Deploying ML Applications at the Edge with AKIDA and MetaTF

K. D. Carlson, D. Corvoysier, D. McClelland, & K. Tsiknos
BrainChip, 23041 Avenida De La Carlota, Suite 250, Laguna Hills CA 92653

Contacts: kcarlson@brainchip.com
rtelson@brainchip.com

Introduction

The Akida AKD1000 is an event-based, neuromorphic system-on-a-chip (NSoC) designed to efficiently accelerate traditional machine learning (ML) algorithms using brain-inspired hardware designs. The AKD1000 differentiates itself from traditional deep learning accelerators (DLAs) by implementing event-based convolutional operations and on-chip, few-shot learning along with very low memory requirements and a focus on collocating computational elements and memory.

However, designing, training, and deploying ML applications at the edge is a non-trivial task that requires a mature ML development framework. To solve this problem, BrainChip has developed MetaTF, an ML development framework that greatly accelerates ML application development and deployment on Akida. Most importantly, MetaTF will be familiar to ML application developers as it is written in Python, supports TensorFlow, and has a high-level API inspired by Keras.

MetaTF ML Framework Overview

The MetaTF ML framework is comprised of three main Python packages:

- The Akida Python package is an interface to the BrainChip Akida NSoC. To allow the development of Akida models without actual Akida hardware, it includes a runtime, a hardware abstraction layer (HAL), and a software backend that simulates the Akida NSoC.
- The CNN2SNN tool converts convolutional neural networks (CNNs) trained using deep learning methods to event-domain, low-latency, and low-power neural networks for use with the Akida runtime.
- The Akida model zoo contains pre-built network models built with the Akida sequential API and the CNN2SNN tool using quantized TF Keras models.

Documentation can be found at: https://doc.brainchipinc.com/

The Akida package is explained in detail below in Figure 1.

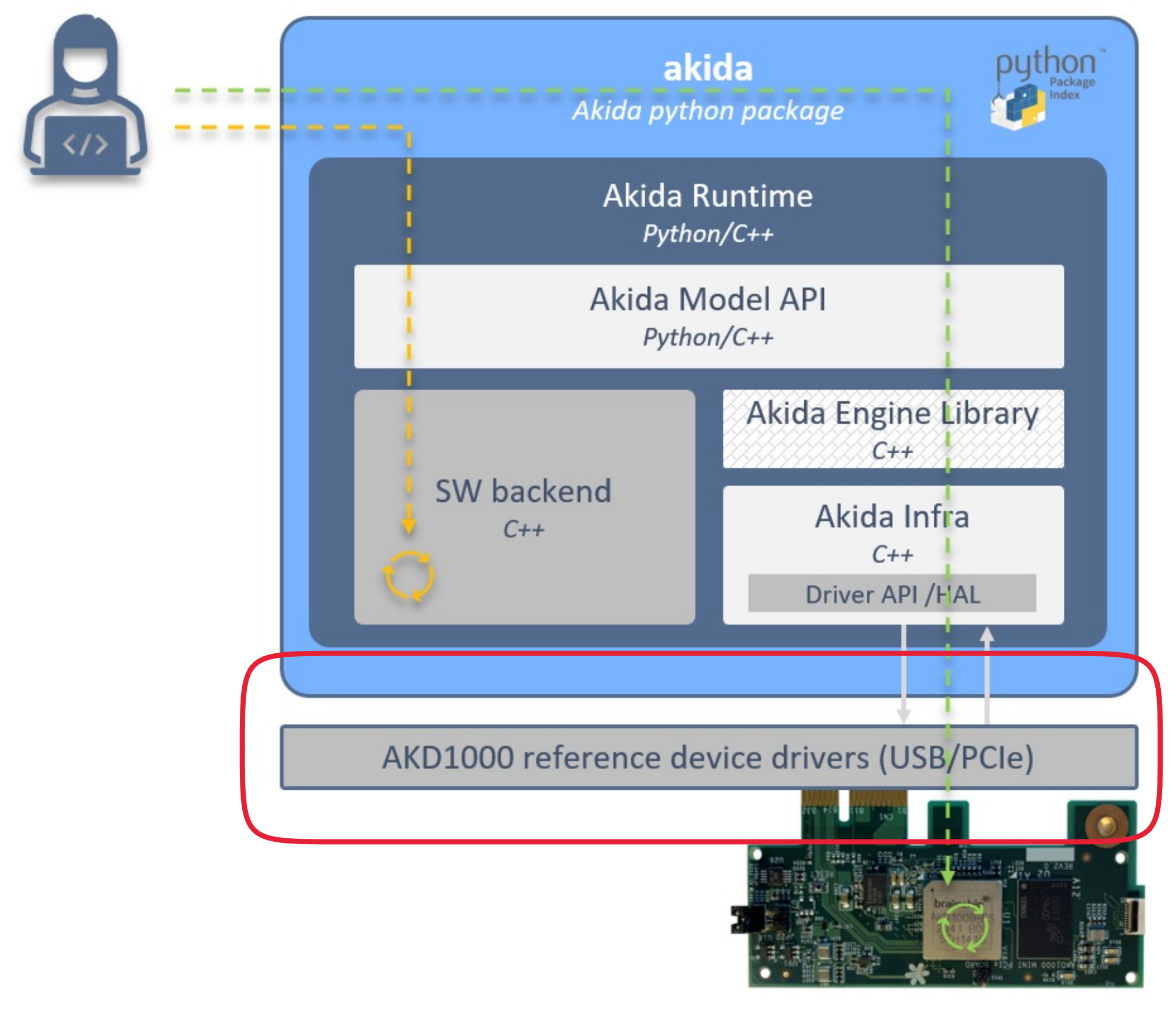

Figure 1. The **Akida Model API** library supports the creation of Akida models, the inferencing and serialization of instantiated models, and their mapping on hardware devices. The **SW backend** simulator is a CPU implementation of the Akida training and inference. The **Akida Engine Library** is a C++ library that supports model instantiation and inference on hardware devices. Finally, **Akida Infra** denotes the HAL.

MetaTF ML Framework Runtime

Figure 2 shows the different runtime configurations. Users can use the software simulation in place of Akida hardware.

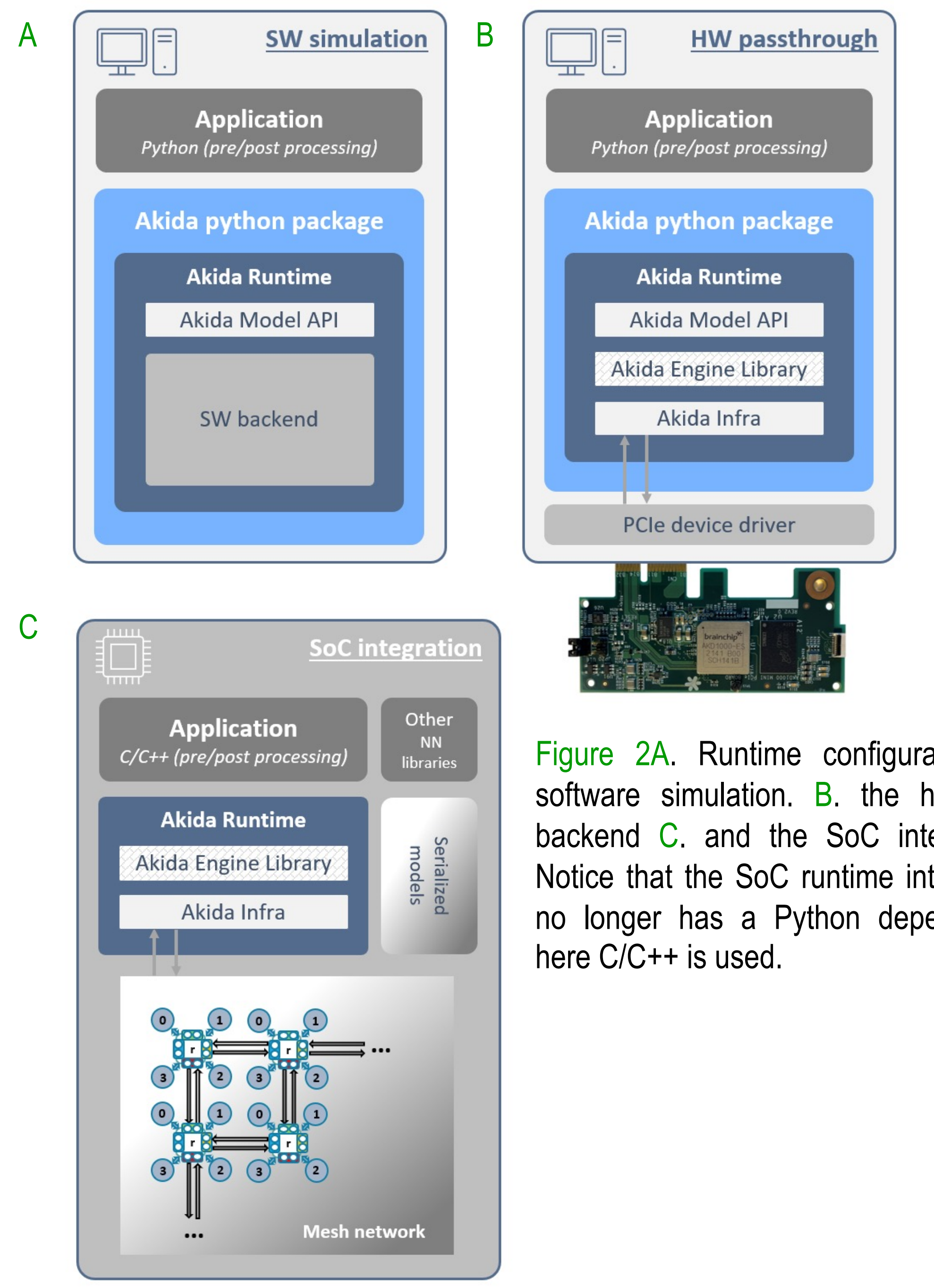

Figure 2A. Runtime configuration for software simulation. B. the hardware backend C. and the SoC integration. Notice that the SoC runtime integration no longer has a Python dependency, here C/C++ is used.

MetaTF ML Workflow

The MetaTF workflow for downloading a pretrained model, converting it to Akida, mapping the model to a device, and displaying a summary is shown below. API calls from cnn2snn and akida Python packages allow users to prototype in a familiar Python/TensorFlow Keras environment. The akida API has been expanded to allow users to map their model to virtual hardware devices.

```python
#!/usr/bin/env python
import os
from akida import AKD1000
from cnn2snn import convert
from akida_models import ds_cnn_kws_pretrained

# Load Keras pre-trained model from Akida model zoo
model_keras = ds_cnn_kws_pretrained()

# Convert Keras model to Akida
model_akida = convert(model_keras)

# Map/compile converted model for the AKD1000 chip
model_akida.map(device=AKD1000())

# Check model mapping: NP allocation and binary size
model_akida.summary()
```

Converted model can be tested with:
- Akida simulator
- Akida and PCIe development board

Output of the model summary is shown here

```
      HW/conv_0-dense_5 (Hardware) - size: 23440 bytes
_____________________________________________________________
Layer (type)              Output shape   Kernel shape    NPs
=============================================================
conv_0 (InputConv.)       [5, 25, 64]    (5, 5, 1, 64)   N/A
_____________________________________________________________
separable_1 (Sep.Conv.)   [5, 25, 64]    (3, 3, 64, 1)   1
_____________________________________________________________
                                         (1, 1, 64, 64)
_____________________________________________________________
separable_2 (Sep.Conv.)   [5, 25, 64]    (3, 3, 64, 1)   1
_____________________________________________________________
                                         (1, 1, 64, 64)
_____________________________________________________________
separable_3 (Sep.Conv.)   [5, 25, 64]    (3, 3, 64, 1)   1
_____________________________________________________________
                                         (1, 1, 64, 64)
_____________________________________________________________
separable_4 (Sep.Conv.)   [1, 1, 64]     (3, 3, 64, 1)   1
_____________________________________________________________
                                         (1, 1, 64, 64)
_____________________________________________________________
dense_5 (Fully.)          [1, 1, 33]     (1, 1, 64, 33)  1
_____________________________________________________________
```

You can use virtual devices to find out how many neural processors (NPs) a model requires without the needing the physical hardware

AkidaNet

Recently, we have developed a replacement for the popular MobileNet v1 model used as a backbone in many applications that we call AkidaNet. AkidaNet's architecture utilizes the Akida hardware more efficiently. Some of our preliminary results are shown below for object classification, face recognition, and face detection. In many cases, switching from MobileNet v1 to AkidaNet results in a slight increase in speed and accuracy accompanied by a 15% to 30% decrease in power usage.

We describe MobileNet and its variants using the same width multiplier (α) convention detailed in Howard et al., 2017. The width multiplier scales the input and output channels of all layers by α, so α = 0.5 produces a model with 50% fewer input and output channels in all layers.

Table 1. Object classification model performance changes MobileNet v1 → AkidaNet

MobileNet Variant	Data Set Resolution	Classification Accuracy (% change)	% Power Change	FPS (% Change)
α = 0.5	ImageNet 224×224	61.30% (+1.54%)	(-15.87%)	33 (+6.45%)
α = 0.25	ImageNet 224×224	46.71% (+1.59%)	(-29.86%)	70 (+7.69%)

Table 2. Face recognition model performance changes MobileNet v1 → AkidaNet

MobileNet Variant	Data Set Resolution	Classification Accuracy (% change)	% Power Change	FPS (% Change)
α = 0.5	LFW 112×96	97.25% (-0.02%)	(-25.48%)	106 (+10.42%)
α = 0.5	Casia Webface 112×96	71.13% (-0.27%)	(-26.27%)	74 (-2.63%)

Table 3. Face detection model performance changes MobileNet v1 → AkidaNet

MobileNet Variant	Data Set Resolution	mAP (% change)	% Power Change	FPS (% Change)
α = 0.5	Widerface (224x224)	72.00 (+0.75%)	(-17.78%)	35 (+2.94%)

Akida Model Zoo

The Akida model zoo is a collection of pretrained Akida-compatible ML models available for download at https://doc.brainchipinc.com/zoo_performances. In the tables below, we show selected models and describe the architecture, input resolution, data set, quantization levels, accuracy, size (in KB), and number of neural processors (NPs) each model requires.

Table 4. Selected image classification models available.

Architecture	Resolution	Dataset	Quantization	Top-1 accuracy	Size (KB)	NPs
AkidaNet	224	ImageNet	8/4/4	69.65%	6322.6	129
AkidaNet 0.5	224	ImageNet	8/4/4	61.30%	1214.4	38
AkidaNet 0.5	160	Cats vs Dogs	8/4/4	96.60%	698.4	24
AkidaNet 0.5	224	Imagenette	8/4/4	95.67%	815.5	32
AkidaNet 0.25	224	Imagenette	8/4/4	91.54%	203.9	22
AkidaNet 0.5	224	SIIM-ISIC Melanoma Classification	8/4/4	98.31% - AUROC 0.7969	811.4	32
AkidaNet 0.5	224	PlantVillage	8/4/4	97.92%	1018.8	33
AkidaNet 0.25	96	Visual Wake Words	8/4/4	82.75%	227.7	17

Table 5. Selected face recognition models available.

Architecture	Resolution	Dataset	Quantization	Top-1 accuracy	Size (KB)	NPs
AkidaNet 0.5	112x96	CASIA Webface face identification	8/4/4	70.18%	1929.8	21
AkidaNet 0.5	112x96	LFW face verification	8/4/4	97.25%	691.2	20

Table 6. Selected time-series and point cloud classification models available.

Architecture	Resolution	Dataset	Quantization	Top-1 accuracy	Size (KB)	NPs
AkidaNet 0.5	224	Physionet2017 ECG classification	8/4/4	73.50% - AUROC 0.7940	1008.4	36
PointNet++	Input Scaling (127,127)	ModelNet40 3D Point Cloud	8/4/4	84.76%	528.5	17

Example Use Case: DVS Gesture

DVS Gesture (Amir et al., 2017) is a well-known event-based dataset, comprising recordings of subjects performing gestures (clapping, waving, circular motions etc.) made with a DVS128 event-based camera. Here we present our approach to achieving state of the art performance on this dataset, using a MetaTF training pipeline.

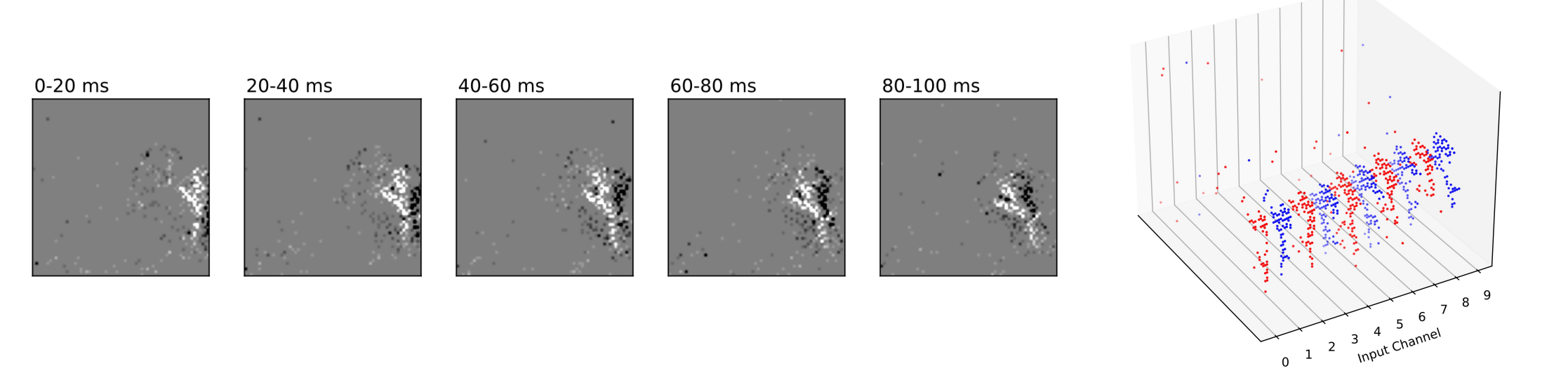

Figure 3. **Data Preparation.** Although the data are event-based, we are able to use a standard Keras training pipeline. Temporal snapshots of the data (100 ms) are prepared as 'images'. We use the 'channel' dimension of the images to encode extra temporal information: the 100 ms time window is subdivided into 5 x 20 ms bins, and each of these is encoded across two channels for event polarity (On vs Off events). For each 'pixel', we simply sum the number of events in each time bin (clipped at 15 for the 4-bit case).

Training:
- Standard Keras training pipeline (16 epochs; Adam optimizer; cyclic LR schedule with max LR 10^{-3} and min 10^{-5}; data augmentation 0.05 random rotation; 0.02 random translation)
- **Quantize** to 4-bits for both weights and activations using default MetaTF quantization tool. Retrain for 8 epochs (as above, but max and min LR reduced to 10^{-4} and 10^{-6} respectively)
- **Conversion** to akida format is simple via the MetaTF method.

Models:
- "Akida Tiny": MP2 – C16 MP2 – C16 MP2 – C32 MP2 – C64 MP2 – C128 GAP – SC256 – Classifier
- "Akida Small": Mp2 – C32 MP2 – C64 MP2 – C64 MP2 – C128 MP2 – C128 GAP – SC256 – Classifier

Results:

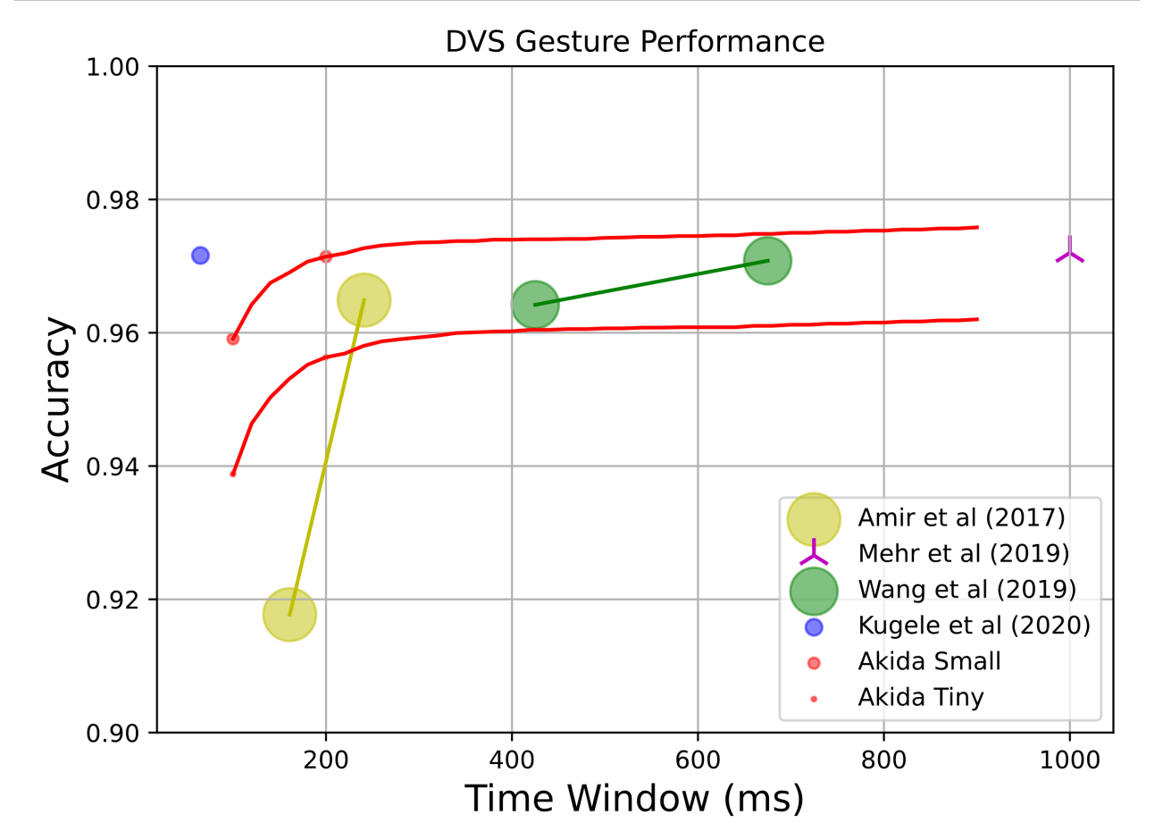

Figure 4. **Test accuracy as a function of temporal processing window.** As is apparent for both Akida models, and for other state-of-the-art solutions, integrating a longer input data window is strong factor in determining accuracy. Symbol diameter is proportional to the number of network parameters in each model. (*Mehr et al. use a true SNN so direct comparison is difficult)

	# Params (million)	# ops (Mops)
Amir	1.53	-
Mehr	*	-
Wang	1.37	-
Kugele	0.48	651
Ak. Small	0.32	11
Ak. Tiny	0.14	2

Table 7. **Number of parameters and operations for SOTA models.** Both Akida models are much smaller than the other networks. The only model with a higher performance on a shorter integration time window (See Fig. 4), is that of Kugele et al. which requires 1-2 orders of magnitude more operations.

References

- Amir, A. et al. (2017). 'A low power, fully event-based gesture recognition system' *IEEE Conference on Computer Vision and Pattern Recognition*, pp 7388-7397, doi: 10.1109/CVPR.2017.781
- Howard, A. G. et al. (2017) 'MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications' arXiv:1704.04861 [cs]
- Kugele, A. et al (2020) 'Efficient Processing of Spatio-Temporal Data Streams with Spiking Neural Networks' *Frontiers in Neuroscience* 14 doi: 10.3389/fnins.2020.00439
- Mehr, A. et al (2019) 'Action recognition using supervised spiking neural networks.' arXiv:1911.03630
- Wang, Q. et al (2020) 'Space-time event clouds for gesture recognition: from RGB cameras to event cameras'. IEE WACV doi: 10.1109/WACV.2019.00199