PROWESS

Technical Research Report

Which Toolkit Provides the Best Optimization for Large Language Models?

In testing commissioned by Intel, Prowess Consulting evaluated the Intel® Distribution of OpenVINO™ toolkit and Core ML® to identify the best LLM deployment pipeline.

Executive Summary

Developers recognize the critical need for efficient AI solutions across diverse computing environments. Hardware-specific software development kits (SDKs) enable seamless integration with on-device hardware, enhancing model execution and accelerating neural network inference, thereby improving a model's ability to apply patterns to new inputs.

With the transition to AI-powered 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 (targeting the CPU, GPU, and neural processing unit [NPU]) or on Apple® silicon–based Mac® (targeting the CPU, GPU, and Apple Neural Engine [ANE]). To help clarify these considerations and assist developers in choosing the most effective on-device large language model (LLM) toolkit, Prowess Consulting tested the Intel® Distribution of OpenVINO™ toolkit on a Dell™ XPS™ 13 AI PC (with an Intel Core Ultra 7 processor 256V) and the Core ML® framework on an Apple MacBook Pro 4 running macOS®.

In testing performed by Prowess Consulting and commissioned by Intel, the Intel Distribution of OpenVINO toolkit delivered a smoother end-to-end pipeline, including download, conversion to OpenVINO intermediate representation (IR), quantization (INT8/INT4, including an NPU-specific INT4), local validation, and containerization. Core ML handled model download, conversion, and weights-only INT8 and INT4 quantization (activations remained in floating-point). However, it still required custom Swift®/Xcode® work for inference (see Table 1).

Table 1. SDK scorecard: the Intel® Distribution of OpenVINO™ toolkit versus the Core ML® framework (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	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★★
Apple® Core ML® framework	★★★★★	★★★☆☆	★★★★☆	★☆☆☆☆	★★★☆☆

Primary drivers influencing our SDK ratings in Table 1 include:

- The Intel Distribution of OpenVINO toolkit enables a straightforward, repeatable pipeline: convert the model for 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.
- The Core ML documentation gaps that we observed increased effort. Developers can request preferred compute units (for example, .cpuOnly, .cpuAndGPU, .cpuAndNeuralEngine, and .all), but there is no documented way to guarantee ANE-only execution.

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 (including Apple silicon systems) are high-performance computers designed with 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. On Apple platforms, AI-capable Mac devices with Apple silicon integrate ANE for on-device ML alongside the CPU and GPU, accelerating on-device models in macOS applications. These systems began shipping from Apple in 2020.[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.

Tooling and Techniques for On-Device 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.

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 Apple silicon, selecting the system that is most appropriate for their workloads. To help AI developers determine the right architecture 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 Core ML framework (on Apple M-series silicon) 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

Models are getting better, faster, and in some cases smaller. Running models locally enables faster, no-API iterations for domain-specific customization, improves performance and accuracy, reduces latency, cuts usage-based API costs (for example, per-1,000-token billing), and helps organizations with strict security requirements protect sensitive data by keeping inference offline.

Open-source AI tools offer flexibility and customization, but developers might require technical expertise to optimize model performance and prevent vulnerabilities that could expose sensitive data. At the same time, tools like the Intel Distribution of OpenVINO toolkit enable 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.

Research Approach

We aligned the two evaluations around the same core activities: model acquisition, conversion, quantization, local inference validation, application integration, and deployment.

For the OpenVINO toolkit, we downloaded the Meta® Llama®-3.2-3B model from Hugging Face and converted it to OpenVINO IR. Using the OpenVINO toolchain, we produced INT8-quantized and INT4-quantized models—including an NPU-specific INT4 variant—and then validated local inference by prompting the model. For packaging and deployment, we built a Docker® image that included the application code, optimized model, and dependencies. The container ran successfully on the target system.

For Core ML, we acquired the same Llama-3.2-3B model via the Hugging Face hub and converted it to Core ML using the Llama-to–Core ML scripts found in the GitHub repository (**andmev/llama-to-coreml**). We added missing modules and saved the result as an .mlpackage for Xcode. Apple's Core ML Python® code examples are illustrative; practical inference requires moving to Swift/Xcode and writing additional custom code.

Because string processing and tokenization are not natively supported in Core ML, a working chatbot demands a custom tokenizer approach. Our proof-of-concept (PoC) Xcode app could load the model and emit text from a small vocabulary file, but it was not a fully intelligent chatbot.

Table 2 | Tools face-off: the Intel® Distribution of OpenVINO™ toolkit versus the Core ML® framework

SDK	Intel® Distribution of OpenVINO™ Toolkit	Core ML® Framework
Platform support	• Open-source tools and high-level APIs • C/C++ for low-level integrations, Python®, and Node.js®	• Apple framework for macOS®/iOS®; integrated with Xcode®/Swift® • Model conversion via Python coremltools • Practical inference requires Swift/Xcode • Models saved as .mlpackage for app use
Hardware acceleration	• Intel® x86-64 architecture (CPUs), Intel integrated and discrete GPUs, and Intel NPUs • Arm® processors (CPUs) since February 2025	• Automatically targets CPU, GPU, or ANE • Device selection is automatic (CPU only/ CPU+GPU/CPU+ANE/all)
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 acquired from Hugging Face® (Llama®-3.2-3B) • Model converter (Core ML tools) required added missing modules (Jupyter®, Torch, and Setuptools) and settings • Weights-only INT8 and INT4 quantization completed via coremltools (activations remained floating-point) • Saved as .mlpackage for Xcode • Runtime considerations: practical inference not executed in Python; requires Swift/Xcode • Full chatbot behavior needed custom tokenizer/ strings processing (not native in Core ML) and additional Swift/Xcode integration
Deployment	• Flexible deployment—write once, deploy anywhere, on device and with cloud AI	• Model must be used in a Swift/Xcode app • Minimal PoC app could load the model and output text from a small vocabulary file • A functional chatbot requires additional custom code
Licensing	• Free for commercial use with Apache® License 2.0	• Proprietary Apple framework included with macOS/iOS SDKs and Xcode
Security	• Model encryption with third-party tools • OpenVINO Security Add-on (OVSA) for access control • Datumaro for encryption of computer vision datasets	• Relies on Apple platform security; no separate Core ML security add-ons noted in testing
Other	• Convolutional neural network (CNN), image classification, object detection, and face recognition	• Apple's example code was conceptual; documentation omitted required steps, increasing setup effort • Platform dependent (macOS/iOS only) • Smaller deployment footprint compared to a full OpenVINO toolkit stack

Intel Distribution of OpenVINO Toolkit Deployment

On a Dell XPS 13 AI PC (powered by an Intel Core Ultra 7 processor 256V), the OpenVINO toolkit used a standard Docker flow. The container was built and loaded, and it ran successfully, confirming local inference and deployment.

SYSTEM UNDER TEST:

MAKE/MODEL: Dell™ XPS™ 13 9350

CPU: Intel® Core™ Ultra 7 processor 256V
GPU: Intel® Arc™ 140V graphics
NPU: Intel® AI Boost

REQUIRED SETUP AND APPLICATIONS

- Windows® 11 (with Docker® Desktop for Windows)
- Intel® Distribution of OpenVINO™ toolkit
- Microsoft® Visual Studio® Code with Docker extension
- Python® 3.10.11

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

SYSTEM UNDER TEST:

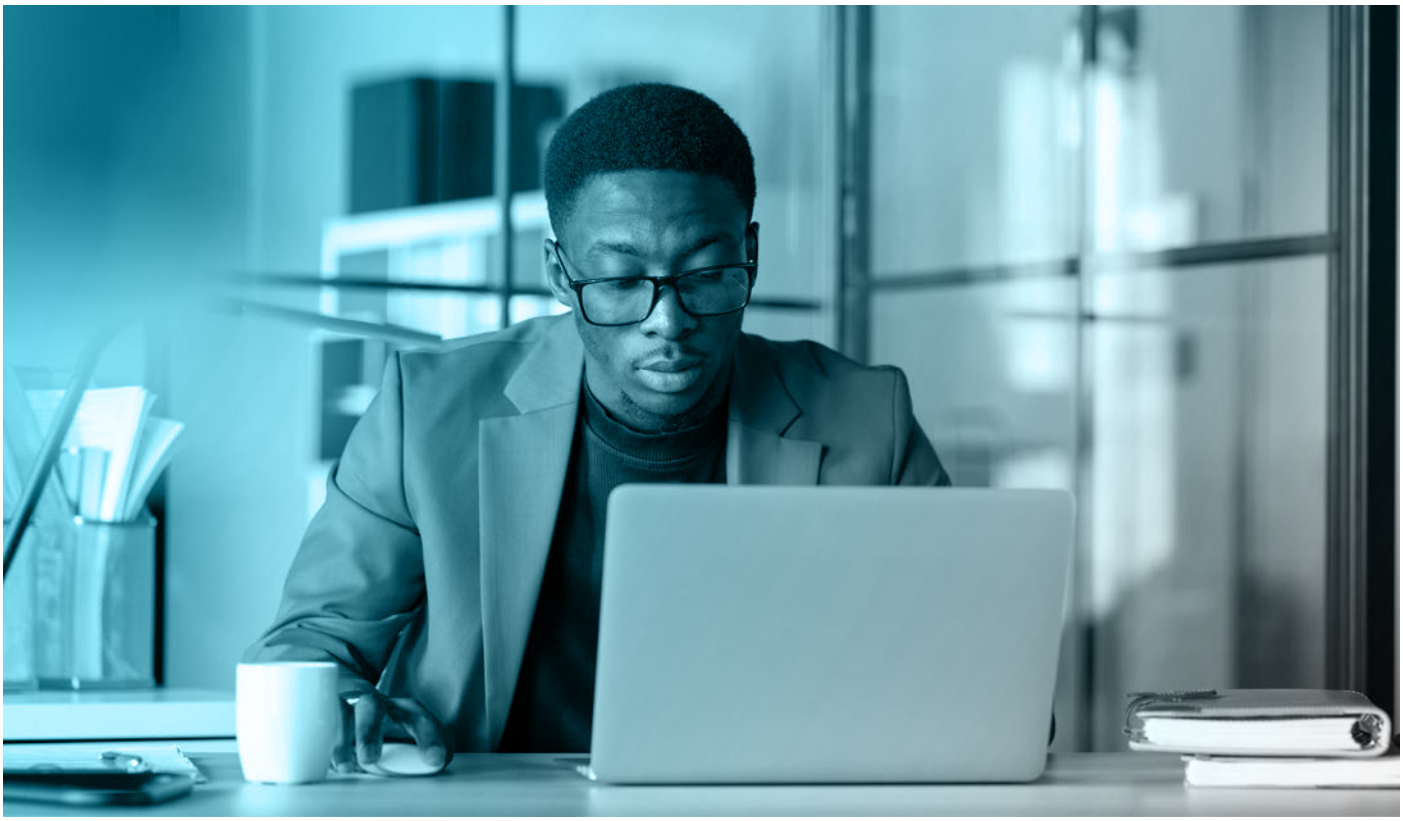
MAKE/MODEL: Apple® MacBook Pro®

CPU: Apple® M4 (Apple silicon)
GPU: Integrated Apple GPU
NPU: Apple Neural Engine (ANE)

REQUIRED SETUP AND APPLICATIONS

- macOS® Sequoia 15.7.2 with Xcode® 26.1 (for Swift® app integration)
- Llama®-to–Core ML® scripts from GitHub repository (andmev/llama-to-coreml) to convert models from Hugging Face® to Core ML format

Figure 2 | AI PC development environment: the Core ML® framework



Comparison of the Intel Distribution of OpenVINO Toolkit and the Core ML Framework 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. Our 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 the target PC.

Core ML® Framework Pipeline Overview

1. Configure the Core ML environment.
2. Download the Meta® Llama-3.2-3B model.
3. Convert the model to Core ML.
4. Complete INT4 and INT8 quantization successfully.
5. Integrate Xcode®/Swift®, import the .mlpackage, and wire up minimal input/output (I/O).
6. Validate interference (requires custom Swift/Xcode code and tokenizer
7. Full chatbot behavior was not validated within test window.

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 optimized (quantized) Llama-3.2-3B INT8 and INT4 models 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.

Issues with the Core ML framework

For the Core ML testing, we downloaded the Llama-3.2-3B model. We converted it to Core ML format using Apple's **published research**, but only after adding missing pieces the guide did not call out (including installing Jupyter®, Torch, and Setuptools) to work around a distutils failure. However, our testing found that running inference required switching to Swift/Xcode and custom application code. We saved the result as an .mlpackage to enable Xcode integration.

On a Mac device powered by Apple silicon and running macOS, we evaluated Core ML using Apple's published example for on-device Llama. Using the Core ML Tools weights-only quantization API, INT8 and INT4 quantization completed (and activations remained floating-point).

To execute the model, we had to move to Swift/Xcode. Even then, a functional chatbot required custom engineering because Core ML lacks native string processing and tokenizer support (for example, SentencePiece/Byte-Pair Encoding [BPE]). As a PoC, we could load the .mlpackage in a minimal Xcode app and emit text from a small vocabulary file, but a real chatbot was out of scope without substantial custom code.

Targeting specific hardware (CPU, GPU, or ANE) is typically decided by Core ML. Developers can request compute units via MLModelConfiguration. computeUnits (for example, .cpuOnly, .cpuAndGPU, .cpuAndNeuralEngine, and .all). See Table 2 for more on the features and differences between the Intel Distribution of OpenVINO toolkit and Apple Core ML.

Overall, the Core ML pipeline proved workable for model conversion (with fixes) and INT8 and INT4 quantization, but end-to-end inference and app integration required significant custom Swift/Xcode work and a bespoke tokenizer strategy.

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

Evaluation Criteria	Intel® OpenVINO™ Toolkit	Apple Core ML®	Notes
Target hardware	★★★★★	★★★★★	The Intel Distribution of OpenVINO toolkit supports Intel CPUs, GPUs, and NPUs, which are present in a wide range of OEM systems. Core ML is optimized for Apple® silicon only.
Platform compatibility	★★★★☆	★★★☆☆	The Intel Distribution of OpenVINO toolkit runs on Windows®, Linux®, and macOS®. Core ML is Apple-only (iOS® and macOS).
Features	★★★★★	★★★☆☆	The Intel Distribution of OpenVINO toolkit offers advanced conversion, compression, and hardware targeting tools. Core ML is streamlined, but has limitations such as platform lock-in, limited deployment flexibility, and Swift®/Xcode® dependency.
Model conversion	★★★★☆	★★★★☆	The Intel Distribution of OpenVINO toolkit has a streamlined conversion tool that supports ONNX®, TensorFlow™, PyTorch®, and so on. CoreML uses the coremltools Python® package, which has limited operator support and stricter constraints, such as partial format support, basic TensorFlow models, and error-prone PyTorch conversions.
Quantization	★★★★★	★★☆☆☆	Quantization to INT8, INT4, and NPU-specific INT4 is intuitive and streamlined with the Intel Distribution of OpenVINO toolkit through the CLI. Core ML supports weights-only INT8/INT4 via coremltools (activations float); ANE targeting cannot be guaranteed.
Inference	★★★★☆	★☆☆☆☆	The Intel Distribution of OpenVINO toolkit offers flexible model interaction with CLI/Python APIs. Core ML requires Swift/Xcode, which can be a barrier for non-Apple developers.
Ease of use	★★★★★	★★☆☆☆	The Intel Distribution of OpenVINO toolkit has intuitive CLI tools and Python APIs. Core ML is easier for Apple-native workflows but is less accessible outside the Apple ecosystem.
Technical proficiency required	★★★★★	★★☆☆☆	The Intel Distribution of OpenVINO toolkit requires some knowledge of basic Python environments and package installs. Core ML assumes familiarity with Swift and Xcode, which can be limiting for non-Apple developers.
Overall score	★★★★★	★★☆☆☆	The Intel Distribution of OpenVINO toolkit excels in hardware support, model conversion, quantization, and flexibility. It is ideal for those who want full control over model optimization. Core ML is streamlined for Apple ecosystems but lacks broader compatibility. Core ML requires Swift/Xcode proficiency, which could be a barrier.

Making Models Smaller

Why does quantization matter for LLMs? Because the smaller numerical footprint of quantized data 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 steps to convert a pre-trained floating-point model to a quantized version with lower precision are shown in Figure 3.

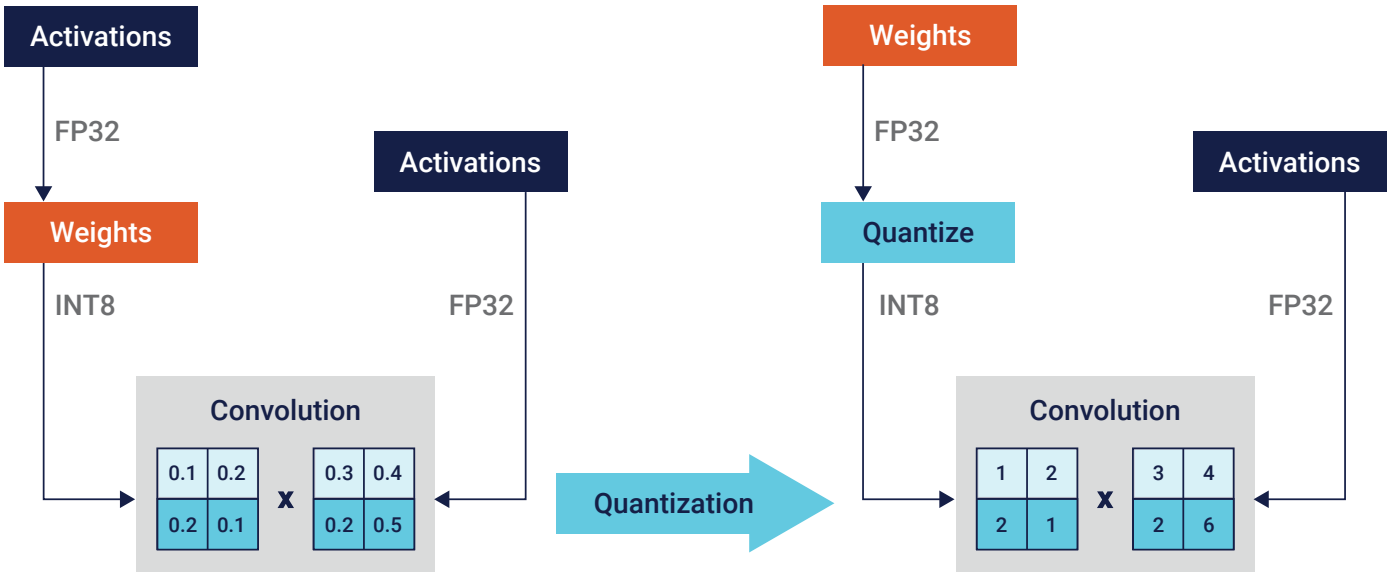


Figure 3 | This diagram illustrates the workflow for post-training quantization. In this example, the process converts an LLM's weights and activations from a higher precision format (FP32) to a lower precision format (INT8). That reduces the model size and improves inference efficiency on AI acceleration hardware with limited impact on accuracy, depending on the model.[5]

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 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

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.

For Core ML, we downloaded Llama-3.2-3B via the Hugging Face hub and successfully converted it to the Core ML format (after adding missing modules/variables and saving it as a .mlpackage). Practical inference was not achievable in the Python environment; running the model required moving to Swift/Xcode, where a functional chatbot demanded significant custom code (for example, tokenizer/string-processing workarounds).

In this evaluation, the Intel Distribution of OpenVINO toolkit provided a repeatable deployment path with an optimized model. The Core ML framework allowed for model download and conversion (including INT8 and INT4 quantization), but required additional modules and Swift/Xcode integration for inference. However, an end-to-end chatbot could not be achieved without substantial custom code, which was outside the scope of this test.

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 Core ML framework. The objective of this test was to create a chatbot application.

Intel Distribution of OpenVINO Toolkit Pipeline

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 Pipeline

The Intel Distribution of OpenVINO toolkit implementation was successful. Next, we tested 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. A 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 with the following contents:

```
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.

Core ML Framework Pipeline

1. Configure the desired developer environment with the following tools and applications:
   a. Python 3.12
   b. Microsoft Visual Studio Code
   c. Apple Xcode developer tools
2. Pull down the Meta-Llama/Llama-3.2-3B model with the Hugging Face hub CLI command.
3. Convert the model to Core ML format.
   a. The code used for Core ML model export can be found on **this Llama-to–Core ML GitHub repository**.
   b. Some working knowledge of installing Python packages and troubleshooting variable errors is required to get the Apple example to work.
4. Use the Core ML tools Python package to quantize the model to INT8 and INT4.
5. Save the model in .mlpackage format or it will simply save in memory with no way to be accessed through Xcode.

```
Mlmodel.save("<MODEL_NAME.mlpackage>")
```

6. Core ML model inference cannot be executed without wrapping an application around it with Swift code.
7. Launch Xcode and create a new project.
8. Import the Core ML model into the project.
9. Create custom code using Swift to build a chatbot with a prompt and response.
10. A trivial application was built and deployed to show a PoC. This entailed creating a vocab.txt file to simplify the process of running inference without building custom tokenizers and having Swift call a Flask server.

Test the Core ML Framework Pipeline

In our testing, the Core ML framework experimentation was unsuccessful in deploying a chatbot.

1. Model acquisition was achieved by pulling Meta-Llama/Llama-3.2-3B via the Hugging Face hub.
2. We followed Apple's example for converting to Core ML format to produce a Core ML model, adding the required pieces (including missing Python modules and a model_id variable and saving it in .mlpackage format) so the model can be imported into Xcode.
3. INT8 and INT4 quantization succeeded.
4. For the inference path, the Python example code was illustrative only; running inference required moving to Swift/Xcode.
5. With custom scaffolding, we created a minimal PoC app that loaded the .mlpackage file and produced output using a small vocabulary file. A functional chatbot was not achieved without additional custom tokenizer and string-processing work.
6. For device selection, Core ML automatically selected CPU/GPU/ANE; however, we did not find a simple, documented way to force ANE usage in the Apple materials.
7. Core ML experimentation validated model conversion and weights-only INT8 and INT4 completed, inference required Swift/Xcode, and ANE could not be forced and was not engaged during quantization. End-to-end chatbot testing was blocked by the need for substantial custom code (including tokenization and string handling).

For more information on Intel, visit
Which Toolkit Provides the Best Optimization for Large Language Models?

Read the technical research reports:
The Intel® Distribution of OpenVINO™ toolkit vs. the Lemonade Server SDK
The Intel® Distribution of OpenVINO™ toolkit vs. the Qualcomm® AI Engine Direct SDK

Endnotes

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

[2] Apple. "**Apple Unleashes M1**." November 2020.

[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] 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.

1225/250101