# Technical Design Specification: Enabling ROCm Support in TorchSharp

## 1. Architectural Overview and Design Strategy

This document provides the technical design specification for extending the.NET TorchSharp library to support the AMD ROCm compute platform, leveraging the existing ROCm (HIP) backend within the underlying C++ LibTorch library. The primary objective is to define a robust engineering path for native Windows 11 support, with a stable alternative provided for the Windows Subsystem for Linux (WSL2).

This specification is predicated on a critical design principle: the TorchSharp C# API will *not* be forked or significantly altered. The underlying LibTorch library intentionally abstracts AMD's HIP runtime behind the existing torch::cuda namespace. Therefore, this project is primarily an exercise in C++ dependency compilation and dynamic C# library loading, not a porting of the C# API.

### 1.1. Analysis of the TorchSharp Three-Layer Architecture

A precise understanding of the existing TorchSharp architecture is essential before specifying modifications. The library operates on a three-layer model, separating the.NET-facing API from the native C++ computation.

#### 1.1.1. Layer 1: C# Frontend (TorchSharp.dll)

This is the public-facing API consumed by.NET developers.

* **API Semantics:** The API is designed to semantically mimic the Python PyTorch API, prioritizing a familiar experience for data scientists and researchers over strict.NET naming conventions.1
* **Structure:** It is composed of C# static classes (e.g., torch, torch.nn, torch.nn.functional) that mirror the PyTorch Python module hierarchy.2
* **Interop Mechanism:** This layer interfaces with the native C++ interop layer (Layer 2) using.NET's Platform Invoke (P/Invoke) mechanism. It uses `` or the modern [LibraryImport] attributes to call C-style function exports from the native wrapper library.1

#### 1.1.2. Layer 2: C++ Interop (LibTorchSharp.Native.dll / .so)

This is the "translation" or "glue" layer that bridges the.NET and C++ worlds.

* **Native C-style ABI:** This is a native C++ project (not C++/CLI) compiled into a shared library.7 It exposes a stable, C-style Application Binary Interface (ABI) that P/Invoke can consume.9
* Function Mapping: This layer adapts the object-oriented libtorch C++ API into a flat C-style API. For example, it defines a C-style exported function:  
  EXPORT\_API bool THS\_Cuda\_is\_available()
* Implementation: The implementation of this function directly calls the corresponding libtorch C++ API:  
  bool THS\_Cuda\_is\_available() { return torch::cuda::is\_available(); }  
  (This is inferred from the API design and common P/Invoke patterns for C++ libraries 10).

#### 1.1.3. Layer 3: C++ Backend (libtorch)

This is the official, high-performance C++ library distributed by the PyTorch team, containing the core tensor (ATen) and autograd logic.12

* **Distribution:** TorchSharp bundles these pre-compiled binaries (e.g., torch\_cpu.dll, torch\_cuda.dll, c10.dll) within its platform-specific NuGet packages.1
* **Compute Kernels:** This layer contains the actual computation kernels. It is this layer that already possesses the latent, compile-time capability to support ROCm in place of CUDA.13

### 1.2. Core Design Principle: Leveraging the torch::cuda Namespace as a HIP Abstraction Layer

This is the foundational principle of this entire specification. The PyTorch development team has intentionally abstracted the AMD HIP (Heterogeneous-compute Interface for Portability) backend behind the *existing* torch::cuda namespace.16

When the libtorch C++ library (Layer 3) is compiled from source with the USE\_ROCM=1 build flag 18, its C++ API calls are "HIP-ified".20 Calls such as torch::cuda::is\_available() and torch::Device(torch::kCUDA) are transparently redirected to call the corresponding AMD HIP runtime functions instead of the NVIDIA CUDA runtime.16

Critically, the C# consumer, the C++ interop layer, and even the libtorch C++ API consumer must continue to use the device string "cuda" (e.g., torch.device("cuda:0")), the torch::kCUDA enum, and the torch::cuda namespace.16 Requesting device "rocm" or "hip" is explicitly documented as *invalid* and will fail.16

### 1.3. Strategic Objective: A Backend-Agnostic Build and Dynamic Loader

Based on the core principle in 1.2, the project scope is dramatically simplified. No new TorchSharp.torch.rocm API is required. The entire existing C# torch.cuda API, the C++ LibTorchSharp.Native interop function signatures, and their C++ implementations will remain *unchanged*.

The strategic objective is therefore twofold:

1. **Build:** Define a reproducible build process that compiles Layer 3 (libtorch) from source with USE\_ROCM=1, and then compiles Layer 2 (LibTorchSharp.Native) by linking it against this new ROCm-enabled libtorch.
2. **Load:** Modify Layer 1 (TorchSharp.dll) to implement a robust, runtime dynamic library loading mechanism. This mechanism must detect the user's hardware (CPU-only, NVIDIA CUDA, or AMD ROCm) and load the *correct set* of native Layer 2 and Layer 3 binaries from the appropriate NuGet package at runtime.

This design transfers the project risk. The risk is *not* in C# API development but in the complexity of creating and maintaining a C++ build-from-source pipeline for libtorch with ROCm support on Windows, a component for which the PyTorch team does not yet provide an official C++ binary release.23

## 2. Target Environment Analysis and Setup

This specification defines two distinct deployment paths. These paths are mutually exclusive and have significant differences in stability, performance, and the availability of pre-compiled dependencies.

### 2.1. Path A: Native Windows 11 (Experimental / Preview)

This is the user's preferred path, leveraging AMD's recent and ongoing effort to support the ROCm stack and AI frameworks natively on Windows 11.25 This is a "preview" solution and must be treated as bleeding-edge.26

#### 2.1.1. Host System Requirements

The following requirements are *strict* and non-negotiable for this path:

* **Operating System:** Windows 11 22H2 or later.30
* **Supported Hardware:** AMD Radeon 7000 and 9000 Series (RDNA 3/4) GPUs, and select Ryzen AI 300 / AI Max APUs (RDNA 3.5).26
* **Driver:** The specialized "AMD Software: PyTorch on Windows Preview Edition" driver (e.g., 25.20.01.14 or newer).26 The standard Adrenalin gaming driver is insufficient.

#### 2.1.2. Build System Environment

The official AMD-sanctioned build system for this platform is the ROCm/TheRock unified build repository.25 This CMake super-project is responsible for building all ROCm components and external projects (like PyTorch) from source on Windows.32

The build environment prerequisites are extensive and specified in the windows\_support.md document 36, including:

* Visual Studio 2022 (MSVC) Build Tools
* CMake (version < 4.0.0 is noted 36)
* Ninja
* Git (with optional Unix tools added to PATH)
* Python 3.12+ 37
* StrawberryPerl

#### 2.1.3. Critical Dependency Gap

A critical gap exists for this path: **no pre-built LibTorch (C++) library for ROCm on Windows is available.**

* The official PyTorch website (pytorch.org) download matrix for LibTorch on Windows only offers "CPU" or "CUDA" compute platforms.23
* AMD's ROCm/TheRock project currently focuses on producing and publishing Python *wheels* (.whl files) for consumption by Python applications.33
* This gap is explicitly confirmed in ROCm/TheRock's own documentation: "This initial effort is focused on publishing torch wheels, but we will also be working towards publishing builds of libtorch on Windows as time goes by".25

**Conclusion:** For Path A (Native Windows 11), the libtorch C++ library (Layer 3) *must* be built from source. This is the primary engineering hurdle for this path.

### 2.2. Path B: Windows Subsystem for Linux (WSL2) (Stable / Recommended)

This is the user's specified fallback path and is the most stable, mature, and well-documented method for using ROCm on a Windows machine.29

#### 2.2.1. Host System Requirements

* **Windows Host:** Windows 10 or 11 with WSL2 and "Virtual Machine Platform" features enabled.41
* **WSL Distribution:** **Ubuntu 22.04 (Jammy Jellyfish)**. This specific version is strongly recommended and is the only version validated in many official guides.41 Using other versions is at-risk.41
* **Windows Driver:** A recent "AMD Software: Adrenalin Edition" driver that includes support for WSL2 (e.g., 25.8.1 or newer).44

#### 2.2.2. WSL2 Instance Setup (Prescriptive)

The following commands must be run on the Windows host and within the Ubuntu 22.04 instance.

1. **On Windows Host (PowerShell):** Install and update WSL.  
   PowerShell  
   wsl --install -d Ubuntu-22.04  
   wsl --update  
     
   (The wsl --update step is crucial for ensuring the latest GPU-compatible kernel 48).
2. **Inside WSL2 (Ubuntu 22.04):** Install the AMD repository.  
   Bash  
   sudo apt update  
   wget https://repo.radeon.com/amdgpu-install/6.1.3/ubuntu/jammy/amdgpu-install\_6.1.60103-1\_all.deb  
   sudo apt install./amdgpu-install\_6.1.60103-1\_all.deb  
     
   (Note: The specific version 6.1.3 is used here as an example 45; the latest compatible version should be used).
3. **Inside WSL2 (Ubuntu 22.04):** Install the ROCm driver stack using the specific WSL use case.  
   Bash  
   sudo amdgpu-install -y --usecase=wsl,rocm --no-dkms  
     
   (This --usecase=wsl,rocm flag is critical. It installs the user-mode ROCm libraries while --no-dkms correctly skips the kernel-mode driver installation, which is handled by the Windows host driver 44).

#### 2.2.3. Verification (Inside WSL2)

* Run rocminfo. This command *must* succeed and list the AMD GPU.49 If it fails, the environment is not set up correctly.48
* **Note:** The rocm-smi utility is *not* supported on WSL2 due to architectural limitations and is expected to fail.48 This is not an error.

#### 2.2.4. Critical Dependency Advantage

This path has a significant advantage over native Windows: **pre-built LibTorch (C++) binaries are available.**

* The official PyTorch website (pytorch.org) *does* provide pre-built LibTorch packages for ROCm on Linux.23
* **Conclusion:** For Path B (WSL2), the libtorch-rocm-linux-\*.zip package can be downloaded directly. This allows us to *skip* the complex and time-consuming "Build LibTorch from Source" step (Section 3.1), making this path significantly more stable and easier to maintain.

### 2.3. ROCm Component Version-Compatibility Matrix

The integration of these distinct, rapidly-evolving software stacks (OS, driver, ROCm, LibTorch, TorchSharp) is highly sensitive to version mismatches. Mismatches are the most common source of failure, leading to "dependency hell" scenarios where, for example, a specific TorchSharp version requires a libtorch version 53 that is not compatible with the installed ROCm driver 20, or where CUDA/PyTorch versions conflict.55

The following table provides a "golden path" of component versions that are validated to work together. This table should be the primary troubleshooting checklist.

**Table 1: ROCm Component Version-Compatibility Matrix**

| **Component** | **Path A: Native Windows 11 (Preview)** | **Path B: WSL2 (Stable)** |
| --- | --- | --- |
| **Host OS** | Windows 11 (22H2+) 30 | Windows 10 or 11 (WSL2 Enabled) 41 |
| **Host GPU Driver** | AMD Software: PyTorch on Windows Preview (e.g., 25.20.01.14+) 26 | AMD Software: Adrenalin Edition (WSL2-compat, e.g., 25.8.1+) 44 |
| **Guest OS** | N/A | Ubuntu 22.04 (Jammy) 44 |
| **Compute Stack** | ROCm 7.1 / 7.9-preview (Built from ROCm/TheRock) 32 | ROCm 6.1.3 / 6.4.2+ (Installed via amdgpu-install --usecase=wsl) 44 |
| **Supported GPUs** | Radeon 7000/9000 Series, Ryzen AI 300+ 26 | Radeon 7000 Series (RDNA3) 31 (Note: 7000s validated, others may work) |
| **libtorch (C++) Source** | **Build from Source (MANDATORY)** 25 | **Download Pre-built Binary** (from pytorch.org) 23 |
| **libtorch Version** | e.g., PyTorch 2.8.0+ (ROCm 7.0+) 54 | e.g., PyTorch 2.6.0+ (ROCm 6.4+) 23 |
| **TorchSharp Version** | 0.10x.x (Tied to libtorch ABI) 53 | 0.10x.x (Tied to libtorch ABI) 53 |

## 3. Component Build Specification: Native Dependencies (C++)

This section details the prescriptive process for compiling the native C++ libraries (Layers 2 and 3). This entire process is *mandatory* for the Native Windows 11 path. For the WSL2 path, only Section 3.2 is required if using the pre-built libtorch from Section 2.2.4.

### 3.1. Building the LibTorch (C++) Backend (from source)

This process is complex and must be performed on the target native Windows 11 machine.

#### 3.1.1. Source Acquisition & Environment Setup (Windows 11)

1. Clone the ROCm/TheRock unified build repository:  
   git clone https://github.com/ROCm/TheRock.git 33
2. cd TheRock
3. Install all build prerequisites as specified in docs/development/windows\_support.md.33 This includes VS 2022, CMake, Ninja, Git, Python 3.12, and StrawberryPerl.36 Enable long file paths and developer mode.36
4. Set the command prompt to UTF-8:  
   chcp 65001 33
5. Initialize the Python virtual environment:  
   python -m venv.venv  
   .venv\Scripts\Activate.bat 33
6. Install Python dependencies:  
   pip install --upgrade pip  
   pip install -r requirements.txt 33
7. Fetch all ROCm submodules and the PyTorch source code:  
   python./build\_tools/fetch\_sources.py 33

#### 3.1.2. Build Configuration & Execution

The ROCm/TheRock repository is a CMake super-project.33 We will not build PyTorch directly, but will invoke TheRock's scripts to build its external-builds/pytorch component.25

1. **Set Critical Environment Variables:** These must be set prior to build invocation.  
   DOS  
   set USE\_ROCM=1  
   set PYTORCH\_ROCM\_ARCH=gfx1100  
     
   (Note: USE\_ROCM=1 is the master flag.19 PYTORCH\_ROCM\_ARCH must be set to the user's specific GPU architecture, e.g., gfx1100 for an RX 7900 XTX 58).
2. Isolating the LibTorch (C++ API) Build:  
   The default ROCm/TheRock build scripts are designed to produce Python wheels.60 This specification requires a modification to produce only the C++ libtorch distributable package.  
   The ROCm/TheRock/external-builds/pytorch/build\_pytorch.py script 40 (or a new script based on tools/build\_libtorch.py from the upstream PyTorch repo 62) must be created or modified to support a --libtorch\_only flag.  
   When this flag is present, the script *must* execute the following logic:
   * **Configure CMake:** Set the necessary CMake variables to build *only* the C++ library, disabling Python:
     + -DBUILD\_SHARED\_LIBS=ON
     + -DUSE\_ROCM=1
     + -DPYTORCH\_ROCM\_ARCH=%PYTORCH\_ROCM\_ARCH%
     + -DBUILD\_PYTHON=OFF
     + -DCMAKE\_INSTALL\_PREFIX=../pytorch-install  
       (This configuration is based on standard libtorch build practices 63).
   * Build and Install: Execute the CMake build and install targets:  
     cmake --build. --target install 63
   * **Result:** This process will populate the ../pytorch-install directory with the required lib/, include/, and share/cmake/ folders.63 This directory *is* the libtorch-rocm-windows package and is the required input for Section 3.2.
   * The script must then exit *before* attempting to build any Python wheels.

### 3.2. Building the LibTorchSharp.Native (C++) Interop Layer

This step compiles the "glue" library (Layer 2) and links it against the ROCm-enabled libtorch (Layer 3) from Section 3.1 (or downloaded for WSL2).

1. Source Acquisition:  
   git clone https://github.com/dotnet/TorchSharp.git
2. **src/Native/CMakeLists.txt Modifications:** This file 67 is the primary target for modification. The following changes are mandatory.

#### 3.2.1. Locate Custom LibTorch Package

The CMakeLists.txt file uses find\_package(Torch REQUIRED).68 We must provide the path to our newly-built libtorch package from 3.1.

CMake

# Add this near the top, pointing to the 'pytorch-install' directory  
set(CMAKE\_PREFIX\_PATH "C:/path/to/pytorch-install/libtorch" CACHE STRING "Path to LibTorch C++ distribution")

(This method is standard for consuming local libtorch builds 66).

#### 3.2.2. Set ROCm Build Flags

We must inform this CMake project that it is building against a ROCm backend.

CMake

# Add these flags to control the build logic  
set(USE\_ROCM ON CACHE BOOL "Enable ROCm support")  
set(USE\_CUDA OFF CACHE BOOL "Disable CUDA support")

(These variables are used by libtorch's own CMake scripts, which we will consume 71).

#### 3.2.3. Critical Build-Order Dependencies: Discovering HIP

A standard build will fail at the linking stage.71 The LibTorchSharp.Native (Layer 2) build will call find\_package(Torch) (Layer 3), which will return a list of ROCm/HIP dependencies (e.g., roc::rocblas). The LibTorchSharp.Native CMake instance will then fail because it does not know what roc::rocblas is or where to find it.

To resolve this, we must discover the HIP SDK *before* discovering libtorch.

CMake

# Add this \*before\* the 'find\_package(Torch REQUIRED)' call  
if (USE\_ROCM)  
 # Add standard ROCm install paths to the search prefix  
 # On Windows, this is typically set by the HIP\_PATH env var  
 # On Linux, it's /opt/rocm  
 list(APPEND CMAKE\_PREFIX\_PATH $ENV{HIP\_PATH} "/opt/rocm/hip" "/opt/rocm") [73]  
  
 # Find the HIP SDK and its components  
 find\_package(HIP REQUIRED) [73, 74, 75]  
endif()  
  
# This call will now succeed  
find\_package(Torch REQUIRED) [68]

This ordering is the key to resolving the linker errors. find\_package(HIP) populates the CMake environment with the locations of all roc::\* targets, allowing find\_package(Torch) to successfully resolve its ROCm-based dependencies.

#### 3.2.4. Target Linkage Specification

The existing linkage line in CMakeLists.txt is sufficient and requires no changes:

CMake

target\_link\_libraries(LibTorchSharp.Native PUBLIC ${TORCH\_LIBRARIES})

(This is based on standard libtorch CMake examples 69). Because of the steps in 3.2.2 and 3.2.3, the ${TORCH\_LIBRARIES} variable will now be correctly populated by TorchConfig.cmake with all required HIP and ROCm libraries (e.g., hip::host, roc::rocblas) instead of CUDA libraries.71

If any custom HIP kernels (.hip files) are added to LibTorchSharp.Native in the future, enable\_language(HIP) must also be added to the CMakeLists.txt.76

## 4. Architectural Modification: C# Wrapper (TorchSharp.dll)

This section details the C# code modifications required to dynamically load the new ROCm backend at runtime.

### 4.1. API Integrity and Backend Abstraction

As established in Section 1.2, the public C# API *must not change*.

* TorchSharp.torch.cuda.is\_available() remains the single method to check for *any* GPU support (NVIDIA or AMD).3
* TorchSharp.torch.device("cuda:0") remains the device string to select the default AMD or NVIDIA GPU.1

This design is a direct parallel to the underlying libtorch HIP abstraction, which reuses the cuda identifier.16

#### 4.1.1. Design Decision: Resolving kCUDA vs. kHIP

A superficial inspection of the libtorch C++ source code (e.g., c10/core/Device.h) reveals distinct c10::DeviceType::CUDA and c10::DeviceType::HIP enum values.81 This may tempt an engineer to modify TorchSharp to expose and send a new DeviceType.HIP.

This would be an architectural error.

The HIP-ification layer within libtorch 20 *maps* the public-facing kCUDA device type to the *internal* kHIP device type at compile time when USE\_ROCM=1 is set. The public C++ API remains torch::cuda::is\_available() 16, which is what LibTorchSharp.Native calls. Therefore, this design specification mandates that the TorchSharp C# layer *must* continue to use DeviceType.CUDA exclusively. The abstraction is handled *inside* libtorch, not by the caller.

### 4.2. Dynamic Backend Loading Mechanism

The current loading mechanism in TorchSharp.torch's static constructor (.cctor()) 77 is insufficient. It relies on internal methods like LoadNativeBackend 77 that use simple NativeLibrary.Load calls with hard-coded library names (e.g., "torch\_cpu", "LibTorchSharp").1

This hard-coded logic fails in complex environments 53 and provides no mechanism to select between different backend implementations (e.g., LibTorchSharp.Native.cuda.dll and LibTorchSharp.Native.rocm.dll).

### 4.3. Prescribed Implementation: Dynamic P/Invoke Resolution

The current LoadNativeBackend logic will be replaced with a robust, centralized resolver using System.Runtime.InteropServices.NativeLibrary.SetDllImportResolver. This is the modern.NET pattern for runtime-aware native library loading.86

Step 1: Register the Resolver

In the TorchSharp.torch static class constructor (.cctor()), a single resolver will be registered for the assembly:

C#

// Inside static torch()  
NativeLibrary.SetDllImportResolver(Assembly.GetExecutingAssembly(), DllImportResolver);

88

Step 2: Implement the Probe Logic

A one-time hardware probe will be implemented. This logic runs once and its result is cached statically.

C#

private enum BackendType { CPU, CUDA, ROCm }  
private static readonly BackendType \_detectedBackend = ProbeForBackend();  
  
private static BackendType ProbeForBackend()  
{  
 // Path 1: Check for explicit user override  
 // (e.g., Environment.GetEnvironmentVariable("TORCHSHARP\_BACKEND"))  
  
 // Path 2: Attempt to load AMD ROCm native driver/runtime  
 // On Windows, try to load the ROCm SMI library  
 // On Linux (WSL2), check for the AMD GPU kernel device  
 if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows)) {  
 if (NativeLibrary.TryLoad("amdsmi.dll", out \_)) {  
 return BackendType.ROCm;  
 }  
 } else {  
 if (File.Exists("/dev/kfd")) { // [59, 91]  
 return BackendType.ROCm;  
 }  
 }  
  
 // Path 3: Attempt to load NVIDIA CUDA native driver/runtime  
 // (e.g., "nvcuda.dll" or "nvapi64.dll" [92])  
 if (NativeLibrary.TryLoad("nvcuda.dll", out \_)) {  
 return BackendType.CUDA;  
 }  
  
 // Path 4: Fallback to CPU  
 return BackendType.CPU;  
}

Step 3: Implement the Resolver Delegate (The "Map")

This delegate will be implemented as defined in.87 It will be called by the.NET runtime for each [LibraryImport] and will map the generic library name to a concrete, backend-specific filename.

C#

private static IntPtr DllImportResolver(string libraryName, Assembly assembly, DllImportSearchPath? searchPath)  
{  
 string concreteName = libraryName;  
  
 // Map the generic 'LibTorchSharp' to its backend-specific version  
 if (libraryName == "LibTorchSharp") {  
 concreteName = \_detectedBackend switch {  
 BackendType.ROCm => "LibTorchSharp.Native.rocm"[85, 93]  
 BackendType.CUDA => "LibTorchSharp.Native.cuda",  
 \_ => "LibTorchSharp.Native.cpu"  
 };  
 }  
   
 // Map the generic 'torch\_cuda' library to its backend-specific version  
 if (libraryName == "torch\_cuda") {  
 concreteName = \_detectedBackend switch {  
 BackendType.ROCm => "torch\_rocm", // Or the name provided by the libtorch build  
 BackendType.CUDA => "torch\_cuda",  
 \_ => "torch\_cpu" // Fallback  
 };  
 }  
   
 // Use TryLoad with platform-specific extensions (.dll,.so)  
 // The runtime automatically handles platform suffixes [88]  
 if (NativeLibrary.TryLoad(concreteName, assembly, searchPath, out IntPtr handle)) {  
 return handle;  
 }  
  
 return IntPtr.Zero;  
}

This pattern 87 provides a flexible, runtime-aware solution that loads the correct native dependency chain based on the hardware present on the user's machine.

### 4.4. New Binary Packaging and Distribution

To support the new loading logic, the build system and NuGet packaging must be updated.

1. **New Native Binaries:** The build process (Section 3.2) will be modified to output:
   * LibTorchSharp.Native.rocm.dll (from the Windows build)
   * libLibTorchSharp.Native.rocm.so (from the Linux build for WSL2)
2. **New libtorch Package:** A new package, libtorch-rocm-win-x64, will be created. This package will contain the C++ build artifacts from Section 3.1.
3. **New Top-Level NuGet Packages:** Two new "bundle" packages will be created to mirror the existing structure.1
   * **TorchSharp-rocm-windows:** This package will depend on the base TorchSharp package and the new libtorch-rocm-win-x64 package. It will contain the runtimes/win-x64/native/LibTorchSharp.Native.rocm.dll binary.
   * **TorchSharp-rocm-linux:** This package will depend on the base TorchSharp package and the existing libtorch-rocm-linux-x64 package. It will contain the runtimes/linux-x64/native/libLibTorchSharp.Native.rocm.so binary.

## 5. Verification, Testing, and Continuous Integration

The project is not complete until it is stable, verifiable, and automatically tested.

### 5.1. Unit and Integration Test Plan (Definition of Done)

This test plan must be executed on a machine with a compatible AMD GPU and the correct ROCm driver stack (either Native or WSL2).

* **Test Case 1: Backend Initialization**
  + **Test:** TorchSharp.torch.cuda.is\_available()
  + **Expected Result:** Must return true.16 This confirms the ROCm backend was successfully loaded by the C# resolver and initialized by libtorch.
* **Test Case 2: Device Enumeration**
  + **Test:** TorchSharp.torch.cuda.device\_count()
  + **Expected Result:** Must return an integer > 0, matching the number of installed AMD GPUs.101
* **Test Case 3: Device Identification (Critical Test)**
  + **Test:** TorchSharp.torch.cuda.get\_device\_name(0)
  + **Expected Result:** Must return the string name of the AMD GPU (e.g., "Radeon RX 7900 XTX").37 This is the definitive test that TorchSharp is communicating with the AMD ROCm driver.
* **Test Case 4: Tensor Lifecycle and Operations**
  + **Test:**  
    C#  
    var device = torch.device("cuda:0");  
    var tensor = torch.rand(3, 3, device: device);  
    var cpu\_tensor = torch.rand(3, 3);  
    var gpu\_tensor = cpu\_tensor.to(device);  
    var result = tensor + gpu\_tensor;  
    var cpu\_result = result.cpu();
  + **Expected Result:** The code must execute without exceptions. The device property of tensor, gpu\_tensor, and result must all report their type as cuda and index as 0.16

### 5.2. Continuous Integration (CI) Pipeline Strategy

A staged CI strategy is required due to the disparity in build complexity between the two target paths.

#### 5.2.1. Path B (WSL2 Pipeline - High Priority / Stable)

This pipeline is practical and can be implemented on standard, cloud-hosted runners with Docker support.

* **Runner:** Standard GitHub Actions Ubuntu-hosted runner.
* **Workflow:**
  1. Pull the official AMD ROCm PyTorch Docker image:  
     docker pull rocm/pytorch:latest.33 This container already includes the ROCm driver stack and a compatible libtorch C++ library.
  2. Start the container with GPU access:  
     docker run... --device=/dev/kfd --device=/dev/dri --group-add video... 59
  3. **Inside Container:**
     + Check out the TorchSharp repository.
     + Run the src/Native/ build (Section 3.2), setting CMAKE\_PREFIX\_PATH to the libtorch directory *inside* the container.
     + Build the C# TorchSharp projects: dotnet build.
     + Execute the C# test plan: dotnet test.98

#### 5.2.2. Path A (Native Windows Pipeline - Aspirational / High-Cost)

This pipeline is logistically complex and expensive, and should be considered a long-term goal.

* **Runner:** This requires a **self-hosted GitHub Actions runner**.34 This runner must be a physical Windows 11 machine with a compatible AMD Radeon 7000/9000 series GPU 25 and the specific PyTorch Preview Driver.26
* **Workflow:**
  1. Install all ROCm/TheRock prerequisites (VS 2022, CMake, etc.).36
  2. Check out the ROCm/TheRock repository.33
  3. **Execute the full libtorch C++ build from source** (Section 3.1). This is a multi-hour step.25
  4. Check out the TorchSharp repository.
  5. Execute the LibTorchSharp.Native and C# builds (Section 3.2), linking against the libtorch built in the previous step.
  6. Execute the.NET test suite (Section 5.1) on the native AMD hardware.37

### 5.3. Conclusions and Staged Rollout Recommendation

The analysis of the build and CI pipeline complexity dictates a staged rollout for ROCm support.

The native Windows CI pipeline (Path A) is prohibitively expensive and slow for a standard open-source CI loop. It requires specialized, self-hosted hardware 28 and a multi-hour C++ compilation of the *entire ROCm stack and PyTorch* from source on every run.25

Conversely, the WSL2/Docker pipeline (Path B) is practical, fast, and maintainable. It leverages AMD's official, pre-built rocm/pytorch Docker images, which *already include* the compatible libtorch C++ library.59 The CI build time is reduced from hours to minutes, covering only the compilation of LibTorchSharp.Native and the.NET tests.

Based on this disparity, the following staged rollout is specified:

* **Phase 1 (Stable Release):** Implement and release TorchSharp-rocm-linux. This package will target the WSL2 environment. It is stable, aligns with AMD's official documentation, and can be fully validated by the practical CI pipeline defined in 5.2.1.
* **Phase 2 (Experimental Release):** Implement and release TorchSharp-rocm-windows. This package will be explicitly marked as "experimental." Its core C# logic will be regression-tested by the WSL2 CI pipeline. The full, native Windows CI pipeline (5.2.2) will remain an aspirational goal, contingent on AMD and the PyTorch team providing official LibTorch C++ binaries for ROCm on Windows. Such a release would eliminate the multi-hour compile step and make the native CI pipeline practical.25

#### Citerede værker

1. dotnet/TorchSharp: A .NET library that provides access to the library that powers PyTorch. - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp>
2. C# Deep Learning Framework TorchSharp: Native Model Training and Image Recognition, tilgået november 16, 2025, <https://www.whuanle.cn/archives/21760>
3. Help with implementing / learning TorchSharp. · Issue #980 - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/issues/980>
4. Using TorchSharp in Unity #874 - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/discussions/874>
5. Platform Invoke (P/Invoke) - .NET - Microsoft Learn, tilgået november 16, 2025, <https://learn.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke>
6. C# P/Invoke on CUDA DLL eventually causes AccessViolationException - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/3686750/c-sharp-p-invoke-on-cuda-dll-eventually-causes-accessviolationexception>
7. C++ interop CLI vs PInvoke? - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/8642620/c-interop-cli-vs-pinvoke>
8. P/Invoke or C++/CLI for wrapping a C library - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/2561898/p-invoke-or-c-cli-for-wrapping-a-c-library>
9. Merging LibTorchSharp in TorchSharp · Issue #79 - GitHub, tilgået november 16, 2025, <https://github.com/xamarin/TorchSharp/issues/79>
10. Torch::cuda::is\_available() returns false - C++ - PyTorch Forums, tilgået november 16, 2025, <https://discuss.pytorch.org/t/torch-is-available-returns-false/73753>
11. Torch::cuda::is\_available() always returns false - C++ - PyTorch Forums, tilgået november 16, 2025, <https://discuss.pytorch.org/t/torch-is-available-always-returns-false/221281>
12. Run PyTorch in .NET with TorchSharp - CPI Consulting, tilgået november 16, 2025, <https://cloudproinc.com.au/index.php/2025/09/21/run-pytorch-in-net-with-torchsharp/>
13. pytorch/pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration - GitHub, tilgået november 16, 2025, <https://github.com/pytorch/pytorch>
14. YAOZZJ/TorchSharpExamples - Gitee, tilgået november 16, 2025, <https://gitee.com/yaozzj/TorchSharpExamples>
15. dotnet/TorchSharpExamples: Repository for TorchSharp examples and tutorials. - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharpExamples>
16. HIP (ROCm) semantics — PyTorch 2.9 documentation, tilgået november 16, 2025, <https://docs.pytorch.org/docs/stable/notes/hip.html>
17. hip.rst.txt - PyTorch, tilgået november 16, 2025, <https://docs.pytorch.org/docs/2.9/_sources/notes/hip.rst.txt>
18. Building libtorch for IBM PLX - C++ - PyTorch Forums, tilgået november 16, 2025, <https://discuss.pytorch.org/t/building-libtorch-for-ibm-plx/52720>
19. Respect user-specified USE\_ROCM/USE\_CUDA · Issue #114388 - GitHub, tilgået november 16, 2025, <https://github.com/pytorch/pytorch/issues/114388>
20. PyTorch compatibility - AMD ROCm documentation, tilgået november 16, 2025, <https://rocm.docs.amd.com/en/docs-6.3.3/compatibility/pytorch-compatibility.html>
21. CUDA to HIP API Function Comparison - AMD ROCm documentation, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/HIP/en/latest/reference/api_syntax.html>
22. Is ROCm viable for ml development with PyTorch - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1ih1487/is_rocm_viable_for_ml_development_with_pytorch/>
23. Get Started - PyTorch, tilgået november 16, 2025, <https://pytorch.org/get-started/locally/>
24. PyTorch, tilgået november 16, 2025, <https://pytorch.org/>
25. [RFC] Enable native Windows CI/CD on ROCm · Issue #159520 - GitHub, tilgået november 16, 2025, <https://github.com/pytorch/pytorch/issues/159520>
26. AMD Software: PyTorch on Windows Preview Edition 25.20.01.14 Release Notes, tilgået november 16, 2025, <https://www.amd.com/en/resources/support-articles/release-notes/RN-AMDGPU-WINDOWS-PYTORCH-PREVIEW.html>
27. The Road to ROCm on Radeon for Windows and Linux - AMD, tilgået november 16, 2025, <https://www.amd.com/en/blogs/2025/the-road-to-rocm-on-radeon-for-windows-and-linux.html>
28. A beginner's guide to deploying LLMs with AMD on Windows using PyTorch, tilgået november 16, 2025, <https://gpuopen.com/learn/pytorch-windows-amd-llm-guide/>
29. ROCm + pyTorch native on Windows - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1enh0z3/rocm_pytorch_native_on_windows/>
30. System requirements (Windows) — HIP SDK installation (Windows) - AMD ROCm documentation, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/install-on-windows/en/latest/reference/system-requirements.html>
31. Use ROCm on Radeon and Ryzen, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/index.html>
32. Build the ROCm Core SDK from source — AMD ROCm 7.9.0 preview, tilgået november 16, 2025, <https://rocm.docs.amd.com/en/7.9.0-preview/install/build-from-source.html>
33. ROCm/TheRock: The HIP Environment and ROCm Kit - A lightweight open source build system for HIP and ROCm - GitHub, tilgået november 16, 2025, <https://github.com/ROCm/TheRock>
34. Windows platform support bringup · Issue #36 · ROCm/TheRock - GitHub, tilgået november 16, 2025, <https://github.com/nod-ai/TheRock/issues/36>
35. Build the ROCm Core SDK from source — AMD ROCm 7.9.0 preview, tilgået november 16, 2025, <https://rocm.docs.amd.com/en/docs-7.9.0/install/build-from-source.html>
36. tilgået november 16, 2025, <https://raw.githubusercontent.com/ROCm/TheRock/main/docs/development/windows_support.md>
37. PyTorch via PIP installation — Use ROCm on Radeon and Ryzen, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/docs/install/installrad/windows/install-pytorch.html>
38. AMD ROCm 6.4.4 Brings PyTorch Support On Windows For Radeon 9000, Radeon 7000 GPUs, & Ryzen AI APUs - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1npn4as/amd_rocm_644_brings_pytorch_support_on_windows/>
39. Install ROCm PyTorch on Windows with AMD Radeon (gfx1151/8060S) – Automated PowerShell Script - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1nffbzt/install_rocm_pytorch_on_windows_with_amd_radeon/>
40. [Announcement] Deleting published therock\_pytorch\_dev docker packages on 2025-11-10 · Issue #2025 · ROCm/TheRock - GitHub, tilgået november 16, 2025, <https://github.com/ROCm/TheRock/issues/2025>
41. How To Run AMD ROCm™ Software in Windows 11, tilgået november 16, 2025, <https://www.amd.com/en/developer/resources/technical-articles/how-to-run-amd-rocm-software-in-windows-11.html>
42. Installation help : r/ROCm - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1j3eyp5/installation_help/>
43. ROCm 6.4.0 (HIP SDK) is available for download on Windows - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1mq0lk0/rocm_640_hip_sdk_is_available_for_download_on/>
44. Install Radeon software for WSL with ROCm, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/docs/install/installrad/wsl/install-radeon.html>
45. ROCm 6.1.3 complete install instructions from WSL to pytorch - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1ep4cru/rocm_613_complete_install_instructions_from_wsl/>
46. Install PyTorch for ROCm — Use ROCm on Radeon and Ryzen, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/docs/install/installrad/wsl/install-pytorch.html>
47. WSL support matrices by ROCm version — Use ROCm on Radeon and Ryzen, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/docs/compatibility/compatibilityrad/wsl/wsl_compatibility.html>
48. Can't run ROCM on WIndows 10 with WSL2, Ubuntu 22.04 LTS - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1doxxuc/cant_run_rocm_on_windows_10_with_wsl2_ubuntu_2204/>
49. [DOCS] WSL2 with AMD ROCm Guide · Issue #6434 · comfyanonymous/ComfyUI - GitHub, tilgået november 16, 2025, <https://github.com/comfyanonymous/ComfyUI/issues/6434>
50. Running ComfyUI in Windows with ROCm on WSL, tilgået november 16, 2025, <https://rocm.blogs.amd.com/software-tools-optimization/rocm-on-wsl/README.html>
51. Complete ROCm 7.0 + PyTorch 2.8.0 Installation Guide for RX 6900 XT (gfx1030) on Ubuntu 24.04.2 - Reddit, tilgået november 16, 2025, <https://www.reddit.com/r/ROCm/comments/1njjixv/complete_rocm_70_pytorch_280_installation_guide/>
52. Run ROCm on Windows Subsystem for Linux (Easy 6-Minute Tutorial) - YouTube, tilgået november 16, 2025, <https://www.youtube.com/watch?v=lIAQeoqSCfo>
53. Not supported exception with libtorch-cpu-win-x64 (2.1.0.1) · Issue #1218 · dotnet/TorchSharp - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/issues/1218>
54. Compatibility matrix - AMD ROCm documentation, tilgået november 16, 2025, <https://rocm.docs.amd.com/en/latest/compatibility/compatibility-matrix.html>
55. Torch.cuda.is\_available() keeps switching to False - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/55717751/torch-cuda-is-available-keeps-switching-to-false>
56. Why `torch.cuda.is\_available()` returns False even after installing pytorch with cuda?, tilgået november 16, 2025, <https://stackoverflow.com/questions/60987997/why-torch-cuda-is-available-returns-false-even-after-installing-pytorch-with>
57. Complete ROCm 7.0 + PyTorch 2.8.0 Installation Guide for RX 6900 XT (gfx1030) on Ubuntu 24.04.2, tilgået november 16, 2025, <https://www.reddit.com/r/comfyui/comments/1njjp79/complete_rocm_70_pytorch_280_installation_guide/>
58. ROCm 6.3 fails to build · Issue #142248 · pytorch/pytorch - GitHub, tilgået november 16, 2025, <https://github.com/pytorch/pytorch/issues/142248>
59. PyTorch on ROCm installation, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/install-on-linux/en/latest/install/3rd-party/pytorch-install.html>
60. [Issue]: Torch rocm not avaiable on gfx1150/gfx1030/gfx906 and windows local build fail #1876 - GitHub, tilgået november 16, 2025, <https://github.com/ROCm/TheRock/issues/1876>
61. tilgået januar 1, 1970, <https://github.com/ROCm/TheRock/blob/main/external-builds/pytorch/build_pytorch.py>
62. Compile libtorch c++ api from source - PyTorch Forums, tilgået november 16, 2025, <https://discuss.pytorch.org/t/compile-libtorch-c-api-from-source/81624>
63. Building libtorch from source gives unexpected contents - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/74944881/building-libtorch-from-source-gives-unexpected-contents>
64. Libtorch build for ROCM error: “aten/src/THH” not exist · Issue #126640 - GitHub, tilgået november 16, 2025, <https://github.com/pytorch/pytorch/issues/126640>
65. Setup.py says USE\_ROCM:OFF, while I set USE\_ROCM=1 - C++ - PyTorch Forums, tilgået november 16, 2025, <https://discuss.pytorch.org/t/setup-py-says-use-rocm-off-while-i-set-use-rocm-1/107141>
66. Issues setting up LibTorch on Windows 11 - PyTorch Forums, tilgået november 16, 2025, <https://discuss.pytorch.org/t/issues-setting-up-libtorch-on-windows-11/199054>
67. Torchscript execution failures (hangs, access violation, Fatal error. Internal CLR fatal error. (0x80131506) ) · Issue #1047 · dotnet/TorchSharp - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/issues/1047>
68. Using CMake on MacOS · dotnet TorchSharp · Discussion #1240 - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/discussions/1240>
69. Torch CMake Error at CMakeLists.txt:10 (find\_package) in C++ - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/53819318/torch-cmake-error-at-cmakelists-txt10-find-package-in-c>
70. Can't compile TorchSharp.sln with Visual Studio · Issue #508 - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/issues/508>
71. Unable to Run PyTorch C++ Frontend Example from Documentation #100246 - GitHub, tilgået november 16, 2025, <https://github.com/pytorch/pytorch/issues/100246>
72. python-torchvision-rocm CMake build error (solved: build in chroot) - Arch Linux Forums, tilgået november 16, 2025, <https://bbs.archlinux.org/viewtopic.php?id=294557>
73. ROCm: libamdhip64.so is linked twice when building a torch C++ program · Issue #141014, tilgået november 16, 2025, <https://github.com/pytorch/pytorch/issues/141014>
74. Trouble loading CUDA support under dotnet-interactive (C#) · Issue #1146 - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/issues/1146>
75. Build Your Own GPT from Scratch with C# and TorchSharp (CPU-Only!) - DEV Community, tilgået november 16, 2025, <https://dev.to/auyeungdavid_2847435260/build-your-own-gpt-from-scratch-with-c-and-torchsharp-cpu-only-3ch5>
76. MissingMethodException: Method not found: '!!0 TorchSharp.ModuleExtensionMethods.to(!!0, Device)' after upgrade - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/78630239/missingmethodexception-method-not-found-0-torchsharp-moduleextensionmethods>
77. Error when using string variable for device - PyTorch Forums, tilgået november 16, 2025, <https://discuss.pytorch.org/t/error-when-using-string-variable-for-device/130738>
78. 7\_torch\_strategy/libtorch/include/c10/core/Device.h · main ... - GitLab, tilgået november 16, 2025, <https://gitlab-03.engr.illinois.edu/ie421_high_frequency_trading_spring_2023/ie421_hft_spring_2023_group_01/group_01_project/-/blob/main/7_torch_strategy/libtorch/include/c10/core/Device.h>
79. CUDA device support does not register allocator to c10::GetAllocator(...) #106533 - GitHub, tilgået november 16, 2025, <https://github.com/pytorch/pytorch/issues/106533>
80. Torchsharp in unity #1244 - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/discussions/1244>
81. Could not load file or assembly 'TorchSharp' · Issue #1039 - GitHub, tilgået november 16, 2025, <https://github.com/dotnet/TorchSharp/issues/1039>
82. Unable to load DLL 'LibTorchSharp' or one of its dependencies: The specified procedure could not be found - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/78932510/unable-to-load-dll-libtorchsharp-or-one-of-its-dependencies-the-specified-pro>
83. Interacting with native libraries in .NET Core 3.0 - Red Hat Developer, tilgået november 16, 2025, <https://developers.redhat.com/blog/2019/09/06/interacting-with-native-libraries-in-net-core-3-0>
84. Native Bindings in CSharp - Lost in Details, tilgået november 16, 2025, <https://www.lostindetails.com/articles/Native-Bindings-in-CSharp>
85. Native library loading - .NET | Microsoft Learn, tilgået november 16, 2025, <https://learn.microsoft.com/en-us/dotnet/standard/native-interop/native-library-loading>
86. NET Core project with native library dependency - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/58955723/net-core-project-with-native-library-dependency>
87. NativeLibrary.SetDllImportResolver(Assembly, DllImportResolver) Method (System.Runtime.InteropServices) | Microsoft Learn, tilgået november 16, 2025, <https://learn.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.nativelibrary.setdllimportresolver?view=net-9.0>
88. Can you dynamically load cross platform Native/Unmanaged dlls/libs in .Net Core?, tilgået november 16, 2025, <https://stackoverflow.com/questions/49270527/can-you-dynamically-load-cross-platform-native-unmanaged-dlls-libs-in-net-core>
89. Building a Better Library Loader in C# - Part 1 - DEV Community, tilgået november 16, 2025, <https://dev.to/thebuzzsaw/building-a-better-library-loader-in-c-part-1-1446>
90. TorchSharp-cuda-linux 0.105.1 - NuGet, tilgået november 16, 2025, <https://www.nuget.org/packages/TorchSharp-cuda-linux/>
91. TorchSharp-cuda-windows 0.105.1 - NuGet, tilgået november 16, 2025, <https://www.nuget.org/packages/TorchSharp-cuda-windows/>
92. How to get ROCM running in Docker to do machine learning on your AMD GPU, tilgået november 16, 2025, <https://drew-gilliesbc.medium.com/how-to-get-rocm-running-in-docker-to-do-machine-learning-on-your-amd-gpu-a521ed72b0fd>
93. Script for testing PyTorch support with AMD GPUs using ROCM - GitHub Gist, tilgået november 16, 2025, <https://gist.github.com/damico/484f7b0a148a0c5f707054cf9c0a0533>
94. torch.cuda.is\_available — PyTorch 2.9 documentation, tilgået november 16, 2025, <https://docs.pytorch.org/docs/stable/generated/torch.cuda.is_available.html>
95. How to invoke CUDA from C# - Stack Overflow, tilgået november 16, 2025, <https://stackoverflow.com/questions/62544641/how-to-invoke-cuda-from-c-sharp>
96. Using C# to pinvoke Cudart64\_75.dll Runtime API functions - NVIDIA Developer Forums, tilgået november 16, 2025, <https://forums.developer.nvidia.com/t/using-c-to-pinvoke-cudart64-75-dll-runtime-api-functions/45434>
97. Installing PyTorch for ROCm — ROCm installation (Linux) - AMD ROCm documentation, tilgået november 16, 2025, <https://rocm.docs.amd.com/projects/install-on-linux/en/docs-6.1.5/how-to/3rd-party/pytorch-install.html>
98. PyTorch inference performance testing - AMD ROCm documentation, tilgået november 16, 2025, <https://rocm.docs.amd.com/en/latest/how-to/rocm-for-ai/inference/benchmark-docker/pytorch-inference.html>
99. Getting started with AMD ROCm containers: from base images to custom solutions, tilgået november 16, 2025, <https://rocm.blogs.amd.com/software-tools-optimization/rocm-containers/README.html>