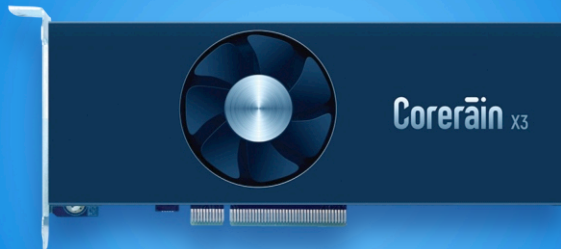# Nebula Accelerator X3

Corerāin
鲲 云 科 技

CAISA chip

Chip Utilization Ratio up to **95.4%**

* Based on Benchmark

ResNet-50	ResNet-152	VGG19	Inception-v4	YOLOv3	SSD-ResNet50	SSD-FPN	*KY-SSD	*U-Net	DeepLabv3+	U-Net Industrial
3.06 ms	8.68 ms	18.33 ms	14.6 ms	31.06 ms	21.96 ms	95.98 ms	2.31 ms	343.01 ms	119.37 ms	74.07 ms
1306.93 FPS	460.27 FPS	218.01 FPS	273.75 FPS	125.75 FPS	182.16 FPS	40.71 FPS	1676.19 FPS	11.39 FPS	32.73 FPS	54.01 FPS
92.32%	95.43%	78.53%	69.21%	82.37%	77.06%	70.64%	80.80%	85.37%	67.08%	64.97%

Note：Batch=4, INT 8. The neural network models above are based on TensorFlow, *KY-SSD and *U-Net are custom network models.

High Performance and Low Latency

High Versatility

High Cost Efficiency

User-friendly

An AI-specific computing acceleration board for deep learning inference at the edge and backend devices.
Empowered by the first commercial streaming AI chip, X3 provides 10.9 TOPS peak performance, and can achieve chip utilization ratio up to 95.4%.
Nebula Accelerator X3 uses PCIe 3.0 x8 interface, compatible with x86 architecture and Arm architecture server.
Users can seamlessly deploy algorithm models developed by TensorFlow, Caffe, PyTorch and ONNX(MXNet) frameworks to the Nebula Accelerator through the algorithm-to-hardware toolchain RainBuilder, providing deep learning computing acceleration solution with high performance, low latency, high versatility and high cost efficiency for edge and IDC devices.

Chip	CAISA
Peak Performance	10.9 TOPS
Chip Utilization Ratio	up to 95.4%
PCIe	PCIe 3.0 x8
Memory	8GB DDR4
Power Consumption	APD 56 W，Average Power Consumption 23.8 W
Power Supply	from PCIe slot without AUX power
Cooling Method	active Cooling Fan
Temperature	-20℃~70℃（Operation temperature）
Dimension	169.5 mm x 69.6mm (Standard half-length, half-height, single-slot)

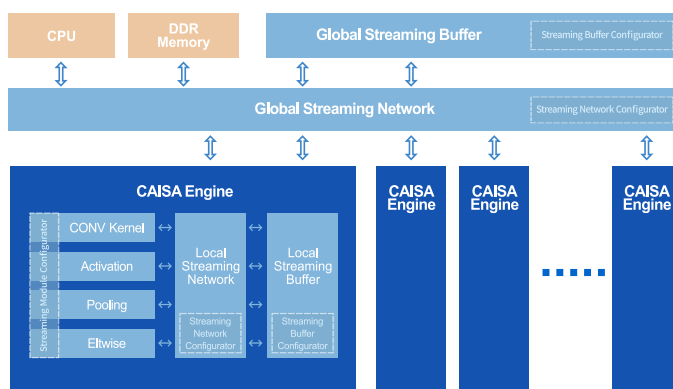**Application Equipment**

IPC   NVR   Server

1 Custom AI Streaming Accelerator
CAISA® Architecture

- High Cost Efficiency: Clock-level accurate calculations
- High Adaptability: Streaming network dynamic reconfiguration
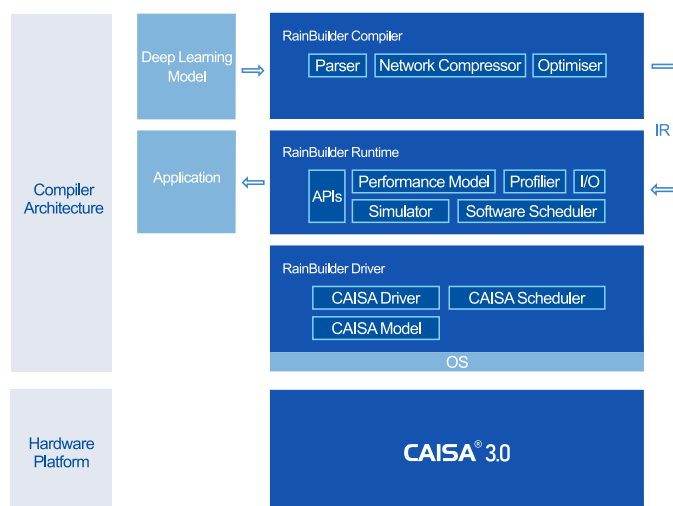- Versatile and User-Friendly: End-to-end automated deployment

CAISA Architecture

| CPU | DDR Memory | Global Streaming Buffer | Streaming Buffer Configurator |

Global Streaming Network — Streaming Network Configurator

CAISA Engine
- Streaming Module Configurator
  - CONV Kernel
  - Activation
  - Pooling
  - Eltwise
- Local Streaming Network
  - Streaming Network Configurator
- Local Streaming Buffer
  - Streaming Buffer Configurator

CAISA Engine · CAISA Engine · · · · · CAISA Engine

---

2 Model-to-Hardware Toolchain RainBuilder®

Specially designed for the CAISA architecture, RainBuilder provides rapid deployment for deep learning algorithms in just two steps. It supports TensorFlow, Caffe, PyTorch, ONNX(MXNet) and mainstream deep learning frameworks with a versatile and user-friendly developing environment.

- Automatically converts models developed in deep learning frameworks such as TensorFlow, Caffe, PyTorch, and ONNX(MXNet)
- High versatility and supports various CNN models such as ResNet, YOLO, DeepLab
- User-friendly, supports C++/Python standard development process, no need to understand the underlying hardware architecture

Compiler Architecture

Deep Learning Model

RainBuilder Compiler
- Parser | Network Compressor | Optimiser

Application

RainBuilder Runtime
- APIs | Performance Model | Profiler | I/O
- Simulator | Software Scheduler

IR

RainBuilder Driver
- CAISA Driver | CAISA Scheduler
- CAISA Model

OS

Hardware Platform

CAISA® 3.0

Operation Instruction

Model → RainBuilder Compiler → IR → RainBuilder Runtime → [CoreRain] → Applications → Smart City / ADAS / Smart Manufacturing

Model:
- Caffe
- TensorFlow
- PyTorch
- ONNX

```
//  set target device as ASIC
cfg.device_priority = {DeviceType::ASIC};
// the 8bit quantized model file
std::string sg_def_file = "./model_8bit.sg";
//create a runner with specified model file and configuration
this->runner_ = BuildRunner(sg_def_file, cfg);
//set input image into input datamap
input_datamap.Set(nodedef.output(0), input_tensor);
input_datamap.SetName(imagefile);
//Run the forward func of network
runner_->RunAsync(input_datamap);
```