The Akida package is explained in detail below in

*collocating*

The training designed developers on 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 NoC. 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 NoC.
- 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://docs.brainchip.com/](https://docs.brainchip.com/)

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

The Akida Model API library supports the creation of Akida models, the inferencing and serialization of instantimized models, and their mapping on hardware devices. The SW back-end 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 net defines the HAL.

**Figure 1. The Akida Model API Library supports the creation of Akida models, the inferencing and serialization of instantimized models, and their mapping on hardware devices. The SW back-end 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 net defines the HAL.**

**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.**

**Figure 2. 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 onnx 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.**

**Converting model can be tested with:**

- Akida simulator
- Akida and POE development board

**Output of the model summary is shown here.**

**Figure 2B. An overview of the Akida framework runtime.**

**Figure 3. A data preparation model showing the AkidaNet performance.**

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 place 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.**

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>Accuracy</th>
<th>% Reduction</th>
<th>% Accuracy</th>
<th>% Power</th>
<th>% Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>224x224</td>
<td>75.00</td>
<td>-15.00%</td>
<td>90.00%</td>
<td>-15.00%</td>
<td>-15.00%</td>
</tr>
<tr>
<td>AkidaNet</td>
<td>224x224</td>
<td>87.60</td>
<td>-15.00%</td>
<td>96.80%</td>
<td>-15.00%</td>
<td>-15.00%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>Resolution</th>
<th>% Change</th>
<th>% Change</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>112x112</td>
<td>(112,112)</td>
<td>-15.00%</td>
<td>-15.00%</td>
<td>-15.00%</td>
</tr>
<tr>
<td>AkidaNet</td>
<td>112x112</td>
<td>(112,112)</td>
<td>78.60%</td>
<td>78.60%</td>
<td>78.60%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>Resolution</th>
<th>% Change</th>
<th>% Change</th>
<th>% Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>224x224</td>
<td>(224,224)</td>
<td>-15.00%</td>
<td>-15.00%</td>
<td>-15.00%</td>
</tr>
<tr>
<td>AkidaNet</td>
<td>224x224</td>
<td>(224,224)</td>
<td>75.00%</td>
<td>75.00%</td>
<td>75.00%</td>
</tr>
</tbody>
</table>

**Table 4. Pre-trained ML models.**

**Table 5. Selected face recognition models.**

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

**Table 7. Summary of parameters and operations for SOTA models.**

**References**