

Tengine嵌入式AI框架的挑战和实践

OPEN AI LAB

王海涛

2020.4.8

目录

嵌入式AI面临的挑战和Tengine的解决方案

Tengine 框架解析

Tengine API 简介

实践1: Tengine 扩展, 定制和添加算子

实践2 : Tengine 在CPU/GPU/NPU/DLA上的推理

OPEN AI LAB

- OPEN AI LAB（开放智能）于2016年成立，公司专注边缘智能计算及应用，致力于推动芯片及算力、算法、工程产品化、行业应用等完整产业链的深度协作，加速人工智能产业化部署和场景的边界拓展，赋能场景化细分行业快速实现+AI。为AIoT产业上下游合作伙伴提供端、边、云一体化人工智能基础软硬件开发平台及应用级解决方案

开放平台 以自主知识产权的**Tengine**为核心，构建跨硬件平台、开放兼容的AI应用开发平台，加速人工智能技术在细分行业领域高效的应用落地

行业覆盖 覆盖芯片公司、算法公司、IDH公司、OEM/ODM公司等

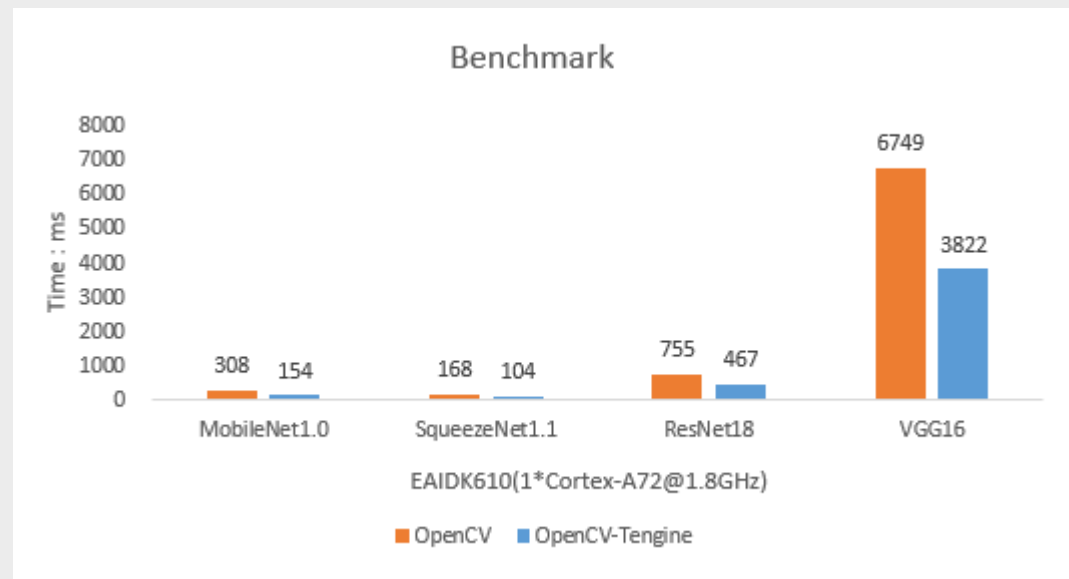
产业赋能 为产业伙伴及行业用户提供智能解决方案、以及技术赋能服务，助力**赋能万物，万物智能**



Tengine as OpenCV Arm DNN Backend



Arm Backend



<https://github.com/opencv/opencv/wiki/Tengine-based-acceleration>

更多介绍,请参考

[OpenCV 4.3.0 with Tengine \(稳定版\)](#)

[OpenCV 4.3.0 with Tengine \(飙车版\) speed up x2](#)



Tengine-自主知识产权的商用级 AIoT智能开发平台

Tengine与众多国内芯片建立深度合作关系，
致力于打造AIoT底层生态圈，
为AIoT应用开发打通在各类硬件平台部署算法的工作流，
加速面向场景的AI算法在嵌入式终端/边缘上快速迁移

AI如电力，将无处不在

产业链碎片化严重，极大影响大量行业的AI进程

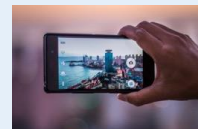
边缘AI推理应用市场爆发，现有软件硬件不能满足需求



缺乏合适的中间层和计算库，AI软件对硬件利用率不高



没有统一的平台和接口，生态碎片化严重.



...

...

目前AI产业链开发效率低下



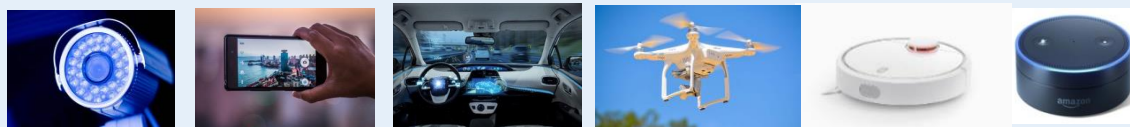
芯片公司被迫花大量精力做上层开发环境和平台



应用/算法公司被迫做大量底层适配优化成为全栈AI公司

CPU GPU DSP NPUs

Tengine 赋能产业链



Caffe

mxnet

PyTorch



...

Tengine

arm

ALLWINER

Rockchip
瑞芯微电子AI
IMICRO

QUALCOMM

HISILICON

NVIDIA

MEDIATEK
联发科技

...

CPU GPU AIPU DSP XPU's

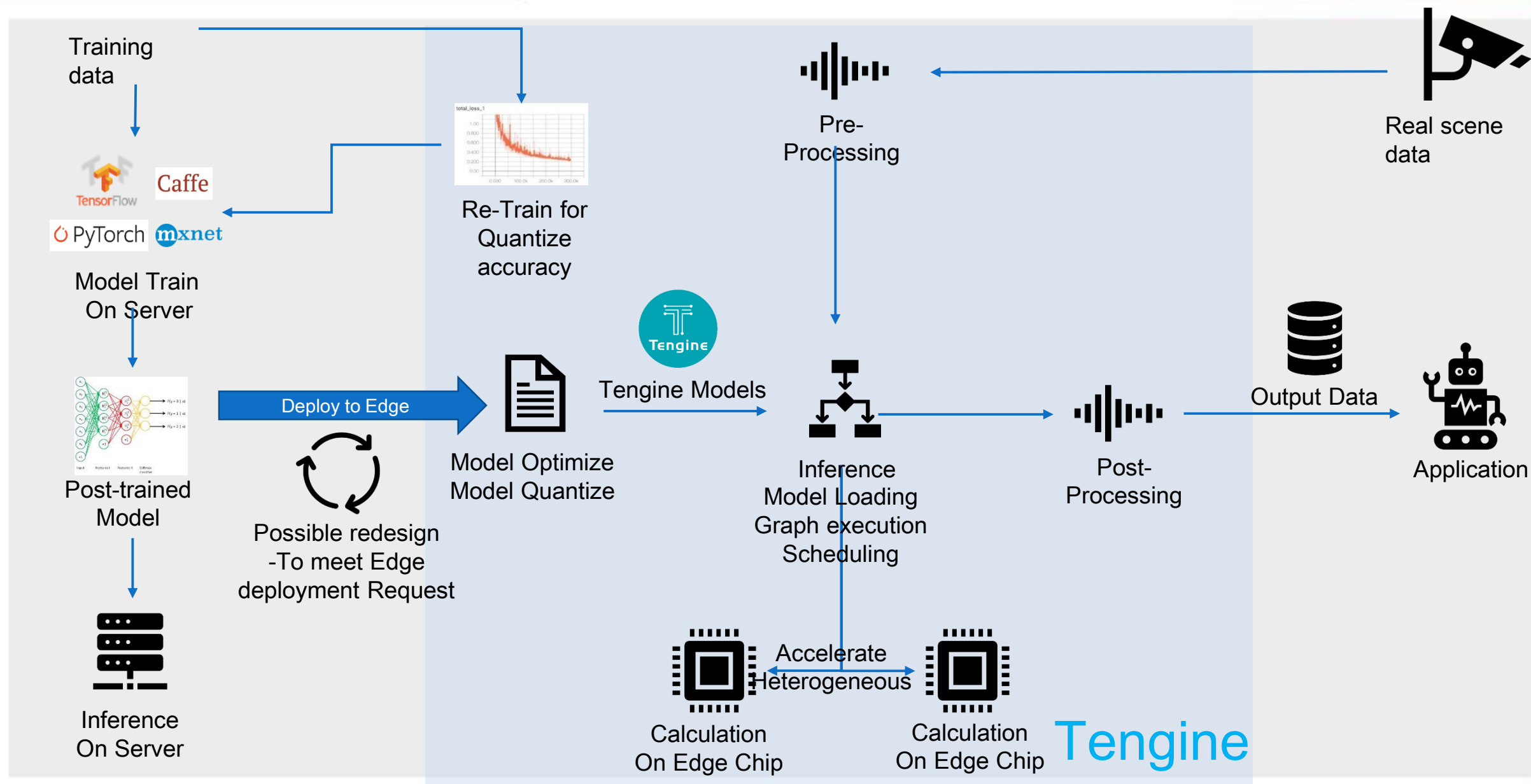
赋能AI开发，助力更多企业进入AI行业



开源项目 <https://github.com/OAID/Tengine>

Tengine助力AIoT应用开发流程

OPEN AI LAB



Tengine 框架解析

Tengine 整体架构简介

训练框架模型适配

计算图执行

高性能计算库

量化精度调优工具



支持主流模型格式



支持主流操作系统



Powered by
Tengin

e



模型转换方案

输入模型

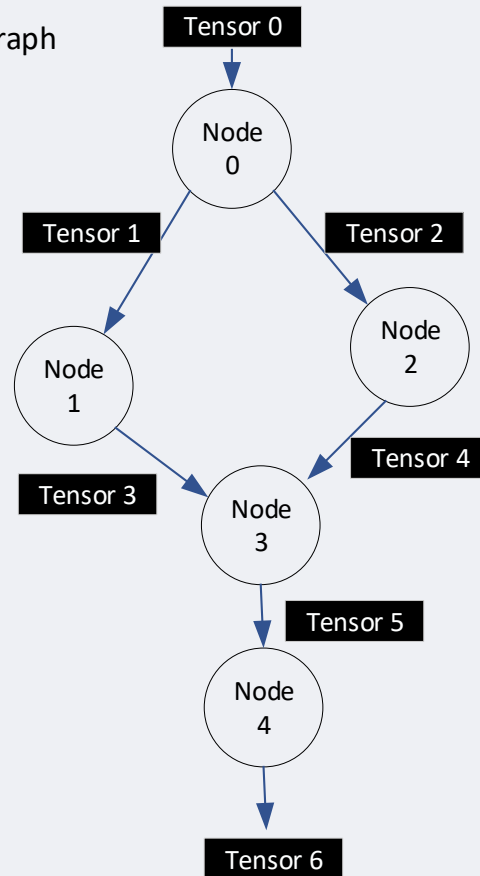


转换工具

Python 脚本

Tengine NNIR

Graph




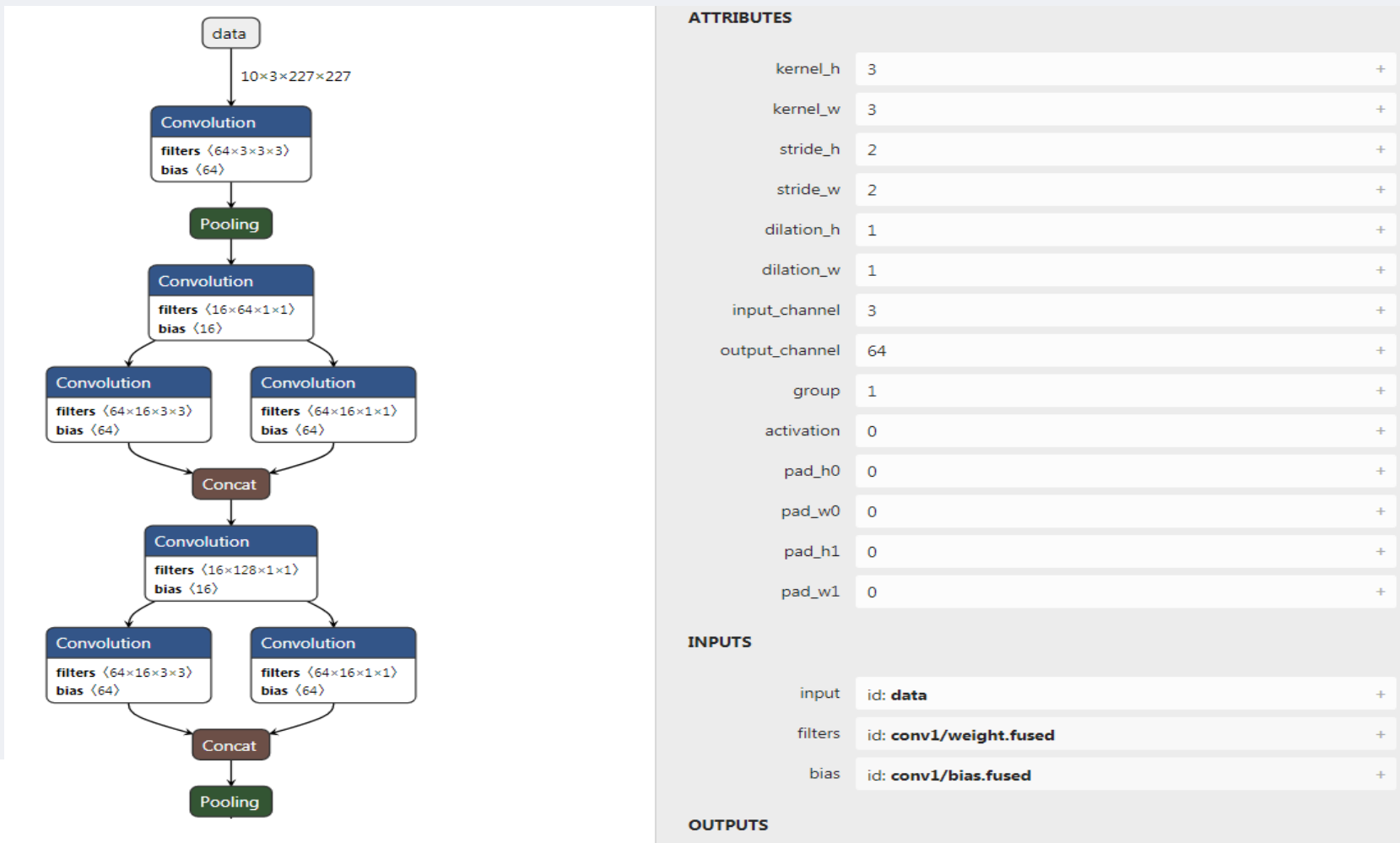
输出模型



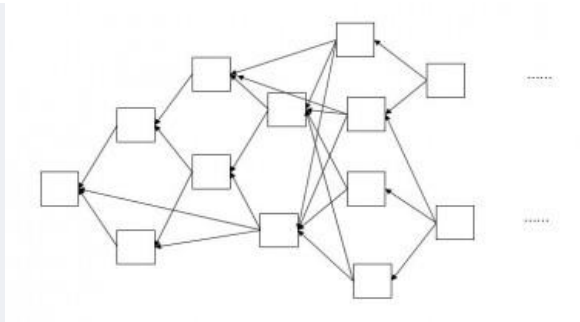
Caffe

Tengine模型可视化工具

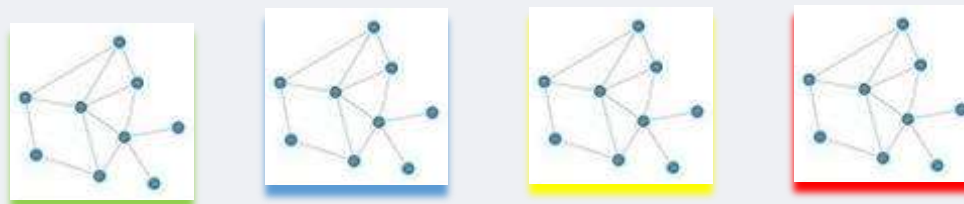
- 与开源社区的主流模型可视化软件 netron 合作, 实现了netron 对Tengine 模型解析的支持.
- NETR  N <https://github.com/lutzroeder/netron/releases>



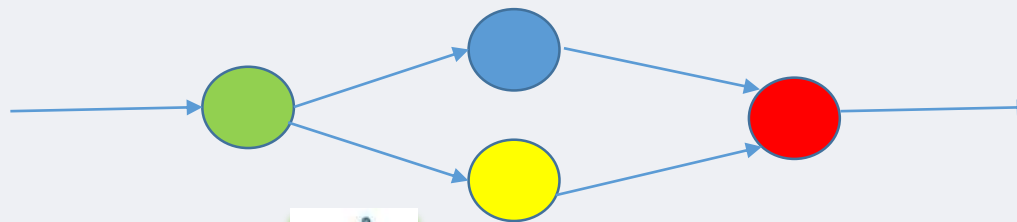
Graph: Tengine NNIR



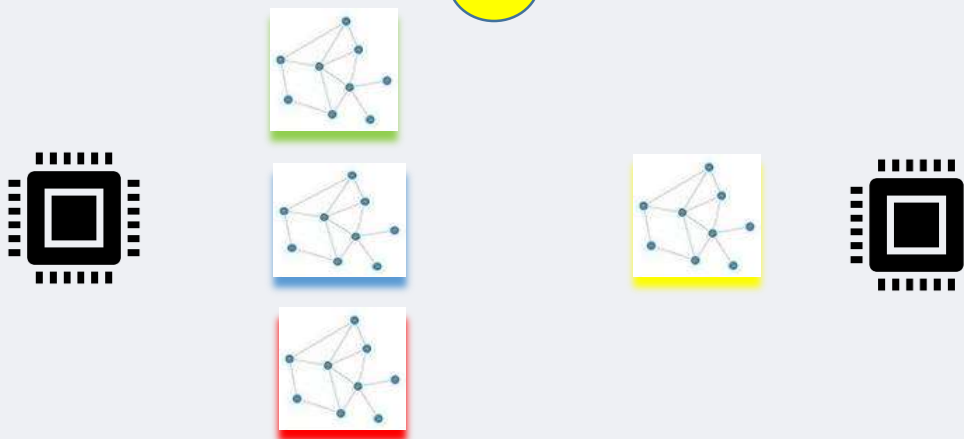
Graph Partition



Graph Scheduling



Device Execution



发挥硬件极致性能的计算

重点优化卷积等最耗时算子

Convolution/Fully Connected/Pooling

多种卷积计算模式 GEMM / Direct / Winograd

双线程加速比达175%，四线程加速比达300%

针对CPU微架构优化汇编

实测Convolution MAC利用率高达70%

适配Arm Cortex-A72/A17/A53/A7/A73/A55/A76

支持FP32/FP16/INT8数据格式计算

INT8相比FP32提速50-90%

混合精度计算模式, 精度损失大的层采用fp32计算

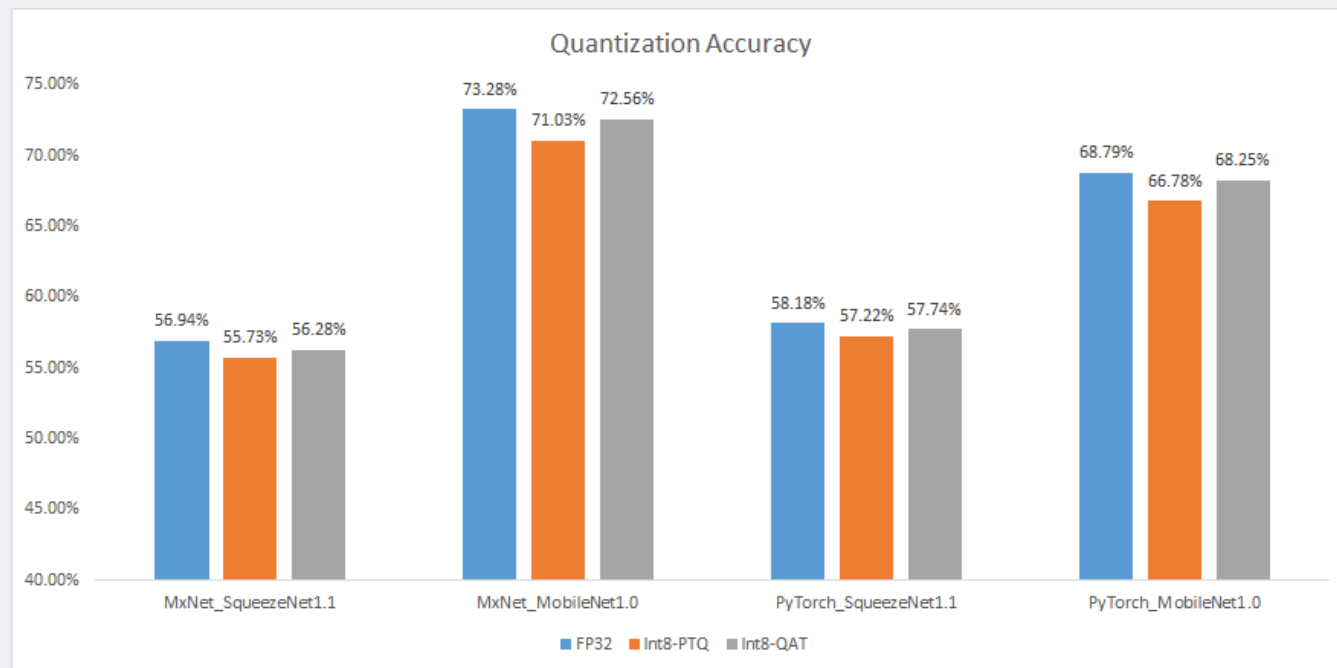
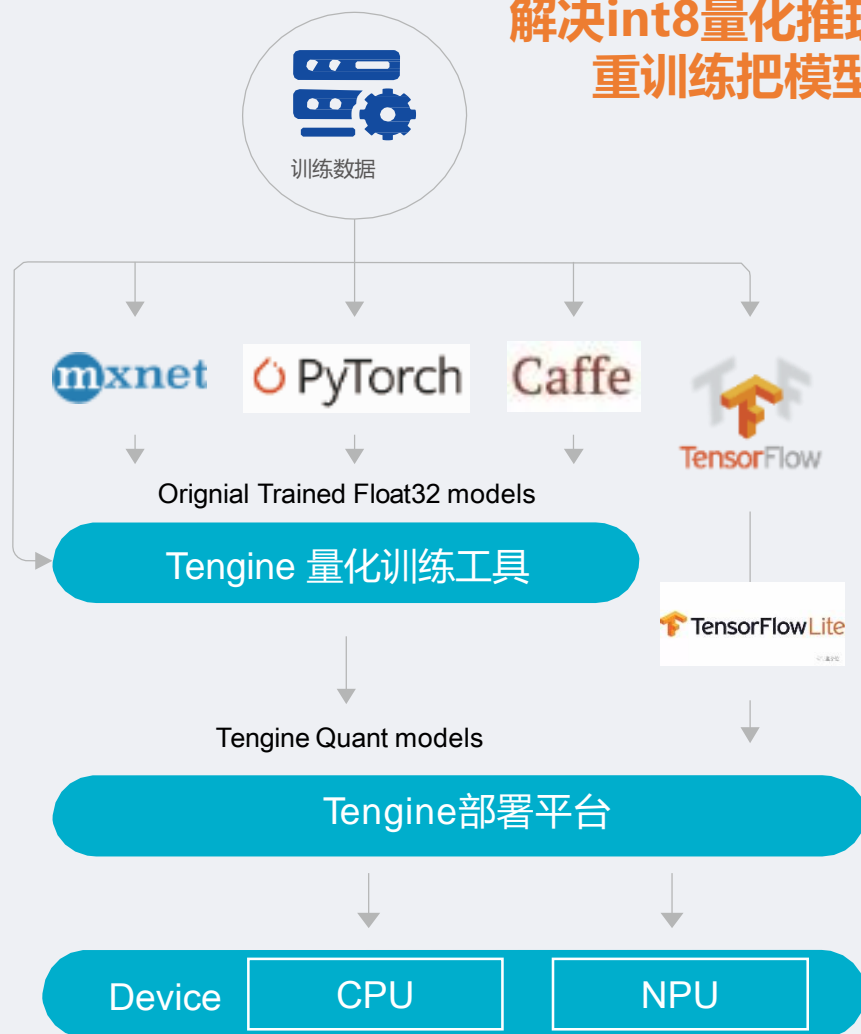
开源版本FP32性能

EAIDK610 RK3399

Model	1 A72	2 A72	GPU
Squeezenet	53.78	36.30	31
MobilenetV1	98.2	57.26	34.4

量化重训练工具

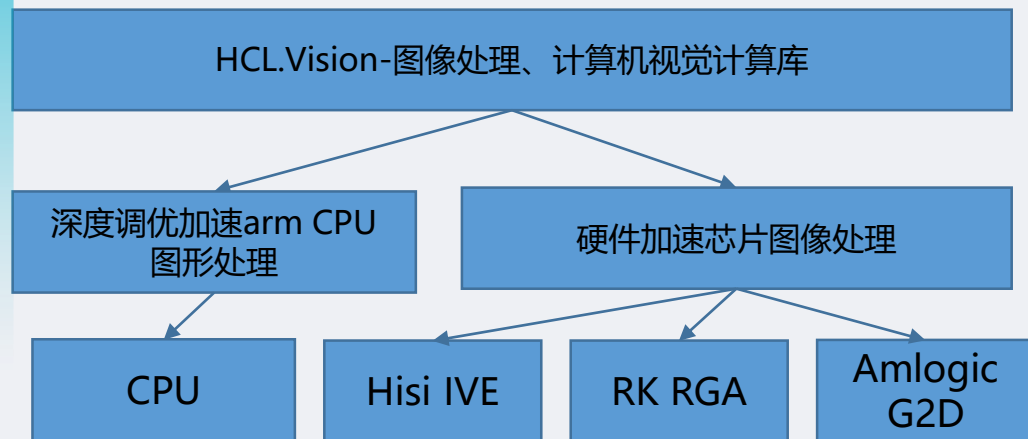
解决int8量化推理精度下降问题的有效方法-量化重训练把模型变成更适合int8推理的模型



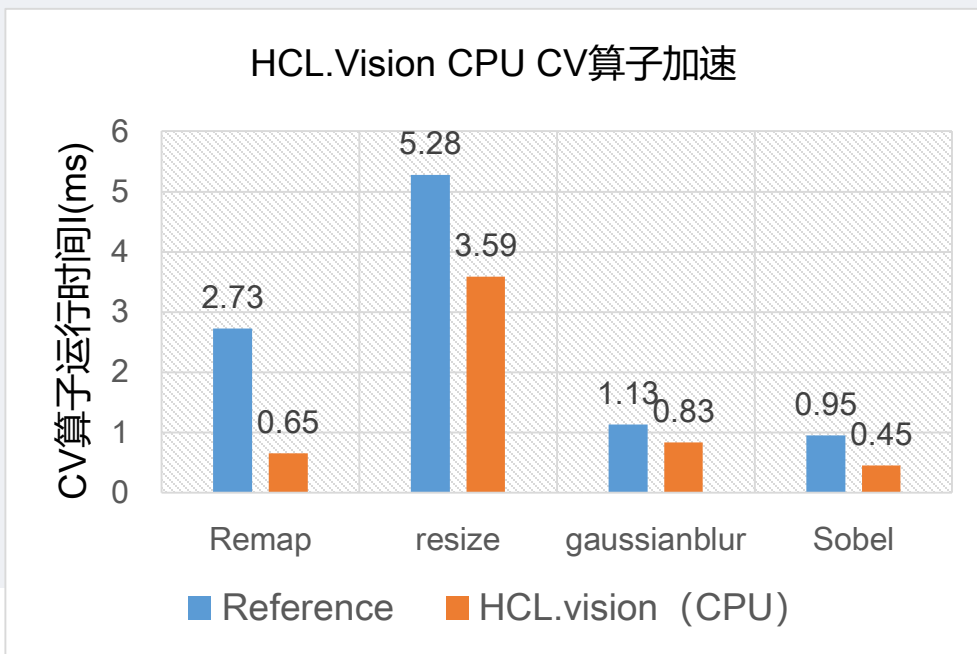
推理精度对比-fp32/离线量化/量化重训练

适合金融、人脸识别等高精度需求应用

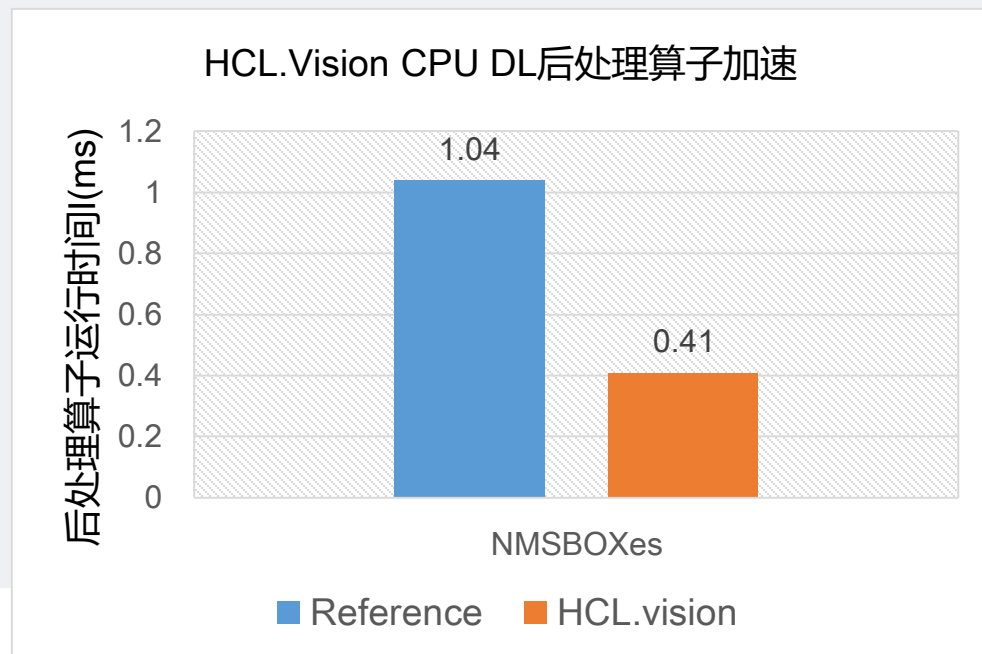
HCL.Vision-专为嵌入式神经网络推理设计的图像处理库



- 涵盖常见图像处理算子，以满足神经网络推理前后处理需求
- **硬件加速器和CPU算子统一接口**，在有图像处理芯片的情况下优先调用硬件加速算子,手工调优汇编的CPU CV算子作为补充
- 神经网络后处理算子CPU实现并深度加速（如NMSBoxes）



Test on RK3399 Linux



Test on RK3399 Linux

Tengine **API**

API 概述与应用举例

C++ API

Python API

C API

Tengine API 概述

参考了 Android NN/OpenVX/TensorRT/ Tensorflow/MXNet 等接口设计思路 and 原则

接口定义与实现完全无关

注重接口稳定性和灵活性, 以利于Tengine 应用和生态的长期稳定性和发展

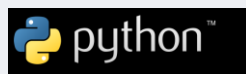
对多硬件支持的友好性

可以支持实现训练框架

采用C作为核心API, 基于C 封装出更易于使用的C++/Python等接口



C++ API



Python API

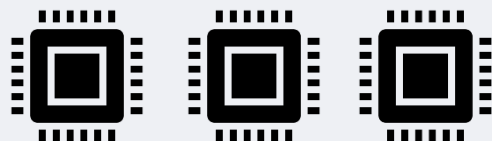


JS API



Java API

Tengine C API



- 应用程序从传统CPU芯片可快速迁移到AI加速器，减少学习成本，提高应用开发部署速度
- 应用程序不需要为多条产品线不同芯片适配接口，统一应用开发平台，减少版本维护工作量

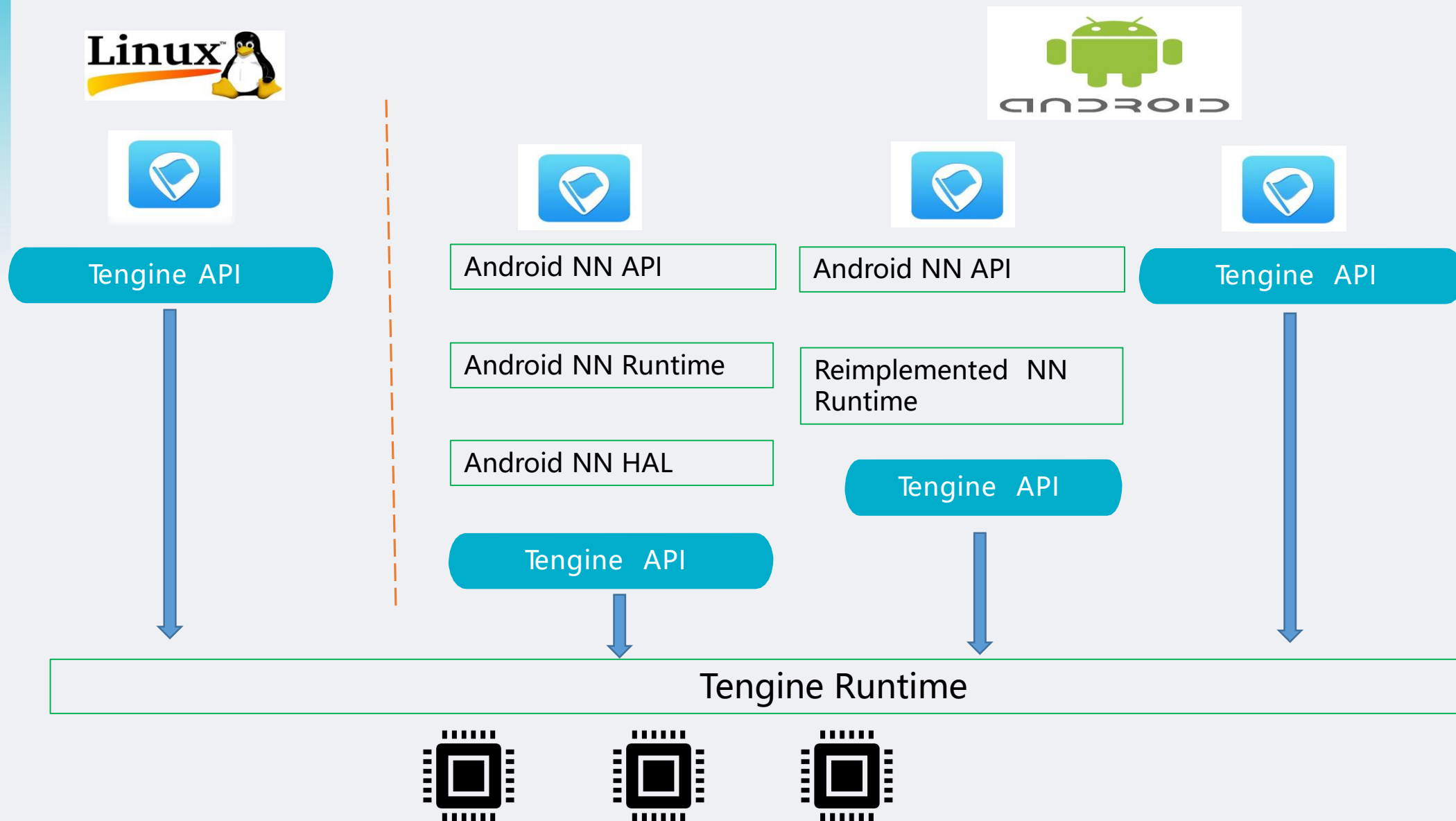
```
// RK3288, Raspberry Pi ...  
graph_t graph = create_graph(NULL, "tengine",  
model_file);
```

```
run_graph(graph,1);
```

```
void * output=get_tensor_buffer(output_tensor);
```

```
// MCU STM32F4 ..  
create_graph(NULL, "tiny", model_mem);  
  
// Arm China AIPU  
create_graph(NULL, "zhouyi", model_file);  
  
// Hi3519av100, Hi3516cv500  
create_graph(NULL, "nnie", model_file,model_config);  
  
// RK3399Pro  
create_graph(NULL, "rk3399pro", model_file);  
  
// Amlogic A311D  
create_graph(NULL, "vx", model_config_file);
```


Tengine 应用方式举例



C++ API

C++ API 为快速构建一个推理应用，提供了简单易用的接口。主要包含 Net和Tensor两个类。

- Tensor: 推理的输入/输出数据，包含数据维度和缓存
- Net: 推理网络

典型的用法如下

- 加载模型: `net.load_model(NULL, "engine", model_file)`
- 设置输入数据: `net.input_tensor(input_tensor_name, input_tensor);`
- 执行推理: `net.run()`
- 获取结果: `net.extract_tensor(output_tensor_name, output_tensor)`
- https://github.com/OAID/Tengine/blob/master/core/include/tengine_cpp_api.h
- https://github.com/OAID/Tengine/blob/master/benchmark/src/bench_sqz.cpp

Python API 提供了在Python下使用Tengine的快速接口，一个用于squeezenet 推理的例子如下：

#加载模型

```
graph = tg.Graph("tengine", model_file)
```

#设置输入数据

```
graph.set_input(image=img, shape=dims, input_idx, mean=mean, scale=scale)
```

#进行推理

```
graph.run()
```

#获取结果

```
result = graph.get_output(output_idx)
```

C API概述

API 可以分为

- Engine的初始化, 销毁, 版本查询和检查
- Graph 的接口: Graph 的创建, 输入输出Node/Tensor, Layout的查询和设置
- Node 的接口: Node 的创建, 输入输出Tensor, 以及Node 属性的查询和设置
- Tensor的接口: Tensor的创建, Layout/Buffer/DataType 的查询和设置
- Graph 执行的接口: Prerun/Run/Postrun
- 其他: 设置log输出, 设置Graph/Node 执行设备等
- https://github.com/OAID/Tengine/blob/master/core/include/tengine_c_api.h

C API 典型推理代码

```
// load the model, create graph
graph_t graph = create_graph(nullptr, "engine", model_file);

// prerun the graph, binding device to run nodes and allocating necessary resource
int ret_prerun = prerun_graph(graph);

// get input tensor
tensor_t input_tensor = get_graph_input_tensor(graph, node_idx, tensor_idx);

// setup input buffer
if(set_tensor_buffer(input_tensor, input_data, 3 * img_h * img_w * 4) < 0)

// do inference
run_graph(graph, 1);

// get output tensor
tensor_t output_tensor = get_graph_output_tensor(graph, node_idx, tensor_idx);

// get output data pointer
float* data = (float*)(get_tensor_buffer(output_tensor));
```

C API 创建图和节点

下面以一个单Convolution 节点的图为例，来说明如何用代码创建一个图并执行

- 创建一个空的Graph

```
graph_t graph = create_graph(nullptr, nullptr, nullptr);
```

- 创建Input节点

```
node_t node = create_graph_node(graph, node_name, "InputOp");  
tensor_t tensor = create_graph_tensor(graph, node_name, TENGINE_DT_FP32);  
set_node_output_tensor(node, 0, tensor, TENSOR_TYPE_INPUT);
```

- 创建Conv节点

```
node_t conv_node = create_graph_node(graph, node_name, "Convolution");  
tensor_t output_tensor = create_graph_tensor(graph, node_name, TENGINE_DT_FP32);  
  
set_node_output_tensor(conv_node, 0, output_tensor, TENSOR_TYPE_VAR);  
set_node_input_tensor(conv_node, 0, input_tensor);  
set_node_input_tensor(conv_node, 1, w_tensor);  
set_node_input_tensor(conv_node, 2, b_tensor);
```

- 设置Conv节点参数

```
set_node_attr_int(conv_node, "kernel_h", &k_size);  
set_node_attr_int(conv_node, "kernel_w", &k_size);  
set_node_attr_int(conv_node, "stride_h", &stride);  
set_node_attr_int(conv_node, "stride_w", &stride);  
set_node_attr_int(conv_node, "pad_h0", &pad);  
set_node_attr_int(conv_node, "pad_w0", &pad);
```


Tengine 实践 1, 定制和扩展算子

定制算子实现
添加新算子

Tengine 算子kernel定制

如果要对Tengine已经实现的算子做定制或者扩展，Tengine提供了两种方式：

- 通过Tengine C API 的custom kernel 接口来做替换，替换时需要指定对哪个节点来做。
- 以 Tengine plugin的形式，来重新实现一遍算子的计算，并在注册算子实现时，把新算子的优先级提高。这样是把整个图的算子实现都做了替换
- 下面介绍 通过Tengine C API custom kernel 的方式来做替换。Plugin的方式会在添加新算子一节介绍

Tengine Custom Kernel 接口

- **struct custom_kernel_ops** 定义了 custom kernel 需要实现的内容

```
const char* op; /* name of the op to be implemented */
void* kernel_param; /* used for kernel impl functions */
int kernel_param_size;
int (*prerun)(struct custom_kernel_ops* ops, struct custom_kernel_tensor* inputs[], int input_num,
              struct custom_kernel_tensor* outputs[], int output_num, int dynamic_shape);
int (*run)(struct custom_kernel_ops* ops, struct custom_kernel_tensor* inputs[], int input_num,
           struct custom_kernel_tensor* outputs[], int output_num);
int (*postrun)(struct custom_kernel_ops* ops, struct custom_kernel_tensor* inputs[], int input_num,
               struct custom_kernel_tensor* outputs[], int output_num);
```

- 在初始化好custom_kernel_ops之后, 根据node 名字或者index, 获得对应的node 句柄
- 调用 **set_custom_kernel**(node_t node, const char* dev_name, struct custom_kernel_ops* kernel_ops) 来替换设备上指定node 的实现

Tengine 新算子添加

- Tengine 模块化的结构使得在Tengine code repo之外添加一个新算子很方便
- 当遇到不支持算子时，我们推荐先用plugin的模式在Tengine Repo外实现。待测试稳定以后，再提交PR合入Repo。
- 下面以添加一个Tensorflow 新算子Ceil为例，
 - 在Tengine NNIR中注册一个新的算子定义
 - 在Tengine Tensorflow Serializer 中注册算子的模型加载
 - 在Tengine Executor 中注册算子的具体实现
- 测试程序执行过程
 - 加载Tensorflow Serializer 模块
 - 加载operatorplugin,先后注册算子定义,算子实现和算子模型加载
 - 加载一个只包含Ceil的Tensorflow模型
 - 设置输入数据,并执行
 - 打印数据,检查结果

Tengine 实践2, 进行推理

CPU/GPU 推理

NPU/DLA 推理

Tengine 演示示例MobilenetSSD

EAIDK610 RK3399

- 单独用CPU跑
- CPU+GPU配合跑
- 多设备MSSD (2CPU+1GPU)

Tengine 演示示例Inception v3

Nvidia: AGX Xavier

TensorRT

Amlogic: A311D

OpenVX



THANK YOU

OPEN AI LAB



赋能万物 万物智能

✉ market@openailab.com

🌐 www.openailab.com



智东西 公开课

专注讲解新兴技术创新与应用