arm

White Paper

Powering the Edge: Driving Optimal Performance with the Arm ML Processor

Powering the Edge

On-device machine learning (ML) is a phenomenon that has exploded in popularity. Smart devices that are able to make independent decisions, acting on locally generated data, are hailed as the future of compute for consumer devices: on-device processing slashes latency; increases reliability and safety; boosts privacy and security...all while saving on power and cost.

Although ML in edge devices has come a long way in a short time, with new networks, new algorithms, and different architectures arriving on the scene in recent years, the basic computational requirements of inference engines have remained constant. Since ML is a process of repetition and refinement, aimed at making sense of a vast amount of information and then drawing a conclusion, functional improvements are still largely driven by high throughput and high efficiency.

Repurposing a CPU, GPU or DSP to implement an inference engine can be an easy way to add ML capabilities to an edge device. Many embedded devices, for example, will need nothing more than a small, low-power microcontroller unit (MCU), and the vast majority of smartphones on the market today are running accurate, performant ML on a CPU. But where responsiveness or power efficiency is critical, CPUs may struggle to meet the demanding performance requirements, and a dedicated neural processing unit (NPU) – such as the Arm ML processor – may be the most appropriate solution.

The Arm ML processor

The ML processor's optimized design provides a massive uplift in efficiency compared to CPUs, GPUs and DSPs through efficient convolution, sparsity and compression.

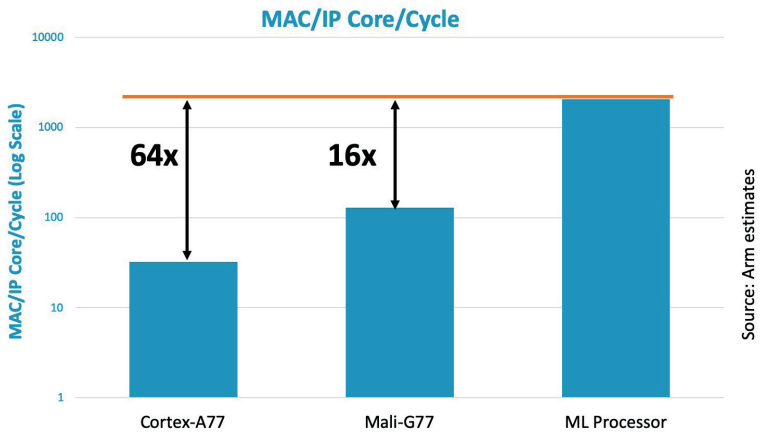
Part of Project Trillium – Arm's heterogeneous compute platform for ML workloads – its scalable architecture delivers the computational determinism required for real-time responses, while offering the flexibility and programmability needed to keep pace with rapidly evolving algorithms.

Performance

The ML processor drives quick, accurate inference while maintaining efficiency and performance. Given that many edge devices are relatively small, battery-powered systems – and thus have limited battery capacity – that means longer operation for a security camera, longer flight time for a drone, or longer range for an electric vehicle.

The ML processor performs convolution with an extremely fast throughput of up to 4 Tera Operations Per Second (TOP/s), and an exceptionally efficient computation rate of 5 TOP/s per Watt (TOPs/W) in a geometry of 7 nm.

For high performance to truly be a benefit, it needs to be coupled with efficiency.



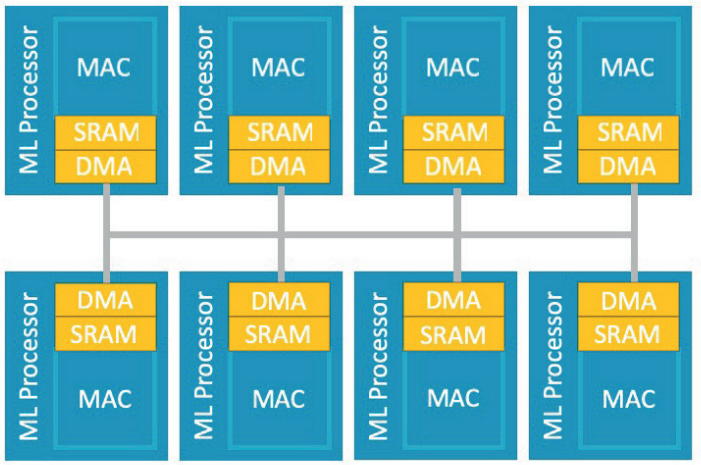
The Arm ML processor delivers single-core performance of < 4 TOP/s with efficiency of < 5 TOPs/W

Scalable Multicore Performance

The ML processor also supports multicore configurations, enabling demanding use cases such as high-resolution segmentation.

- < 32 TOP/s
  Up to eight processors can be arranged in a tightly coupled cluster that processes multiple networks in parallel, or a single, large network split across cores.

- > 250 TOP/s
  Larger configurations are supported through Arm CoreLink Interconnect mesh technology.



Up to eight cores, 32 TOP/s, closely coupled

Arm donated Arm NN to the Linaro Machine Intelligence Initiative, where it is now developed fully in open source. To find out more, visit **mlplatform.org**.

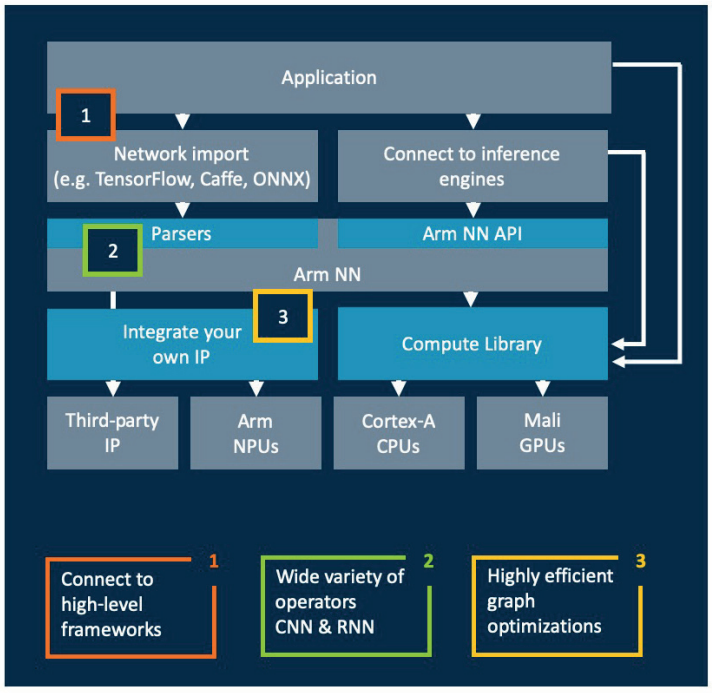
Efficiency

For high performance to truly be a benefit, it needs to be coupled with efficiency. The ML processor provides an industry-leading power efficiency of 5 TOPs/W, achieved through state-of-the-art optimizations such as neural compilation, efficient convolutions and bandwidth reduction mechanisms.

The processor is supported by Arm NN, open-source software that facilitates complete, futureproof solutions. Easy to implement, Arm NN enables efficient translation of NNs in existing frameworks – such as TensorFlow, PyTorch and Caffe – and compiles the workloads to run across Arm Cortex CPUs, Arm Mali GPUs and the ML processor. This reduces the need for processor-specific optimization and lowers the software investment associated with each piece of hardware.

Arm NN

- Connects the ML Processor to all popular frameworks: TensorFlow, TensorFlow Lite, PyTorch, Caffe, and others via ONNX
- Supports an increasing number of operators on the ML processor and across the Arm platform
- Drives efficient execution through integrated graph optimizations and operator fusing

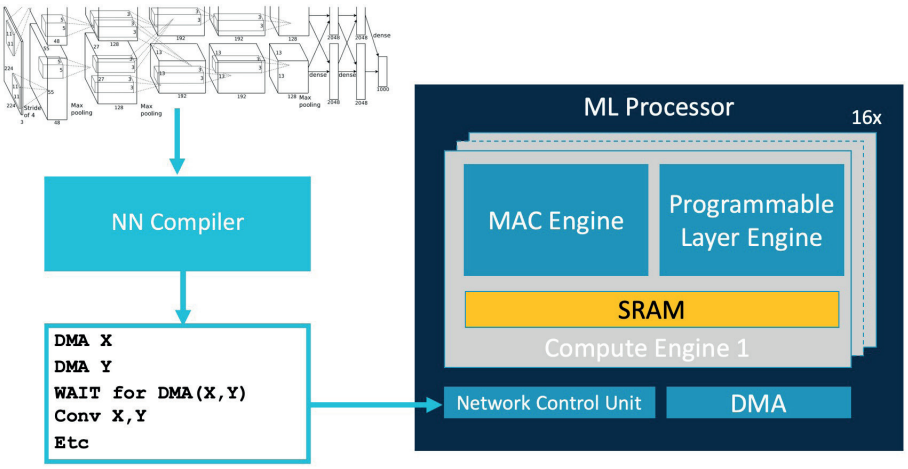


Neural Compilation

Neural compilation allows more data to be stored on the chip and re-used between fused layers, avoiding a trip to DRAM and significantly reducing power consumption. Arm's compression technology also provides 1.5-3x savings on external memory bandwidth to DRAM; Winograd-efficient convolutions accelerate common filters by 225% compared to other NPUs, allowing more performance in a smaller area.

The compiler takes the full network graph and automatically tiles the work between the 16 compute engines, fitting a 64 kB slice of SRAM into each compute engine – equivalent to a full megabyte of SRAM for the processor as a whole. It also uses determinism and static scheduling to streamline operations and increase efficiency – reducing development effort for new neural features, hiding complex memory transfers and further improving battery life.

Next, the compiler inspects the neural network and maps it to a command stream specially designed for use by components in the ML processor. This command stream then schedules tasks for efficient DMA handling and simpler flow control between high-level control tasks and compute engines performing convolution. The result is an optimized workload that can run with higher utilization and faster, more efficient execution, improving the overall responsiveness of the core.



Efficient operation driven by Arm NN

Since memory accesses are predictable, a complex hierarchy of memory caches to support scheduling is not required. Giving each compute engine its own SRAM also brings memory operations closer to the compute operations, for faster processing. The result is efficient yet predictable hardware performance, with greater flexibility to define dependencies and specify processing order.

This technique of gating for zeros helps reduce power consumption in the convolution layers by as much as 50%.

Efficient Convolutions

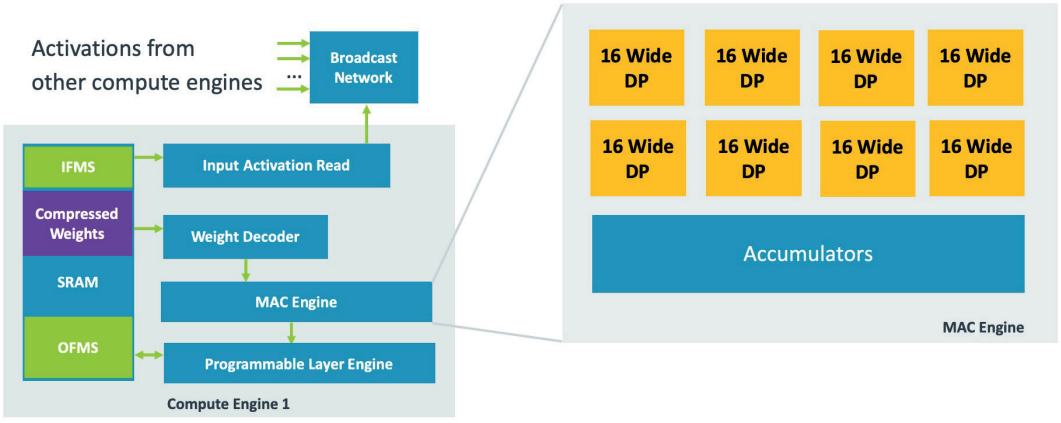
Common neural networks consist of 70-90% convolutions[1]. At the heart of these convolution operations are time-consuming multiple accumulates (MACs). The deeper the architecture, the more MACs are required and the longer it takes to process the workload – which, in the case of an edge device, can lead to user frustration.

To mitigate this, the ML processor's MAC engine – which performs the bulk of the convolutional computations – has been designed for efficiency, with the ability to execute eight 16x16, 8-bit dot product (DP) operations per cycle. That is, each 16-wide DP unit in the MAC engine performs an 8-bit, 16-deep DP operation with 32-bit accumulators. This translates to 256 operations per cycle which, multiplied across the 16 compute engines, equates to 4096 operations per cycle – yielding a computational rate of 4 TOPs at 1 GHz.

The processor also uses several techniques to make convolutions faster and more efficient:

- **Sharing feature maps**

  As shown in the block diagram below, input data or activations are read and shared through the broadcast network, reducing the number of redundant, high-energy reads to memory. Each engine receives the same activations and combines them with unique weights, different in each engine. The weights are compressed or decompressed locally to minimize the distance and energy required to execute each part of the computation. The result of these computations is multiple output feature maps (OFMs), pushed to external memory at the end of each operator.



Convolution layer computations in the ML processor

- **Operator fusing**

  Multiple operations are also intelligently fused together. These may be sequential operations from the MAC engine, the PLE engine or both. When this fusion occurs,

1. A Scalable Multi-TeraOPS Deep Learning Processor Core for AI Training and Inference, B. Fleischer et al., VLSI Symposium 2018

the resulting OFMs are stored locally and re-used as the next operators' activations. This exploitation of temporal locality reduces the number of memory transactions to DRAM, further reducing energy consumption.

- **Zero power gating**

  To avoid wasting computational effort calculating values that will eventually be truncated to zero, the data path uses techniques that predict zero-valued activations and disables parts of the associated logic and hardware. This technique of gating for zeros helps reduce power consumption in the convolution layers by as much as 50%.

- **Winograd**

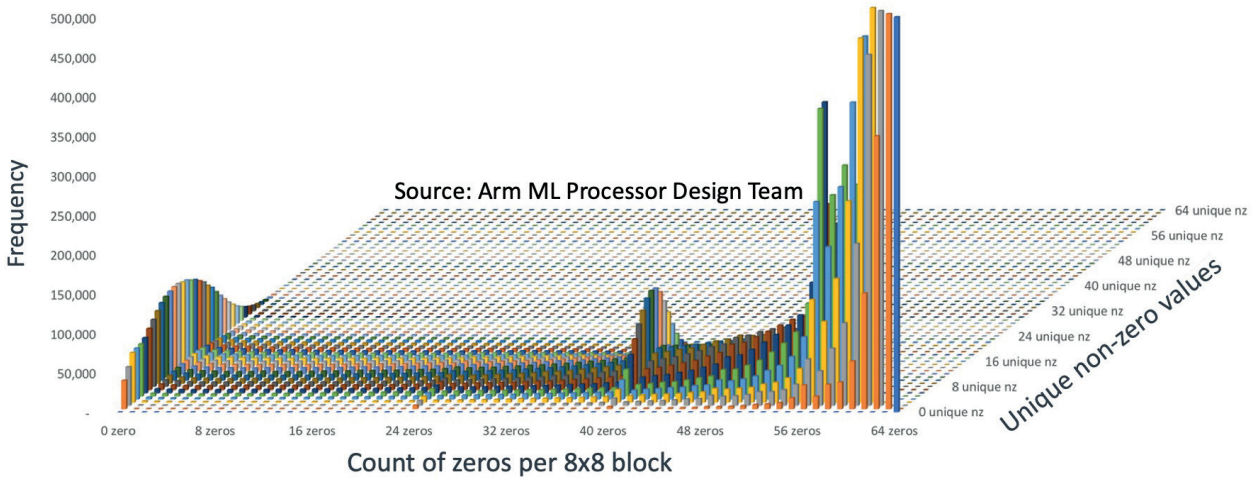
  The MAC engine also contains hardware acceleration for Winograd convolution and required transformations. Winograd accelerates common filters by 225% compared to other NPUs, allowing for higher performance execution on networks that use 3x3 convolution filters, such as Inception v1-v4, Resnet-v1, Resnet-v2, VGG, and YOLOv2.

Bandwidth Reduction

The ML processor not only delivers highly efficient performance, it also minimizes memory bandwidth. Designing with memory in mind, right from the start, is critical for full system integration and ensuring a consistent flow of data. In order to achieve this, multiple optimizations were included such as lossless compression – end to end throughout the processor – clustering and pruning, as well as workload tiling.

- **Lossless Compression**

  The ML processor also uses lossless compression to reduce the size of weights and activations, saving a significant amount of energy (or bandwidth) for DRAM operation and further increasing efficiency. Compression is also used to manage zero and non-zero values, giving an overall bandwidth reduction of <3x compared with architectures with no compression.



Insight into the distribution of computed values in common neural networks

- **End-to-End Compression**

  The ML processor is able to compress models throughout the toolchain before they are read from the internal SRAM – increasing efficiency in later layers of the network, where weights tend to use more bandwidth. This improves performance, particularly when working with deep CNNs that use higher numbers of convolution layers.

- **Clustering and Pruning**

  The processor also provides hardware support for clustering and pruning. Clustering is a model deployment technique that identifies and 'clusters' similar input patterns and groups or 'snaps' the remaining non-zero weights to a smaller set of possible non-zero values, making them easier to manage. Pruning removes whole nodes in a network to reduce the number of connections and required calculations. To exploit the benefits of both clustering and pruning, the processor allows neural network models to be compressed offline during the compilation phase, providing hardware support when reading from each engine's local memory.

- **Workload Tiling**

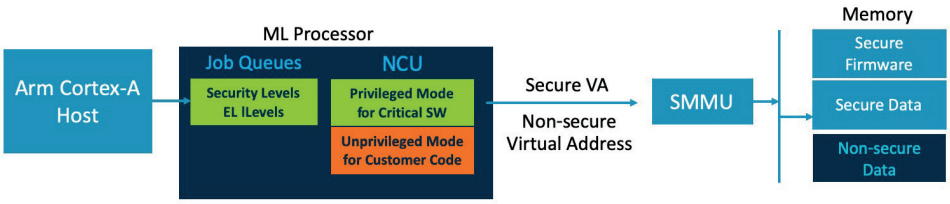
  Compiler-based scheduling reduces bandwidth further. Scheduling makes more efficient use of memory bandwidth by keeping the working set in SRAM. The use of tiling, which applies (or 'tiles') convolution operations across the compute engines, avoids trips to DRAM. In addition, multiple output feature maps are calculated in parallel using identical input feature maps and different weights. The intermediate stages are pipelined between the MAC engine and the PLE. All these features minimize bandwidth and make the ML processor more efficient.

Security

It hardly needs to be said that security is an essential part of system design. A key concept is reduction of the attack surface, which will – ideally – be small, simple and auditable.

To meet this need, the ML processor allows several implementation choices to address multiple risk profiles and a wide range of use cases, some of which are shown below. It uses industry-proven Arm microcontroller technology, with standard privilege levels and firmware that clears the SRAMs, making it easier to audit. No other solutions have these security features built in from the start.

The system can be configured with an optional system memory management unit (SMMU) necessary to support processing in a trusted or secure execution environment (TEE or SEE), through configurable secure queues, supporting multiple users who may be processing content for high-value media streams.

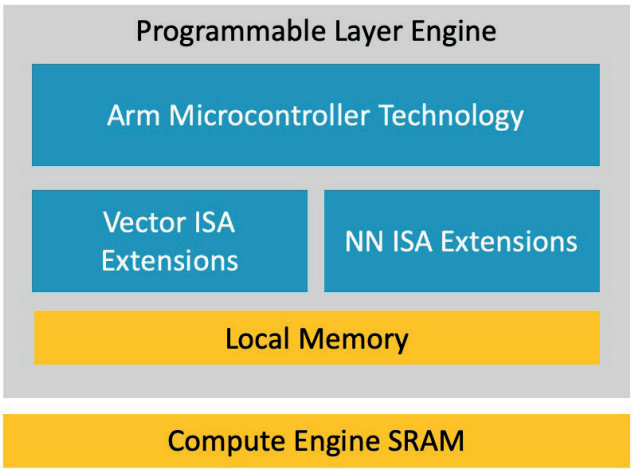


Secure and flexible operating modes of the Arm ML Processor

Arm's end-to-end approach to security provides seamless, system-wide layered protection for processors, subsystems, acceleration, and offloading. Arm Platform Security Architecture (PSA) is the framework for a common security best practice. Arm's security portfolio helps protect against a broad spectrum of attacks, allowing partners to deploy the security level that best matches application needs.

Futureproofing

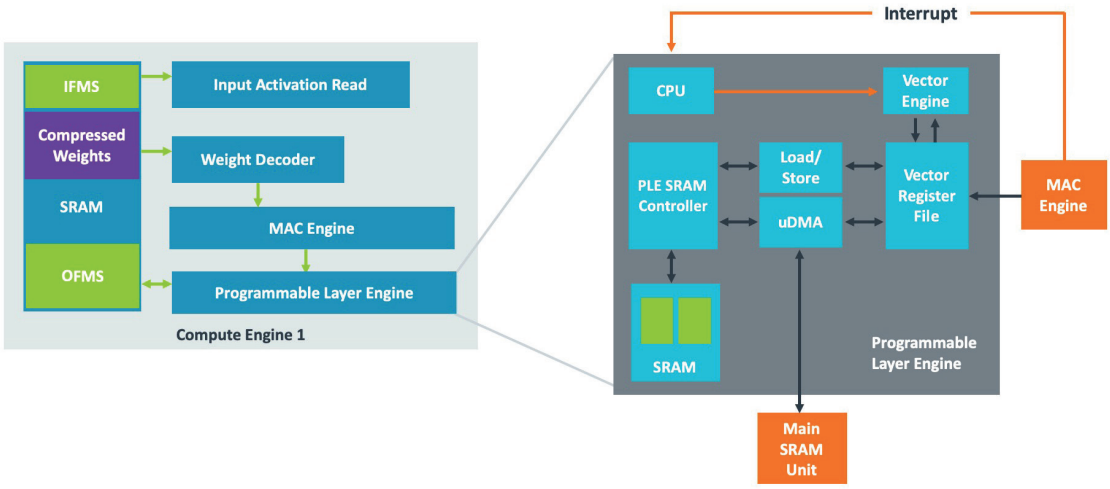
ML is constantly evolving, so it's important to be able to evolve a design to keep it operating optimally. To address this, the programable layer engine (PLE) enables new operators to be developed in software, after the hardware has been designed. These new operators can be included in future firmware updates, enabling feature updates after tape-out or after the device has shipped.



The programmable layer engine (PLE) in the Arm ML processor

The PLE refines and enhances results from the MAC engine, offering a way to implement new techniques as they emerge – even those not associated with convolutional operation. It includes Arm microcontroller technology, enhanced with add-ons for different instruction set architectures (ISAs), including vector and neural network extensions. These extensions are often targeted for non-convolutional operations, such as pooling, ReLu, and transcendentals.

As shown in the block diagram above, the PLE receives the computational results of the MAC engine and populates them in the PLE register file. Interrupts are sent to activate PLE processing. Since operations often serve to pool or reduce, the majority are performed by a 128-bit wide, 16-lane vector engine. Results are sent back to the PLE SRAM and then written out by a micro-DMA unit as output feature maps (OFMs) to the main SRAM in each compute engine. The PLE results are subsequently pulled back into the compute engine for processing.

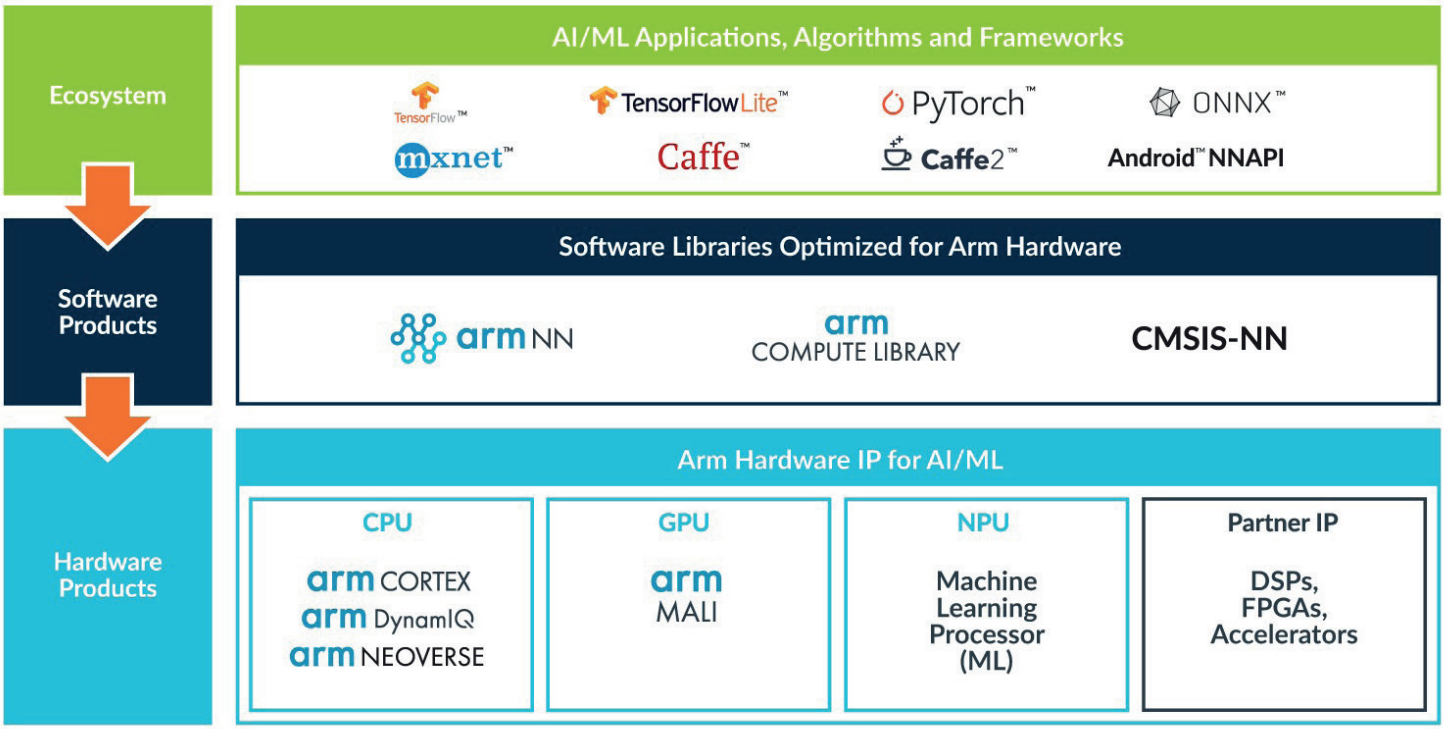


PLE operation

Seamless Integration; Boundless Support

The ML processor is part of a broad ML platform that emphasizes interoperability and convenience, and is designed to work seamlessly across Arm products – including compilers, drivers and libraries – as well as frameworks that reduce complexity, support scalability, and reduce development effort.

This wider platform – supported by a diverse and thriving ecosystem – is driving the next generation of smart devices through design flexibility, optimized performance and faster time to market.



Take the Next Step...

- Find out more about the Arm ML processor
- Download the ML processor datasheet
- Learn more about machine learning on Arm