DirectML Readiness and Example

Technical Training Material

WW26, June 2024

intel®

Legal Disclaimer

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis. You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

All product plans and roadmaps are subject to change without notice.

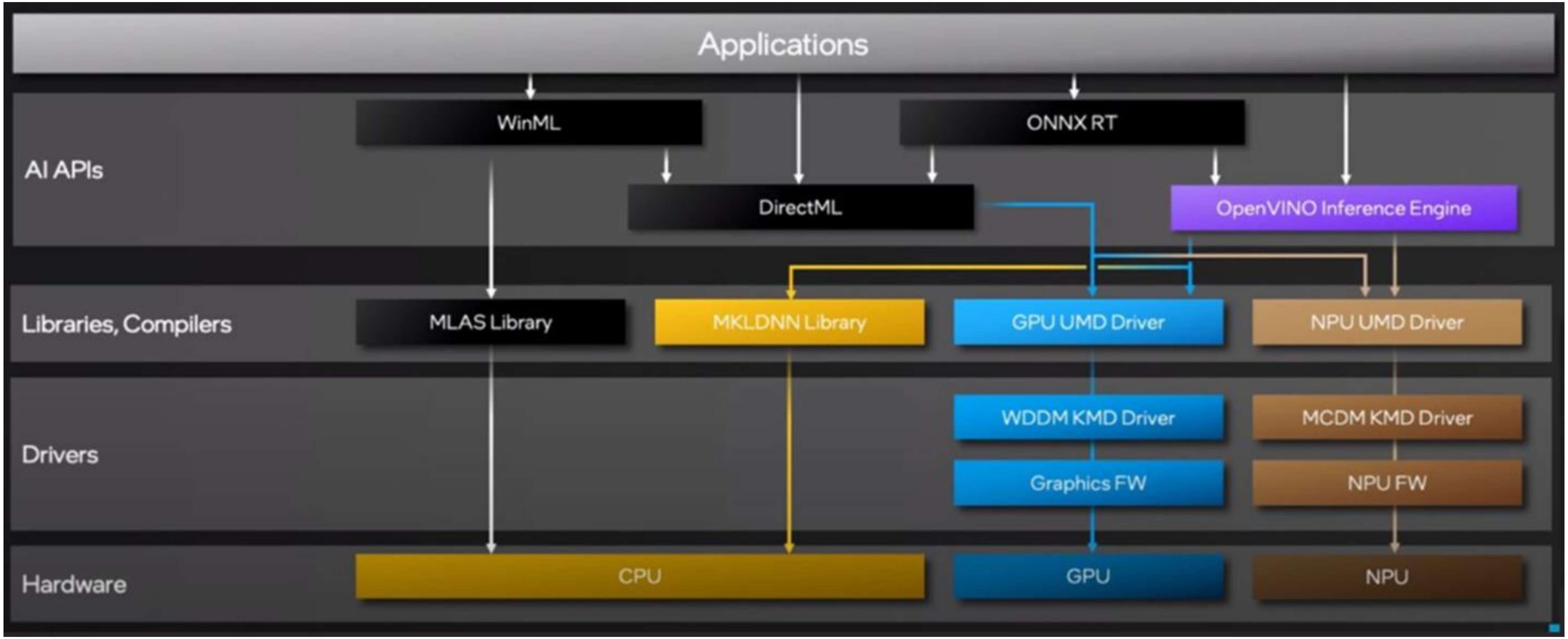
The products described may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Altering clock frequency, voltage, or memory interface speeds may void any product warranties and reduce stability, security, performance, and life of the processor and other components. Intel has not validated processor running memory above Plan-Of-Record (POR) speed. DRAM/DIMM devices should support desired speed, check with DRAM/DIMM vendors for details. System manufacturers are responsible for all validation and assume the risk of any stability, security, performance, or other functional issues resulting from such alterations.

Intel Optimized AI PC Stacks



*Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

DirectML Readiness

Below models are already supported on NPU

Model Type	Model Name
Public	whisper_base_encoder_lm_fp16_layernorm_gelu
Public	whisper_base_decoder_static_kvcache_128_lm_fp16_layernorm_gelu_4dmask
Public	whisper_base_decoder_static_non_kvcache_lm_fp16_layernorm_gelu_4dmask
Public	mobilenetv2-7-fp16-optimized
Public	resnet50-v1-12-fp16
Public	efficientnet-lite4-11-fp16
Public	squeezenet1.0-12-fp16

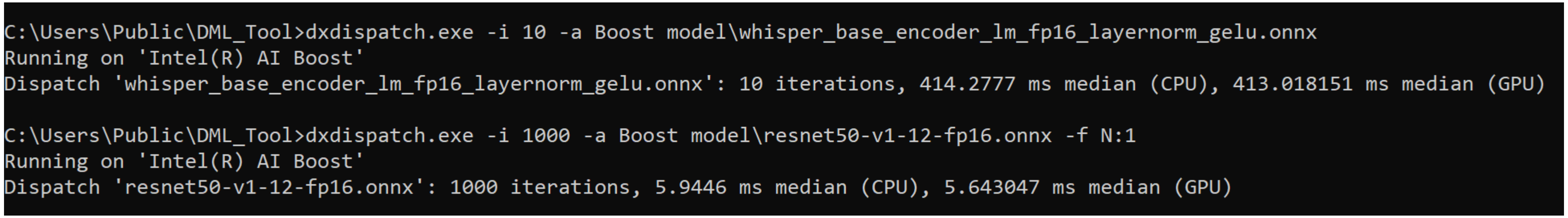
DML Benchmarking

Wait for public binaries from MSFT, its expected to be available beginning of July

DirectML/Releases.md at master · microsoft/DirectML (github.com) directml.dll v1.15

Need to enable Developer Mode in Settings> System> For developers> Developer Mode ON

- WinMLRunner:
  - WinMLRunner.exe -model **whisper_base_encoder_lm_fp16_layernorm_gelu.onnx** -GpuAdapterName boost -perf -iterations 10
  - WinMLRunner.exe -model **squeezenet1.0-12-fp16.onnx** -GpuAdapterName boost -perf -iterations 1000
- DxDispatch:
  - dxdispatch.exe -i 10 -a Boost **whisper_base_encoder_lm_fp16_layernorm**_gelu.onnx
  - dxdispatch.exe -i 1000 -a Boost **resnet50-v1-12-fp16.onnx** -f N:1



```
C:\Users\Public\DML_Tool>dxdispatch.exe -i 10 -a Boost model\whisper_base_encoder_lm_fp16_layernorm_gelu.onnx
Running on 'Intel(R) AI Boost'
Dispatch 'whisper_base_encoder_lm_fp16_layernorm_gelu.onnx': 10 iterations, 414.2777 ms median (CPU), 413.018151 ms median (GPU)

C:\Users\Public\DML_Tool>dxdispatch.exe -i 1000 -a Boost model\resnet50-v1-12-fp16.onnx -f N:1
Running on 'Intel(R) AI Boost'
Dispatch 'resnet50-v1-12-fp16.onnx': 1000 iterations, 5.9446 ms median (CPU), 5.643047 ms median (GPU)
```

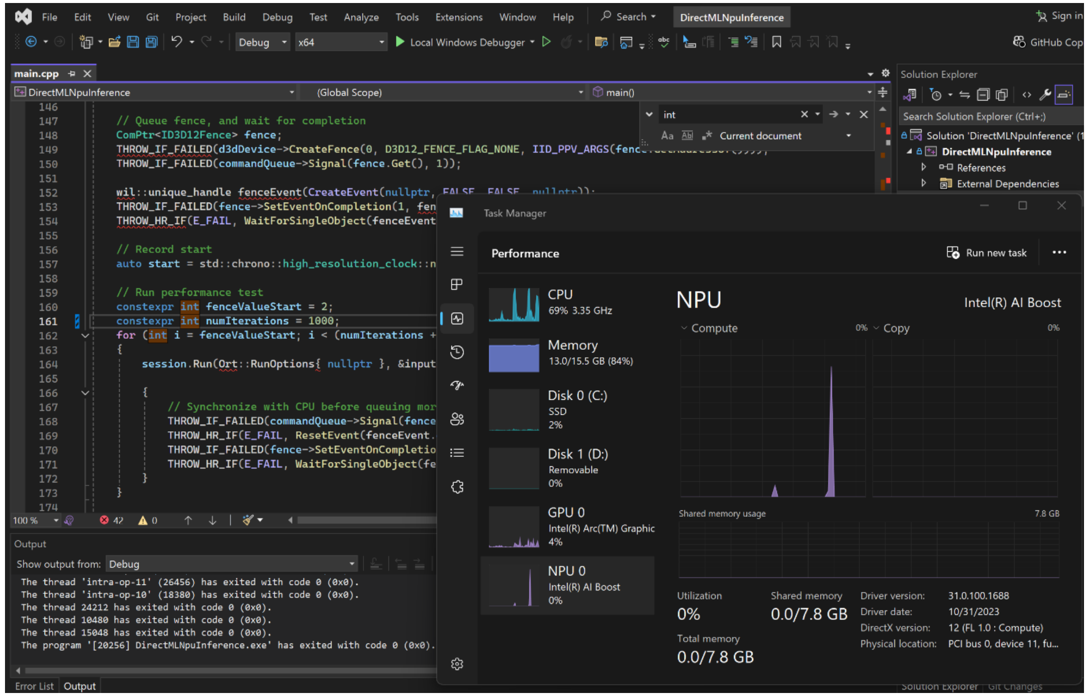
Setup Environment

1. Test configuration:
   - Platform: MTL
   - OS: Win11 22H2 or 23H2
   - NPU driver: 32.0.100.2540
   - GPU driver: 31.0.101.5334+
2. Install Visual Studio 2022 (community version is ok):
   - install "Desktop development with C++)
3. Copy Lab1 DirectML “Samples” and “Libraries” folders to C:\Users\Public
4. Build DirectMLNpuInference:
   - Launch C:\Users\Public\Samples\DirectMLNpiInference\DirectMLNpuInference.sln by Visual Studio 2022
   - Start build by selecting debug /x64 and click “Local Windows Debugger”
5. Build DirectMLSuperResolution:
   - Launch C:\Users\Public\Samples\DirectMLSuperResolution\DirectMLSuperResolution.sln by Visual Studio 2022
   - Start build by selecting debug /x64 and click “Local Windows Debugger”

DirectML Example (NPU Inference)

DirectMLNpuInference

A minimal but complete DirectML sample that demonstrates how to perform OnnxRuntime inference via D3D12 and DirectML on a NPU. Select a NPU device, create a OnnxRuntime session, execute the model on the NPU, and retrieve the results. This sample executes the mobilenet model.



```
        // Queue fence, and wait for completion
        ComPtr<ID3D12Fence> fence;
        THROW_IF_FAILED(d3dDevice->CreateFence(0, D3D12_FENCE_FLAG_NONE, IID_PPV_ARGS(fenc
        THROW_IF_FAILED(commandQueue->Signal(fence.Get(), 1));

        wil::unique_handle fenceEvent(CreateEvent(nullptr, FALSE, FALSE, nullptr));
        THROW_IF_FAILED(fence->SetEventOnCompletion(1, fen
        THROW_HR_IF(E_FAIL, WaitForSingleObject(fenceEvent

        // Record start
        auto start = std::chrono::high_resolution_clock::n

        // Run performance test
        constexpr int fenceValueStart = 2;
        constexpr int numIterations = 1000;
        for (int i = fenceValueStart; i < (numIterations +
        {
            session.Run(Ort::RunOptions{ nullptr }, &input

            {
                // Synchronize with CPU before queuing mor
                THROW_IF_FAILED(commandQueue->Signal(fence
                THROW_HR_IF(E_FAIL, ResetEvent(fenceEvent.
                THROW_IF_FAILED(fence->SetEventOnCompletio
                THROW_HR_IF(E_FAIL, WaitForSingleObject(fe
            }
        }
```

Output

Show output from: Debug

```
The thread 'intra-op-11' (26456) has exited with code 0 (0x0).
The thread 'intra-op-10' (18380) has exited with code 0 (0x0).
The thread 24212 has exited with code 0 (0x0).
The thread 10480 has exited with code 0 (0x0).
The thread 15048 has exited with code 0 (0x0).
The program '[20256] DirectMLNpuInference.exe' has exited with code 0 (0x0).
```

Task Manager

Performance

- CPU: 69% 3.35 GHz
- Memory: 13.0/15.5 GB (84%)
- Disk 0 (C:): SSD 2%
- Disk 1 (D:): Removable 0%
- GPU 0: Intel(R) Arc(TM) Graphic 4%
- NPU 0: Intel(R) AI Boost 0%

NPU — Intel(R) AI Boost

Compute 0% · Copy 0%

Shared memory usage 7.8 GB

Utilization: 0%
Shared memory: 0.0/7.8 GB
Total memory: 0.0/7.8 GB

Driver version: 31.0.100.1688
Driver date: 10/31/2023
DirectX version: 12 (FL 1.0 : Compute)
Physical location: PCI bus 0, device 11, fu...

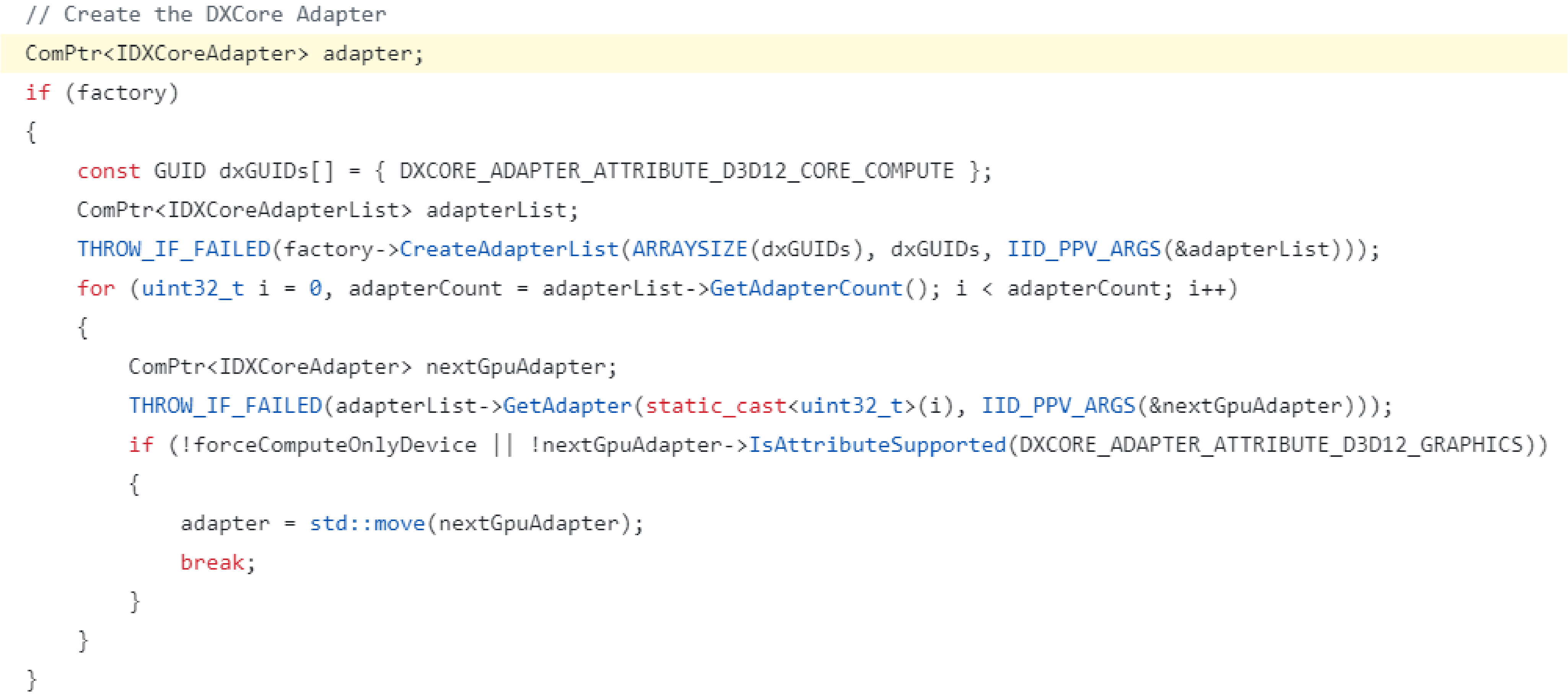
DirectML Example (NPU Inference) (Cont.)

Currently, we can only run this sample on MTL with SV2 OS:

- NPU has moved to being generic ml device instead of core compute on 24H2:
  - const GUID dxGUIDs[] = { DXCORE_ADAPTER_ATTRIBUTE_D3D12_CORE_COMPUTE }; (SV2)
  - DXCORE_ADAPTER_ATTRIBUTE_**D3D12_GENERIC_ML** (24H2)
- Need generic ML update to support LNL

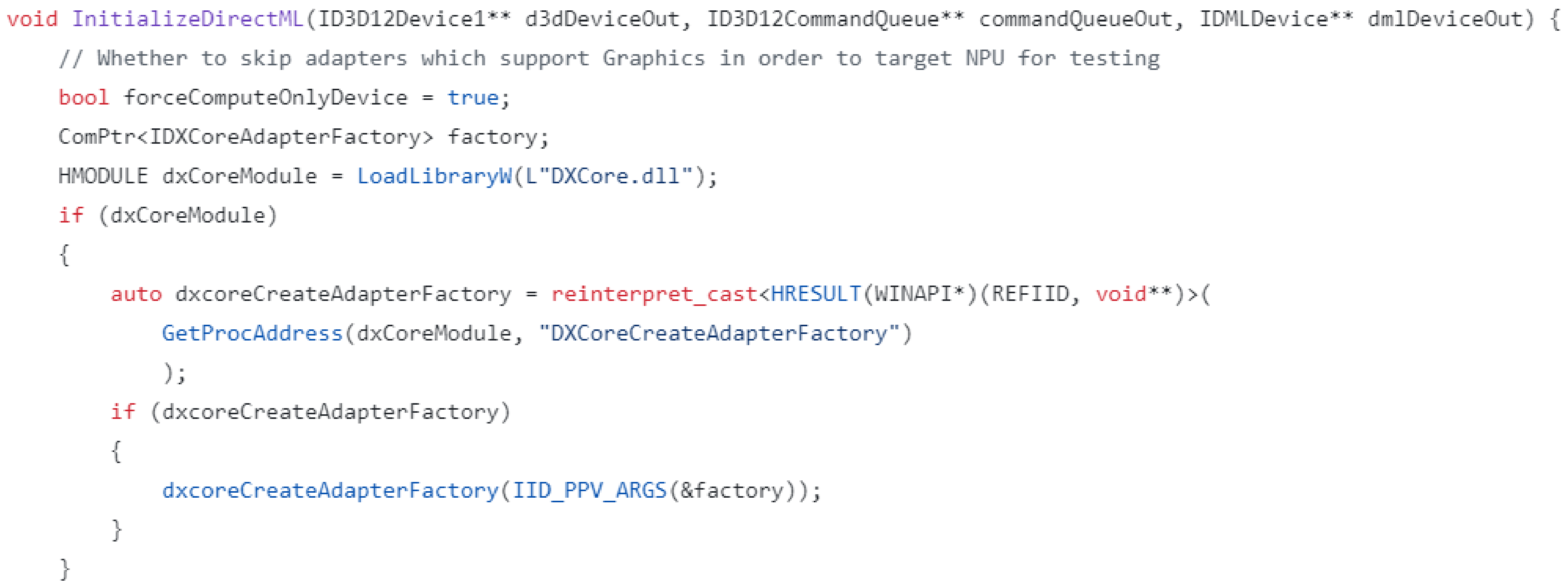
*Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

DirectML Example (NPU Inference) (Cont.)



```
// Create the DXCore Adapter
ComPtr<IDXCoreAdapter> adapter;
if (factory)
{
    const GUID dxGUIDs[] = { DXCORE_ADAPTER_ATTRIBUTE_D3D12_CORE_COMPUTE };
    ComPtr<IDXCoreAdapterList> adapterList;
    THROW_IF_FAILED(factory->CreateAdapterList(ARRAYSIZE(dxGUIDs), dxGUIDs, IID_PPV_ARGS(&adapterList)));
    for (uint32_t i = 0, adapterCount = adapterList->GetAdapterCount(); i < adapterCount; i++)
    {
        ComPtr<IDXCoreAdapter> nextGpuAdapter;
        THROW_IF_FAILED(adapterList->GetAdapter(static_cast<uint32_t>(i), IID_PPV_ARGS(&nextGpuAdapter)));
        if (!forceComputeOnlyDevice || !nextGpuAdapter->IsAttributeSupported(DXCORE_ADAPTER_ATTRIBUTE_D3D12_GRAPHICS))
        {
            adapter = std::move(nextGpuAdapter);
            break;
        }
    }
}
```

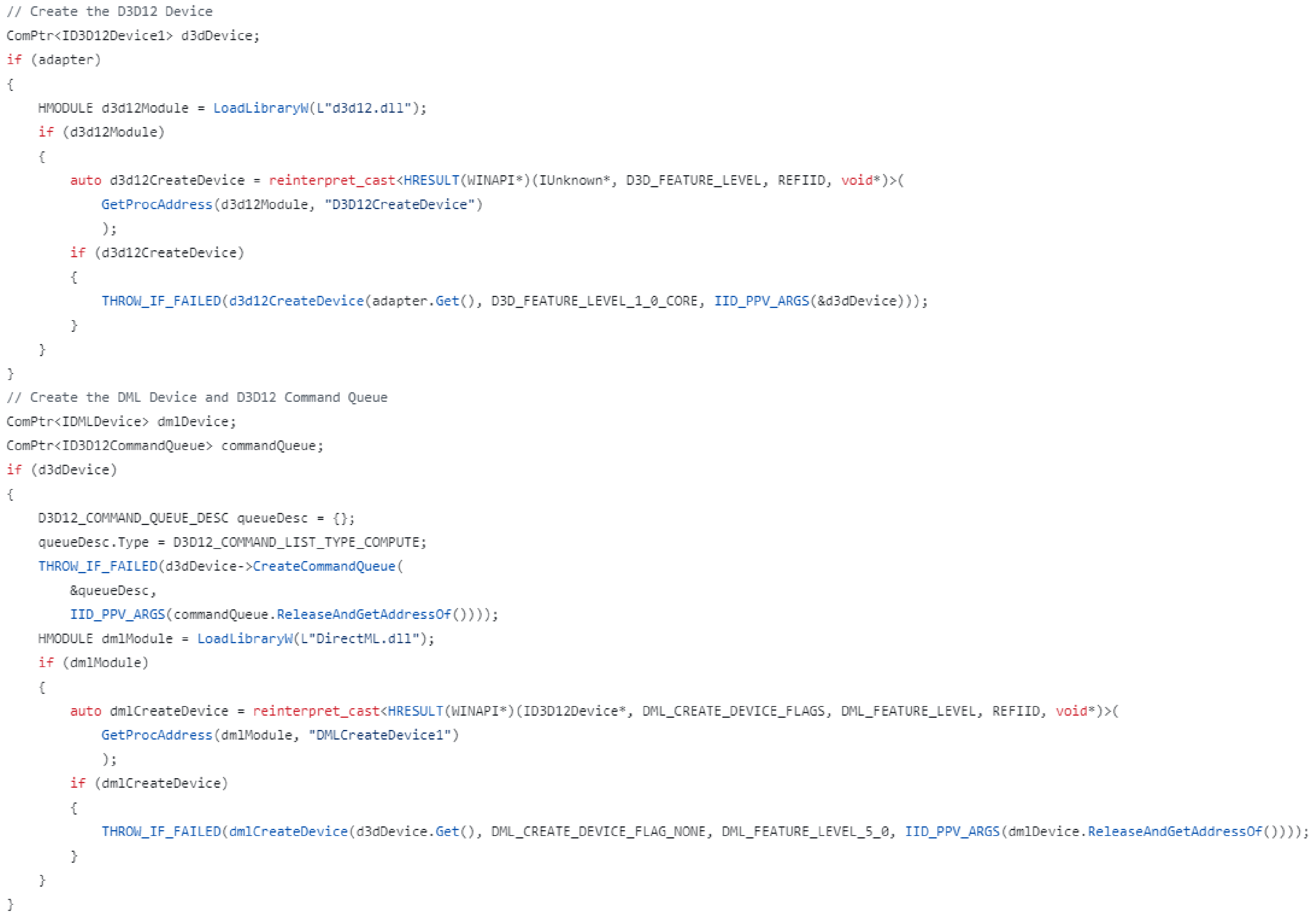
DirectML Example (NPU Inference) (Cont.)



```
void InitializeDirectML(ID3D12Device1** d3dDeviceOut, ID3D12CommandQueue** commandQueueOut, IDMLDevice** dmlDeviceOut) {
    // Whether to skip adapters which support Graphics in order to target NPU for testing
    bool forceComputeOnlyDevice = true;
    ComPtr<IDXCoreAdapterFactory> factory;
    HMODULE dxCoreModule = LoadLibraryW(L"DXCore.dll");
    if (dxCoreModule)
    {
        auto dxcoreCreateAdapterFactory = reinterpret_cast<HRESULT(WINAPI*)(REFIID, void**)>(
            GetProcAddress(dxCoreModule, "DXCoreCreateAdapterFactory")
            );
        if (dxcoreCreateAdapterFactory)
        {
            dxcoreCreateAdapterFactory(IID_PPV_ARGS(&factory));
        }
    }
```

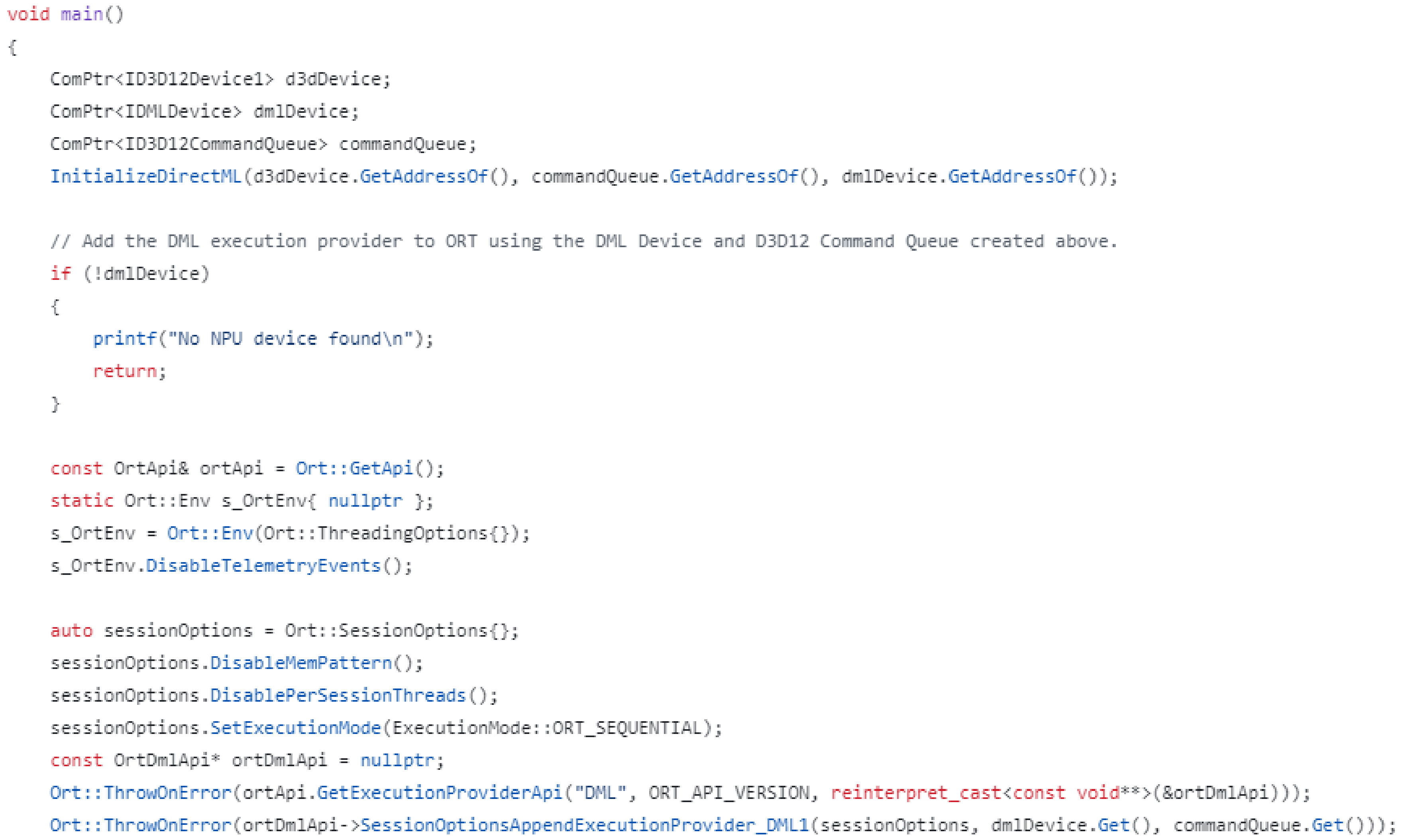
*Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

DirectML Example (NPU Inference) (Cont.)



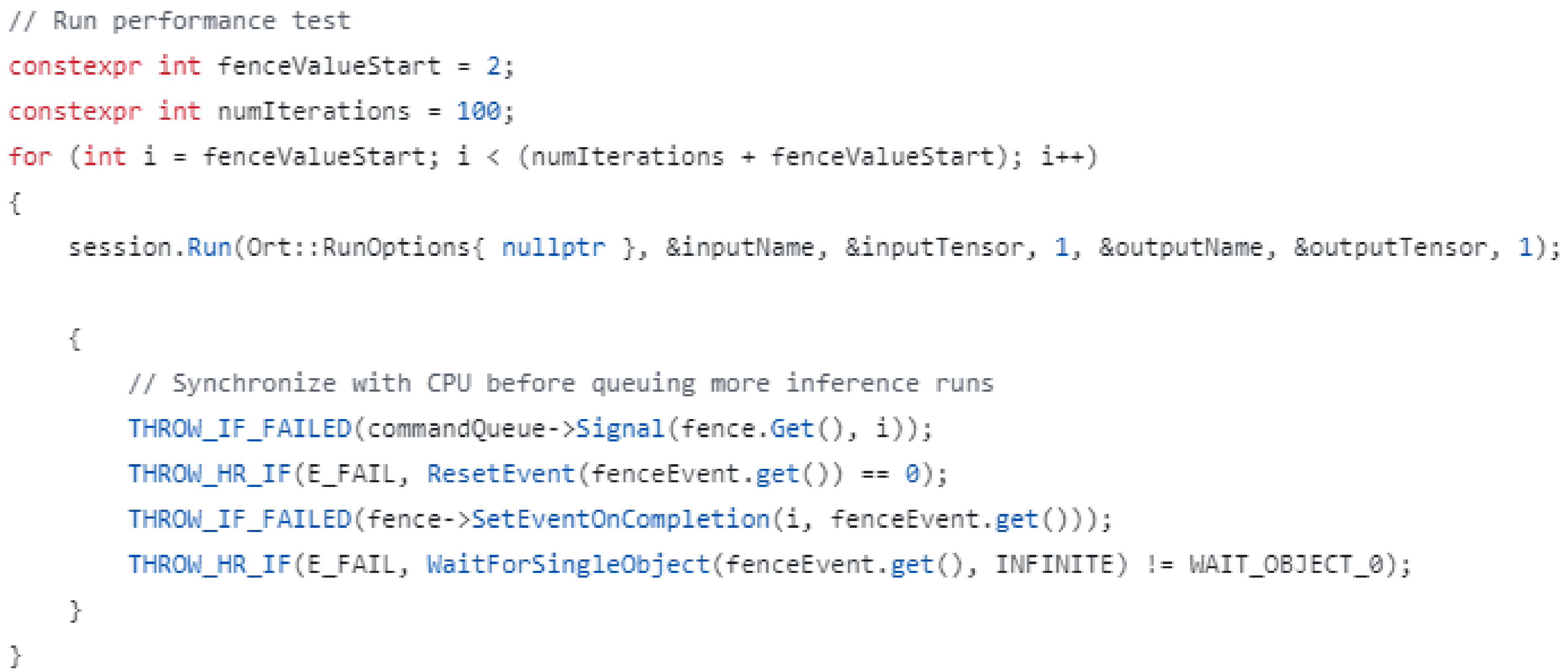
```
// Create the D3D12 Device
ComPtr<ID3D12Device1> d3dDevice;
if (adapter)
{
    HMODULE d3d12Module = LoadLibraryW(L"d3d12.dll");
    if (d3d12Module)
    {
        auto d3d12CreateDevice = reinterpret_cast<HRESULT(WINAPI*)(IUnknown*, D3D_FEATURE_LEVEL, REFIID, void*)>(
            GetProcAddress(d3d12Module, "D3D12CreateDevice")
            );
        if (d3d12CreateDevice)
        {
            THROW_IF_FAILED(d3d12CreateDevice(adapter.Get(), D3D_FEATURE_LEVEL_1_0_CORE, IID_PPV_ARGS(&d3dDevice)));
        }
    }
}
// Create the DML Device and D3D12 Command Queue
ComPtr<IDMLDevice> dmlDevice;
ComPtr<ID3D12CommandQueue> commandQueue;
if (d3dDevice)
{
    D3D12_COMMAND_QUEUE_DESC queueDesc = {};
    queueDesc.Type = D3D12_COMMAND_LIST_TYPE_COMPUTE;
    THROW_IF_FAILED(d3dDevice->CreateCommandQueue(
        &queueDesc,
        IID_PPV_ARGS(commandQueue.ReleaseAndGetAddressOf())));
    HMODULE dmlModule = LoadLibraryW(L"DirectML.dll");
    if (dmlModule)
    {
        auto dmlCreateDevice = reinterpret_cast<HRESULT(WINAPI*)(ID3D12Device*, DML_CREATE_DEVICE_FLAGS, DML_FEATURE_LEVEL, REFIID, void*)>(
            GetProcAddress(dmlModule, "DMLCreateDevice1")
            );
        if (dmlCreateDevice)
        {
            THROW_IF_FAILED(dmlCreateDevice(d3dDevice.Get(), DML_CREATE_DEVICE_FLAG_NONE, DML_FEATURE_LEVEL_5_0, IID_PPV_ARGS(dmlDevice.ReleaseAndGetAddressOf())));
        }
    }
}
```

DirectML Example (NPU Inference) (Cont.)



```
void main()
{
    ComPtr<ID3D12Device1> d3dDevice;
    ComPtr<IDMLDevice> dmlDevice;
    ComPtr<ID3D12CommandQueue> commandQueue;
    InitializeDirectML(d3dDevice.GetAddressOf(), commandQueue.GetAddressOf(), dmlDevice.GetAddressOf());

    // Add the DML execution provider to ORT using the DML Device and D3D12 Command Queue created above.
    if (!dmlDevice)
    {
        printf("No NPU device found\n");
        return;
    }

    const OrtApi& ortApi = Ort::GetApi();
    static Ort::Env s_OrtEnv{ nullptr };
    s_OrtEnv = Ort::Env(Ort::ThreadingOptions{});
    s_OrtEnv.DisableTelemetryEvents();

    auto sessionOptions = Ort::SessionOptions{};
    sessionOptions.DisableMemPattern();
    sessionOptions.DisablePerSessionThreads();
    sessionOptions.SetExecutionMode(ExecutionMode::ORT_SEQUENTIAL);
    const OrtDmlApi* ortDmlApi = nullptr;
    Ort::ThrowOnError(ortApi.GetExecutionProviderApi("DML", ORT_API_VERSION, reinterpret_cast<const void**>(&ortDmlApi)));
    Ort::ThrowOnError(ortDmlApi->SessionOptionsAppendExecutionProvider_DML1(sessionOptions, dmlDevice.Get(), commandQueue.Get()));
```

DirectML Example (NPU Inference) (Cont.)



```cpp
// Run performance test
constexpr int fenceValueStart = 2;
constexpr int numIterations = 100;
for (int i = fenceValueStart; i < (numIterations + fenceValueStart); i++)
{
    session.Run(Ort::RunOptions{ nullptr }, &inputName, &inputTensor, 1, &outputName, &outputTensor, 1);

    {
        // Synchronize with CPU before queuing more inference runs
        THROW_IF_FAILED(commandQueue->Signal(fence.Get(), i));
        THROW_HR_IF(E_FAIL, ResetEvent(fenceEvent.get()) == 0);
        THROW_IF_FAILED(fence->SetEventOnCompletion(i, fenceEvent.get()));
        THROW_HR_IF(E_FAIL, WaitForSingleObject(fenceEvent.get(), INFINITE) != WAIT_OBJECT_0);
    }
}
```

DirectML Example (Super Resolution)

DirectML Super Resolution

This sample demonstrates the DirectML API by implementing a super-resolution machine learning (ML) model on the GPU. This includes converting between image and tensor formats, initializing and executing ML operators, and interleaving graphics and ML work. The ML model performs a smart upscale of an image to double its original resolution. For example, it can scale a 540p image to 1080p.

DirectML Example (Super Resolution) (Cont.)

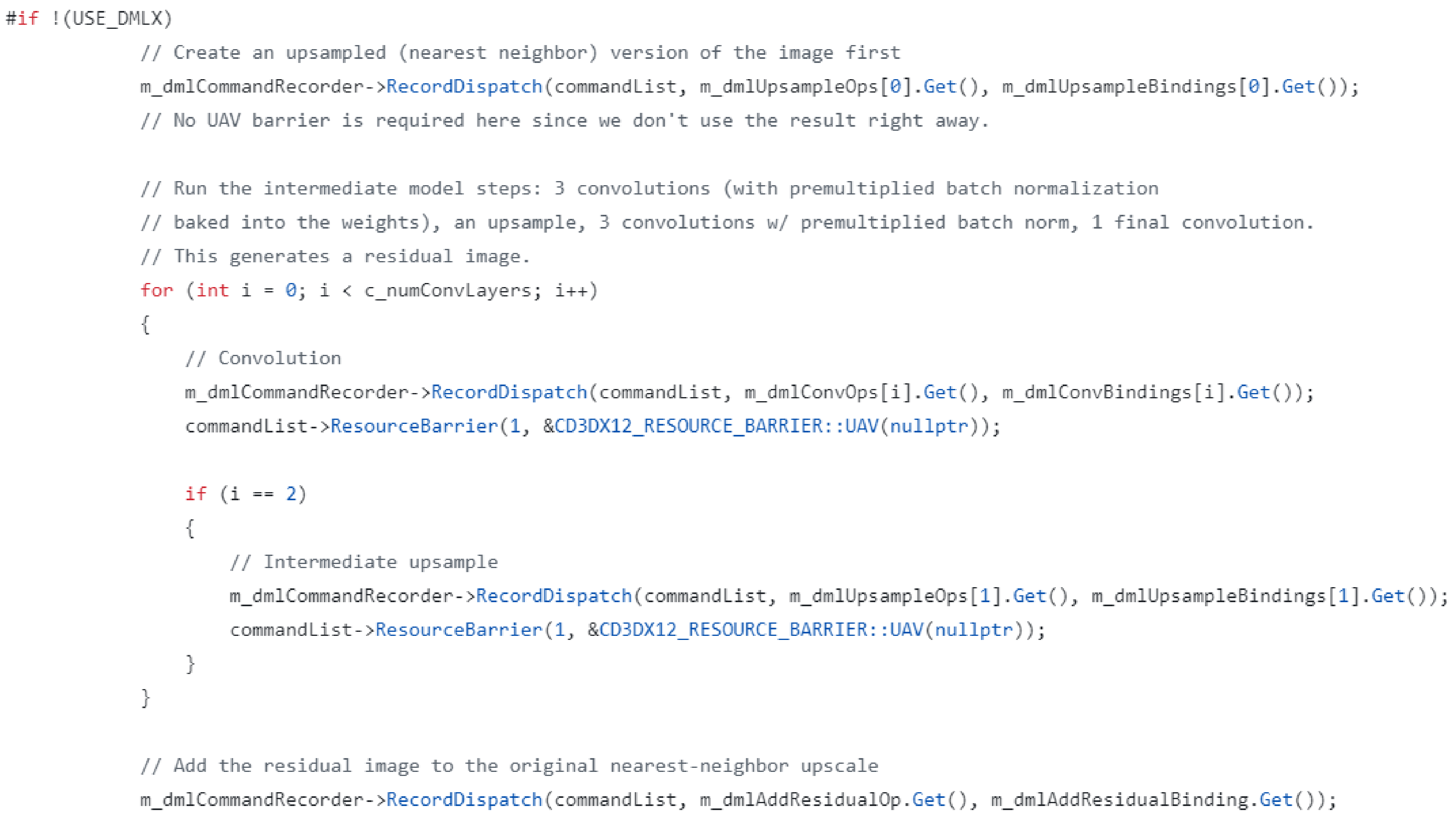
The primary purpose of this sample is to demonstrate usage of the DirectML API for real-time graphics processing. The particular super-resolution model is rather heavyweight, so a GPU and driver with dedicated ML support is recommended to run at higher framerates:

- CreateDeviceDependentResources: This calls into several methods to create and initialize the resources required to run the ML model and render the results. In particular, CreateDirectMLResources creates all the DML operators that comprise the model, as well as the D3D resources used to store intermediate results, and shaders to convert between texture and tensor formats. InitializeDirectMLResources binds the D3D resources to the operators and runs operator initialization.
- Render: If DirectML processing is enabled, this runs the operators using their bound resources to perform the super-resolution method on the current video frame. When DML is disabled, it uses a simple bilinear upscale. The result is rendered to the screen.

*Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

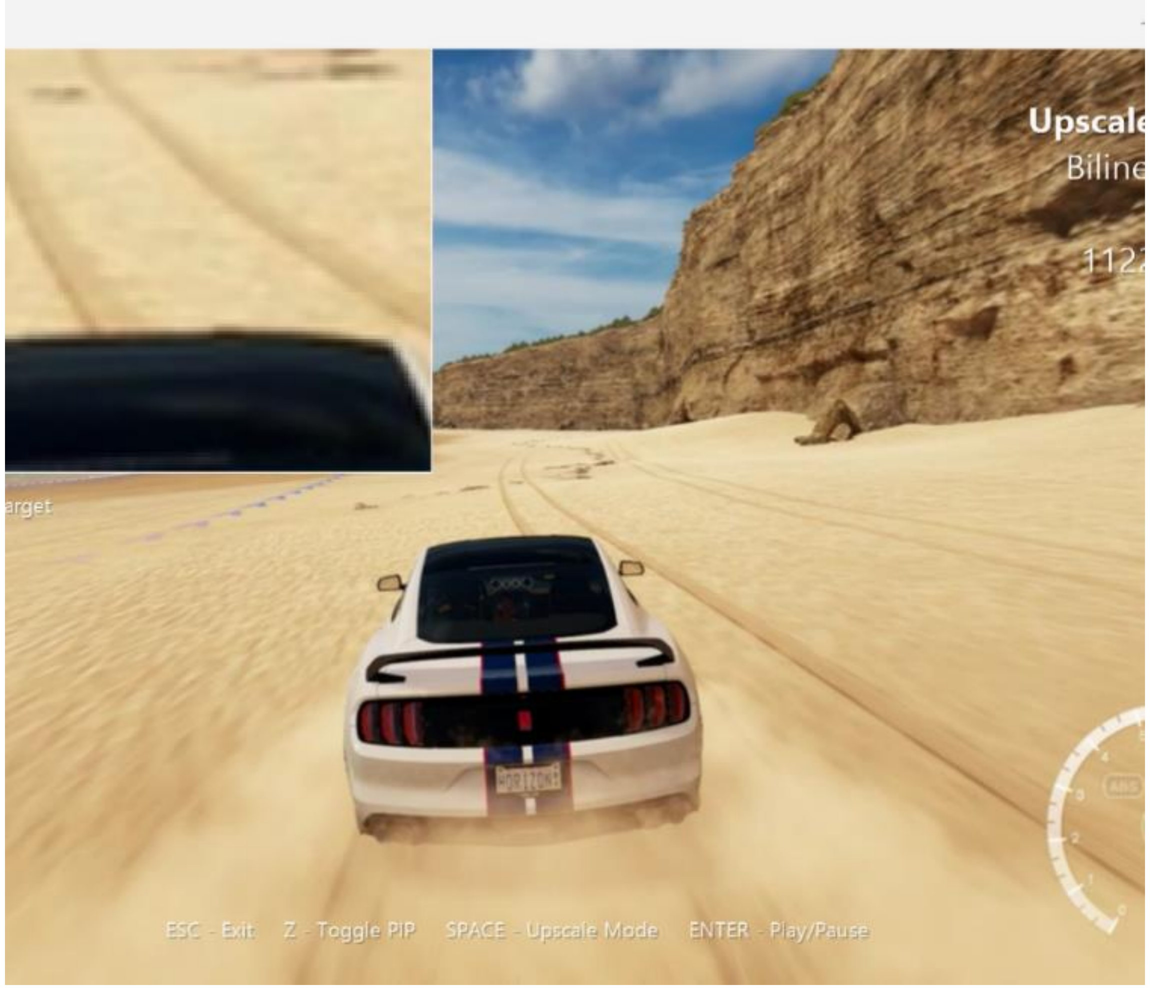
DirectML Example (Super Resolution) (Cont.)

- Image to Tensor
- Get residual image
- Tensor to image

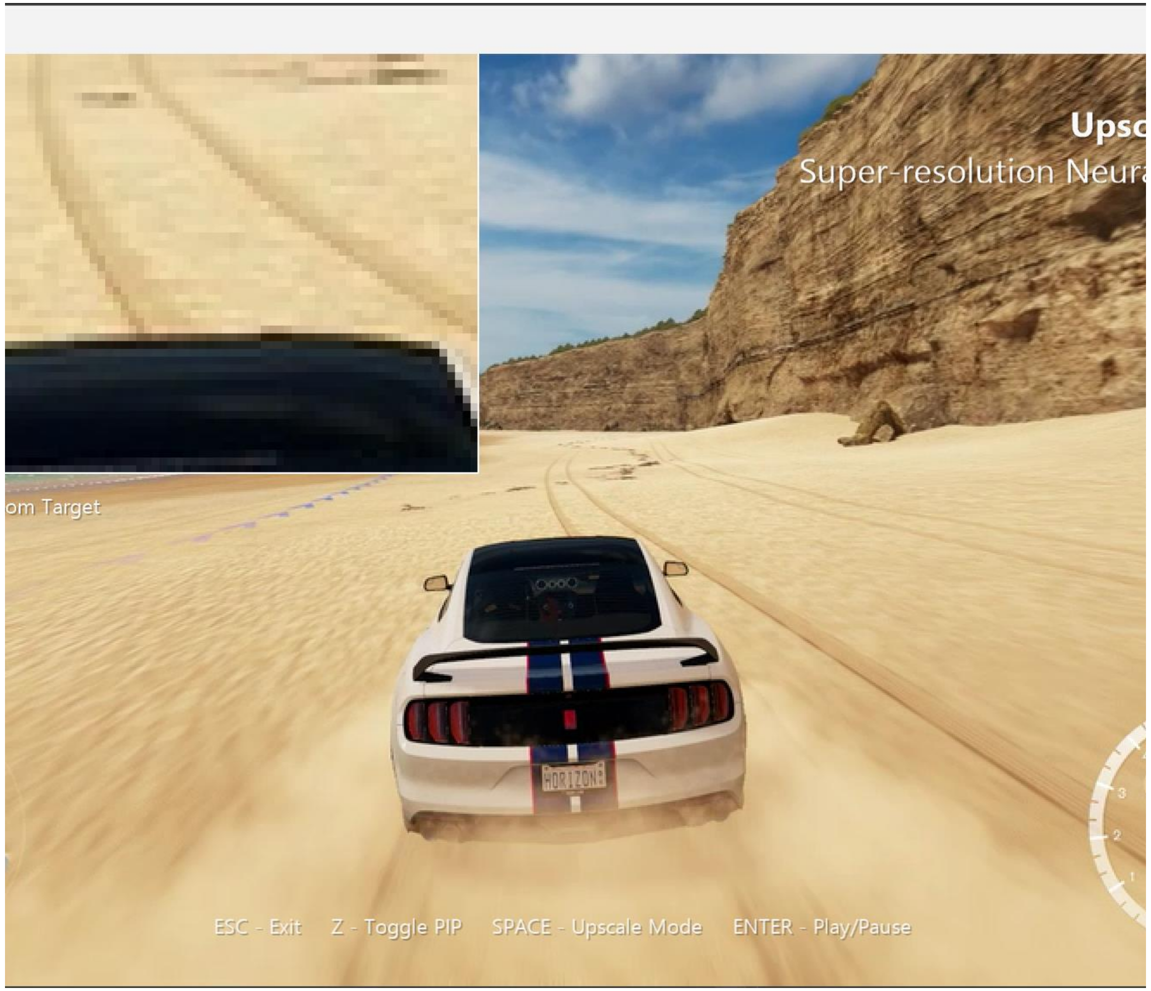


```
#if !(USE_DMLX)
            // Create an upsampled (nearest neighbor) version of the image first
            m_dmlCommandRecorder->RecordDispatch(commandList, m_dmlUpsampleOps[0].Get(), m_dmlUpsampleBindings[0].Get());
            // No UAV barrier is required here since we don't use the result right away.

            // Run the intermediate model steps: 3 convolutions (with premultiplied batch normalization
            // baked into the weights), an upsample, 3 convolutions w/ premultiplied batch norm, 1 final convolution.
            // This generates a residual image.
            for (int i = 0; i < c_numConvLayers; i++)
            {
                // Convolution
                m_dmlCommandRecorder->RecordDispatch(commandList, m_dmlConvOps[i].Get(), m_dmlConvBindings[i].Get());
                commandList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::UAV(nullptr));

                if (i == 2)
                {
                    // Intermediate upsample
                    m_dmlCommandRecorder->RecordDispatch(commandList, m_dmlUpsampleOps[1].Get(), m_dmlUpsampleBindings[1].Get());
                    commandList->ResourceBarrier(1, &CD3DX12_RESOURCE_BARRIER::UAV(nullptr));
                }
            }

            // Add the residual image to the original nearest-neighbor upscale
            m_dmlCommandRecorder->RecordDispatch(commandList, m_dmlAddResidualOp.Get(), m_dmlAddResidualBinding.Get());
```

DirectML Example (Super Resolution) (Cont.)



Upscale
Biline
1122



Upsc
Super-resolution Neura

*Other names and brands may be claimed as the property of others. All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.