MemryX MX3 M.2 Accelerator Module Performance and Ease of Use: A Hands-On Evaluation

By the Staff of BDTI

May 2025

Executive Summary

This report presents an independent, hands-on evaluation of the MemryX MX3 M.2 AI accelerator module, focusing on its ease of use and performance. The evaluation was performed by BDTI, an independent technology analysis firm.

The module is an M.2-compliant AI accelerator capable of providing 14-24 TFLOPS of AI inference capacity. It uses on-module memory to support neural network models of up to 84 million 4-bit parameters, 42 million 8-bit parameters, or 21 million 16-bit parameters. Activations are stored in BF16 format, simplifying quantization of floating-point models while preserving accuracy. The module can be used off the shelf with Linux or Windows host machines (including smaller machines such as the Raspberry Pi 5), and its tools support models written in Keras, TensorFlow, TensorFlow Lite, and ONNX. PyTorch and other frameworks are supported through ONNX conversion.

For this evaluation we installed the M.2 module in an x86 Linux PC, downloaded and compiled several neural network models using the MemryX tools, ran these networks on the module, and measured inference performance and power consumption. We also developed a small stand-alone example application that takes images from a USB webcam and runs object detection on them using the module.

We found the MX3 M.2 module to be exceptionally easy to use while providing good performance and consuming little power. It is the first AI accelerator we've encountered for which both the hardware and the software "just works." We were particularly impressed by the scope of models tested in MemryX's Model eXplorer website, and the fact that the software tools are sufficiently capable that MemryX doesn't need to provide its own model zoo of modified models—rather, models can be easily compiled and achieve good performance from their original source code form.

1. Introduction

This report presents BDTI's independent, hands-on evaluation of the MemryX MX3 M.2 AI accelerator module, focusing on its performance and ease of use. Our target audience consists of engineers, developers, and managers considering using the MX3 module in products. For the evaluation, BDTI obtained an MX3 M.2 module, installed it in a Linux PC, downloaded several neural network models, compiled them using the MemryX tools, ran these networks on the M.2 module, and measured inference performance and power consumption. We also developed a small stand-alone example application that takes images from a USB webcam and runs object detection on them using the module.

This evaluation was performed by Berkeley Design Technology, Inc. (BDTI), a technology analysis and software development firm specializing in embedded computer vision and AI applications. BDTI has extensive experience developing, optimizing, and deploying computer vision applications across many different platforms. In addition to its software development work, for more than 30 years BDTI has performed in-depth, hands-on evaluations of numerous processors, development kits, and tools. For more information about BDTI, please visit www.bdti.com. For questions about this report, please email us at info@bdti.com.

This report is organized as follows:

- Overview of MemryX MX3 M.2 module
    - Hardware and architecture
    - Software tools
    - Documentation and website
- BDTI evaluation
    - Hardware installation
    - Software installation and initial test
    - Compilation experience and Model eXplorer
    - Application examples
    - Throughput and latency performance
    - Power consumption
- Conclusions

Typographical note: *As is traditional in BDTI reports, we attempt to separate fact from opinion. We set our opinions in indented, italicized paragraphs, like this.*

2. Overview of MemryX MX3 M.2 Module

2.1 Hardware and Architecture

The MemryX MX3 M.2 module (see Figure 1) is an AI accelerator module designed to facilitate high-performance AI inference for edge computing. The module is in an M.2 M-key 2280 form factor (22 x 80 mm) and can be used in systems providing a compatible M.2 socket.

The MX3 M.2 module consists of four MemryX MX3 AI accelerator chips. In a typical PC, due to power supply constraints of the M.2 interface, the MX3 chips are clocked at 600 MHz and the four chips provide a total of 14 TFLOPS of processing capacity. That said, in systems adequate ventilation and with M.2 interfaces capable of supplying more power, the chips can be clocked at 850 MHz, providing 20 TFLOPS of processing. And, with a special M.2 interposer board capable of providing even more power to the module, it is possible to clock the chips at 1 GHz, resulting in 24 TFLOPS of processing capacity.

Although MX3 chips are available for “chip-down” designs, this evaluation focuses on the M.2 module.

The M.2 module supports neural network models with up to 84 million 4-bit weight parameters, 42 million 8-bit weight parameters, or 21 million 16-bit weight parameters. Weight parameters are stored on the module (i.e., within each of the four chips), eliminating the need for external DRAM. Figure 2 shows examples of popular neural network models that can be deployed on the module.

The module’s MX3 chips use the bfloat16 (BF16) data type for activations, and 4-, 8-, or 16-bit integer data types (INT4/8/16) for weights. Since most neural networks are designed using 32-bit floating-point weights (FP32), such networks must be quantized to run on the M.2 module. The use of BF16 data types for activations preserves model accuracy and makes it much easier for MemryX’s tools to quantize input networks to a smaller bit precision without needing “tuning data” for quantization. MemryX’s neural network compiler automatically performs this quantization, or allows the user to specify per-layer weight quantization levels.

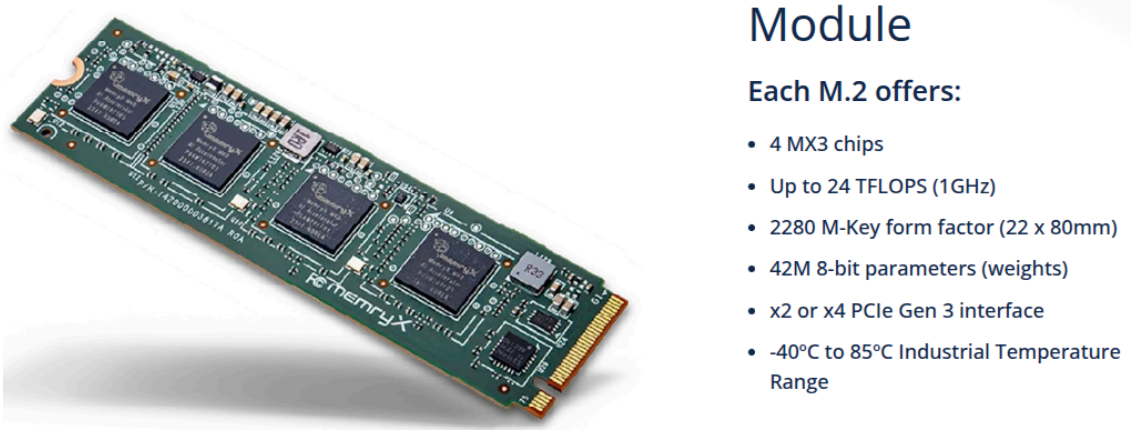

Figure 1. MemryX M.2 module showing four MX3 chips.

The use of BF16 data type for activations is clever. The dynamic range for activations is much larger than the dynamic range for weights. Weights are well behaved and usually symmetrical statistically so they can be quantized safely, albeit with some predicted loss in accuracy. However, activations usually cannot be safely quantized unless the quantization range is tuned via calibration data. A calibration ("tuning") dataset is a small set of representative inputs the model will encounter in the real world and is used for gathering statistics on the model's activations. These statistics determine the optimal quantization parameters such as scaling factors and zero points for converting the activations to a lower precision data format. MemryX's use of floating-point for activations increases the available dynamic range, making quantization safer and easier and eliminating the need for a calibration dataset. This in turn allows MemryX's tools to automatically quantize a wider variety of neural networks, improving ease of use.

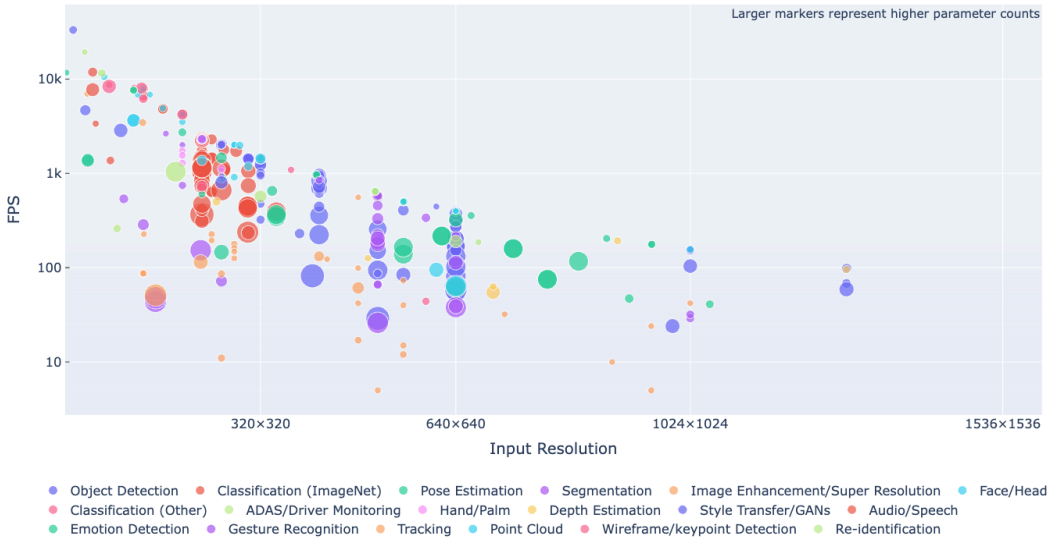

Figure 2. MemryX has benchmarked the MX3 M.2 module on more than 350 popular neural networks designed for a variety of use cases. An interactive version of the above chart is available on the MemryX website as part of their Model eXplorer tool.

Although the MX3 processes data as single samples (equivalent to a batch size of one), its pipelined architecture allows it to achieve performance comparable to other accelerators that use batch processing.

MemryX states that the MX3 M.2 module typically consumes 3-10 watts of power, depending on the model being run and the frame rate at which it is run. Peak power consumption is 15 watts maximum (i.e., the limit of the system power supply to the M.2 module). MemryX says that in Q3 of 2025 their module will allow the user to set a maximum power level limit in order to achieve maximum performance while staying within a specified power budget.

The MemryX architecture is a streaming, many-core, near-memory dataflow design intended to accelerate AI workloads such as neural networks. MemryX's dataflow architecture features tensor compute-cores (which they refer to as MemryX Compute Engines, or MCEs) that are configured once at compile-time and then communicate directly with each other through the input and output data they consume and produce. An illustration of the dataflow concept is shown in Figure 3.

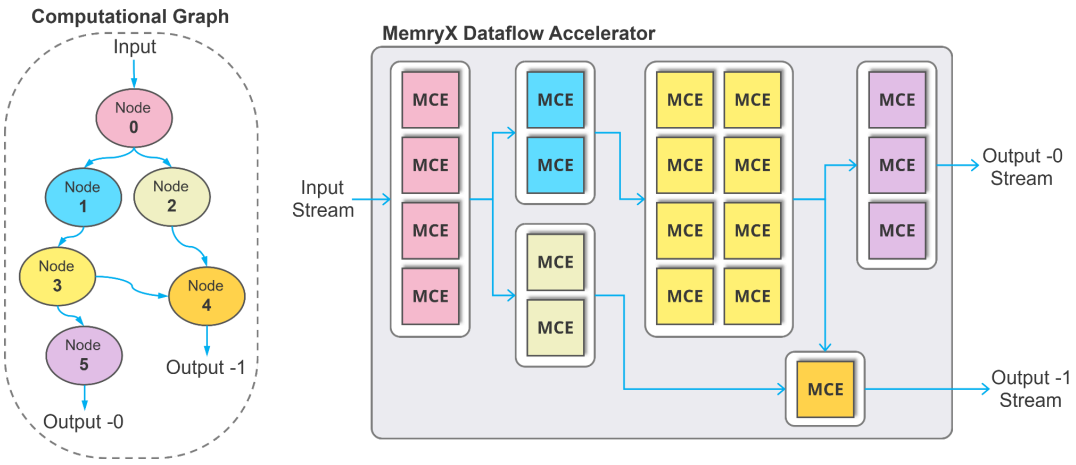

Figure 3. Illustration of the dataflow concept used in MemryX architecture showing mapping of dataflow nodes to MemryX Compute Engines (MCEs).

The MX3 contains two main types of MCEs: MAC cores (M-Cores) and ALU cores (A-Cores). By dividing the workload among multiple cores, the MemryX architecture naturally supports space multiplexing, allowing each neural network layer to be mapped efficiently and streamed through the dataflow pipeline. On-chip distributed memories play a central role. Two separate memory types are used: (1) weight memories that store neural network parameters and (2) feature-map memories that hold input, output, and intermediate data during inference. Feature-map memories also act as the communication medium between processing elements, eliminating the need for a centralized on-chip memory and enabling software-defined data pathways. Each compute "tower" contains multiple groups of M-Cores and A-Cores alongside local weight memory, interleaved with feature-map towers as shown in Figure 4.

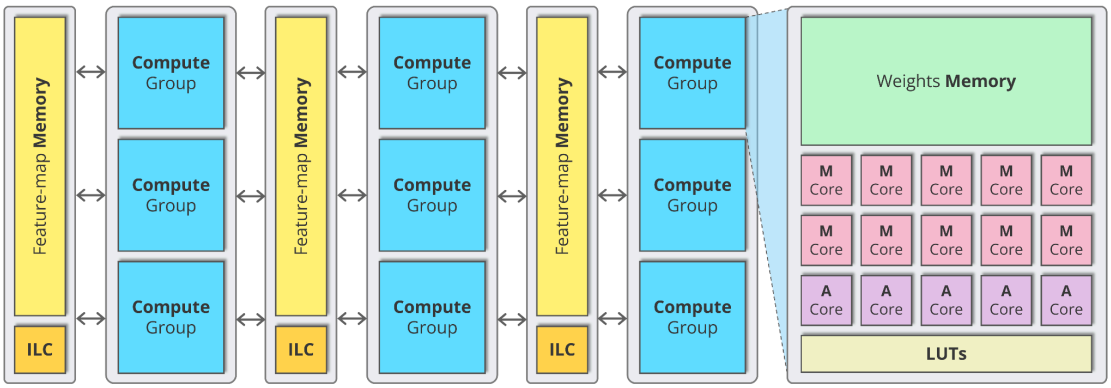

Figure 4. Illustration of the MX3's tower architecture, showing the interleaving stacks of compute and memory towers.

Every M-Core interfaces with local weight memory, adjacent feature-map towers, and neighboring M-Cores. A-Cores similarly interface with local look-up tables (LUTs) and adjacent

feature-map towers. The M-Core and A-Core organization is shown in Figure 5. The M-Core is a high-throughput vector/matrix compute core specialized for feature-map-by-weight operations such as convolutions. The A-Core handles feature-map-by-feature-map operations such as add, multiply, and concatenate operations in addition to specialized arithmetic such as reciprocals, softmax, and LUT-based approximations.

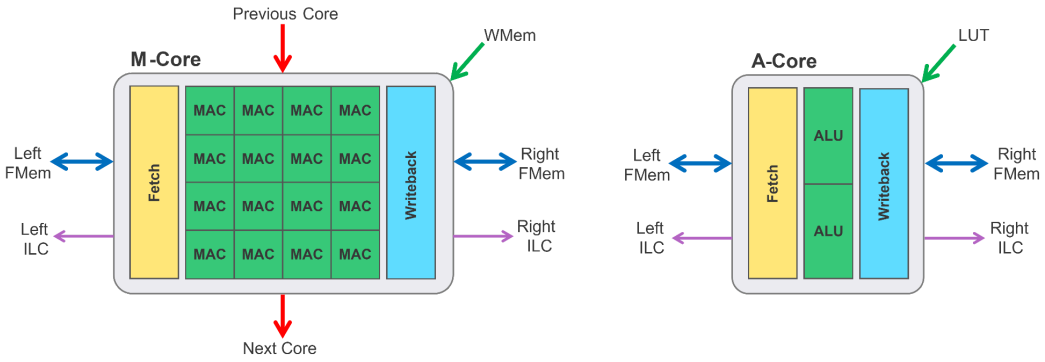

Figure 5. A top-level view of the M-Core and A-Core organizations, illustrating how each core type connects to memory resources and adjacent cores.

> *Data-streaming many-core architectures offer several advantages, among which are increased computational parallelism, efficient data processing, flexibility, and scalability. In addition, near-memory designs (in which data is stored close to where it is processed) provide bandwidth savings, lower power consumption, and lower execution latency. All these features work in tandem to improve the overall performance of complex workloads such as neural networks.*

The MemryX architecture supports a wide variety of networks; indeed, as discussed below, MemryX provides public throughput and latency benchmark results for more than 350 models on the module. These include 89 models for classification, 95 for object detection, 31 for segmentation, 35 for pose estimation, 30 for super-resolution, 25 for face/head detection/recognition, and others for ADAS/driver monitoring, hand/palm detection, depth estimation, style transfer/GANs, emotion detection, audio and speech, gesture recognition, tracking, etc. In addition, MemryX tells us that in their validation testing, they test more than 3,000 public and private models for each software release.

> *We were impressed by the breadth and depth of the available models, and more so by the fact that these models do not require any modification to run on the module—the tools are of sufficient quality that the models can be compiled and run "out of the box" in their original source code form.*

At present, transformer models are not natively supported (including YOLOv10 and YOLOv11), and it is not currently possible to compile transformer models using the same "one-click" methodology described above. MemryX states that customers needing transformer models on

the module should contact them for support. They add that in a future software release, transformer models will be supported with one-click compilation. In the meantime, they point to the TinyStories small language model on their website as an example of a transformer-based model that can run on the MX3 architecture.

> *While the current lack of transformer models today is inconvenient, we are heartened to hear that support for YOLOv10/v11 and certain other transformer-based models will be available in MemryX's next software release (Q2 2025), and one-click compiler support of additional transformer-based models is expected in a software release later this year.*

2.2 Software Tools

MemryX provides an SDK that includes both offline tools (compiler, simulator) and runtime software (drivers and APIs). APIs are available in C++ and Python. MemryX's SDK supports both native Windows (including a signed driver) and Linux. Under Windows, MemryX's offline tools currently require WSL (Windows Subsystem for Linux), with MemryX stating that native Windows support for offline tools is coming soon. Linux-based systems include support for x86, Arm, and RISC-V architectures. Supported Arm boards include Raspberry Pi 5, Orange Pi 5 Plus, Orange Pi 5 Max, and Radxa Rock 5B.

The tools include:

- A neural network compiler (mx_nc)
- Benchmarking tools available in both C++ (acclBench) and Python (mx_bench)
- Software libraries for application development
- A utility (DFP Inspect) to parse and display the metadata of a dataflow program (DFP) file
- Documentation, examples, and tutorials
- A cycle-accurate simulator (mostly intended for hardware development in a chip-down design)

The MemryX neural network compiler, mx_nc, supports four of the most widely used neural network frameworks: Keras, TensorFlow, TensorFlow Lite (TFLite), and ONNX. It accepts models in their original FP32 data type format, analyzes the weights and activations, and automatically quantizes the model. The activations are always quantized to BF16 data type, while the weights are quantized to INT4, INT8, or INT16 either automatically or as specified by the user. The MemryX mapper then optimizes the compute resources for the resulting model for maximum throughput (by default) on the MX3. The output of this compiler/mapper toolchain is called a dataflow program (DFP), which runs on the MX3.

Figure 6 shows the chain of processes that occur under the hood.

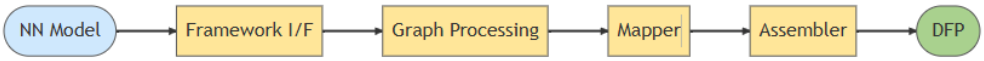

Figure 6. Steps involved in compiling a neural network model to a dataflow program (DFP).

> *MemryX's tools make it easy to take existing floating-point neural network models and automatically map them to and run them on the MX3 M.2 module without the user having to do any "pre-work," such as quantization-aware training, pruning, or compression. We were impressed by the scope of models supported.*

The tool suite also provides benchmarking utilities that can be instrumented into inference code on the MX3 and allows developers to measure latency and throughput (FPS) metrics during the execution of a model.

As mentioned, MemryX also provides a cycle-accurate simulator for their MX3, primarily designed for engineers developing a custom design with a different number of chips than the 4-chip M.2. MemryX's tools support hardware designs using from 1 to 16 MX3 chips.

2.3 Documentation and Website

MemryX provides extensive documentation in the Developer Hub section of their website to support their tools and user development. The site also features numerous examples and tutorials that guide developers through the steps of compiling, running, and benchmarking different neural network models. The Developer Hub is divided into several sections:

- Getting started
- Running your first model
- Model exploration (i.e., see what models are available and get them running quickly)
- Tutorials (shorter examples written in a step-by-step style)
- End-to-end examples
- Reference materials (tools, APIs, specifications, etc.)

The Model eXplorer feature on the MemryX website provides a searchable, categorized database of more than 350 models that MemryX has tested with their M.2 accelerator. For each model, Model eXplorer provides the type of model (e.g., classification, object detection, pose estimation, etc.), input resolution, parameter count, model format (e.g., ONNX, Keras, TFLite, etc.), benchmark results in terms of throughput (FPS) and latency (ms) in both 14 TFLOPS and 20 TFLOPS M.2 accelerator modes, and then links to the model's original source and a pre-compiled DFP file. MemryX notes that the models listed have not been modified from their original form, and developers are encouraged to compile the models themselves; the provided

DFP files are merely a convenience. In this way, MemryX stresses that they do not provide a conventional model zoo—that is, a collection of models that have been ported, modified, or optimized to work on their accelerator—since their tools are sufficiently robust to not require such modifications.

3. BDTI Evaluation

BDTI evaluated the MemryX M.2 module and software on an x86 Linux host platform consisting of an Intel Core i9-13900K 13th generation processor on a MSI Pro Z790-A motherboard running Ubuntu 22.04.4 LTS.

3.1 Hardware Installation

We received an MX3 M.2 module with a top-mounted heat sink, bottom-mounted heat sink holder, and a ⅛"-long screw to attach the module to the motherboard. Unfortunately, due to the thickness of the bottom-mounted heat sink holder, we were unable to plug the M.2 module into slots on the surface of the motherboard. Instead, we used vertically stacked M.2 slots that were available. The provided ⅛"-long screw was too short to anchor the module, but we improvised using a different fastener of the correct size.

3.2 Software Installation and Initial Test

With the hardware installed, we proceeded with the installation of the SDK, runtime, and other utilities, following the instructions on MemryX's Get Started page. The instructions were clear and the procedure straightforward, taking well under an hour. We then selected a few models (YOLOv8m in PyTorch, ResNet-50, and Inception-v3 in Keras) from the Tutorials section of the MemryX website, compiled them, and ran inference on the MX3 module. These models compiled quickly and easily and ran with no issues.

3.3 Compilation Experience and Model eXplorer

We next compiled a selection of models from the MemryX Model eXplorer to evaluate ease of use of the toolchain. We used the Model eXplorer tool on the MemryX website to understand what DNN models were supported by the model and to get an idea of their features and performance.

Once we chose models to test, we downloaded the models from the original source website; we did this intentionally, to understand how easy it was to compile models "from scratch." The answer was: very easy. To start, we experimented with image classification models like ResNet-50 and Inception-v3 in Keras format and object detection models like YOLO in ONNX format. The compilation process from input to the generation of the executable DFP file consisted of us issuing a single command for each model.

Once we had compiled the models from scratch, we then used Model eXplorer to download the pre-compiled DFP models to verify that our compilation resulted in the same performance as the versions on the MemryX website.

> *We found Model eXplorer both easy to use and valuable for getting a quick understanding of what models had been tested with the module, and what their benchmarked performance was.*
>
> *We liked the fact that when we attempted to compile a model that would not fit on the MX3 module (e.g., YOLOv8l) or had unsupported operations (e.g., YOLOv11), the compiler gave an informative error message describing this limitation right away. This avoids the wasted time that we have experienced in debugging compilation errors with other toolchains where errors are indicated much later in the process. In cases where we compiled large models destined for the MX3 M.2 module and the resource requirements were beyond the four-chip capacity of the module, we could simply instruct the compiler via an inline parameter to compile the model for any number of MX3 chips. For example, YOLOv8l could fit successfully in a six-chip MX3 configuration. Where the compilation issue was an unsupported operation, such as in the case of YOLOv11, the compiler immediately issued an unsupported operator error with specific details about the missing operator.*

3.4 Application Examples

For creating end-to-end applications, MemryX provides a collection of practical application examples for real-time video inferencing, object detection, and more. Developers can take an example as a template and adapt it to their own needs.

We selected and implemented an end-to-end single- and a multi-stream object detection application in Python. We experimented with two input sources: a live USB camera and a video source file. Our application was based on the 640x640 YOLOv8m model. For a live USB camera with a 640x480 pixels resolution video, the throughput was 24 FPS (limited by the USB camera, and not by YOLOv8m, which reached 45 FPS in our test). For a 3-stream input video source file, each at 1920x1080 resolution frames, the reported throughput was 12 FPS per stream, with pre-processing (e.g., downsampling from 1920x1080 to 640x640) and post-processing of the model output (e.g., non-maximum suppression) performed on the host CPU. Only the model execution was run on the MX3 chips.

Developing the above application based on MemryX's example code was easy and took less than two hours.

> *We liked the high quality of the tools and developer's documentation on the MemryX website, finding them to be among the best we have encountered. The toolchain was easy and simple to use. We liked the fact that no manual pre-processing of the model, in terms of compression or quantization, was necessary before compilation—the compiler*

handled all that for us. We found that the examples for end-to-end application development were useful starting points to develop one's own application.

3.5 Throughput and Latency Performance

We chose three models to use in evaluating the inference throughput and latency performance of the MX3 M.2 module. Our selection criteria were that we considered them to be popular models used in real-world use cases, and that MemryX considered them to be relevant in their target markets.

The three models we selected were YOLOv8s, EfficientNet-Lite4, and OSNet-x1, as shown in Figure 7. (Note that OSNet-x1 is not a model listed in Model eXplorer as being supported on the module; it was intentionally chosen to stretch the tools a bit by feeding them an untested model.)

Model	Resolution	Parameters (Millions)	GOPS per Frame	Notes
YOLOv8s	640x640x3	11	29.7	Popular object detection model. Model source.
EfficientNet-Lite4	300x300x3	12.4	5	Efficient classification model. Model source.
OSNet-x1	256x128x3	2.2	1	Small model for person re-identification. Model source.

Figure 7. Evaluated models and their characteristics.

For each of these models, we downloaded the pretrained model from its original source location using the links provided in the Model eXplorer—i.e., we did not download them from the MemryX website. The YOLOv8s and OSNet-x1 models were in PyTorch and EfficientNet-Lite4 in TFLite. Models in TFLite format can be natively compiled by the MemryX tools, while PyTorch models had to be converted to ONNX first. We then used the MemryX compiler to optimize the ONNX and the TFLite models and generated executable DFP files.

The MemryX compiler provides multiple optimization effort levels, including "lazy," "normal," "hard," and "closed loop." These modes provide a trade-off between compilation time and optimization. We compiled all three models above using "normal," "hard," and "closed loop" optimization modes. The "normal" mode took from 1 to 4 minutes to finish. For the "hard" and "closed loop" modes, the compilation time varied significantly by the model. On our Core i9-13900K system, compiling EfficientNet-Lite4, YOLOv8s, and OSNet-x1 in the "hard" mode took 6, 23, and 38 minutes, respectively. The "closed loop" mode took 19, 39, and 55 minutes, respectively.

We ran all three model executables on the MX3 M.2 module in the default 14 TFLOPS mode and used both MemryX-provided benchmarking utilities (acclBench and mx_bench) to measure latency and throughput. We examined the source code of the benchmarking utilities to confirm that the correct measurements were being computed for these two metrics and then verified the results independently with our own benchmarking code.

As illustrated in Figure 8, the in-to-out latency measurement—which MemryX refers to as “full system latency” or “latency 2”—measures the time elapsed between image input (including any host-related delays) and final output. The reported value is the mean of 20 trials.

In contrast (and also illustrated in Figure 8), frame-to-frame latency (which MemryX refers to as “latency 1”) is the time between sequential output frames after the processing pipeline is up and running. The reciprocal of this frame-to-frame latency is throughput, measured in frames per second (FPS). FPS measurements were carried out with multi-threading enabled on the host. This enables the host to feed a new image frame into the MX3 as soon as the device is ready to consume it. The MX3 processes the images with batch size of one but in a pipelined operation (i.e., it can begin processing a new image before the previous one is fully processed). The throughput result in FPS is 1,000x the reciprocal of the processing time for 1,000 image frames.

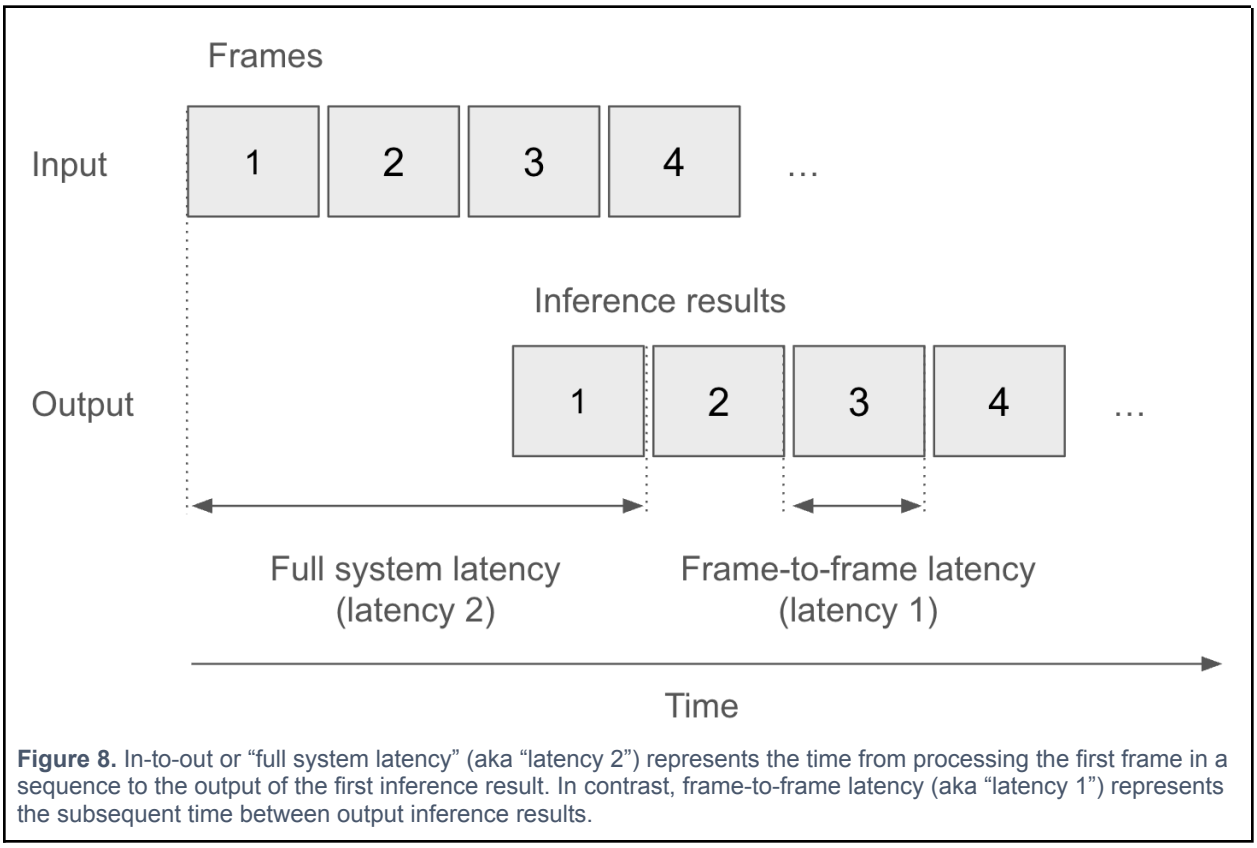

Figure 8. In-to-out or “full system latency” (aka “latency 2”) represents the time from processing the first frame in a sequence to the output of the first inference result. In contrast, frame-to-frame latency (aka “latency 1”) represents the subsequent time between output inference results.

MemryX's Python Benchmark API enables the developer to measure the latency or FPS in real-world applications. The API has a "threading" option to let the user specify whether to measure the latency or throughput (FPS) in the same way as their benchmarking tools do.

Figure 9 shows BDTI's measurements of in-to-out latency, frame-to-frame latency, and throughput for the selected models with two different compiler effort levels.

	In-to-Out Latency (ms)		Frame-to-Frame Latency (ms)		Throughput (FPS)	
Model	**Hard**	**Closed Loop**	**Hard**	**Closed Loop**	**Hard**	**Closed Loop**
YOLOv8s	12.51	12.47	5.64	4.48	177	223
EfficientNet-Lite4	9.85	9.68	1.37	1.32	732	752
OSNet-x1	5.23	5.09	1.15	0.95	871	1,054

Figure 9. BDTI-measured latency and throughput measurements for selected networks running in 14 TFLOPS mode. Measurements were taken using both acclBench and mx_bench and confirmed using separately developed BDTI code. Models were compiled using either the "hard" or "closed loop" optimization method, as indicated.

For both YOLOv8s and EfficientNet-Lite4, BDTI results matched the published results on the MemryX website. (As mentioned, OSNet-x1 results are not listed on the MemryX website.)

Although we did not test for accuracy, MemryX has a section in their Developer Hub Tutorials devoted to accuracy measures. Given MemryX's use of BF16 activations and no need to recover inaccuracies due to quantization errors, MemryX's tutorial claims of high accuracy and posted results make sense. Furthermore, MemryX provided us a white paper that showed the ability for a user to trade off small reduction in accuracy in exchange for materially higher FPS performance using the same power.

> *The ability to trade off AI model accuracy for performance is an important and commonly used technique for optimizing system performance. Although we did not evaluate this specifically, we felt it important to highlight the fact that MemryX's module and tools support this capability.*

3.6 Power Consumption

MemryX provided us with a pre-release version of its M.2 module that is instrumented to measure the module's current and power consumption. The circuitry on this module measures the current and power consumption of the entire module, which is to say the four MX3 chips and any support circuitry.

Power measurement is complicated by several factors:

First, as with all semiconductor devices, the module's power consumption is dependent on temperature, and increases with increasing temperature. So, for example, a module at 70°C will consume less power than a module at 90°C.

Second, power consumption increases non-linearly with supply voltage, and the MX3 module's power management is intelligent enough to reduce voltage when the module does not need maximum performance, reducing power consumption.

Third, to protect against damage, the module has a temperature sensor that will throttle the clock speed when module temperatures reach 100°C.

Thus, power measurements are dependent on temperature, workload, and whether throttling is occurring. Temperature, in turn, is dependent on workload and thermal setup (e.g., active vs. passive cooling, airflow, ambient temperature, etc.).

Figure 10 presents power consumption numbers on our selected networks when the module is running at maximum throughput (i.e., processing as many frames per second as possible), without throttling, at several different temperatures. This might occur, for example, in a network video recording (NVR) application performing object detection on as many video streams as the MX3 M.2 module is capable of. The reported power numbers indicate the average power consumption over the last three seconds (30 measurements at 10 readings per second).

Model	Throughput (FPS)	Power Consumption at 50°C (W)	Power Consumption at 70°C (W)	Power Consumption at 90°C (W)
YOLOv8s	223	10.8	11.3	12.2
EfficientNet-Lite4	752	9.5	10.0	10.8
OSNet-x1	1,054	6.7	7.1	7.7

Figure 10. BDTI-measured power consumption for selected models when running at maximum throughput in 14 TFLOPS mode.

Of course, not all applications require maximum throughput. For example, another network video recording system might only be connected to a handful of cameras and thus not be running the module at its maximum capacity. Figure 11 shows power consumption for 1, 2, and 3 streams of 30 FPS video.

Model	Power Consumption at 30 FPS at 50°C (W)	Power Consumption at 60 FPS at 50°C (W)	Power Consumption at 90 FPS at 50°C (W)
YOLOv8s	3.7	4.7	5.9
EfficientNet-Lite4	2.8	3.1	3.3
OSNet-x1	2.7	2.8	2.9

Figure 11. BDTI-measured power consumption for selected models when running frame rates of 30, 60, and 90 FPS in 14 TFLOPS mode.

4. Conclusions

We found the MemryX MX3 M.2 AI accelerator module to be exceptionally easy to use while providing good performance and consuming little power. It is the first AI accelerator we've encountered for which both the hardware and the software "just work."

We were particularly impressed by the scope of models tested in MemryX's Model eXplorer website, and the fact that the software tools are sufficiently capable that MemryX doesn't need to provide its own model zoo of modified models—rather, models can be easily compiled and achieve good performance from their original source code form.

MemryX's architecture and use of BF16 data format for activations means that there is no need to modify most networks to get them to work on the module, nor is any special training or calibration/tuning data required for quantization. The compiler automatically optimizes the model graph and maps it for best performance on the MX3 without impact on model accuracy.

In addition we found the MemryX documentation to be thorough and well written, allowing us to get started quickly and move from simple examples to complete applications in short order.

5. References

MemryX website:

- Main website: https://memryx.com
- MX3 module and chip specifications: https://developer.memryx.com/specs/index.html
- Developer Hub: https://developer.memryx.com/index.html
- Tools overview: https://developer.memryx.com/tools/index.html
- Model eXplorer: https://developer.memryx.com/model_explorer/models.html
- APIs: https://developer.memryx.com/api/index.html

Models:

- YOLOv8: https://docs.ultralytics.com/models/yolov8/#performance-metrics
- EfficientNet-Lite4: www.kaggle.com/models/tensorflow/efficientnet/tfLite/lite4-fp32
- OSNet-x1: https://kaiyangzhou.github.io/deep-person-reid/MODEL_ZOO