PROWESS

Technical Research Report

Which Toolkit Provides the Best Optimization for Large Language Models?

Prowess Consulting, commissioned by Intel, conducted research and testing to determine whether the Intel® Distribution of OpenVINO™ toolkit or the Lemonade Server SDK can help build the best pipeline for large language model (LLM) deployment on Intel® processors and AMD Ryzen™ processors.

Executive Summary

Developers recognize the critical need for efficient AI solutions across diverse computing environments. Hardware-specific SDKs are designed to enable seamless integration with on-device hardware, enhancing model execution and accelerating neural network inference to improve a 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 a hybrid architecture or on devices powered by AMD Ryzen™ processors. Prowess Consulting conducted testing to help shed light on those considerations and help developers choose the most effective in-device large language model (LLM) toolkit.

We tested the Intel® Distribution of OpenVINO™ toolkit on a Dell™ XPS™ 13 AI PC (with an Intel Core Ultra 7 processor 256V) and the Lemonade Server SDK on an ASUS ZenBook® 14 (with an AMD Ryzen AI 7 350 processor) to help developers find the solution that best suits their workflows. Based on our testing, the Intel Distribution of OpenVINO toolkit earned higher scores in target hardware support, platform compatibility, model conversion, inference, and community support. Table 1 highlights our findings.

Table 1 | SDK scorecard: The Intel® Distribution of OpenVINO™ toolkit versus the Lemonade Server SDK (using a five-star rating system, from 1 [poor] to 5 [excellent])

Software Development Toolkits	Target Hardware	Platform Compatibility	Model Conversion	Inference	Community Support
Intel® Distribution of OpenVINO™ toolkit	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★★
Lemonade Server SDK	★★★★☆	★★★☆☆	★★☆☆☆	★★★★☆	★★★☆☆

Primary drivers influencing our SDK ratings in Table 1 include:

- The Intel Distribution of OpenVINO toolkit enables a straightforward, repeatable pipeline: convert the model to the Intel Distribution of OpenVINO toolkit, perform quantization (for example, INT4, INT8, and the tested NPU-specific INT4 quantization), and then package the optimized build with standard dependencies into a container that runs successfully without special workarounds.
- Lemonade Server SDK provides an easy graphical user interface (GUI) on-ramp, but with inconsistent model reliability and dev/Python® Package Index (PyPI) roadblocks (including a gated neural processing unit [NPU] package), stalled and unsupported quantization, and extra containerization workarounds, which contribute to reduced production readiness.

The AI Technology Challenge

Companies are investing in AI development with various degrees of success. More industries are exploring the use of LLMs to train smaller models and utilize retrieval-augmented generation (RAG) for specific tasks. As the push to integrate AI capabilities into enterprise systems fundamentally redefines line-of-business (LOB) applications, enterprise development teams can leverage AI-enhanced hardware and software to automate tasks and enhance their development processes.

Initially targeted at commercial markets, AI PCs are high-performance computers designed with neural processing units (NPUs), along with CPUs and GPUs, which enable machine learning (ML) and other AI functions to be performed locally on the device. The first Windows® AI PCs were previewed at the Microsoft Build 2023 conference.[1] These AI PCs were equipped with 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 (AMD Ryzen processor–based PCs) from OEM partners such as ASUS, HP, and Lenovo.[2]

By 2028, most consumer PCs are expected to support AI acceleration hardware, according to Gartner research.[3] These AI-enhanced systems offer developers improved performance, reduced latency, and enhanced data privacy and security. Additional benefits include longer battery life and lower costs compared to cloud-based AI processing.[4] 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 (for example, floating point to integer) during or after training. This smaller numerical footprint can speed up inference, though reduced precision might affect accuracy. With hardware-optimized tools, software developers can make use of an AI PC's hardware to enhance performance and resource efficiency.

AI PCs can offer developers improved performance, reduced latency, enhanced data privacy and security, and lower costs compared to cloud-based AI processing.[5] These systems also differ from standard PCs, which depend on cloud-based infrastructure to run AI applications due to the high memory and computational requirements of AI workloads, particularly those involving LLMs.

As AI PCs become more prevalent, developers face hardware-optimization choices across CPU, GPU, and NPU targets. For example, they might build on devices powered by Intel Core Ultra processors or on systems powered by AMD Ryzen processors, selecting the system that is most appropriate for the workload.

To help AI developers determine the right platform for their needs, Prowess Consulting tested the Intel Distribution of OpenVINO toolkit on a Dell XPS 13 AI PC (powered by an Intel Core Ultra 7 processor 256V) and the Lemonade Server SDK on an ASUS ZenBook 14 (powered by an AMD Ryzen AI 7 350 processor) to see which tool offers the most significant benefits. Specifically, we evaluated the target hardware, platform compatibility, features, ease of use, documentation, community support, trade-offs between open source and proprietary solutions, required skill sets, and costs.

Working with Local LLMs

Companies like Microsoft, Intel, Meta, and others are working to create open platforms for AI development. 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 model for natural language processing (NLP),[6] 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. Additionally, local customization of models for specific use cases or domains, such as healthcare, legal, finance, and customer service, can improve performance through faster iterations (without 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.

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 might need technical experience to optimize model performance and avoid vulnerabilities that could expose sensitive data. At the same time, tools like the Intel Distribution of OpenVINO toolkit 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 AMD collaborate with Hugging Face®, an open-source repository, to ensure that open-source models are accessible and safe to use in their environments.

While Lemonade Server SDK is also open source, its NPU acceleration can depend on AMD-gated components—the npu-llm-artifacts.zip package is in early access and requires a registered account to acquire it—and reliability can vary by model. See Table 2 for more information on the features and differences between the Intel Distribution of OpenVINO toolkit and the Lemonade Server SDK.

Research Approach

We conducted extensive research and testing of the Intel Distribution of OpenVINO toolkit and the Lemonade Server SDK on Windows systems, exercising model handling, inference behavior, NPU enablement, quantization attempts, and containerized deployment. Each toolkit's pipeline included model conversion, quantization (where supported), and local inference validation. We evaluated both SDKs with the goal of creating a chatbot application that leverages the compute devices on Intel and AMD systems. We attempted to use the NPU to test its functionality and capabilities with LLMs. While both toolkits support dev workflows (the entire development process), Lemonade Server SDK also offers a GUI pipeline path. This study focuses on the dev pipelines (activities performed using the SDK).

Testing Workflows

For OpenVINO, we converted the Hugging Face Llama-3.2-3B model to OpenVINO Intermediate Representation (IR) and compressed it to INT8, INT4, and an NPU-specific INT4 for our test. We then packaged the optimized model into a Docker® image based on python:3.10-slim, installed standard requirements, and ran the application. Our deployment was successful.

For Lemonade Server SDK, we evaluated both the GUI (.exe) and dev (PyPI/Miniforge) flows on an ASUS ZenBook 14 with an AMD Ryzen AI 7 350 processor and an AMD Radeon™ 860M GPU. We pulled several models (Llama-3.2-3B, DeepSeek-R1, and Gpt-oss-20b, among others) via the Lemonade Server SDK's model management and from Hugging Face. The container was successfully built, but we were unable to fully validate the pipeline due to model load failures and gated NPU components.

Lemonade SDK NPU Enablement

An NPU-enabled Llama-3.2-3B model ran via the Lemonade GUI. However, we could not enable NPU inference in the developer (PyPI/Miniforge) workflow because the required ONNX® Runtime GenAI component wasn't available in that environment. The additional NPU libraries distributed through AMD's early-access program were not used for this test, as AMD requires an internal technical contact to grant that access.

Quantization with AMD Quark, a comprehensive cross-platform toolkit designed to simplify and enhance the quantization of deep learning (DL) models, stalled on the CPU because ROCm-based GPU acceleration wasn't supported on the test's AMD Radeon 860M GPU. Windows Subsystem for Linux® (WSL) for AMD Quark is recommended in the Quark documentation and was carried out as part of the test, but it crashed in multiple runs. Dockerizing an application with a Lemonade Server SDK model required avoiding the requirements.txt file due to dependency issues and pulling the model during image build. After these workarounds, we successfully deployed a simple chatbot with Lemonade Server SDK.

Lemonade Server SDK Issues and Deployment Workarounds
Where applicable, we downloaded models from Hugging Face. In the Lemonade Server SDK (dev/PyPI) flow, we installed the Hugging Face command-line interface (CLI) and pulled models for local testing. For the Intel Distribution of OpenVINO toolkit deployment, we downloaded the original Hugging Face PyTorch® Llama-3.2-3B model and then performed quantization to INT8 (with the ability to target INT4 and NPU-specific INT4) before packaging the optimized model for inference.

Testing on an ASUS ZenBook 14 (powered by an AMD Ryzen AI 7 350 processor) used both the Lemonade Server GUI and the Lemonade Server Dev (PyPI/Miniforge) on Windows 11 with Python 3.10.11. The GUI allowed model downloads via Model Management and ran an NPU variant of Llama-3.2-3B, while enabling the NPU in the developer workflow (PyPI/Miniforge) was blocked due to the need for an ONNX Runtime GenAI library and early-access NPU artifacts that weren't available.

After setting up AMD Quark for post-training quantization, the CPU runs stalled, and GPU acceleration wasn't available because ROCm doesn't support the tested AMD Radeon 860M GPU.[7] Attempts via WSL were unstable and crashed under load. Models pulled through the Lemonade Server SDK lacked a config.json file. This file typically serves as a configuration blueprint that guides the quantization process, with a model type for the Quark PTQ workflow. The lack of this file necessitated a "bring-your-own-model" approach for quantization; such a manual workaround blocked standard, automated quantization flows and complicated reproducibility. For containerization, the dev setup installed lemonade-sdk and pulled the model during the Docker build to meet cache and path requirements. Deployment then succeeded after these workarounds.

Table 2 | Tools face-off: the Intel® Distribution of OpenVINO™ toolkit versus the Lemonade Server SDK

SDK	Intel® Distribution of OpenVINO™ Toolkit	Lemonade Server SDK
Platform support	• Open-source tools and high-level APIs • C/C++ for low-level integrations, Python®, and Node.js®	• Open-source project (GitHub®) with a Windows® GUI (tray app) and a dev/PyPI CLI flow (Miniforge/conda) • Dev server auto-launches in browser • GPU quantization was attempted via Windows Subsystem for Linux® (WSL) with ROCm on the test system; ROCm was not supported on the AMD Radeon™ 860M GPU7
Hardware acceleration	• Intel® x86-64 architecture (CPUs), Intel integrated and discrete GPUs, and Intel NPUs • Arm® processors (CPUs) since February 2025	• CPU, GPU, and NPU recipes (OGA CPU/OGA NPU/OGA Hybrid; llama.cpp) • NPU confirmed for a specific model in GUI; dev NPU blocked by a required ONNX® runtime GenAI component and NPU artifacts gated behind AMD early-access on the test system • Quantizing models pulled via Lemonade Server with AMD Quark failed because the ONNX exports lacked the required config.json/model_type metadata expected by the Quark quantize_quark.py post-training quantization workflow; CPU runs hung, and the GPU path—attempted via WSL with ROCm—was blocked because ROCm doesn't support the tested AMD Radeon 860M
Model handling	• Model downloader (Open Model Zoo) • Model converter (OpenVINO intermediate representation format) • Model quantizer • Neural Network Compression Framework (NNCF) and training and post-training algorithms for optimizing inference in OpenVINO • Optimum Intel command-line interface (CLI)	• Model management user interface (UI): Hot models, by recipe, by category; supports pulling custom models (for example, from Hugging Face®) • Several curated/custom models failed to load or crashed in GUI tests • ONNX exports pulled via Lemonade Server SDK lacked config.json/model_type required by AMD Quark PTQ; BYO-model recommended for quantization

SDK	Intel® Distribution of OpenVINO™ Toolkit	Lemonade Server SDK
Deployment	• Flexible deployment—write once, deploy anywhere, on device and with cloud AI	• GUI usage is straightforward; dev/Docker required extra steps (avoid requirements.txt due to dependency issues; pull model during image build for cache/path constraints); deployment succeeded after workarounds
Licensing	• Free for commercial use with Apache® License 2.0	• Open-source project; note that certain NPU components are gated (for example, ONNX Runtime GenAI bits/npu-llm-artifacts early-access package) and not included by default
Security	• Model encryption with third-party tools • OpenVINO Security Add-on (OVSA) for access control • Datumaro for encryption of computer vision datasets	• No dedicated security add-ons; relies on platform/toolchain components; some NPU binaries are distributed via AMD's gated/early-access channels
Other	• Convolutional neural network (CNN), image classification, object detection, and face recognition	• Default CPU chat model was fast/responsive in GUI; multiple "hot"/custom models failed to load or crashed; dev path showed dependency churn and WSL instability under quantization load; community-driven support with AMD contact required for certain NPU packages

Comparison of the Intel Distribution of OpenVINO Toolkit and the Lemonade Server SDK Development Results

To highlight practical differences between the two development scenarios, we documented the steps required to bring a local LLM online (see the **Appendix**) and then executed each toolkit's pipeline. The evaluation covered model handling, inference behavior, quantization attempts, NPU enablement, and containerized deployment, with a focus on where workflows succeeded and where they stalled.

Intel® Distribution of OpenVINO™ Toolkit Pipeline Overview

1. Configure the OpenVINO toolkit environment.
2. Convert the model to OpenVINO.
3. Quantize the model to INT8/INT4.
4. Prompt the model to validate local inference.
5. Build a Docker® image including all dependencies.
6. Deploy the image and validate on target PC.

Lemonade Server SDK Pipeline Overview

1. Configure the Lemonade Server SDK environment.
2. Download and prepare the model for Lemonade Server SDK.
3. Attempt post-training quantization with AMD Quark (INT8/INT4).
4. Prompt the model to validate local inference.
5. Build a Docker® image, including all required Lemonade Server SDK dependencies.
6. Deploy the image and validate on target PC.

LLM Build with the Intel Distribution of OpenVINO Toolkit

On a Dell XPS 13 9350 AI PC with an Intel Core Ultra 7 processor 256V running Windows 11, we packaged an optimized (quantized) Llama-3.2-3B INT8 model together with the application and dependencies, built a container, and ran the solution locally. The build, load, and run steps were completed without requiring any special workarounds beyond the standard Docker workflow, and deployment was successful.

Roadblocks with Lemonade Server SDK

On an ASUS ZenBook 14 with AMD Ryzen AI 7 350 processor, we ran both the Lemonade Server SDK GUI and Lemonade Server SDK dev (PyPI/Miniforge) on Windows 11 with Python 3.10.11. In the GUI, we obtained models through AMD Model Management tools and software. The default CPU chat model was fast and responsive, and an NPU variant of Llama-3.2-3B ran successfully, producing more detailed output but at a slower rate than the default CPU chat model. However, multiple curated or custom models failed to load or crashed during GUI testing.

In the developer workflow, the NPU route could not be validated because a required ONNX Runtime GenAI library was unavailable in the environment, and additional NPU components are restricted to AMD's early-access program. Quantization attempts using AMD Quark stalled on the CPU; the ROCm-based GPU route was unsupported on the test iGPU; and in multiple runs, the WSL environment crashed under load. Containerized deployment was achieved only after extra steps to work around dependency and model-cache constraints, including installing Lemonade Server SDK's development extras and pulling the model during the image build. At that point, deployment was successful.

In this test, the Intel Distribution of OpenVINO toolkit's path provided a straightforward package-and-run experience with INT4, INT8, and the NPU-specific model that was tested. The Lemonade Server SDK provided a quick GUI on-ramp but exhibited inconsistent model reliability and gaps in the developer path for NPU access, quantization on CPU/GPU, and containerization, requiring additional workarounds.

We found the Intel Distribution of OpenVINO toolkit's runtime to be exponentially easier to work with than the Lemonade Server SDK, especially for LLMs. With the Intel Distribution of OpenVINO toolkit, a developer can download very large models, export to the OpenVINO runtime, perform quantization to INT4 and INT8, 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. The Lemonade Server SDK, on the other hand, provides sufficient features to run full AI pipelines, but some tools are limited by restricted access to gated resources, and others have limited capabilities—especially when manipulating LLMs.

Intel now offers Intel OpenVINO GenAI, a library of the most popular generative AI model pipelines, optimized execution methods, and samples that run on top of the highly performant OpenVINO Runtime.

Table 3 | Rating the AI PC developer experience: the Intel® Distribution of OpenVINO™ toolkit versus the Lemonade Server SDK (using a five-star rating system: 1 [poor] to 5 [excellent])

Evaluation Criteria	Intel® Distribution of OpenVINO™ Toolkit	Lemonade Server SDK	Notes
Target hardware	★★★★★	★★★★☆	OpenVINO tested on a Dell™ XPS™ 13 9350 AI PC (with an Intel® Core™ Ultra 7 processor 256V); Lemonade Server SDK tested on ASUS ZenBook® 14 (with an AMD Ryzen™ AI 7 350 processor). The Lemonade Server GUI ran an NPU variant of Llama-3.2-3B; the NPU hardware was blocked by the AMD early-access program.
Platform compatibility	★★★★☆	★★★☆☆	OpenVINO runs on Windows® 11 with Docker® and Microsoft® Visual Studio® Code; Lemonade Server SDK uses Windows 11 plus a PyPI/Miniforge environment and Windows Subsystem for Linux® (WSL) environment for GPU targeting.
Features	★★★★★	★★★☆☆	OpenVINO notes focused on model packaging/deployment; Lemonade Server SDK provided model management (hot/by recipe/by category) and CLI pulls, plus a chatbot via Lemonade Server GUI. The latest OpenVINO model server can expose an OpenAI-compatible endpoint to simplify app integration.
Model conversion	★★★★★	★☆☆☆☆	Model conversion with OpenVINO was simple and straightforward. For Lemonade Server SDK, several pulled models could not be prepared for quantization due to missing configuration files, making a bring-your-own-model approach preferable.
Quantization	★★★★★	★☆☆☆☆	OpenVINO used an INT8 model. Lemonade Server SDK's CPU quantization stalled; GPU quantization was not supported on the test GPU; the WSL environment was unstable under load.
Inference	★★★★☆	★★★★☆	The OpenVINO containerized application executed successfully. In Lemonade Server SDK, the default CPU chat model responded quickly and an NPU variant of Llama-3.2-3B ran in the GUI, but multiple curated or custom models failed to load or crashed; the NPU could not be exercised in the developer workflow.
Ease of use	★★★★★	★★★☆☆	OpenVINO provided a straightforward package-and-run experience using standard tooling. The Lemonade Server SDK's GUI was easy to set up and use, though the developer path required additional environment setup and troubleshooting.
Technical proficiency required	★★★★★	★★☆☆☆	OpenVINO required familiarity with basic Python® packaging and containerization. The Lemonade Server SDK's developer workflow required comfort with conda environments, WSL configuration for GPU attempts, and dependency troubleshooting.
Overall score	★★★★★	★★★☆☆	OpenVINO achieved a clean deployment with an optimized model. The Lemonade Server SDK offered a fast GUI on-ramp but showed inconsistent model reliability, blocked NPU use in the developer path, and required additional steps to stabilize deployment. In addition, OpenVINO provides cross-hardware compatibility with a single runtime, enabling models to execute seamlessly across the CPU, GPU, and NPU. By contrast, the Lemonade Server SDK relies on multiple runtimes (LLama.cpp, ONNX, and RyzenAI-EPs), which makes it more complex for developers to navigate across different flows.

Making Models Smaller

Why does quantization matter for LLMs? Quantization is a technique to convert a 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, GPUs, and NPUs to handle models whose parameters increasingly represent billions of values.

Post-Training Quantization

Post-training quantization reduces a model's computation and memory footprint by converting the weights and activations of an LLM from higher precision formats (such as FP32 and FP16) to lower precision formats (such as INT8 and 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 loss of accuracy and can improve inference efficiency.

The optimization applied in our workflow was weight-only quantization (WOQ) as part of the OpenVINO toolkit's weight compression feature. This approach to quantization is commonly used with LLMs to reduce model size and improve efficiency by quantizing models' weights to lower precision (in this case, INT8) while keeping the activations at the original floating-point precision. The steps to convert a pre-trained floating-point model to a quantized version with lower precision are shown in Figure 1.

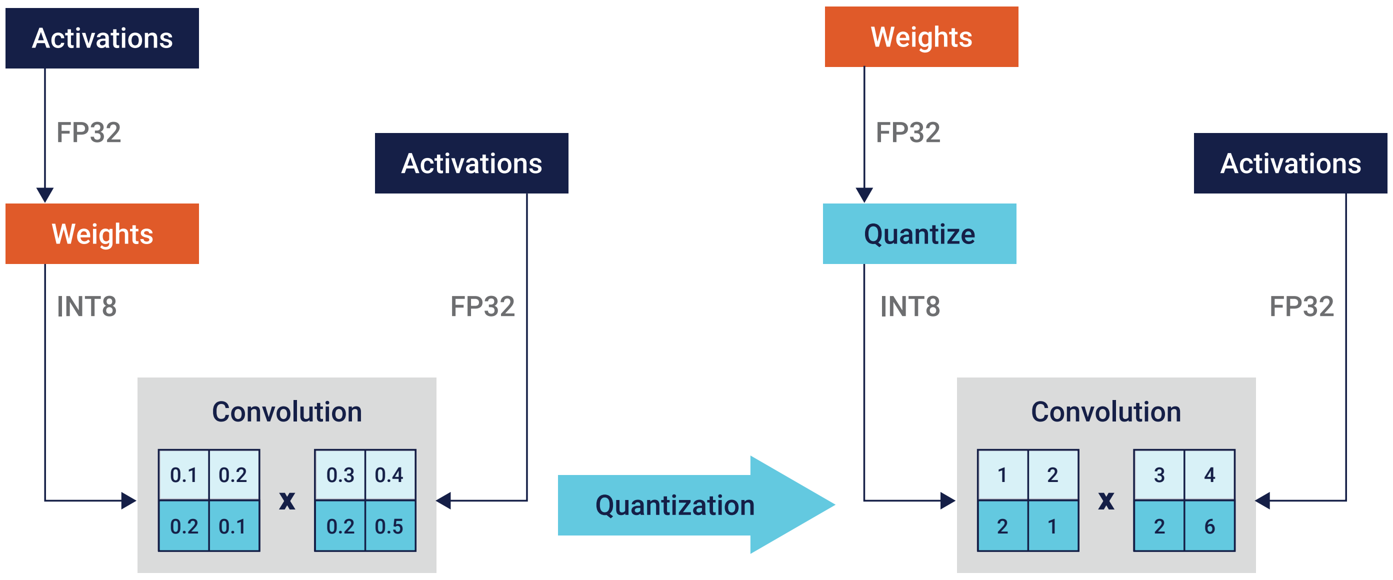

Figure 1 | This diagram illustrates the workflow for post-training quantization; in this example, the process converts an LLM's weights and activations from a higher precision format (FP32) to a lower precision format (INT8), reducing the model size and improving inference efficiency on AI acceleration hardware with limited impact on accuracy, depending on the model[9]

Hardware Compatibility and Precision Support

It is essential to verify that the target hardware supports the selected quantization method. Some CPUs and GPUs are optimized for FP64, FP32, or lower precisions such as FP16, Bfloat16 (BF16), and INT8. 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 compare on-device LLM workflows, we conducted a side-by-side evaluation of the Intel Distribution of OpenVINO toolkit (on a Dell XPS 13 9350 AI PC with an Intel Core Ultra 7 processor 256V) and the Lemonade Server SDK (on an ASUS ZenBook 14 with an AMD Ryzen AI 7 350 processor). Our assessment focused on model handling, inference behavior, quantization attempts, NPU enablement (for installing Python packages on a specific device), and containerized deployment on Windows 11. No formal performance benchmarks or cost analysis were recorded for this run.

The Intel Distribution of OpenVINO toolkit delivered a straightforward package-and-run experience. We added an (INT8-quantized) Llama-3.2-3B model to the project, containerized it with standard dependencies, and deployed a chatbot successfully with no special workarounds.

The Lemonade Server SDK's GUI provided a quick on-ramp. The default CPU chat model was fast and responsive, and an NPU variant of Llama-3.2-3B ran in the GUI (producing more detailed output but doing so slower than the default CPU model). However, multiple curated or custom models, such as Llama-3.2-3B-Instruct-NPU and Deepseek-R1-Distill-Llama-8B-CPU, among others, failed to load or crashed in the GUI. In the developer workflow, NPU use could not be validated because a required runtime component was unavailable, and additional NPU pieces are restricted to AMD early-access.

Quantization attempts stalled on the CPU, the GPU route was unsupported on the test iGPU, and, in some instances, WSL crashed under load. Containerized deployment succeeded only after extra steps (avoiding a standard requirements bundle, adjusting dependencies, and pulling the model during image build).

In this evaluation, the Intel Distribution of OpenVINO toolkit offered a repeatable deployment path with an optimized model, while the Lemonade Server SDK was well-suited for quick trials in the GUI, but required workarounds in the developer path for NPU access, quantization, and containerization on the tested hardware.

In the Prowess Consulting team's tests, the Intel Distribution of OpenVINO toolkit earned higher scores in target hardware support, platform compatibility, model conversion, inference, and community support, making it an excellent option when looking for a solution that best suits developer workflows.

Appendix

The following pipelines outline the complete workflows for developing and optimizing AI applications powered by LLMs using the Intel Distribution of OpenVINO toolkit and the Lemonade Server SDK. The objective of this test was to create a chatbot application.

Intel Distribution of OpenVINO Toolkit Pipeline Experimentation

1. Download and install the following applications:
    a. **Python 3.10.11 (64-bit)**
    b. **Microsoft® Visual Studio® Code**
    c. **Git for Windows**
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**.
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
```

Test the Successful Intel Distribution of OpenVINO Toolkit Experimentation

Intel Distribution of OpenVINO toolkit experimentation is successful. Next, test utilizing a Docker container.

1. This assumes the virtual environment (vEnv) has already been created and the model has been quantized and is ready for deployment.
2. Download and install **Docker Desktop for Windows**.
3. Create a Docker Image file structure with the following contents:
   a. Main.py (or whatever the name of the chatbot application is)
   b. Requirements.txt
      i. Frozen from the vEnv setup environment
   c. Models (folder containing the optimized model)
4. Launch Visual Studio Code and install the Docker extension.
5. In Visual Studio Code, open the folder that was created in Step 3.
6. Create a new Dockerfile:

```
FROM python:3.10-slim
WORK dir /app
COPY main.py ./
COPY models/Llama3B_int8/ /app/models/Llama3B-
int8/
COPY requirements.txt ./
RUN pip install -r requirements.txt
CMD ["python", "chat.py"]
```

7. Build the Docker image:

```
Docker build -t openvino-app .
```

8. Save the Docker image:

```
Docker save -o openvino_app.tar openvino-app
```

9. Copy the TAR image to a local drive and place it on the target system.
10. Download and install Docker Desktop on the target system.
11. Load the Docker image:

```
Docker load -I openvino_app.tar
```

12. Run the Docker image:

```
Docker run --rm openvino-app
```

13. Deployment was successful.

Lemonade Server SDK Pipeline Experimentation

1. Download and install AMD Lemonade-Server utilizing the GUI, **Lemonade Server**.
2. Download and load the default LLM chat model: **Qwen2.5-0.5B-Instruct-CPU**.
3. Verify the ability to download and load the following models:
   a. **Qwen3-Coder-30B-A3B-Instruct-GGUF** (Hot llama.cpp model)
      i. Easy to download.
      ii. Fails to load in the LLM chat window.
   b. **DeepSeek-R1-Distill-Llama-8B-Hybrid** (OGA Hybrid model)
      i. Easy to download.
      ii. Lemonade-server/Python.exe crashes when attempting to load this model.
   c. **Llama-3.2-3B-Instruct-NPU** (OGA NPU model)
      i. Easy to download.
      ii. Successfully loaded and output tokens.
      iii. Confirmed running against the NPU (Task Manager).
      iv. Slower than the default model, but outputs much more detailed information.
   d. **Deepseek-R1-Distill-Llama-8B-CPU** (OGA CPU model)
      i. Easy to download.
      ii. Successfully loads.
      iii. Very slow response time.
      iv. Poor token output.
   e. **Gpt-oss-20b-GGUF** (Hot llama.cpp model)
      i. Easy to download.
      ii. Fails to load in the LLM chat window.
   f. **Llama-3.2-3B-Instruct-Hybrid**
      i. Easy to download.
      ii. Lemonade-server/Python.exe crashes when attempting to load this model.
   g. **Qwn3-30B-A3B-Instruct-2507-GGUF**
      i. Easy to download.
      ii. Fails to load in the LLM Chat window.
   h. **Meta-llama/Llama3.2-3B** (custom model from Hugging Face)
      i. Same model as was used in Prowess Consulting's technical research report looking at the **Intel Distribution of OpenVINO versus the Qualcomm QNN**.
      ii. Download successfully.
      iii. Fails to load in the LLM Chat window.

Lemonade Server SDK Developer Setup

1. Download and install AMD Lemonade-Server utilizing the GUI, **Lemonade Server.**
2. Download and install **Python 3.10.11**.
3. Download and install **Conda Minforge3**.
4. Add Python and Conda miniforge3 directories to the user environment path.
5. Open a Windows Terminal session and launch Python.
6. From the Python environment, execute conda init.
7. From the Python environment, execute conda activate lemon.
8. Install packages based upon the targeted device:
   a. NPU/Hybrid = OGA
   b. CPU = OGA, llama.cpp, or PyTorch
   c. GPU = llama.cpp
9. Download the targeted model:
   a. **Lemonade-server-dev pull <MODEL_NAME>** (note: reference the model list above)
10. Run the model:
    a. **Lemonade-server-dev run <MODEL_NAME>** (note: reference the model list above)
    b. The lemonade-server will auto-launch in a local browser.
    c. Interaction with the model is via GUI, much like in the EXE/GUI version.
11. Verify the ability to download and load the following models:
    a. **Qwen2.5-0.5B-Instruct-CPU**
       i. Successfully ran.
    b. **DeepSeek-R1-Distill-Llama-8B-CPU**
       i. Successfully ran.
    c. **Gemma-3-4B-it-GGUF**
       i. Successfully ran GPU bound model
    d. **Qwen3-8B-GGUF**
       i. Successfully ran GPU bound model.
    e. **Llama-3.2-3B-Instruct-NPU** (OGA NPU model)
       i. Able to run in GUI environment.
       ii. Fails in dev CLI environment.
          3. Missing /lib/onnxruntime.dll binary
          4. NPU LLM artifacts package: npu-llm-artifacts_1.3.0.zip from **https://account.amd.com/en/member/ryzenai-sw-ea.html**
             e. Failed to gain access to this package.
          6. **Llama-3.2-3B-Instruct-NPU and Llama-3.2-3B-Instruct-Hybridfails** present the same onnxruntime.dll issue.

Test the Lemonade Server SDK Experimentation

Lemonade Server SDK experimentation is successful. Next, test utilizing a Docker container:

1. This assumes the vEnv has already been created and the model has been pulled down from Lemonade Server.
2. Download and install **Docker Desktop for Windows**.
3. Create the Docker image file structure containing Main.py (or the name of the chatbot application).
4. Launch Visual Studio Code and install the Docker extension.
5. In Visual Studio Code, open the folder that was created in Step 3.
6. Create a new Docker file that includes the following:

```
FROM python:3.12-slim
WORKDIR /app
COPY main.py ./
RUN pip install “lemonade-sdk[dev,oga-cpu]”
RUN lemonade-server-dev pull Qwen2.5-0.5B-Instruct-CPU
EXPOSE 8000
CMD [“python”, “main.py”}
```

7. Build the Docker image by running the following command:

```
Docker build -t amd-app
```

8. Save the Docker image by running the following command:

```
Docker save -o amd_app.tar amd-app
```

9. Copy the TAR image to a local drive and place it on the target system.
10. Download and install Docker Desktop on the target system.
11. Load the Docker image by running the following command:

```
Docker load -i amd_app.tar
```

12. Run the Docker image by running the following command:

```
Docker run --rm amd-app
```

Read the technical research reports:

The Intel® Distribution of OpenVINO™ toolkit vs. the Qualcomm® AI Engine Direct SDK

The Intel® Distribution of OpenVINO™ toolkit vs. Core ML®

Endnotes

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

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

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

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

[5] Microsoft. "**Copilot+ PCs expand availability with new AMD and Intel silicon**." September 2024.

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

[7] AMD. **System requirements**. Accessed October 2025.

[8] For more on Weights-Only Quantization, see "**Weight Compression**". Accessed October 2025.

[9] Intel. "**Post-training Quantization**." Accessed October 2025.

PROWESS

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.
Prowess Consulting and the Prowess logo are trademarks of Prowess Consulting, LLC.

Copyright © 2025 Prowess Consulting, LLC. All rights reserved.
Other trademarks are the property of their respective owners.

1025/250101