch model into the TorchScript format—an intermediate representation that supports a subset of Python—to enable execution in high-performance environments, such as neural networks, without relying on standard Python dependencies.

Several workarounds were required to successfully convert the TorchScript model to ONNX format and generate a valid ONNX model file. We compiled the ONNX model through the Qualcomm AI Hub, targeting the Snapdragon X Elite compute reference device (CRD). Compiling the code through the AI Hub required refactoring the model into separate .onnx and .data files to work around the 2 GB message size limitation of Google Protocol Buffers (Protobuf™).

To compile the model for inference to hardware-accelerated platforms, we needed to generate separate ONNX and associated data files.

- The ONNX file encodes the model's computation graph—including layers, data flow, and tensor dimensions.
- The associated data files store the trained weights and numerical tensors needed for runtime inference.

Maintaining a separation between the computation graph (ONNX) and the model's parameters—weights and biases—enables modular updates to either component without retraining or re-exporting the entire model. It also contributes to efficient file size management, as the ONNX file is considerably smaller than the data files, in addition to meeting platform requirements such as memory usage, storage size, and power consumption for compiler optimization.

AI PCs such as the Dell XPS 13 9345 laptop with the Qualcomm Snapdragon X Elite X1E-80-100 processor are designed to accelerate quantized models. The enhanced AI PCs, combined with hardware-specific software tools, can make the models faster and more efficient by supporting the compressed model size, speeding up tensor computation, and using less memory.

Developed by Google and released as an open-source project in 2008, Protobuf™ data interchange format is a language- and platform-neutral mechanism for serializing and deserializing complex data structures (objects) in a transmissible format (bytes) for persistence and storage.

We were not able to perform quantization on the LLM using the Qualcomm tools and the Snapdragon X Elite Hexagon NPU, which is designed to optimize inference on 8-bit integer (INT8) quantized models. Despite significant time spent on the process, we consistently ran into shape errors and weight issues and were unable to achieve successful quantization. Several failed attempts returned "Internal quantizer" errors. We reached out to Qualcomm, and the company confirmed a software bug and said they have plans to fix it. (At the time of our research, AI Hub Quantization was still in beta.)[10]

We could build the LLM with Qualcomm's Gen AI Inference Extensions (GENIE), designed for AI acceleration on the NPU. In this case, the model was already optimized, which bypassed a large portion of our testing. Qualcomm's GENIE is integrated with the Qualcomm AI Engine Direct SDK. See Table 3 for more on Prowess Consulting's evaluation of the Qualcomm AI Engine Direct SDK and the Intel OpenVINO toolkit.

Table 3 | Rating the AI PC developer experience: The Intel® OpenVINO™ toolkit versus the Qualcomm® AI Engine Direct SDK (five-star rating system: 1 [poor] to 5 [excellent])

Evaluation Criteria	Intel® OpenVINO™	Qualcomm® AI Engine Direct SDK	Notes
Target hardware	★★★★★	★★★☆☆	Less availability of Qualcomm® Snapdragon® processor–based systems compared to Intel®-based systems; requires Windows® on Arm®
Platform compatibility	★★★★☆	★★★☆☆	Intel® OpenVINO™ toolkit model does not complete a build with Python® 3.13 (the latest version); reverted to Python 3.10. In Qualcomm, Python® for Arm is native but fails to build the required QNN dependencies; must run Python 64-bit
Features	★★★★★	★★★☆☆	Qualcomm provides enough features to run full AI pipelines, but some are still in beta, and others have limited capabilities, especially when manipulating LLMs
Model conversion	★★★★★	★☆☆☆☆	Optimum Intel simplifies model conversion with a single CLI command. Qualcomm required advanced custom Python code to convert from PyTorch® to TorchScript and ultimately to the ONNX® model format
Quantization	★★★★★	Failed	Converted Hugging Face® Models to FP16, INT8, INT4, and NPU-specific INT4 in OpenVINO. Internal quantizer error in Qualcomm (Qualcomm confirmed this is a bug on the AI Hub side and will deliver a fix)
Inference	★★★★☆	★★☆☆☆	Only a few lines of Python code needed for chatbot inference in OpenVINO. Able to run basic inference in Qualcomm but failed to submit an inference job to the AI Hub; "object is not subscriptable" error
Performance/benchmarking	★★★★☆	N/A	Used OpenVINO GenAI benchmarking tool to provide latency performance data against all models tested, including the original PyTorch model. N/A for Qualcomm given the quantization failure
Ease of use	★★★★★	★★☆☆☆	Tracing the model (required by Qualcomm's AI Hub) proved difficult, resolved with custom Python code. Compiling the model through the AI Hub requires refactoring the model to address a Google Protobuf™ 2 GB size limitation
Documentation	★★★★☆	★★★☆☆	Qualcomm's AI Hub documentation is primarily for SLMs; complex code required for interacting with LLMs is not provided
Community support	★★★★★	★★☆☆☆	Vast open-source community for OpenVINO, including support and examples from Stack Overflow, GitHub, Reddit, and Quora. There is not a large community presence for the Qualcomm AI Hub environment, requiring developers to reach out to Qualcomm directly.
Technical proficiency required	★★★★★	★★☆☆☆	Manipulating LLMs within the QNN environment is not for beginner developers; advanced Python programming techniques are required
Overall score	★★★★★	★★☆☆☆	Intel OpenVINO 2025.1 provides a more streamlined approach. Qualcomm AI Engine Direct SDK works as a proof of concept for Arm-related model manipulation, but is not practical in real-world scenarios, especially when interacting with LLMs

Making Models Smaller

Why does quantization matter for LLMs in ML? Quantization is a technique to convert the model's weights and activations from higher precision to lower precision (for example, from floating point to integer) during the training or post-training phase with minimal accuracy loss. This smaller numerical footprint can speed up inference. It also reduces power consumption, memory bandwidth, and cost—allowing hardware such as CPUs and GPUs to handle models whose parameters increasingly represent billions of values.

Common techniques for performing post-training quantization at different stages of the model deployment pipeline include static, dynamic, and quantization-aware training. In static quantization, which is used after model training and before deployment, quantization parameters—such as optimal scaling factors for each layer—are determined by running the model through a calibration phase using a representative dataset before inference. Static quantization is often used when deploying models on edge devices.

In dynamic quantization, only the model's weights are reduced to lower-precision numbers (that is, integers) before inference. The activations—the floating-point outputs from layers as data flows through the network—are quantized on the fly during inference. Dynamic quantization is often applied to pretrained models, converting them into quantized versions prior to deployment. For applications where any accuracy loss is unacceptable, such as medical imaging, quantization-aware training (QAT) is sometimes used. QAT simulates quantization during training, using the quantize/dequantize (QDQ) format, by inserting QDQ nodes around tensors to mimic quantization. This approach is supported by the ONNX format, PyTorch framework, and NVIDIA® TensorRT™ SDK, and it allows flexibility by enabling the neural network to learn how to operate during model training and inference.

Post-training quantization reduces a model's computation and memory footprint by converting the weights and activations of an LLM from higher precision formats (FP32, FP16) to lower precision formats (INT8, INT4). This transformation is performed after training and does not require access to the original training data or retraining the model. This method can reduce model size without significant accuracy loss and improve inference efficiency. The steps to convert a pre-trained floating-point model to a quantized version with lower precision are shown in Figure 3.

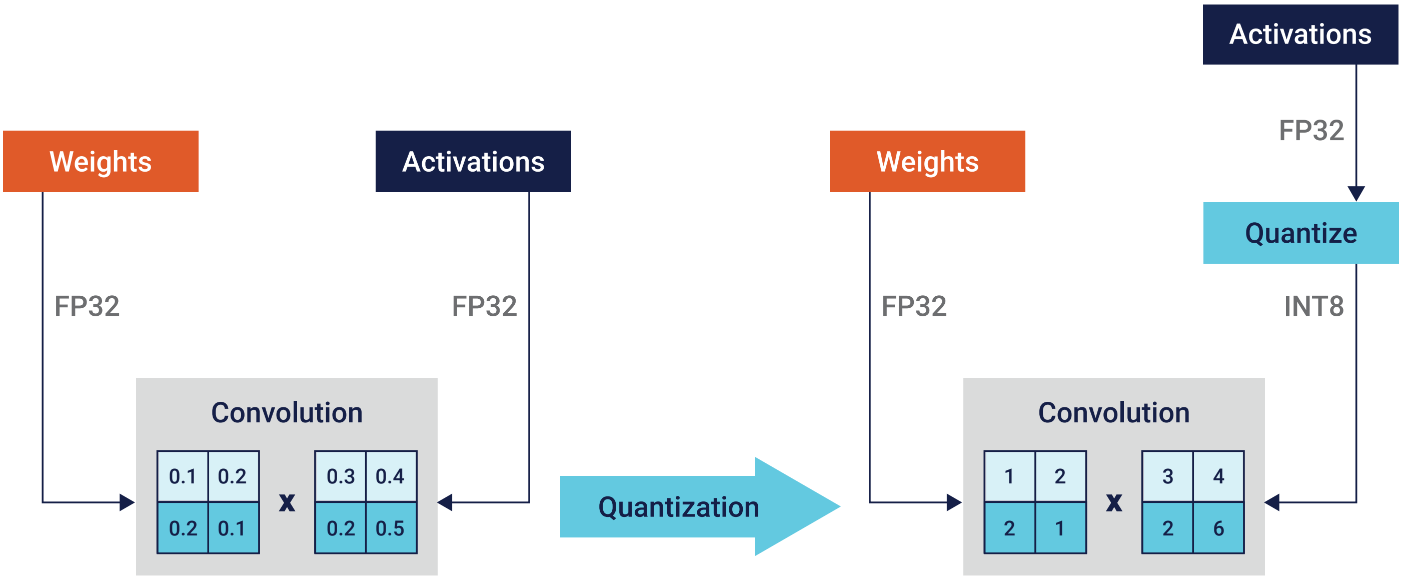

Figure 3 | This diagram illustrates the workflow for post-training quantization. In this example, the process converts an LLM's weights and activations from a higher precision format (FP32) to a lower precision format (INT8), reducing the model size and improving inference efficiency on AI acceleration hardware with limited impact on accuracy, depending on the model.[11]

It is important to ensure that the target hardware supports the chosen quantization method. Some CPUs and GPUs are optimized for FP64, FP32, or lower precisions such as FP16 and BFloat16 (BF16). With AI and ML, more hardware platforms are being optimized to support FP16 (half-precision) or integer-level quantization operations. Quantization techniques are particularly critical for LLMs. Beyond improving inference speed, they can significantly reduce infrastructure demands and power consumption, making them vital for scalable and efficient model deployment.

Key Findings

To help developers build AI models and applications locally on the best architecture, we conducted a comparative evaluation of the Intel OpenVINO toolkit and the Qualcomm AI Engine Direct SDK on Dell XPS 13 AI PCs equipped with dedicated NPUs and integrated AI accelerators. Our analysis focused on real-world inference workloads and end-to-end model-development pipelines. The Intel OpenVINO toolkit earned higher scores in target hardware support, platform compatibility, features (model optimization, quantization, and deployment tooling), ease of use, quality and depth of documentation, level of community and vendor support, licensing models (open source), and required technical proficiency.

Based on our testing and research, we found the Intel OpenVINO toolkit to be the better choice for most users because it offers:

- **AI-enhanced performance and efficiency**: It is built to fully leverage Intel Core Ultra processors with a dedicated NPU, alongside GPU and CPU acceleration.
- **Streamlined workflow**: It provides integrated tools for model conversion, quantization, and deployment.
- **Broad compatibility**: It delivers a mature development platform that is well supported on Intel Core Ultra processors and capable of running smaller AI tasks on previous-generation Intel processors.
- **Ease of use for new developers**: It provides a streamlined process with Optimum Intel and related tools, with no heavy coding required.
- **Security and privacy**: On-device processing and local AI model inferencing reduce the risks associated with sensitive data in the cloud.
- **Strong community backing**: It accommodates open-source solutions with active developer support.

The Qualcomm AI Engine Direct SDK is designed to offer the best tooling for AI optimization on Qualcomm Snapdragon hardware. However, Prowess Consulting's research and testing found that using LLMs meant finding numerous workarounds for the Qualcomm QNN SDK, because of its incomplete functionality. (At the time of this research, AI Hub Quantization was still in beta and had known bugs.) While a smaller model could potentially complete the quantization process, the external weights and QDQ serialization bug prevent that on large models.

In today's competitive market, choosing the right LLMs for your applications isn't just important—it's a strategic advantage. The Intel OpenVINO toolkit stands out with broader deployment across Intel-based AI PCs, giving developers the freedom and flexibility to build without limitations.

This is especially true for developers targeting a large market of deployed systems and supporting a broad range of models. Released in 2018, Intel OpenVINO began expanding its capabilities to include voice and NLP in 2019. Since the December 2023 release of Intel Core Ultra Series 1 processors, which introduced a dedicated NPU, the Intel OpenVINO toolkit has focused on the demand for AI acceleration and faster inference of LLMs.

As the transition to AI and ML expands across industries, developers will have an increasingly wide array of tools and hardware to choose from. AI PCs equipped with hardware-specific SDKs like the Intel OpenVINO toolkit are poised to set the standard for AI innovation and drive the success of projects—today and in the years ahead.

Appendix

The following pipelines outline the complete workflows for developing and optimizing AI applications powered by LLMs using the Intel OpenVINO toolkit and the Qualcomm AI Engine Direct SDK.

Intel OpenVINO toolkit pipeline:

1. Download and install the following applications:
   a. Python 3.10.11 (64-bit)
      i. **www.python.org/downloads/release/python-31011/**
   b. Microsoft® Visual Studio® Code
      i. **https://code.visualstudio.com/download**
   c. Git for Windows
      i. **https://gitforwindows.org/**
2. Configure a Python virtual environment.
3. Install required dependencies:
   a. OpenVINO 2025.1.0
   b. Neural Network Compression Framework (NNCF)
   c. Optimum Intel
   d. OpenVINO tokenizers
   e. OpenVINO Genai
   f. Diffusers
   g. Librosa

    h. Hugging Face Hub
    i. Auto-GPTQ

4. Log in to **https://huggingface.co/login**.
5. Gain access to the model under test at **https://huggingface.co/meta-llama/Llama-3.2-3B**.
6. Create a Hugging Face access token.
7. Use the access token with the Hugging Face CLI.
8. Download the meta-Llama/Llama-3.2-3B model with the Hugging Face CLI.
9. Convert the model to INT8, INT4, and INT4 for the NPU.
10. Launch Visual Studio Code and run inference on the model using the following example code:

```
Import openvino_genai as ov_genai
model_path = “metallama_INT8”
pipe=ov.genai.LLMPipeline(model_path, “GPU) # Change to NPU or leave blank to run on CPU.
print(pipe.generate(“What is generative AI?, max_new_token=100))
```

11. Clone the GenAI LLM benchmark with Git.
    a. **https://github.com/openvinotoolkit/openvino.genai**
12. Benchmark each model, including the original, for comparison using the following example code:

```
python .\benchmark.py -m C:\Users\<UserName>\Documents\IntelOpenVINO\.openvino\metallama_huggingface -p
“What is generative AI?” -n 2 -f pt
```

QNN pipeline:

1. Download and install the following applications:
    a. Python 3.10.11 (64-bit)
        i. **www.python.org/downloads/release/python-31011/**
    b. Microsoft Visual Studio Code (Arm64)
        i. **https://code.visualstudio.com/download**
    c. Git for Windows
        i. **https://gitforwindows.org/**
2. Configure Qualcomm AI Runtime SDK version 2.32.6.250402.
    a. **https://qpm.qualcomm.com/#/main/tools/details/Qualcomm_AI_Runtime_Community**
3. Install the qai_hub_models AI Hub package.
    a. **https://pypi.org/project/qai-hub/**
4. Install the Hugging Face CLI.
5. Install required Python dependencies:
    a. Torch
    b. Transformers
    c. ONNX
    d. NumPy®
6. Download the meta-Llama/Llama-3.2-3B model with the Hugging Face CLI.
7. Launch Visual Studio Code.

Note: The remainder of the QNN pipeline is executed with custom Python code not provided in any Qualcomm AI Hub tutorials.

8. Trace the Hugging Face model to TorchScript.
    a. This requires the WrappedLlamaModel(torch.nn.Module) class in order to prevent structured output issues and to extract logits.
9. Load the model.
10. Wrap the model with **WrappedLlamaModel(torch_model)**.
11. Prepare the input tensor with the **torch.randint()** function.
12. Convert the model with the **torch.jit.trace()** function.
    a. This is required to convert to ONNX format.
13. Export to ONNX format and save using external data formats:

```
onnx.save_model(onnx_model, external.onnx, save_as_external_data=True)
```

    a. This resolves the 2 GB Protobuf size limitation error when compiling to the AI Hub.

14. Create a directory structure ending with .onnx (required by AI Hub).
15. Submit the model for compiling using the **hub.submit_compile_job()** function.
16. Load the tokenizer using the ***AutoTokenizer.from_pretrained()*** function.

17. Convert to NumPy arrays.
18. Convert the data into the dictionary format expected by AI Hub.
19. Submit the quantization job to AI Hub using the **hub.submit_quantize_job()** function.
20. The quantization job submission will fail on Qualcomm's AI Hub server with an internal quantizer error.

Endnotes

[1] Gartner. "**Gartner Predicts 30% of Generative AI Projects Will Be Abandoned After Proof of Concept by end of 2025**." July 2024.

[2] Intel. "**AI Coming to the PC at Scale**." May 2023.

[3] Microsoft Blog. "**Introducing CoPilot+ PCs**." May 2024.

[4] Gartner. "**Gartner Forecasts Worldwide GenAI Spending to Reach $644 Billion in 2025**." March 2025.

[5] Andrew Hewitt. "**Forrester: Preparing for the Era of the AI PC**." *Computer Weekly*. May 2024.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. "**Attention Is All You Need**." Cornell University. June 2017.

[7] GitHub. "**Neural Network Compression Framework (NNCF)**." Accessed June 2025.

[8] Meta. "**The Future of AI: Built with Llama**." December 2024.

[9] Chien Van Nguyan, Xuan Shen, Ryan Aponte, et al. "**A Survey of Small Language Models**." October 2024.

[10] Qualcomm. "**Overview of Qualcomm AI Hub: Quantization (Beta)**." Accessed May 2025.

[11] Intel. "**Post-training Quantization**." Accessed June 2025.

Legal Notices and Disclaimers

The analysis in this document was done by Prowess Consulting and commissioned by Intel.

Results have been simulated and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

0725/240157