

ESPRESSIF

ESP-DL 用户指南

发行版本 *latest*

乐鑫信息科技

2026 年 11 月 06 日

Contents

1 Introduction **3**
- 1.1 ESP-DL 简介 . . . 3
- 1.2 ESP-DL 项目组织 . . . 4

2 入门指南 **7**
- 2.1 硬件要求 . . . 7
- 2.2 软件要求 . . . 8
- 2.3 快速开始 . . . 9
- 2.4 模型量化 . . . 9
- 2.5 模型部署 . . . 10

3 Tutorials **11**
- 3.1 如何量化模型 . . . 11
- 3.2 自动量化 . . . 15
- 3.3 如何加载、测试和性能分析模型 . . . 26
- 3.4 如何进行模型推理 . . . 31
- 3.5 如何创建新模块（算子） . . . 34
- 3.6 使用 AI Agent 自动实现算子 . . . 36
- 3.7 如何部署 MobileNetV2 . . . 44
- 3.8 如何部署 YOLO11n . . . 52
- 3.9 如何部署 YOLO11n-pose . . . 61
- 3.10 如何部署流式模型 . . . 71
- 3.11 使用 TQT 量化模型 . . . 75

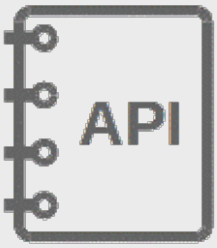
4 API Reference **89**
- 4.1 Tensor API Reference . . . 89
- 4.2 Module API Reference . . . 100
- 4.3 Model API Reference . . . 104
- 4.4 Fbs API Reference . . . 123



入门指南



使用教程



API Reference

CHAPTER 1

Introduction

1.1 ESP-DL 简介

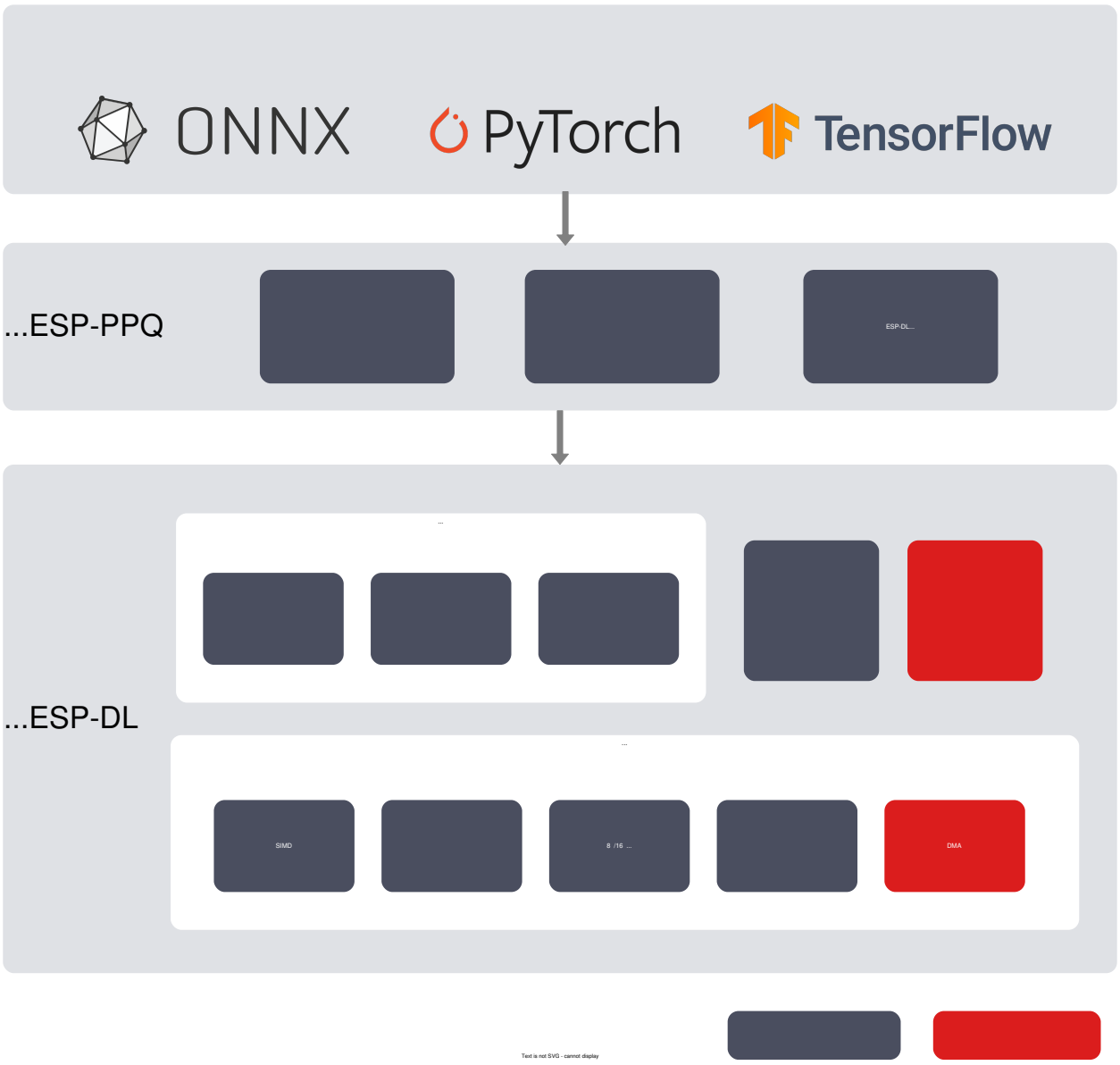
ESP-DL 是一个专为 ESP 系列芯片设计的轻量级且高效的神经网络推理框架。通过 ESP-DL，您可以轻松快速地使用乐鑫的系统级芯片 (SoC) 开发 AI 应用。

1.1.1 概述

ESP-DL 提供了加载、调试和运行 AI 模型的 API。该框架易于使用，并且可以与其他乐鑫 SDK 无缝集成。ESP-PPQ 作为 ESP-DL 的量化工具，能够量化来自 ONNX、Pytorch 和 TensorFlow 的模型，并将其导出为 ESP-DL 标准模型格式。

- **ESP-DL 标准模型格式**：该格式类似于 ONNX，但使用 FlatBuffers 而不是 Protobuf，使其更轻量级并支持零拷贝反序列化，文件后缀为 '.espdl'。
- **高效算子实现：** ESP-DL 高效地实现了常见的 AI 算子，如 Conv、Pool、Gemm、Add 和 Mul 等。目前支持的算子 operator_support_state.md
- **静态内存规划器：**内存规划器根据用户指定的内部 RAM 大小，自动将不同层分配到最佳内存位置，确保高效的整体运行速度同时最小化内存使用。
- **双核调度：**自动双核调度允许计算密集型算子充分利用双核计算能力。目前，Conv2D 和 DepthwiseConv2D 支持双核调度。
- **8bit LUT Activation：**除了 Relu, PRelu(n>1) 之外的所有激活函数，ESP-DL 默认使用 8bit LUT(Look Up Table) 方式实现, 以加速推理。

ESP-DL 系统框架图如下所示:



1.2 ESP-DL 项目组织

ESP-DL 的模块化设计使其开发、维护和扩展变得高效。项目的组织结构如下：

1.2.1 dl（深度学习）

核心深度学习模块和工具，分为子模块：

- **model** 加载、管理和分配深度学习模型的内存。包含 `dl_model_base` 和 `dl_memory_manager`。
- **module** 60+ 个神经网络算子接口（卷积、池化、激活等）。文件：`dl_module_base.hpp`，`dl_module_conv.hpp`，`dl_module_pool.hpp`，`dl_module_relu.hpp` 等。
- **base** 具体的算子实现，包括对芯片（esp32, esp32s3, esp32p4）的 ISA 特定汇编加速。包含算子实现文件如 `dl_base_conv2d.cpp/hpp`，`dl_base_avg_pool2d.cpp/hpp` 等，以及 `isa/` 子目录中的 ISA 特定代码。
- **math** 数学操作（矩阵函数）。文件：`dl_math.hpp` 和 `dl_math_matrix.hpp`。
- **tool** 辅助功能（实用工具）。文件：`dl_tool.hpp` 和 `dl_tool.cpp`。包含 `isa/` 子目录中的 ISA 特定工具。
- **tensor** 张量类和操作。文件：`dl_tensor_base.hpp`。

1.2.2 vision（计算机视觉）

计算机视觉模块，分为子模块：

- **classification** 图像分类（模型推理）。推理：`dl_cls_base`。后处理器：`imagenet_cls_postprocessor`，`hand_gesture_cls_postprocessor`，`dl_cls_postprocessor`。
- **recognition** 特征提取（模型推理）。特征数据库管理（注册、删除、查询）。预处理器：`dl_feat_image_preprocessor`。推理：`dl_feat_base`。后处理器：`dl_feat_postprocessor`。数据库：`dl_recognition_database`
- **image** 图像处理（调整大小、裁剪、仿射变换）。颜色转换（像素、图像）。图像预处理器（调整大小、裁剪、颜色转换、规范化、量化的管道）。图像解码/编码（JPEG/BMP）。绘制工具（点、空心矩形）。

  图像处理：`dl_image_process`。颜色转换：`dl_image_color`。图像预处理器：`dl_image_preprocessor`。图像解码/编码：`dl_image_jpeg`、`dl_image_bmp`。绘制工具：`dl_image_draw`。
- **detect** 目标检测（模型推理）。推理：`dl_detect_base`。后处理器：`dl_detect_yolo11_postprocessor`、`dl_detect_espdet_postprocessor`、`dl_detect_msr_postprocessor`、`dl_detect_mnp_postprocessor`、`dl_detect_pico_postprocessor`。姿态估计：`dl_pose_yolo11_postprocessor`。

1.2.3 audio（音频处理）

音频处理模块，分为子模块：

- **common** 通用音频工具。文件：`dl_audio_common.cpp/hpp`，`dl_audio_wav.cpp/hpp`。
- **speech_features** 语音特征提取。文件：`dl_speech_features.cpp/hpp` (base class), `dl_fbank.cpp/hpp` (Filter Bank), `dl_mfcc.cpp/hpp` (MFCC), `dl_spectrogram.cpp/hpp` (Spectrogram)。

1.2.4 fbs_loader（FlatBuffers 加载器）

处理 FlatBuffers 模型：

- **include** 头文件：`fbs_loader.hpp`，`fbs_model.hpp`。
- **src** 实现：`fbs_loader.cpp`。
- **lib/** 针对不同目标的预编译库：`esp32/`，`esp32s3/`，`esp32p4/`。
- **espidl.fbs** FlatBuffers 模式文件。
- **pack_espdl_models.py** 模型打包脚本。

1.2.5 其他文件

- **CMakeLists.txt** 项目构建配置。
- **idf_component.yml** 组件元数据（名称、版本、依赖项）。
- **README.md** 项目文档和使用说明。
- **LICENSE** 许可条款。

CHAPTER 2

入门指南

2.1 硬件要求

- 一块 ESP32-S3 或 ESP32-P4 开发板。推荐使用：ESP32-S3-EYE 或 ESP32-P4-Function-EV-Board
- 一台 PC（Linux 系统）

备注:

- 部分开发板目前采用 Type C 接口。请确保使用正确的线缆连接开发板!
- ESP-DL 也支持 ESP32，但其算子实现采用 C 编写，因此 ESP32 运行速度会远慢于 ESP32-S3 或 ESP32-P4。如有需要，可在项目中自行添加编译配置文件，ESP-DL 的函数接口调用方式完全一致。需要注意的是:
  - 使用 **ESP-PPQ** 量化 **ESP32** 平台模型时，需将 target 设置为 `c`。
  - 使用 **ESP-DL** 部署 **ESP32** 平台模型时，项目编译 target 则设置为 `esp32`。

2.2 软件要求

2.2.1 ESP-IDF

ESP-DL 基于 ESP-IDF 运行。有关如何获取 ESP-IDF 的详细说明，请参阅 ESP-IDF 编程指南。

备注： 请使用 ESP-IDF 的 `release/v5.3` 或更高版本。

2.2.2 ESP-PPQ

ESP-PPQ 是基于 ppq 的量化工具，其 代码 已全部开源。ESP-PPQ 在 PPQ 的基础上添加了乐鑫定制的 quantizer 和 exporter，方便用户根据不同的芯片选择和 ESP-DL 匹配的量化规则，并导出为 ESP-DL 可以直接加载的标准模型文件。ESP-PPQ 兼容 PPQ 所有的 API 和量化脚本。更多细节请参考 PPQ 文档和视频。如果您想量化自己的模型，可以使用如下方式安装 esp-ppq：

方式一: 使用 pip 安装包

```
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/
↪cpu
pip install esp-ppq
```

方式二: 使用 pip 安装源码，以便保持与 master 分支同步

```
git clone https://github.com/espressif/esp-ppq.git
cd esp-ppq
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/
↪cpu
pip install -e .
```

方式三: 使用 uv 安装包

```
uv pip install "esp-ppq[cpu]" --torch-backend=cpu
# GPU
# uv pip install "esp-ppq[cpu]" --torch-backend=cu124
# AMD GPU
# uv pip install "esp-ppq[cpu]" --torch-backend=rocm6.2
# Intel XPU
# uv pip install "esp-ppq[cpu]" --torch-backend=xpu
```

方式四: 使用 uv 安装源码，以便保持与 master 分支同步

```
git clone https://github.com/espressif/esp-ppq.git
cd esp-ppq
uv pip install torch torchvision torchaudio --index-url https://download.pytorch.org/
↪whl/cpu
uv pip install -e .
```

方式五: 在 docker 中使用 esp-ppq

```
docker build -t esp-ppq:your_tag https://github.com/espressif/esp-ppq.git
```

备注:

- 示例代码中安装的是 linux pytorch cpu 版本，请根据实际情况安装对应的 pytorch。
- 如果使用 uv 安装包，仅需要更改 `--torch-backend` 参数即可，其会忽略项目中配置的 pytorch URLs 索引。

2.3 快速开始

ESP-DL 提供了一些开箱即用的 示例

2.3.1 示例编译 & 烧录

```
idf.py set-target [Soc]
idf.py flash monitor
```

使用具体的芯片替换 `[Soc]`，目前支持 `esp32s3` 和 `esp32p4`。示例暂未添加 `esp32` 的模型和编译配置文件。

2.3.2 示例配置

```
idf.py menuconfig
```

一些示例包含可配置的选项，可以在使用 `idf.py set-target` 指定芯片之后使用 `idf.py menuconfig` 进行配置。

2.3.3 故障排除

查看 ESP-IDF 文档

请参阅 ESP-IDF DOC

擦除 FLASH 和清除示例

```
idf.py eras-flash -p [PORT]
```

删除 `build/`、`sdkconfig`、`dependencies.lock`、`managed_components/` 并重试。

2.4 模型量化

首先，请参考 ESP-DL 算子支持状态 operator_support_state.md，确保您的模型中的算子已经得到支持。

ESP-DL 必须使用专有格式 `.espdl` 进行模型部署，深度学习模型需要进行量化和格式转换之后才能使用。ESP-PPQ 提供了 `espdl_quantize_onnx` 和 `espdl_quantize_torch` 两种接口以支持 ONNX 模型和 PyTorch 模型导出为 `.espdl` 模型。其他深度学习框架，如 TensorflFlow, PaddlePaddle 等都需要先将模型转换为 ONNX 。因此请确保您的模型可以转换为 ONNX 模型。更多详细信息，请参阅：

- 如何量化模型

- 如何量化 *MobileNetV2*
- 如何量化 *YOLO11n*
- 如何量化 *YOLO11n-pose*
- 如何量化流式模型

2.5 模型部署

ESP-DL 提供了一系列 API 来快速加载和运行模型。更多详细信息，请参阅：

- 如何加载和测试模型
- 如何进行模型推理
- 如何部署流式模型

CHAPTER 3

Tutorials

3.1 如何量化模型

ESP-DL 必须使用专有格式 `.espdl` 进行模型部署。这是一种量化模型格式，支持 8bit (w8a8)、16bit (w16a16) 以及混合精度 (w8a16) 量化。这三种量化方式可在模型中任意组合使用，在保持模型精度的同时最大化推理效率。在本教程中，我们将以 quantize_sin_model 为例，介绍如何使用 ESP-PPQ 量化并导出 `.espdl` 模型，量化方法为 Post Training Quantization (PTQ)。

- 准备工作
- 预训练模型
- 量化并导出 *.espdl*
  - 量化方案 *(quant_type)*
  - 混合量化
  - 添加测试输入/输出
  - 量化模型推理 & 精度评估
- 高级量化方法
  - 训练后量化 *(PTQ)*
  - 量化感知训练 *(QAT)*

3.1.1 准备工作

安装 ESP_PPQ

3.1.2 预训练模型

```
python sin_model.py
```

执行 sin_model.py 。该脚本会训练一个简单的 Pytorch 模型用于拟合 [0, 2pi] 范围内的 sin 函数。训练结束会保存相应的.pth 权重，并导出 ONNX 模型。

备注： ESP-PPQ 提供了 `espdl_quantize_onnx` 和 `espdl_quantize_torch` 两种接口以支持 ONNX 模型和 PyTorch 模型。其他深度学习框架，如 TensorfFlow, PaddlePaddle 等都需要先将模型转换为 ONNX 。

- TensorFlow 转 ONNX tf2onnx
- TFLite 转 ONNX tflite2onnx
- TFLite 转 TensorFlow tflite2tensorflow
- PaddlePaddle 转 ONNX paddle2onnx

3.1.3 量化并导出 `.espdl`

参考 quantize_torch_model.py 和 quantize_onnx_model.py ，了解如何使用 `espdl_quantize_onnx` 和 `espdl_quantize_torch` 接口量化并导出 `.espdl` 模型。

执行脚本后会导出三个文件，分别是：

- `**.espdl`: ESPDL 模型二进制文件，可直接部署于芯片端执行推理，并支持通过 Netron 可视化查看模型结构。
- `**.info`: ESPDL 模型文本文件，用于调试和确定 `.espdl` 模型是否被正确导出。包含了模型结构，量化完的模型权重，测试输入/输出等信息。
- `**.json`: 量化信息文件，用于量化信息的保存和加载。

量化方案 (`quant_type`)

通过 `espdl_quantize_onnx` / `espdl_quantize_torch` 的 `quant_type` 参数选择量化方案：

quant_type	权重	激活	典型用途
`w8a8` (默认)	INT8	INT8	推理速度最快，大多数模型的默认选择
`w16a16`	INT16	INT16	精度最高，计算和内存开销更大
`w8a16`	INT8	INT16	混合精度：权重 8bit、激活 16bit。适用于带静态权重的 Conv / Gemm / MatMul（ESP-DL ≥ v3.3.11）
`none`	FP32	FP32	跳过量化（仅用于调试 / 导出）

```
quant_ppq_graph = espdl_quantize_torch(
    model=model,
    espdl_export_file="sin_model.espdl",
    calib_dataloader=dataloader,
    calib_steps=32,
    input_shape=[1, 1],
    target="esp32s3",           # "esp32p4", "esp32s31", "esp32s3" 或 "c"
    quant_type="w8a8",          # "w8a8", "w8a16", "w16a16" 或 "none"
    collate_fn=collate_fn,
    device="cpu",
    error_report=True,
    export_test_values=True,
    verbose=1,
)
```

备注:

- 请优先使用 `quant_type`。当同时传入时，它会覆盖旧的 `num_of_bits` 参数。`num_of_bits=8` 等价于 `quant_type="w8a8"`；`num_of_bits=16` 等价于 `quant_type="w16a16"`。
- `w8a16` 从 ESP-DL v3.3.11 开始全面支持。并非所有算子都有独立的 w8a16 kernel：带静态权重的 Conv / Gemm / MatMul 使用 INT8 权重和 INT16 激活；depthwise Conv (`group != 1`) 及其它算子会回退到 INT16。详见 operator_support_state.md。
- `debug=True` 会在 PPQ 中启用高精度 INT16 仿真，仅对 `w16a16` 生效 (旧参数别名: `hi_precision=True`)。

混合量化

可以在同一模型中混合使用 `w8a8`、`w8a16` 和 `w16a16`。先用 `quant_type` 设置默认方案，再通过 `QuantizationSetting.dispatching_table` 将指定算子调度到其它方案：

```
from esp_ppq import QuantizationSettingFactory
from esp_ppq.api import espdl_quantize_onnx, get_target_platform

TARGET = "esp32p4"
quant_setting = QuantizationSettingFactory.espdl_setting()
# 将精度敏感层提升为混合精度或全 INT16
quant_setting.dispatching_table.append(
    "/features/features.1/conv/conv.0/conv.0.0/Conv",
    get_target_platform(TARGET, quant_type="w8a16"),
)
quant_setting.dispatching_table.append(
    "/features/features.1/conv/conv.0/conv.0.2/Clip",
    get_target_platform(TARGET, quant_type="w16a16"),
)

quant_ppq_graph = espdl_quantize_onnx(
    onnx_import_file="model.onnx",
    espdl_export_file="model.espdl",
    calib_dataloader=dataloader,
    calib_steps=32,
    input_shape=[1, 3, 224, 224],
    target=TARGET,
    quant_type="w8a8",
```

(续下页)

(接上页)

```
    setting=quant_setting,
    collate_fn=collate_fn,
    device="cpu",
)
```

推荐使用 `get_target_platform(target, quant_type="w8a8")`。旧写法 `get_target_platform(target, 16)` (或 `num_of_bits=16`) 仍然可用，并映射为 `w16a16`。

请通过 `setting.dispatching_table` 配置混合精度。`espdl_quantize_*` 的 `dispatching_override` 和 `dispatching_method` 参数已废弃，将在后续版本中移除。

备注:

1. 不同平台的 `.espdl` 模型不能混用，推理结果会有误差。
    - `ESP32` 采用 `Per-Tensor` 量化策略, ROUND 策略为 `ROUND_HALF_UP`。
        - 使用 **ESP-PPQ** 量化 **ESP32** 平台模型时，需将 target 设置为 `c`，因为在 ESP-DL 中，其算子实现采用 C 语言编写。
        - 使用 **ESP-DL** 部署 **ESP32** 平台模型时，项目编译 target 则设置为 `esp32`。
    - `ESP32S3` 采用 `Per-Tensor` 量化策略, ROUND 策略为 `ROUND_HALF_UP`。
    - `ESP32P4` 和 `ESP32S31` 的 `Conv`, `GEMM` 算子采用 `Per-Channel` 量化策略，其它算子采用 `Per-Tensor` 量化策略, ROUND 策略为 `ROUND_HALF_EVEN`。
2. 目前 ESP-DL 使用的量化策略是对称量化 + POWER OF TWO。

添加测试输入/输出

验证模型在板端的推理结果是否正确，首先需要记录 PC 端的一组测试输入/输出。开启 api 中的 `export_test_values` 选项，就能将一组测试输入/输出固化在 `.espdl` 模型中。`input_shape` 参数和 `inputs` 参数必须指定其中的一个，`input_shape` 参数使用随机的测试输入，`inputs` 则可以指定一个特定的测试输入。`.info` 文件中可以查看测试输入/输出的值。搜索 `test inputs value` 和 `test outputs value` 查看它们。

量化模型推理 & 精度评估

`espdl_quantize_onnx` 和 `espdl_quantize_torch` API 会返回 `BaseGraph`。使用 `BaseGraph` 构建相应的 `TorchExecutor` 就可以在 PC 端使用量化模型进行推理了。

```
executor = TorchExecutor(graph=quanted_graph, device=device)
output = executor(input)
```

量化模型推理得到的输出可以用来计算各种精度指标。由于 `esp-dl` 板端推理的结果是能和 `esp-ppq` 对齐的，可以直接用该指标评估量化完模型的性能。

备注:

1. 当前 esp-dl 仅支持 batch_size 为 1，不支持多 batch 或者动态 batch。
2. `.info` 文件中的测试输入/输出，以及量化完的模型权重都是 16 字节对齐的，也就是说如果不满 16 字节，会在后面填充 0。

3.1.4 高级量化方法

如果你的模型使用默认的 w8a8 量化无法达到满意的结果，可以将整个模型切换为 `w8a16` / `w16a16`，或按混合量化 混合使用多种方案，也可以使用如下量化方法进一步减少量化模型的性能损失：

训练后量化 (PTQ)

- *混合量化 (w8a8 / w8a16 / w16a16)*
- *8bit 后量化*

量化感知训练 (QAT)

- *YOLO11n-pose 量化感知训练*

3.2 自动量化

3.2.1 使用 AutoQuant 自动量化模型

- 准备工作
- 怎么使用 *AutoQuant*
- 结果获取
- 示例

在面向 ESP32 芯片部署模型时，我们通常会使用 `espdl_quantize_onnx` 这类接口完成模型量化。只要提供 ONNX 模型、校准数据以及一组量化设置，就可以导出用于部署的 `.espdl` 模型。

但对于自定义模型而言，真正困难的部分往往并不在于量化流程本身，而在于如何选择合适的量化配置。例如，量化位宽应如何设置，应选择哪种校准算法，是否需要启用额外的优化策略，以及这些优化策略的参数应该如何调整等。这些因素都会影响量化误差，并进一步影响模型量化后的精度表现以及在 ESP32 芯片上的推理延迟。

因此，在实际开发过程中，寻找一组合适的量化配置通常需要一定经验积累和反复实验。开发者往往需要不断调整量化参数，重新执行量化与测试，再根据结果决定下一步的尝试方向。AutoQuant 想解决的，正是这类重复、繁琐且依赖经验的调参过程，将其自动化完成。

下面我们来看如何使用 AutoQuant 来自动找到合适的量化配置，并获取量化后的模型。

准备工作

1. *安装 ESP_PPQ*

怎么使用 AutoQuant

AutoQuant 的使用方式和 `espdl_quantize_onnx` 类似，同样通过接口调用完成模型量化，只不过这里使用的是 `espdl_auto_quantize_onnx` 来启动自动搜索流程。使用时依然需要准备 ONNX 模型、校准数据以及输入 shape 等。

两者的主要区别在于：`espdl_quantize_onnx` 用于执行一次固定配置的量化，因此需要用户明确给出量化设置；而 `espdl_auto_quantize_onnx` 用于在多组量化设置中自动进行搜索、评测与筛选，用户只需提供搜索相关设置即可。

下面以这段代码为例进行说明：

```
from esp_ppq.api import AutoQuantSearchSetting, espdl_auto_quantize_onnx

def evaluate_fn(graph):
    # Replace this with your validation logic.
    top1, top5 = ...
    return top1, {"top1": top1, "top5": top5}

setting = AutoQuantSearchSetting(
    search_mode="fast",
    num_of_candidates=5,
    score_direction="maximize",
    run_dir="outputs/auto_quant_mobilenetv2",
)

espdl_auto_quantize_onnx(
    onnx_import_file="model.onnx",
    espdl_export_file="outputs/model.espdl",
    calib_dataloader=calib_loader,
    calib_steps=32,
    input_shape=[3, 224, 224],
    evaluate_fn=evaluate_fn,
    target="esp32p4",
    setting=setting,
    device="cuda",
)
```

在该示例中，我们给 `espdl_auto_quantize_onnx` 传入了 9 个参数：

参数	说明
`onnx_import_file`	ONNX 模型路径
`espdl_export_file`	`.espdl` 模型导出路径
`calib_dataloader`	校准数据
`calib_steps`	校准时使用的 batch 数量
`input_shape`	模型输入 shape，不包含 batch 维度。例如 `[3, 224, 224]`
`evaluate_fn`	可选的评测函数
`target`	部署平台
`setting`	AutoQuant 的搜索设置
`device`	运行设备，例如 `"cpu"` 或 `"cuda"`

其中，我们重点关注 evaluate_fn 和 setting，其他参数的使用方法和 espdl_quantize_onnx 基本一致。

evaluate_fn

evaluate_fn 是 AutoQuant 用来比较不同量化配置结果的评测函数，是一个可选参数。

- **提供该函数时**：AutoQuant 会按照 evaluate_fn 返回的指标排序，例如分类模型的 top1 / top5、或检测模型的 mAP。
- **未提供该函数时**：AutoQuant 会使用下方的 **默认评测方式**进行排序。

默认评测方式：首先调用 graphwise_error_analyse，在校准数据上计算模型量化的 graphwise SNR 误差，然后取误差最大的 3 层求平均，将该平均值作为评测结果。

如果需要自定义 evaluate_fn，则必须满足以下要求：

- 返回值必须是 (score, extras) 二元组。
- score 必须是有限数字，不能是 nan、inf 或 bool。
- extras 必须是 dict。
- extras 中不能使用 AutoQuant 的保留字段：score、hash、index、folder、files、strategy、params

setting

setting 用来控制 AutoQuant 的搜索行为。例如搜索模式、保留多少个候选结果、实验结果保存路径，以及是否从已有结果继续搜索等，都由 AutoQuantSearchSetting 配置。

AutoQuantSearchSetting 的常用构造参数如下：

参数	默认值	说明
search_mode	"exhaustive"	搜索模式。"exhaustive" 会枚举所有候选配置；"fast" 会先快速筛选一轮，再继续搜索更有希望的配置。
num_of_candidat	5	最终保留的候选模型数量，也就是 Top-K 的 K。
score_directior	"maximize"	score 的排序方向。准确率、mAP 等指标越大越好，使用 "maximize"；误差、loss 等指标越小越好，使用 "minimize"。例如未传入 evaluate_fn 时，应使用 "minimize"
candidate_filte	low_latency_candida	Top-K 候选过滤器。默认更偏向保留部署成本较低的结果；如果只想严格按 score 排序，可以设置为 None
run_dir	"outputs/ auto_quant"	实验结果保存目录
resume	False	是否从已有 run_dir 继续搜索

如果使用 "fast" 搜索模式，还可以进一步设置：

```
setting.top_strategy = 10
setting.early_stop_patience = 3
```

对应属性如下：

属性	默认值	说明
top_strategy	10	第一轮筛选后保留多少组策略继续搜索
early_stop_patience	3	当前策略连续多少次没有提升后停止搜索

真正决定“要尝试哪些量化策略和参数”的，是 `setting.strategy_space` 和 `setting.param_space`：

字段	说明
`strategy_space`	控制每个量化策略是否参与搜索，例如是否尝试 mixed_precision
`param_space`	控制每个策略对应的参数候选，例如校准算法候选、优化步数、学习率等

通常不需要调整这两个字段，如果你有需要，可以修改需要调整的部分：

```
setting.strategy_space["mixed_precision"] = [True, False]
setting.param_space["calib_algorithm"]["method"] = ["kl", "mse"]
```

完整默认值可以查看 `esp_ppq/autoquant/setting.py`。

结果获取

AutoQuant 每完成一个候选配置的搜索，都会将结果写入 `run_dir`。目录结构如下：

```
<run_dir>/
    summary.json
    candidates.json
    0000/
        config.json
        model.espdl
        model.info
        model.json
        model.native
    0001/
        ...
```

其中，`summary.json` 记录所有已经完成的实验。`candidates.json` 记录当前 Top-K 候选。每个编号目录保存一次实验的配置和导出的模型文件。

开启 `resume=True` 后，AutoQuant 会读取 `summary.json`，跳过已经完成的 `(strategy, params)` 组合。这个功能主要用于搜索中断后的继续执行，以及细化搜索。

示例

ESP-PPQ 提供了两个可以直接运行的 AutoQuant 示例：

```
python -m esp_ppq.samples.AutoQuant.mobilenetv2
python -m esp_ppq.samples.AutoQuant.yolo11n
```

3.2.2 使用 espdl-quantize Skill 自动调优量化精度

- 什么是 *espdl-quantize skill*
- 依赖要求
- *skill* 仓库路径

- 安装 *espdl-quantize skill*
  - 方法一：使用 *npx*（推荐 - 最简单）
  - 方法二：手动安装
- 如何使用 *espdl-quantize skill*
  - 快速开始示例
- 完整示例
- 查看最终产出结果
  - *final_report.md*
  - *best* 目录
- 故障排查

本文档介绍如何使用 **espdl-quantize skill** 对深度学习模型进行自动量化调优。espdl-quantize 是一个 ** 在线迭代、分布感知 ** 的 Agent skill：它会先跑 baseline，再逐轮分析 layerwise 误差分布，并据此自动选择最适合的 esp-ppq 量化方法（如 TQT、混合精度、权重均衡、偏置校正等），反复迭代直至精度收敛或达到上限。

你只需提供校准数据加载和模型评估逻辑；skill 负责驱动 `QuantizationSettingFactory.espdl_setting()` 的迭代搜索。

什么是 espdl-quantize skill

esp-ppq 暴露了十余种可调量化 passes（校准算法、层间权重均衡、偏置校正、水平权重拆分、混合精度、TQT、LSQ、块重建等），每种又有 2-6 个参数。手动尝试不仅耗时，还容易因参数冲突导致 silent degradation。

espdl-quantize skill 将”看误差报告 → 猜测参数 → 重跑”的人工循环改造为 ** 结构化、分布感知的自动搜索 **，核心能力包括：

1. **知识库**——`references/ppq_methods.md` 中编码了每种 esp-ppq 方法的原理、参数、适用场景与反模式。
2. **决策手册**——`references/decision_playbook.md` 根据 Top-K 最差层的输入/权重/输出分布，将观察到的模式映射到候选方法。
3. **固定 harness** ——`scripts/run_iteration.py` 接收你的契约模块（`user_quant.py`）和一个 JSON setting，输出结构化产物（metrics、layerwise error、per-layer stats），Agent 只需读取 JSON 即可做出下一轮决策。
4. **搜索状态机**——`scripts/compare_iterations.py` 检查已尝试过的配置，告诉 Agent 下一轮该跑什么。
5. **目标感知安全网**——自动检测与目标芯片量化策略冲突的 passes：
   - 针对 **POWER_OF_2 目标**（`esp32p4 / esp32s3 / c`），skill 会自动跳过 LSQ：LSQ 学习任意 scale，与 POWER_OF_2 的 2 的幂次约束冲突，因此改用 TQT（它直接训练 `log2_scale`，天然适配 POWER_OF_2）。
   - **esp32p4** 上启用逐层均衡（Layerwise Equalization）时，skill 会发出警告：该机制与 Conv/Gemm 的 per-channel 量化策略在功能上重叠，esp-ppq 官方标记为 “Not recommend”，但实验表明部分 MobileNet 族/深度可分离网络仍可从中受益，因此 skill 仍会选择性尝试。

依赖要求

1. *安装 ESP_PPQ*
2. 安装 skill 额外依赖：

   在 skill 目录（包含 `SKILL.md` 的目录）下执行：

   ```
   pip install -r assets/extra_requirements.txt
   ```

   额外依赖包括：`pandas`、`scipy`、`tqdm`。

   备注： skill 直接运行在当前的 Python 解释器中，**无需 Docker**。

skill 仓库路径

espdl-quantize skill 在 esp-dl 仓库中的路径为：

```
esp-dl/tools/agents/skills/espdl-quantize/
```

目录结构：

```
esp-dl/tools/agents/skills/espdl-quantize/
├── SKILL.md                          # skill 入口与使用说明
├── assets/
│   ├── extra_requirements.txt        # 额外 Python 依赖
│   ├── user_quant_torch_example.py   # Torch 契约模板
│   ├── user_quant_onnx_example.py    # ONNX 契约模板
│   ├── example_quantize_mobilenetv2/ # MobileNet-V2 完整示例
│   └── example_quantize_yolo11n/     # YOLO11n 完整示例
├── references/
│   ├── contract.md                   # 用户契约完整规范
│   ├── decision_playbook.md          # 分布→方法决策手册
│   └── ppq_methods.md                # esp-ppq 各方法详解
└── scripts/
    ├── run_iteration.py              # 单轮迭代 harness
    └── compare_iterations.py         # 迭代比较与状态机
```

安装 espdl-quantize skill

espdl-quantize skill 是 Agent 目录无关的，可以安装在任何 Agent 的 skills 文件夹下。常见安装路径如下。

方法一：使用 npx（推荐 - 最简单）

为 OpenCode、Cursor、Claude Code 及其他兼容工具安装 skill 最简单的方法是使用 npx：

```
npx skills add espressif/esp-dl --skill espdl-quantize
```

备注： npx 是 Node.js（通过 npm）自带的包执行器。如果你尚未安装 Node.js，请参考 Node.js 安装指南 进行安装。

执行命令后，skill 将被自动安装并在你的 Coding Agent 工具中可用。

方法二：手动安装

如果你偏好手动安装，或者电脑未安装 Node.js，请根据你使用的具体工具按照以下说明操作：

Cursor 用户

将 skill 目录复制到 Cursor 的 skills 文件夹 (以用户级 skills 目录为例)：

```
# Linux / macOS
cp -r esp-dl/tools/agents/skills/espdl-quantize \
      ~/.cursor/skills/espdl-quantize

# Windows (PowerShell)
Copy-Item -Recurse esp-dl\tools\agents\skills\espdl-quantize \
          $env:USERPROFILE\.cursor\skills\espdl-quantize
```

OpenCode 用户

将 skill 目录复制到 OpenCode 的 skills 文件夹：

```
# 复制到项目的 skills 目录
cp -r esp-dl/tools/agents/skills/espdl-quantize \
      .opencode/skills/espdl-quantize

# 或复制到用户级 skills 目录
cp -r esp-dl/tools/agents/skills/espdl-quantize \
      ~/.agents/skills/espdl-quantize
```

如何使用 espdl-quantize skill

使用本 skill 前，你需要准备一份 **用户契约文件**（通常命名为 `user_quant.py`），其中包含：

- `QUANT_CONFIG`：量化配置字典（模型类型、输入形状、目标芯片、primary_metric 等）。
- `create_calib_dataloader()`：返回 PyTorch `DataLoader`，用于校准。
- `get_torch_model()` 或提供 `onnx_path`：返回 eval 模式的 `nn.Module`（torch 模型）或 ONNX 路径。
- `evaluate(quant_graph) -> dict`：接收 PPQ 量化后的图，返回指标字典（必须包含 `QUANT_CONFIG["primary_metric"]` 对应的 key）。
- （可选）“evaluate_fast(quant_graph) -> dict“：用于快速中间迭代评估。

完整契约规范请参考 skill 目录下的 `references/contract.md`。

快速开始示例

以下命令展示了典型的使用方式。你可以直接复制到 Agent 对话中使用。

示例 1：基本的量化调优请求

```
// 量化 mobilenet_v2 模型
我想用 espdl-quantize skill 量化 example_quantize_mobilenetv2/user_quant.py 这个␣
↪mobilenet_v2 模型。先跑 baseline，然后根据 layerwise 误差迭代 13 轮，目标 top1␣
↪越接近 71.878 越好，能超越，那就更好了。

// 量化 yolo11n 模型
我想用 espdl-quantize skill 量化 example_quantize_yolo11n/user_quant.py 这个 yolo11n␣
↪模型。先跑 baseline，然后根据 layerwise 误差迭代 13 轮，目标 map5095 越接近 39.0␣
↪越好，能超越，那就更好了。
```

示例 2：追加迭代

```
// 量化 mobilenet_v2 模型
我想用 espdl-quantize skill 量化 example_quantize_mobilenetv2/user_quant.py 这个␣
↪mobilenet_v2 模型。之前已经迭代了13轮，我还想再迭代10轮，目标 top1 越接近 71.878␣
↪越好，能超越，那就更好了。

// 量化 yolo11n 模型
我想用 espdl-quantize skill 量化 example_quantize_yolo11n/user_quant.py 这个 yolo11n␣
↪模型。之前已经迭代了13轮，我还想再迭代10轮，目标 map5095 越接近 39.0␣
↪越好，能超越，那就更好了。
```

skill 接收到指令后会自动执行以下流程：

1. **Phase 0 —基线评估**：使用默认 `QuantizationSettingFactory.espdl_setting()` 跑一次量化，得到 baseline 指标。
2. **Phase 1 — 误差分析**：读取 `layerwise_error.json`、`layer_stats.json`、`non_computing_hot_ops.json`、`graphwise_jumps.json`，识别 Top-K 最差层。
3. **Phase 2 —决策与配置生成**：根据 decision playbook 选择候选方法，生成下一轮 `setting.json`。
4. **Phase 3 —迭代执行**：运行 `scripts/run_iteration.py` 进行量化，评估指标。
5. **Phase 4 —比较与收敛判断**：使用 `scripts/compare_iterations.py` 比较历史迭代，若达到 `target_metric` 或迭代上限则停止，否则回到 Phase 1。

每轮迭代的结果保存在 `outputs/iter_<N>/` 目录下，包含：

- `metrics.json` ——当前迭代的评估指标。
- `layerwise_error.json` ——各层的孤立误差。
- `layer_stats.json` / `layer_stats_full.json` ——各层张量的分布统计。
- `setting.json` ——本轮使用的完整量化配置。
- `model.espdl`、`model.info`、`model.json`、`model.native` ——量化后的模型文件。

备注:

- Agent 指令中的目标指标（如 top1、map5095）必须同时满足两个一致性：与 QUANT_CONFIG[“primary_metric”] 一致，且与 user_quant.py 中 evaluate() 函数返回的指标 key 一致。
- Agent 需支持上下文压缩或自动摘要功能；上下文窗口更大的模型稳定性更佳，例如 deepseek_v4_pro。

- 在自动搜索过程中，skill 会根据收敛规则（连续 3 轮结果相对 best 变化均小于 0.1%）自动判断迭代是否结束。但 AI Agent 可能因上下文丢失或误判而在到达用户指定迭代轮次前提前终止。此时你只需追加迭代指令（如”再迭代 N 轮”），skill 会从 `outputs/` 下已有产物恢复搜索，无需重新开始。

完整示例

skill 目录下提供了两个可直接运行的完整示例，供你参考契约实现和项目结构：

1. **MobileNet-V2 图像分类（Torch）**

   相对路径：`assets/example_quantize_mobilenetv2/`

   包含：

   - `user_quant.py` ——完整的 Torch 模型契约实现（ImageNet 校准与评估）。
   - `datasets/` ——模型 evaluate 函数实现。

2. **YOLO11n 目标检测（ONNX）**

   相对路径：`assets/example_quantize_yolo11n/`

   包含：

   - `user_quant.py` ——完整的 ONNX 模型契约实现（COCO 校准与评估）。
   - `yolo11n_eval.py` ——检测任务 evaluate 辅助脚本。

从 esp-dl 仓库根目录出发的绝对路径示例：

```
# MobileNet-V2 示例
cd esp-dl/tools/agents/skills/espdl-quantize/assets/example_quantize_mobilenetv2

# YOLO11n 示例
cd esp-dl/tools/agents/skills/espdl-quantize/assets/example_quantize_yolo11n
```

查看最终产出结果

当 skill 完成所有迭代（或达到收敛条件）后，会在 `outputs/` 目录下生成以下关键产物，便于你查阅和部署最优模型：

final_report.md

`outputs/final_report.md` 是每次量化调优的 ** 最终总结报告 **，包含：

- **Summary** ——最优迭代轮次、最佳指标值、与目标的差距。
- **Iteration history** ——所有迭代轮次的完整历史表，包括每轮改动的方法、指标变化（delta）和效果评估（improvement / regression / new best）。
- **Best setting** ——最优迭代的完整量化配置（JSON 格式），可直接复用。
- **Python snippet** ——可直接复制到代码中使用的 `QuantizationSettingFactory.espdl_setting()` 配置代码片段。
- **Key findings** ——Skill 对搜索过程的总结：最有效的方法、回归的方法、剩余误差分布热点。
- **Remaining gap** ——若未达到目标指标，报告会分析剩余差距并提供后续优化建议（如增大数据量、混合精度、blockwise reconstruction 等）。

以下是一份 MobileNet-V2 on ESP32-S3 示例中的 `final_report.md` 片段：

```
## Summary
- Best iteration: iter_8
- top1: 71.5000 (target_metric=71.8780)
- Other metrics: top5=89.4500

## Iteration history

| iter | method changed | top1 | delta | outcome | rank | affects inference speed |
|---|---|---|---|---|---|---|
| 0 | default espdl_setting() baseline | 60.5000 | +0.0000 = | baseline | 11 | No |
| 1 | Phase 2: calib=kl × TQT(default) | 71.2250 | +10.7250 ↑ | improvement | 2 | No |
| 2 | Phase 2: calib=mse × TQT(default) | 70.5250 | +10.0250 ↑ | improvement | 10 |␣
↪No |
| 3 | Phase 2: calib=percentile × TQT(default) | 70.9000 | +10.4000 ↑ | improvement |␣
↪9 | No |
| 8 | Phase 3 lever 3c: Fusion alignment | 71.5000 | +11.0000 ↑ | **best** | **1** |␣
↪No |

## Best setting
```json
{
  "iteration_id": 8,
  "rationale": "Phase 3 lever 3c: Fusion alignment for elementwise ops...",
  "calib_algorithm": "kl",
  "tqt_optimization": { "enabled": true, "lr": 1e-05, "steps": 500, ... },
  "fusion_alignment": { "align_elementwise_to": "Align to Large" }
}
```

Key findings
- **Best vs baseline**: +11.0000 (from 60.5000 to 71.5000).
- **Most effective lever**: Phase 3 lever 3c (Fusion alignment) ...
- **Largest regression**: iter_12 (Blockwise reconstruction) ...
```

best 目录

`outputs/best/` 目录下保存了 ** 最优迭代 ** 的完整产物，便于直接部署或进一步分析：

- `model.espdl` ——最优量化模型文件，可直接用于 esp-dl 部署。
- `metrics.json` ——最优迭代的评估指标。
- `setting.json` ——最优迭代的完整量化配置（与 `final_report.md` 中的 Best setting 一致）。
- `iteration_index.json` ——指向最优迭代的元数据。

以下是最优迭代的 `metrics.json` 示例：

```
{
  "_primary_key": "top1",
  "_primary_value": 71.5,
  "metric_direction": "max",
  "top1": 71.5,
  "top5": 89.45
}
```

以及 `setting.json` 示例：

```
{
  "iteration_id": 8,
  "rationale": "Phase 3 lever 3c: Fusion alignment for elementwise ops...",
  "calib_algorithm": "kl",
  "tqt_optimization": { "enabled": true, "lr": 1e-05, "steps": 500, ... },
  "fusion_alignment": { "align_elementwise_to": "Align to Large" }
}
```

小技巧: 你可以直接复制 `final_report.md` 中的 Python snippet 到生产代码中复现最优配置，无需手动拼凑参数。

故障排查

`ModuleNotFoundError: No module named 'pandas'`（或 pandas / scipy / tqdm）
未安装 skill 额外依赖。在 skill 目录下运行:

```
pip install -r assets/extra_requirements.txt
```

`KeyError: 'primary_metric'` 或 `AssertionError: primary_metric=... not in metrics=...`
`QUANT_CONFIG["primary_metric"]` 设置的 key 与 `evaluate()` 实际返回的 dict key 不一致。请对齐两者后重新运行。

`OSError: [Errno 2] No such file or directory: '.../user_quant.py'`
skill 找不到契约文件。确认：- 你当前的工作目录包含 `user_quant.py`；或 - 在指令中显式指定了正确的相对/绝对路径。

某轮迭代耗时极长（TQT / LSQ / blockwise reconstruction）
这些方法需要训练，PC 时间较高。若急需结果，可以在指令中要求 skill 跳过 TQT/LSQ 等高耗时 passes，优先尝试 layerwise equalization、bias correction 等零推理开销的方法。

`LSQ pass was skipped because target policy is POWER_OF_2`
正常现象。esp-ppq 在 POWER_OF_2 目标（所有 esp-dl 芯片）上 LSQ 会静默退化，skill 自动跳过并以 TQT 替代。无需处理。

`Layer-wise equalization is not recommended for per-channel weight targets`
warn-only。esp-ppq 使用逐层均衡量化时会发出警告，这是正常现象。

SKILL 没有反应 / 没有执行完所有迭代
可能是 Agent 没有正确触发 skill，或中途出现错误但未正确反馈。尝试:

- 使用显式触发词:”使用 espdl-quantize skill…”。
- 确认 skill 目录结构完整，且已安装额外依赖。
- 使用指令遵循能力更强的模型。
- 换用上下文窗口更大的模型，并开启 Agent 的上下文压缩/自动摘要功能。

迭代多轮后精度不再提升
检查 `outputs/iter_<N>/layer_stats.json`，若所有层的 `Noise:Signal Power Ratio` 均低于 0.05（5%），说明量化误差已接近噪声下限，继续调参收益有限。此时可考虑:

- 增加校准数据量（`calib_steps`）。
- 尝试混合精度（mixed precision）为敏感层保留更高位宽。
- 检查输入预处理是否与训练时完全一致。

3.3 如何加载、测试和性能分析模型

在本教程中，我们将介绍如何加载、测试和分析一个 espdl 模型。参考例程

- 准备工作
- 从 *rodata* 中加载模型
- 从 *partition* 中加载模型
- 从 *sdcard* 中加载模型
- 测试模型板端推理是否正确
- 分析模型内存使用情况
- 分析模型推理延迟
- 组合性能分析：*profile()* 方法

3.3.1 准备工作

1. 安装 *ESP_IDF*
2. 量化导出 *espdl* 模型

3.3.2 从 rodata 中加载模型

此方法将模型文件直接嵌入到应用程序 FLASH 的 `.rodata` 段中。这是最简单的方法，但缺点是每次应用程序代码更改时模型都会被重新烧录。

1. **在 `CMakeLists.txt` 中添加模型文件**

   要将 `.espdl` 模型文件嵌入到 `.rodata` 段，请在 `CMakeLists.txt` 中添加以下代码。前几行应放在 `idf_component_register()` 之前，最后一行放在 `idf_component_register()` 之后。

```
idf_build_get_property(component_targets __COMPONENT_TARGETS)
if ("___idf_espressif__esp-dl" IN_LIST component_targets)
   idf_component_get_property(espdl_dir espressif__esp-dl COMPONENT_DIR)
elseif("___idf_esp-dl" IN_LIST component_targets)
   idf_component_get_property(espdl_dir esp-dl COMPONENT_DIR)
endif()
set(cmake_dir ${espdl_dir}/fbs_loader/cmake)
include(${cmake_dir}/utilities.cmake)
set(embed_files your_model_path/model_name.espdl)

idf_component_register(...)

target_add_aligned_binary_data(${COMPONENT_LIB} ${embed_files} BINARY)
```

2. **在程序中加载模型**

   包含头文件：

```
#include "dl_model_base.hpp"
```

声明模型符号并创建模型：

```
// 符号名由三部分组成：前缀 "_binary_"，文件名 "model_espdl"，后缀 "_start"
extern const uint8_t model_espdl[] asm("_binary_model_espdl_start");

// 基本用法 - 使用默认参数加载模型
dl::Model *model = new dl::Model((const char *)model_espdl, fbs::MODEL_LOCATION_
↪IN_FLASH_RODATA);

// 高级用法 - 自定义参数：
// - 将参数保留在 FLASH 中（节省 PSRAM/内部 RAM，但性能较低）
// - 限制内部 RAM 使用为 0 字节（优先使用 PSRAM）
// - 使用贪婪内存管理器
// - 无加密密钥
// - param_copy = false（将参数保留在 FLASH 中）
// dl::Model *model = new dl::Model((const char *)model_espdl,
//                                  fbs::MODEL_LOCATION_IN_FLASH_RODATA,
//                                  0,  // max_internal_size
//                                  dl::MEMORY_MANAGER_GREEDY,
//                                  nullptr,  // key
//                                  false);  // param_copy
```

备注：性能与内存权衡：

- **烧录时间：**使用从 *rodata 中加载模型* 时，模型文件嵌入在应用程序二进制文件中，每次修改代码时都会重新烧录。对于大型模型，这会增加烧录时间。考虑使用从 *partition 中加载模型* 或从 *sdcard 中加载模型* 来避免此问题。
- **内存 vs 性能：**`param_copy` 参数控制模型参数是否从 FLASH 复制到更快的内存（PSRAM/内部 RAM）。设置 `param_copy=false` 可以节省 RAM，但由于 FLASH 访问速度较慢，会降低推理性能。仅在 RAM 极其紧张时才禁用参数复制。
- **应用程序分区大小：**嵌入在 `.rodata` 中的大型模型可能需要增加 `partition.csv` 中的应用程序分区大小。

3.3.3 从 `partition` 中加载模型

此方法将模型存储在单独的 FLASH 分区中，允许您独立于应用程序代码更新模型。

1. **在 `partition.csv` 中添加模型信息**

   创建或修改您的 `partition.csv` 文件以包含模型分区。有关分区表的详细信息，请参阅 ESP-IDF 分区表文档。

   ```
   # Name,   Type, SubType, Offset,  Size, Flags
   factory,  app,  factory,  0x010000,  4000K,
   model,    data,  spiffs,         ,  4000K,
   ```

   - **Name:** 任何有意义的名称（包括空终止符最多 16 个字符）
   - **Type:** `data`
   - **SubType:** `spiffs`（模型存储必需）
   - **Offset:** 留空以自动计算

- **Size:** 必须大于模型文件大小

2. **在 `CMakeLists.txt` 中添加模型烧录信息**

```
idf_component_register(...)
set(image_file your_model_path/model_name.espdl)
esptool_py_flash_to_partition(flash "model" "${image_file}")
```

`esptool_py_flash_to_partition` 中的第二个参数必须与 `partition.csv` 中的 `Name` 字段匹配。

3. **在程序中加载模型**

包含头文件：

```
#include "dl_model_base.hpp"
```

创建模型实例：

```
// 基本用法 - 使用默认参数加载模型
dl::Model *model = new dl::Model("model", fbs::MODEL_LOCATION_IN_FLASH_PARTITION);

// 高级用法 - 将参数保留在 FLASH 中以节省 RAM
// dl::Model *model = new dl::Model("model",
//                                  fbs::MODEL_LOCATION_IN_FLASH_PARTITION,
//                                  0,  // max_internal_size
//                                  dl::MEMORY_MANAGER_GREEDY,
//                                  nullptr,  // key
//                                  false);   // param_copy
```

第一个参数（分区标签）必须与 `partition.csv` 中的 `Name` 字段匹配。

备注：烧录优化： 使用 `idf.py app-flash` 代替 `idf.py flash`，可以仅烧录应用程序分区而不重新烧录模型分区。这显著减少了开发期间的烧录时间。

3.3.4 从 `sdcard` 中加载模型

此方法从 SD 卡加载模型，当 FLASH 空间有限或需要频繁更新模型而无需重新烧录时非常有用。

1. **准备 SD 卡**
   - **格式：** SD 卡应格式化为 FAT32。如果未格式化，挂载时将自动格式化（数据会丢失）。
   - **备份：** 在使用 ESP-DL 之前，请始终备份 SD 卡数据。
2. **挂载 SD 卡**
   - **使用 BSP（板级支持包）：**

     在 menuconfig 中启用 `CONFIG_BSP_SD_FORMAT_ON_MOUNT_FAIL` 以允许自动格式化。

```
#include "bsp/esp-bsp.h"
ESP_ERROR_CHECK(bsp_sdcard_mount());
```

   - **不使用 BSP：**

     配置挂载选项，设置 `format_if_mount_failed = true`。

```
#include "esp_vfs_fat.h"
#include "sdmmc_cmd.h"

esp_vfs_fat_sdmmc_mount_config_t mount_config = {
    .format_if_mount_failed = true,
    .max_files = 5,
    .allocation_unit_size = 16 * 1024
};
// 挂载 SD 卡（具体实现取决于您的硬件）
```

3. **复制模型到 SD 卡**

   将您的 `.espdl` 模型文件复制到 SD 卡（例如，复制到根目录作为 `model.espdl`）。

4. **在程序中加载模型**

   包含头文件：

```
#include "dl_model_base.hpp"
```

- 如果不使用 BSP(Board Support Package)

```
// 挂载sdcard.
const char *model_path = "/your_sdcard_mount_point/your_model_path/model_name.
↪espdl";
Model *model = new Model(model_path, fbs::MODEL_LOCATION_IN_SDCARD);
```

备注： 使用从 *sdcard 中加载模型* 时，模型加载过程将花费更长的时间，因为模型数据需要从 sdcard 复制到 PSRAM 或者 internal RAM。如果你的 FLASH 空间紧张，这种方法很有用。

3.3.5 测试模型板端推理是否正确

`test()` 方法通过将推理结果与模型文件中嵌入的基准真值进行比较，验证模型是否产生正确的推理结果。

前提条件：

- `.espdl` 模型必须在 ESP-PPQ 中导出时启用 ** 测试输入和输出 **（使用 `export_test_values` 选项）。
- 对于部署，您可以导出一个没有测试数据的版本以减小模型大小。

API： `esp_err_t dl::Model::test()`

返回值： 如果所有测试通过则返回 `ESP_OK`，否则返回 `ESP_FAIL`。

用法：

```
#include "dl_model_base.hpp"

// 创建模型后...
esp_err_t ret = model->test();
if (ret == ESP_OK) {
    ESP_LOGI(TAG, "模型测试通过！");
} else {
    ESP_LOGE(TAG, "模型测试失败！");
}
```

(续下页)

(接上页)

```
// 或使用便捷宏：
ESP_ERROR_CHECK(model->test());
```

工作原理：

1. 加载模型中嵌入的测试输入张量，所以 test() 不需要外部输入
2. 通过所有模型层运行推理
3. 将每个输出与基准真值进行比较（考虑量化误差的容差）
4. 报告每个输出的成功或失败

INT16 模型注意事项：由于量化舍入误差，INT16 模型允许比较时有 ±1 的差异。

3.3.6 分析模型内存使用情况

`profile_memory()` 方法打印跨不同内存类型（内部 RAM、PSRAM、FLASH）的内存使用详细明细。

API： `void dl::Model::profile_memory()`

用法：

```
#include "dl_model_base.hpp"

// 创建并测试模型后...
model->profile_memory();
```

输出包括：

名称	解释
fbs_model parameter	flatbuffers 模型，包含一个子项，模型参数 parameter。flatbuffers 模型除了模型参数之外，还包括测试输入输出，模型参数/变量的形状，模型结构等信息。
parameter_copy	复制的模型参数，当 flatbuffers 模型位于 FLASH 的时候，默认情况下会复制到 PSRAM 或者 internal RAM 以提高推理性能。
variable	内存管理模块申请的内存，模型输入/输出以及中间的计算结果都会使用这部分空间。
others	类成员变量所需要的空间，heap_caps_aligned_alloc / heap_caps_aligned_calloc 申请过程中对齐的额外部分（很小）。

显示的内存类型：每个类别的内部 RAM、PSRAM 和 FLASH 使用情况。

3.3.7 分析模型推理延迟

`profile_module()` 方法打印模型中每个模块（层）的详细延迟信息。

API： `void dl::Model::profile_module(bool sort_module_by_latency = false)`

参数： - `sort_module_by_latency`：如果为 `true`，模块按延迟排序（最高优先）。如果为 `false`（默认），模块按拓扑顺序显示。

用法：

```
// 默认：拓扑顺序
model->profile_module();

// 按延迟排序（最高优先）
model->profile_module(true);
```

输出包括： - 模块名称 - 模块类型（操作类型）- 推理延迟（微秒，如果启用 `DL_LOG_LATENCY_UNIT` 则为周期数）- 末尾的总推理延迟

相关 API：

- `std::map<std::string, module_info> get_module_info()` - 以编程方式返回模块信息
- `void print_module_info(const std::map<std::string, module_info> &info, bool sort_module_by_latency = false)` - 从映射打印模块信息

3.3.8 组合性能分析：profile() 方法

`profile()` 方法结合了 `profile_memory()` 和 `profile_module()`，进行综合分析。

API： `void dl::Model::profile(bool sort_module_by_latency = false)`

用法：

```
// 拓扑顺序的综合性能分析
model->profile();

// 按延迟排序的综合性能分析
model->profile(true);
```

这是获取内存和性能分析的最便捷方式。

3.4 如何进行模型推理

在本教程中，我们将介绍最基本的模型推理流程。参考例程

- 准备工作
- 加载模型
- 获取模型输入/输出。
- 量化输入

      - 量化单个值
      - 量化 *dl::TensorBase*
  - 反量化输出
      - 反量化单个值
      - 反量化 *dl::TensorBase*
  - 模型推理

3.4.1 准备工作

安装 ESP_IDF

3.4.2 加载模型

如何加载模型

3.4.3 获取模型输入/输出。

```
std::map<std::string, dl::TensorBase *> model_inputs = model->get_inputs();
dl::TensorBase *model_input = model_inputs.begin()->second;
std::map<std::string, dl::TensorBase *> model_outputs = model->get_outputs();
dl::TensorBase *model_output = model_outputs.begin()->second;
```

可以通过 `get_inputs()` 和 `get_outputs()` api 获得输入/输出的名字和对应的 `dl::TensorBase`。更多信息，请参阅*dl::TensorBase 文档* 。

备注： ESP-DL 的内存管理器会为每个模型的输入/中间结果/输出分配一整块的内存。由于它们共用这部分内存，所以当模型进行推理的时候，后面的结果会覆盖前面的结果。也就是说，`model_input` 中的数据，在执行完模型推理之后，可能就会被 `model_output` 或者其他中间结果所覆盖。

3.4.4 量化输入

8bit 和 16bit 量化的模型，分别接受 `int8_t` 和 `int16_t` 类型的输入。`float` 类型的输入必须先根据 `exponent` 量化成对应的整数类型之后才能喂入模型。计算公式：

$$Q = \text{Clip}\left(\text{Round}\left(\frac{R}{\text{Scale}}\right), \text{MIN}, \text{MAX}\right)$$

$$\text{Scale} = 2^{\text{Exp}}$$

其中：

- R 是要量化的浮点数。
- Q 是量化后的整数值，需要在 [MIN, MAX] 范围内进行裁剪。
- MIN 整数最小值，8bit 时，MIN = -128, 16bit 时，MIN = -32768。
- MAX 整数最大值，8bit 时，MAX = 127, 16bit 时，MAX = 32767。

量化单个值

```
float input_v = VALUE;
// Note that dl::quantize accepts inverse of scale as the second input, so we use DL_
↪RESCALE here.
int8_t quant_input_v = dl::quantize<int8_t>(input_v, DL_RESCALE(model_input->
↪exponent));
```

量化 dl::TensorBase

```
// assume that input_tensor already contains the float input data.
dl::TensorBase *input_tensor;
model_input->assign(input_tensor);
```

3.4.5 反量化输出

8bit 和 16bit 量化的模型，分别得到 int8_t 和 int16_t 类型的输出。必须根据 exponent 反量化之后才能得到浮点输出。计算公式：

$$R' = Q \times \text{Scale}$$

$$\text{Scale} = 2^{\text{Exp}}$$

其中：

- R’是反量化后恢复的近似浮点值。
- Q 是量化后的整数值。

反量化单个值

```
int8_t quant_output_v = VALUE;
float output_v = dl::dequantize(quant_output_v, DL_SCALE(model_output->exponent));
```

反量化 dl::TensorBase

```
// create a TensorBase filled with 0 of shape [1, 1]
dl::TensorBase *output_tensor = new dl::TensorBase({1, 1}, nullptr, 0, dl::DATA_TYPE_
↪FLOAT);
output_tensor->assign(model_output);
```

3.4.6 模型推理

请参阅：

- 参考例程
- *void dl::Model::run(runtime_mode_t mode)*
- *void dl::Model::run(TensorBase *input, runtime_mode_t mode)*
- *void dl::Model::run(std::map<std::string, TensorBase*> &user_inputs, runtime_mode_t mode, std::map<std::string, TensorBase*> user_outputs)*

3.5 如何创建新模块（算子）

本教程将指导您在 `dl::module` 命名空间中创建一个新模块。`Module` 类是所有模块的基类，您将扩展这个基类来创建您的自定义模块。

备注： ESP-DL 中的模块接口应与 ONNX 对齐。

3.5.1 理解基类 `Module`

基类提供了几个必须在派生类中重写的虚方法。

- **方法：**
  - *dl::module::Module::Module()*：构造函数，用于初始化模块。
  - *dl::module::Module::~Module()*：析构函数，用于释放资源。
  - *dl::module::Module::get_output_shape()*：根据输入形状计算输出形状。
  - *dl::module::Module::forward()*：运行模块，高级接口。
  - *dl::module::Module::forward_args()*：运行模块，低级接口。
  - *dl::module::Module::deserialize()*：从序列化信息创建模块实例。
  - *dl::module::Module::print()*：打印模块信息。

更多信息，请参考 Module Class Reference。

3.5.2 创建新模块类

要创建一个新模块，您需要从 `Module` 基类派生一个新类并重写必要的方法。

示例：创建 MyCustomModule 类

更多示例，请参考 esp-dl/dl/module。

```
#include "module.h" // 包含定义 Module 类的头文件

namespace dl {
namespace module {

class MyCustomModule : public Module {
public:
    // 构造函数
    MyCustomModule(const char *name = "MyCustomModule",
                module_inplace_t inplace = MODULE_NON_INPLACE,
                quant_type_t quant_type = QUANT_TYPE_NONE)
        : Module(name, inplace, quant_type) {}

    // 析构函数
    virtual ~MyCustomModule() {}

    // 重写 get_output_shape 方法
    std::vector<std::vector<int>> get_output_shape(std::vector<std::vector<int>> &
↪input_shapes) override {
        // 实现根据输入形状计算输出形状的逻辑
        std::vector<std::vector<int>> output_shapes;
        // 示例：假设输出形状与输入形状相同
        output_shapes.push_back(input_shapes[0]);
        return output_shapes;
    }

    // 重写 forward 方法
    void forward(std::vector<dl::TensorBase *> &tensors, runtime_mode_t mode =␣
↪RUNTIME_MODE_AUTO) override {
        // 实现运行模块的逻辑
        // 示例：对张量执行某些操作
        for (auto &tensor : tensors) {
            // 对每个张量执行某些操作
        }
    }

    // 重写 forward_args 方法
    void forward_args(void *args) override {
        // 实现低级接口的逻辑
        // 示例：根据参数执行某些操作
    }

    // 从序列化信息反序列化模块实例
    static Module *deserialize(fbs::FbsModel *fbs_model, std::string node_name){
        // 实现反序列化模块实例的逻辑
        // 接口应与 ONNX 对齐
    }

    // 重写 print 方法
    void print() override {
        // 打印模块信息
        ESP_LOGI("MyCustomModule", "Module Name: %s, Quant type: %d", name.c_str(),␣
↪quant_type);
    }
```

(续下页)

(接上页)

```
};

} // namespace module
} // namespace dl
```

注册 MyCustomModule 类

当您实现了 MyCustomModule 类后，请在 dl_module_creator 中注册您的模块，使其全局可用。

```
void register_dl_modules()
{
    if (creators.empty()) {
        ...
        this->register_module("MyCustomModule", MyCustomModule::deserialize);
    }
}
```

3.6 使用 AI Agent 自动实现算子

本文档介绍如何在各种 Coding Agent 工具（如 Claude、Cursor、OpenCode 等）中安装和使用 espdl-operator skill，以便在 ESP-DL 框架中完成对神经网络算子的自动化实现。**以下说明以 Linux 环境为例**。

- 什么是 espdl-operator skill
- 依赖要求
- 安装/放置 Skill
- 快速开始示例
- Skill 触发使用
- 主要功能流程
- 故障排除
- 相关资源

3.6.1 什么是 espdl-operator skill

espdl-operator 是一个供 Coding Agent 使用的自动化开发 skill，用于在 ESP-DL 框架中实现、测试和优化神经网络算子。当你向 Coding Agent（如 Claude、Cursor、OpenCode 等）提出算子相关请求时，该 skill 会指导 AI 自动完成以下工作：

Coding Agent 将自动为你：

1. **分析算子需求** - 解析 ONNX 算子规范，确定算子类型和数据类型支持
2. **生成 esp-dl C++ 代码** - 自动创建 Module 层和 Base 层的头文件/实现文件
3. **修改 esp-ppq 量化配置** - 在量化工具中注册算子支持，配置 layout pattern

4. **创建测试用例** - 生成 PyTorch/ONNX 测试模型，配置测试参数
5. **执行构建与测试** - 运行 Docker 构建，生成测试数据，执行硬件测试
6. **验证结果对齐** - 确保 esp-dl 和 esp-ppq 的推理结果一致

Skill 的核心价值：

- **端到端自动化** - 从需求到可运行代码，Coding Agent 自动完成所有步骤
- **跨仓库协调** - 同时修改 esp-dl（C++）和 esp-ppq（Python）两个代码库
- **遵循最佳实践** - 自动应用 ESP-DL 的代码规范、目录结构和测试流程
- **增量式开发** - 支持新算子实现、数据类型扩展等多种场景

适用场景：

场景	示例请求
实现新算子	“实现 HardSwish 算子，支持 int8 和 float32”
添加数据类型支持	“给 Tanh 算子添加 int16 支持”
量化支持	“为 Mod 添加量化支持”
结果对齐	“验证 LogSoftmax 在 esp-dl 和 esp-ppq 的结果是否一致”

3.6.2 依赖要求

在使用 `espdl-operator` skill 之前，你需要提前安装以下依赖：

必需提前安装的依赖

依赖	用途	安装命令
Docker	用于构建和测试环境	官方安装指南
uv	Python 包管理器	`curl -LsSf https://astral.sh/uv/install.sh | sh`
Git	版本控制	`apt install git` (Ubuntu/Debian)

由 Skill 自动处理的依赖

以下依赖 ** 无需手动安装 **, `espdl-operator` skill 会在执行过程中自动处理：

- **esp-ppq**: Python 量化工具包 - skill 会自动在 Docker 容器中以源码的形式安装
- **文档生成脚本**: `gen_ops_markdown.py` 等工具 - skill 会自动运行
- **Docker 镜像**: `espdl/idf-ppq` 镜像 - skill 会自动构建（如不存在）
- **ESP-IDF**: 开发框架 - 已包含在 Docker 镜像中

验证依赖安装

安装完成后，请验证以下命令可以正常执行：

```
# 检查 Docker
docker --version

# 检查 uv
uv --version

# 检查 Git
git --version
```

如果以上命令都能正常输出版本信息，说明环境已就绪，可以开始使用 skill。

3.6.3 安装/放置 Skill

项目结构

首先，确认你的项目目录结构如下（esp_dl_project_1 是项目根目录，根目录目录名称可以任意）。以 opencode 为例，目录结构全景图如下：

```
esp_dl_project_1/                    <-- 项目根目录（所有命令在此执行）
├── esp-dl/                          # ESP-DL 主代码库
│   ├── esp-dl/                      # 核心库源代码（dl/, vision/, audio/ 等）
│   ├── examples/                    # 示例程序
│   ├── test_apps/                   # 测试应用
│   ├── tools/                       # 工具脚本
│   └── ...
├── esp-ppq/                         # 量化工具（与 esp-dl 同级）
│   ├── esp_ppq/                     # 主包源代码
│   ├── pyproject.toml               # 项目配置文件
│   └── ...
└── .opencode/                       # OpenCode 配置（需要创建）
    └── skills/espdl-operator/       # skill 安装位置（指向 esp-dl/tools/agents/
↪skills/espdl-operator/）
```

注意: skill 的源代码位于 esp-dl/tools/agents/skills/espdl-operator/，你需要将其复制或链接到 .opencode/skills/espdl-operator/（或其他 Agent 对应的目录）。skill 文件包括 SKILL.md（主文件）和 references/（参考模板和检查清单）。

重要：执行命令的位置

所有以下命令都必须在项目根目录 "esp_dl_project_1/" 下执行。

如果你不确定当前位置，请先执行：

```
# 查看当前目录
pwd

# 应该输出类似：/home/username/workspace/esp_dl_project_1
# 或 /path/to/esp_dl_project_1

# 如果不在项目根目录，请先跳转
cd /path/to/esp_dl_project_1
```

方法一：使用 npx（推荐 - 最简单）

为 OpenCode、Cursor、Claude Code 及其他兼容工具安装 skill 最简单的方法是使用 npx：

```
npx skills add espressif/esp-dl --skill espdl-operator
```

备注： npx 是 Node.js（通过 npm）自带的包执行器。如果你尚未安装 Node.js，请参考 Node.js 安装指南 进行安装。

执行命令后，skill 将被自动安装并在你的 Coding Agent 工具中可用。

方法二：手动安装

如果你偏好手动安装，或者你的工具不支持 npx，请根据你使用的具体工具按照以下说明操作：

备注： 不同 Agent 工具的 skill 安装目录可能有所不同，请根据实际使用的工具选择对应的安装路径。

OpenCode

方法一：复制文件

```
# 确保你在项目根目录 esp_dl_project_1/
cd /path/to/esp_dl_project_1

# 创建 .opencode/skills 目录
mkdir -p .opencode/skills/espdl-operator

# 从 esp-dl/tools/agents/skills/espdl-operator 复制到 .opencode/skills/espdl-operator
cp -r esp-dl/tools/agents/skills/espdl-operator/* .opencode/skills/espdl-operator/
```

方法二：使用符号链接（推荐用于开发，保持同步更新）

```
# 确保你在项目根目录 esp_dl_project_1/
cd /path/to/esp_dl_project_1

# 创建 .opencode/skills 目录
mkdir -p .opencode/skills

# 创建符号链接（使用相对路径）
# 注意：从 .opencode/skills/espdl-operator 指向 esp-dl/tools/agents/skills/espdl-
↪operator
ln -s ../../esp-dl/tools/agents/skills/espdl-operator .opencode/skills/espdl-operator

# 验证链接是否成功
ls -la .opencode/skills/espdl-operator
# 应该显示 SKILL.md 和 references/ 目录
```

启动 OpenCode 后，系统会自动加载该 skill。

Cursor

方法一：复制文件

```
# 确保你在项目根目录 esp_dl_project_1/
cd /path/to/esp_dl_project_1

# 创建 Cursor skills 目录
mkdir -p .cursor/skills/espdl-operator

# 复制 skill 文件
cp -r esp-dl/tools/agents/skills/espdl-operator/* .cursor/skills/espdl-operator/
```

方法二：使用符号链接

```
# 确保你在项目根目录 esp_dl_project_1/
cd /path/to/esp_dl_project_1

# 创建 .cursor/skills 目录
mkdir -p .cursor/skills

# 创建符号链接
ln -s ../../esp-dl/tools/agents/skills/espdl-operator .cursor/skills/espdl-operator
```

Claude Desktop (Claude Code)

方法一：复制文件

```
# 确保你在项目根目录 esp_dl_project_1/
cd /path/to/esp_dl_project_1

# 创建 Claude skills 目录
mkdir -p .claude/skills/espdl-operator

# 复制 skill 文件
cp -r esp-dl/tools/agents/skills/espdl-operator/* .claude/skills/espdl-operator/
```

方法二：使用符号链接

```
# 确保你在项目根目录 esp_dl_project_1/
cd /path/to/esp_dl_project_1

# 创建 .claude/skills 目录
mkdir -p .claude/skills

# 创建符号链接
ln -s ../../esp-dl/tools/agents/skills/espdl-operator .claude/skills/espdl-operator
```

3.6.4 快速开始示例

假设你要实现一个新的算子 `MyOp`:

1. **确保 skill 已安装**

```
ls -la .opencode/skills/espdl-operator/SKILL.md
```

2. **在 Agent 中提问**

```
"帮我实现一个 MyOp 算子，支持 int8、int16 和 float32"
```

3. **Agent 会自动**
   - 加载 skill
   - 按照 9 个阶段指导 Coding Agent 工作
   - 生成必要的代码文件
   - 运行 Docker 测试

3.6.5 Skill 触发使用

安装完成后，你可以通过以下方式触发 `espdl-operator` skill:

自然语言触发

直接在对话中使用以下关键词:

中文触发词	英文触发词
“实现算子”	“implement operator”
“添加算子”	“add operator”
“量化支持”	“quantization support”
“算子对齐”	“operator alignment”
“添加新的算子”	“add a new op”

示例对话

```
用户："帮我实现一个 Mod 算子"
Agent: [自动加载 espdl-operator skill 并开始指导]

用户："添加 LogSoftmax 算子到 esp-dl"
Agent: [自动加载 skill 并提供实现步骤]
```

显式调用

如果自动触发未生效，可以显式要求 Agent 使用该 skill：

```
"使用 espdl-operator skill 帮我实现 Softmax 算子"
"根据 espdl-operator skill 的指导，给 LogSoftmax 添加 int16 支持"
```

3.6.6 主要功能流程

该 skill 指导 Coding Agent 完成以下主要阶段：

Phase 1: 研究与分类

- 理解 ONNX 算子规范
- 确定算子类型（Elementwise、Convolution、Pooling 等）
- 确定支持的数据类型（int8、int16、float32）

Phase 2: 实现 esp-dl Module 层

- 创建算子模块头文件 (`dl_module_<op>.hpp`)
- 实现 `get_output_shape()` 和 `forward()` 方法
- 在 `dl_module_creator.hpp` 中注册算子

Phase 3: 实现 esp-dl Base 层

- 创建 C 参考实现 (`dl_base_<op>.hpp/cpp`)

Phase 4: esp-ppq 集成

- 在 `EspdlQuantizer.py` 中注册量化支持
- 在 `espdl_typedef.py` 中配置 layout pattern

Phase 5: 配置测试用例

- 添加 PyTorch/ONNX 测试模型构建器
- 在 `op_cfg.toml` 中配置测试参数

Phase 6: Docker 构建与测试

- 生成测试用例（int8、int16、float32）
- 构建测试应用
- 在硬件上运行测试

Phase 7: SIMD 优化（可选）

- 这一部分暂未支持，后续会迭代优化

Phase 8: 算子对齐验证

- 确保 esp-dl 和 esp-ppq 推理结果一致

Phase 9: 更新文档

- 运行 `gen_ops_markdown.py` 更新算子支持状态文档

3.6.7 故障排除

Skill 未触发

- 确认 skill 目录位于正确的目录（以 opencode 为例：`.opencode/skills/espdl-operator`）
- 尝试使用显式触发词:”使用 espdl-operator skill…”
- 确认 Agent 工具支持 skill

Skill 流程未全部执行

- 中间有流程未执行，显式调用 `espdl-operator` skill 执行，例如:

```
# 未执行 docker 命令进行硬件烧录/测试，则在对话中下达命令:
基于 espdl-operator skill 的说明，对硬件进行烧录测试，硬件已连接
```

算子实现效果不理想

该 skill 的作用是指导 AI 自动完成算子代码的编写。所以最终效果会受到两个因素影响:

1. **你使用的 AI 工具** (如 OpenCode、Cursor、Claude 等)
2. **AI 模型本身的代码能力** (不同模型写代码的水平，理解能力，调用 tool 的能力有差异)

根据我们的测试经验，以下组合在实现 C 语言版本的算子时，效果较好（因资源有限，许多组合还未覆盖，可自行尝试）:

Coding Agent 工具	使用的模型
Cursor	Claude Opus 4.6
OpenCode	Kimi 2.5 + GLM 5 或 Claude Opus 4.6

如果发现生成的代码质量不好，可以尝试：

- 换用更强的 AI 模型（如 Claude Opus、GPT-4 等）
- 尝试下效果更好的 Coding Agent 工具
- 参考 SKILL.md 中的详细指导手动实现

Docker 问题

```
# 检查 Docker 是否运行
docker ps

# 重新构建镜像
cd esp-dl/tools/agents/skills/espdl-operator/assets/docker
docker build -t espdl/idf-ppq:latest .
```

权限问题

```
# 确保设备访问权限 (Linux)
sudo usermod -a -G dialout $USER
# 重新登录以生效
```

3.6.8 相关资源

- **SKILL.md**: `esp-dl/tools/agents/skills/espdl-operator/SKILL.md` - 完整的开发指南
- **模板**: `esp-dl/tools/agents/skills/espdl-operator/references/esp-dl-templates.md`
- **检查清单**: `esp-dl/tools/agents/skills/espdl-operator/references/esp-ppq-checklist.md`
- **esp-dl**: `esp-dl/` - esp-dl 主代码库
- **esp-ppq**: `esp-ppq/` - esp-ppq 量化工具（与 esp-dl 同级目录）

3.7 如何部署 MobileNetV2

在本教程中，我们介绍如何使用 ESP-PPQ 对预训练的 MobileNetV2 模型进行量化，并使用 ESP-DL 部署量化后的 MobileNetV2 模型。

- 准备工作
- 模型量化
  - 预训练模型
  - 校准数据集
  - *8bit* 后量化

- 模型部署
  - 图像分类基类
  - 前处理
  - 后处理

3.7.1 准备工作

1. 安装 *ESP_IDF*
2. 安装 *ESP_PPQ*

3.7.2 模型量化

备注:

- **ESP32P4**：对 Conv 和 Gemm 算子的权重采用 **per-channel** 量化策略，其余算子仍使用 **per-tensor**。
- **ESP32S3 及其他芯片**：因指令集限制，所有算子均统一采用 **per-tensor** 量化策略。

由于 **per-channel** 量化在细节保留上通常优于 **per-tensor**，因此相同模型在 ESP32-P4 上的量化精度往往高于 ESP32-S3。

量化脚本

预训练模型

从 torchvision 加载 MobileNet_v2 的预训练模型，你也可以从 ONNX models 或 TensorFlow models 下载：

```
import torchvision
from torchvision.models.mobilenetv2 import MobileNet_V2_Weights

model = torchvision.models.mobilenet.mobilenet_v2(weights=MobileNet_V2_Weights.
↪IMAGENET1K_V1)
```

校准数据集

校准数据集需要和你的模型输入格式一致，校准数据集需要尽可能覆盖你的模型输入的所有可能情况，以便更好地量化模型。这里以 ImageNet 数据集为例，演示如何准备校准数据集。

使用 torchvision 加载 ImageNet 数据集：

```
import torchvision.datasets as datasets
from torch.utils.data.dataset import Subset
dataset = datasets.ImageFolder(
   CALIB_DIR,
   transforms.Compose(
         [
            transforms.Resize(256),
            transforms.CenterCrop(224),
```

(续下页)

(接上页)

```
            transforms.ToTensor(),
            transforms.Normalize(
               mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
            ),
         ]
   ),
)
dataset = Subset(dataset, indices=[_ for _ in range(0, 1024)])
dataloader = DataLoader(
   dataset=dataset,
   batch_size=BATCH_SIZE,
   shuffle=False,
   num_workers=4,
   pin_memory=False,
   collate_fn=collate_fn1,
)
```

8bit 后量化

下面的量化设置通过 AutoQuant 搜索得到。要使用 AutoQuant，请更新 esp-ppq 为最新版本并参考 教程。

ESP32-P4 量化设置

```
# default: Replace ReLU6 with ReLU.

quant_setting = QuantizationSettingFactory.espdl_setting()
quant_setting.quantize_activation_setting.calib_algorithm = 'kl'

quant_setting.equalization = True
quant_setting.equalization_setting.iterations = 4
quant_setting.equalization_setting.value_threshold = 0.1
quant_setting.equalization_setting.opt_level = 2
quant_setting.equalization_setting.interested_layers = None

quant_setting.tqt_optimization = True
tqt_setting = quant_setting.tqt_optimization_setting
tqt_setting.lr = 5e-5
tqt_setting.steps = 800
tqt_setting.block_size = 4
tqt_setting.is_scale_trainable = True
tqt_setting.gamma = 0.0
tqt_setting.int_lambda = 0.0
tqt_setting.collecting_device = 'cuda'
```

ESP32-P4 量化误差

```
Analysing Graphwise Quantization Error(Phrase 1):: 100
↪%|████████████████████████████████████████████████████████████████████████████████
↪8/8 [00:03<00:00,  2.45it/s]
Analysing Graphwise Quantization Error(Phrase 2):: 100
↪%|████████████████████████████████████████████████████████████████████████████████
↪8/8 [00:07<00:00,  1.13it/s]
Layer                                              | NOISE:SIGNAL POWER RATIO
/features/features.17/conv/conv.0/conv.0.0/Conv:   | ████████████████████ | 16.879%
/features/features.16/conv/conv.2/Conv:            | ████████████████████ | 15.993%
```

(续下页)

(接上页)

```
/features/features.15/conv/conv.2/Conv:           | ████████████        | 11.915%
/features/features.1/conv/conv.1/Conv:            | ███████████         | 10.603%
/features/features.17/conv/conv.2/Conv:           | ██████████          | 10.027%
/features/features.14/conv/conv.2/Conv:           | ██████████          | 9.947%
/features/features.18/features.18.0/Conv:         | ██████████          | 9.698%
/features/features.6/conv/conv.2/Conv:            | ████████            | 8.669%
/features/features.13/conv/conv.2/Conv:           | ████████            | 8.486%
/features/features.12/conv/conv.2/Conv:           | ███████             | 7.475%
/features/features.16/conv/conv.0/conv.0.0/Conv:  | ███████             | 7.409%
/features/features.3/conv/conv.2/Conv:            | ███████             | 7.102%
/features/features.5/conv/conv.2/Conv:            | ███████             | 6.808%
/features/features.16/conv/conv.1/conv.1.0/Conv:  | ███████             | 6.629%
/features/features.10/conv/conv.2/Conv:           | ██████              | 6.346%
/features/features.15/conv/conv.1/conv.1.0/Conv:  | ██████              | 6.103%
/features/features.15/conv/conv.0/conv.0.0/Conv:  | ██████              | 5.871%
/features/features.11/conv/conv.2/Conv:           | ██████              | 5.859%
/features/features.9/conv/conv.2/Conv:            | ██████              | 5.553%
/features/features.4/conv/conv.2/Conv:            | ████                | 4.576%
/features/features.7/conv/conv.2/Conv:            | ████                | 4.275%
/features/features.14/conv/conv.1/conv.1.0/Conv:  | ████                | 3.975%
/features/features.2/conv/conv.2/Conv:            | ███                 | 3.867%
/features/features.14/conv/conv.0/conv.0.0/Conv:  | ███                 | 3.843%
/features/features.8/conv/conv.2/Conv:            | ███                 | 3.688%
/features/features.2/conv/conv.1/conv.1.0/Conv:   | ███                 | 3.620%
/features/features.5/conv/conv.1/conv.1.0/Conv:   | ███                 | 3.545%
/features/features.13/conv/conv.1/conv.1.0/Conv:  | ███                 | 3.508%
/features/features.12/conv/conv.1/conv.1.0/Conv:  | ███                 | 3.423%
/features/features.13/conv/conv.0/conv.0.0/Conv:  | ███                 | 3.422%
/features/features.6/conv/conv.1/conv.1.0/Conv:   | ███                 | 3.398%
/features/features.3/conv/conv.1/conv.1.0/Conv:   | ███                 | 3.311%
/features/features.10/conv/conv.1/conv.1.0/Conv:  | ███                 | 3.184%
/features/features.2/conv/conv.0/conv.0.0/Conv:   | ██                  | 2.923%
/features/features.11/conv/conv.1/conv.1.0/Conv:  | ██                  | 2.919%
/features/features.9/conv/conv.1/conv.1.0/Conv:   | ██                  | 2.899%
/features/features.8/conv/conv.1/conv.1.0/Conv:   | ██                  | 2.801%
/features/features.7/conv/conv.0/conv.0.0/Conv:   | ██                  | 2.687%
/features/features.4/conv/conv.0/conv.0.0/Conv:   | ██                  | 2.534%
/features/features.12/conv/conv.0/conv.0.0/Conv:  | ██                  | 2.532%
/features/features.10/conv/conv.0/conv.0.0/Conv:  | ██                  | 2.404%
/classifier/classifier.1/Gemm:                    | ██                  | 2.323%
/features/features.11/conv/conv.0/conv.0.0/Conv:  | ██                  | 2.314%
/features/features.17/conv/conv.1/conv.1.0/Conv:  | ██                  | 2.218%
/features/features.1/conv/conv.0/conv.0.0/Conv:   | ▌                   | 2.154%
/features/features.4/conv/conv.1/conv.1.0/Conv:   | ▌                   | 2.094%
/features/features.6/conv/conv.0/conv.0.0/Conv:   | ▌                   | 1.591%
/features/features.7/conv/conv.1/conv.1.0/Conv:   | ▏                   | 1.313%
/features/features.9/conv/conv.0/conv.0.0/Conv:   | ▏                   | 1.147%
/features/features.8/conv/conv.0/conv.0.0/Conv:   | ▏                   | 1.126%
/features/features.5/conv/conv.0/conv.0.0/Conv:   | ▏                   | 0.995%
/features/features.3/conv/conv.0/conv.0.0/Conv:   | ▏                   | 0.810%
/features/features.0/features.0.0/Conv:           |                     | 0.109%
Analysing Layerwise quantization error:: 100
 ↪%|████████████████████████████████████████████████████████████████████████████████████████████████████████████
 ↪53/53 [05:01<00:00,  5.69s/it]
Layer                                             | NOISE:SIGNAL POWER RATIO
/features/features.1/conv/conv.0/conv.0.0/Conv:   | ████████████████    | 27.681%
/features/features.1/conv/conv.1/Conv:            | ███████████████     | 26.101%
```

(续下页)

(接上页)

```
/features/features.0/features.0.0/Conv:          | ▉                    | 1.908%
/features/features.14/conv/conv.2/Conv:          | ▉                    | 1.741%
/features/features.2/conv/conv.1/conv.1.0/Conv:  | ▉                    | 1.665%
/features/features.3/conv/conv.1/conv.1.0/Conv:  | ▉                    | 1.123%
/features/features.16/conv/conv.2/Conv:          | ▉                    | 0.902%
/features/features.3/conv/conv.2/Conv:           | ▉                    | 0.872%
/features/features.2/conv/conv.2/Conv:           | ▉                    | 0.806%
/features/features.2/conv/conv.0/conv.0.0/Conv:  | ▉                    | 0.735%
/features/features.17/conv/conv.0/conv.0.0/Conv: |                      | 0.529%
/features/features.14/conv/conv.1/conv.1.0/Conv: |                      | 0.510%
/features/features.15/conv/conv.2/Conv:          |                      | 0.409%
/features/features.4/conv/conv.2/Conv:           |                      | 0.352%
/features/features.14/conv/conv.0/conv.0.0/Conv: |                      | 0.340%
/features/features.4/conv/conv.0/conv.0.0/Conv:  |                      | 0.339%
/features/features.3/conv/conv.0/conv.0.0/Conv:  |                      | 0.333%
/classifier/classifier.1/Gemm:                   |                      | 0.295%
/features/features.17/conv/conv.2/Conv:          |                      | 0.244%
/features/features.15/conv/conv.0/conv.0.0/Conv: |                      | 0.231%
/features/features.16/conv/conv.0/conv.0.0/Conv: |                      | 0.214%
/features/features.7/conv/conv.2/Conv:           |                      | 0.211%
/features/features.15/conv/conv.1/conv.1.0/Conv: |                      | 0.206%
/features/features.4/conv/conv.1/conv.1.0/Conv:  |                      | 0.194%
/features/features.13/conv/conv.2/Conv:          |                      | 0.179%
/features/features.11/conv/conv.2/Conv:          |                      | 0.160%
/features/features.7/conv/conv.0/conv.0.0/Conv:  |                      | 0.137%
/features/features.18/features.18.0/Conv:        |                      | 0.134%
/features/features.13/conv/conv.0/conv.0.0/Conv: |                      | 0.131%
/features/features.6/conv/conv.2/Conv:           |                      | 0.122%
/features/features.11/conv/conv.1/conv.1.0/Conv: |                      | 0.106%
/features/features.5/conv/conv.2/Conv:           |                      | 0.103%
/features/features.16/conv/conv.1/conv.1.0/Conv: |                      | 0.098%
/features/features.11/conv/conv.0/conv.0.0/Conv: |                      | 0.090%
/features/features.10/conv/conv.2/Conv:          |                      | 0.083%
/features/features.12/conv/conv.2/Conv:          |                      | 0.080%
/features/features.8/conv/conv.2/Conv:           |                      | 0.072%
/features/features.5/conv/conv.1/conv.1.0/Conv:  |                      | 0.070%
/features/features.8/conv/conv.1/conv.1.0/Conv:  |                      | 0.069%
/features/features.6/conv/conv.1/conv.1.0/Conv:  |                      | 0.067%
/features/features.5/conv/conv.0/conv.0.0/Conv:  |                      | 0.052%
/features/features.12/conv/conv.0/conv.0.0/Conv: |                      | 0.049%
/features/features.10/conv/conv.0/conv.0.0/Conv: |                      | 0.044%
/features/features.13/conv/conv.1/conv.1.0/Conv: |                      | 0.036%
/features/features.7/conv/conv.1/conv.1.0/Conv:  |                      | 0.033%
/features/features.9/conv/conv.2/Conv:           |                      | 0.029%
/features/features.6/conv/conv.0/conv.0.0/Conv:  |                      | 0.029%
/features/features.8/conv/conv.0/conv.0.0/Conv:  |                      | 0.024%
/features/features.10/conv/conv.1/conv.1.0/Conv: |                      | 0.024%
/features/features.12/conv/conv.1/conv.1.0/Conv: |                      | 0.023%
/features/features.9/conv/conv.0/conv.0.0/Conv:  |                      | 0.019%
/features/features.17/conv/conv.1/conv.1.0/Conv: |                      | 0.019%
/features/features.9/conv/conv.1/conv.1.0/Conv:  |                      | 0.016%

* Prec@1 72.025 Prec@5 89.425
```

ESP32-P4 量化结果

实验结果表明：在 ESP32P4 上采用 PTQ 量化时，量化模型 Prec@1 为 72.025%，与浮点模型（71.878%）接近。

ESP32-S3 量化设置

```
# default: Replace ReLU6 with ReLU.

quant_setting = QuantizationSettingFactory.espdl_setting()
quant_setting.quantize_activation_setting.calib_algorithm = 'kl'

quant_setting.equalization = True
quant_setting.equalization_setting.iterations = 4
quant_setting.equalization_setting.value_threshold = 0.1
quant_setting.equalization_setting.opt_level = 2
quant_setting.equalization_setting.interested_layers = None

quant_setting.tqt_optimization = True
tqt_setting = quant_setting.tqt_optimization_setting
tqt_setting.lr = 1e-5
tqt_setting.steps = 1000
tqt_setting.block_size = 4
tqt_setting.is_scale_trainable = True
tqt_setting.gamma = 0.0
tqt_setting.int_lambda = 0.0
tqt_setting.collecting_device = 'cuda'
```

ESP32-S3 量化误差

```
Analysing Graphwise Quantization Error(Phrase 1):: 100
↪%|████████████████████████████████████████████████████████████████████████████████
↪8/8 [00:03<00:00,  2.34it/s]
Analysing Graphwise Quantization Error(Phrase 2):: 100
↪%|████████████████████████████████████████████████████████████████████████████████
↪8/8 [00:07<00:00,  1.01it/s]
Layer                                                | NOISE:SIGNAL POWER RATIO
/features/features.17/conv/conv.0/conv.0.0/Conv:     | ████████████████████ | 34.420%
/features/features.16/conv/conv.2/Conv:              | ██████████████       | 24.168%
/features/features.15/conv/conv.2/Conv:              | ██████████           | 16.545%
/features/features.14/conv/conv.2/Conv:              | ████████             | 13.160%
/features/features.1/conv/conv.1/Conv:               | ████████             | 12.979%
/features/features.17/conv/conv.2/Conv:              | ██████               | 11.054%
/features/features.18/features.18.0/Conv:            | ██████               | 10.546%
/features/features.13/conv/conv.2/Conv:              | ██████               | 10.353%
/features/features.16/conv/conv.0/conv.0.0/Conv:     | ██████               | 9.501%
/features/features.6/conv/conv.2/Conv:               | █████                | 9.038%
/features/features.12/conv/conv.2/Conv:              | █████                | 8.405%
/features/features.16/conv/conv.1/conv.1.0/Conv:     | █████                | 8.007%
/features/features.3/conv/conv.2/Conv:               | ████                 | 7.629%
/features/features.10/conv/conv.2/Conv:              | ████                 | 7.459%
/features/features.15/conv/conv.0/conv.0.0/Conv:     | ████                 | 7.418%
/features/features.5/conv/conv.2/Conv:               | ████                 | 6.852%
/features/features.15/conv/conv.1/conv.1.0/Conv:     | ████                 | 6.680%
/features/features.11/conv/conv.2/Conv:              | ████                 | 6.381%
/features/features.9/conv/conv.2/Conv:               | ███                  | 5.971%
/features/features.14/conv/conv.1/conv.1.0/Conv:     | ███                  | 5.716%
/features/features.14/conv/conv.0/conv.0.0/Conv:     | ███                  | 5.159%
/features/features.4/conv/conv.2/Conv:               | ███                  | 5.090%
/features/features.7/conv/conv.2/Conv:               | ███                  | 4.734%
/features/features.2/conv/conv.2/Conv:               | ██                   | 4.244%
/features/features.8/conv/conv.2/Conv:               | ██                   | 4.103%
/features/features.2/conv/conv.1/conv.1.0/Conv:      | ██                   | 4.070%
```

(续下页)

(接上页)

```
/features/features.12/conv/conv.1/conv.1.0/Conv: | ██                  | 3.780%
/features/features.13/conv/conv.1/conv.1.0/Conv: | ██                  | 3.755%
/features/features.5/conv/conv.1/conv.1.0/Conv:  | ██                  | 3.752%
/features/features.13/conv/conv.0/conv.0.0/Conv: | ██                  | 3.726%
/features/features.2/conv/conv.0/conv.0.0/Conv:  | ██                  | 3.653%
/features/features.6/conv/conv.1/conv.1.0/Conv:  | ██                  | 3.541%
/classifier/classifier.1/Gemm:                   | ██                  | 3.391%
/features/features.10/conv/conv.1/conv.1.0/Conv: | ██                  | 3.327%
/features/features.3/conv/conv.1/conv.1.0/Conv:  | ██                  | 3.197%
/features/features.11/conv/conv.1/conv.1.0/Conv: | ██                  | 3.168%
/features/features.8/conv/conv.1/conv.1.0/Conv:  | ██                  | 3.083%
/features/features.9/conv/conv.1/conv.1.0/Conv:  | ██                  | 3.063%
/features/features.7/conv/conv.0/conv.0.0/Conv:  | ██                  | 2.819%
/features/features.12/conv/conv.0/conv.0.0/Conv: | ██                  | 2.757%
/features/features.4/conv/conv.0/conv.0.0/Conv:  | ██                  | 2.656%
/features/features.11/conv/conv.0/conv.0.0/Conv: | █                   | 2.573%
/features/features.10/conv/conv.0/conv.0.0/Conv: | █                   | 2.570%
/features/features.17/conv/conv.1/conv.1.0/Conv: | █                   | 2.513%
/features/features.4/conv/conv.1/conv.1.0/Conv:  | █                   | 2.172%
/features/features.6/conv/conv.0/conv.0.0/Conv:  | █                   | 1.688%
/features/features.7/conv/conv.1/conv.1.0/Conv:  | █                   | 1.399%
/features/features.9/conv/conv.0/conv.0.0/Conv:  | █                   | 1.272%
/features/features.8/conv/conv.0/conv.0.0/Conv:  | █                   | 1.250%
/features/features.5/conv/conv.0/conv.0.0/Conv:  | █                   | 1.083%
/features/features.3/conv/conv.0/conv.0.0/Conv:  |                     | 0.813%
/features/features.1/conv/conv.0/conv.0.0/Conv:  |                     | 0.763%
/features/features.0/features.0.0/Conv:          |                     | 0.049%
Analysing Layerwise quantization error:: 100
↪%|████████████████████████████████████████████████████████████████████████████████████████████████████████████
↪53/53 [04:31<00:00,  5.11s/it]
Layer                                            | NOISE:SIGNAL POWER RATIO
/features/features.1/conv/conv.0/conv.0.0/Conv:  | ████████████████████ | 10.635%
/features/features.1/conv/conv.1/Conv:           | ████████████████     | 9.070%
/features/features.14/conv/conv.2/Conv:          | ███                  | 1.356%
/features/features.0/features.0.0/Conv:          | ██                   | 1.057%
/features/features.17/conv/conv.0/conv.0.0/Conv: | ██                   | 0.905%
/features/features.3/conv/conv.1/conv.1.0/Conv:  | █                    | 0.649%
/features/features.17/conv/conv.2/Conv:          | █                    | 0.631%
/features/features.2/conv/conv.2/Conv:           | █                    | 0.605%
/features/features.16/conv/conv.2/Conv:          | █                    | 0.572%
/features/features.3/conv/conv.2/Conv:           | █                    | 0.529%
/features/features.2/conv/conv.1/conv.1.0/Conv:  | █                    | 0.505%
/features/features.14/conv/conv.1/conv.1.0/Conv: | █                    | 0.285%
/features/features.2/conv/conv.0/conv.0.0/Conv:  | █                    | 0.284%
/features/features.4/conv/conv.2/Conv:           | █                    | 0.277%
/features/features.14/conv/conv.0/conv.0.0/Conv: |                      | 0.244%
/features/features.15/conv/conv.2/Conv:          |                      | 0.242%
/features/features.4/conv/conv.0/conv.0.0/Conv:  |                      | 0.235%
/features/features.16/conv/conv.0/conv.0.0/Conv: |                      | 0.147%
/features/features.7/conv/conv.2/Conv:           |                      | 0.133%
/features/features.11/conv/conv.2/Conv:          |                      | 0.126%
/features/features.7/conv/conv.0/conv.0.0/Conv:  |                      | 0.105%
/classifier/classifier.1/Gemm:                   |                      | 0.104%
/features/features.12/conv/conv.2/Conv:          |                      | 0.102%
/features/features.3/conv/conv.0/conv.0.0/Conv:  |                      | 0.100%
/features/features.16/conv/conv.1/conv.1.0/Conv: |                      | 0.099%
/features/features.15/conv/conv.1/conv.1.0/Conv: |                      | 0.091%
```

(续下页)

(接上页)

```
/features/features.15/conv/conv.0/conv.0.0/Conv: |                       | 0.090%
/features/features.4/conv/conv.1/conv.1.0/Conv:  |                       | 0.079%
/features/features.5/conv/conv.2/Conv:           |                       | 0.079%
/features/features.13/conv/conv.2/Conv:          |                       | 0.072%
/features/features.11/conv/conv.1/conv.1.0/Conv: |                       | 0.068%
/features/features.5/conv/conv.1/conv.1.0/Conv:  |                       | 0.064%
/features/features.11/conv/conv.0/conv.0.0/Conv: |                       | 0.062%
/features/features.13/conv/conv.0/conv.0.0/Conv: |                       | 0.061%
/features/features.6/conv/conv.2/Conv:           |                       | 0.060%
/features/features.10/conv/conv.2/Conv:          |                       | 0.056%
/features/features.6/conv/conv.1/conv.1.0/Conv:  |                       | 0.051%
/features/features.17/conv/conv.1/conv.1.0/Conv: |                       | 0.047%
/features/features.8/conv/conv.2/Conv:           |                       | 0.040%
/features/features.12/conv/conv.0/conv.0.0/Conv: |                       | 0.040%
/features/features.8/conv/conv.1/conv.1.0/Conv:  |                       | 0.039%
/features/features.13/conv/conv.1/conv.1.0/Conv: |                       | 0.038%
/features/features.12/conv/conv.1/conv.1.0/Conv: |                       | 0.034%
/features/features.5/conv/conv.0/conv.0.0/Conv:  |                       | 0.033%
/features/features.18/features.18.0/Conv:        |                       | 0.029%
/features/features.10/conv/conv.0/conv.0.0/Conv: |                       | 0.025%
/features/features.7/conv/conv.1/conv.1.0/Conv:  |                       | 0.023%
/features/features.6/conv/conv.0/conv.0.0/Conv:  |                       | 0.022%
/features/features.9/conv/conv.2/Conv:           |                       | 0.021%
/features/features.10/conv/conv.1/conv.1.0/Conv: |                       | 0.020%
/features/features.8/conv/conv.0/conv.0.0/Conv:  |                       | 0.017%
/features/features.9/conv/conv.0/conv.0.0/Conv:  |                       | 0.011%
/features/features.9/conv/conv.1/conv.1.0/Conv:  |                       | 0.007%

* Prec@1 70.543 Prec@5 89.525
```

ESP32-S3 量化结果

实验结果表明：在 ESP32-S3 上采用 PTQ 量化时，量化模型 Prec@1 为 70.543%，略高于在 ESP32-P4 上的量化精度。

备注： 如果想进一步降低量化误差，可以尝试使用 QAT (Auantization Aware Training)。具体方法请参考 PPQ QAT example。

3.7.3 模型部署

参考示例

图像分类基类

- dl_cls_base.hpp
- dl_cls_base.cpp

前处理

`ImagePreprocessor` 类中封装了常用的图像前处理流程，包括 `color conversion`, `crop`, `resize`, `normalization`, `quantize`。

- dl_image_preprocessor.hpp
- dl_image_preprocessor.cpp

后处理

- dl_cls_postprocessor.hpp
- dl_cls_postprocessor.cpp
- imagenet_cls_postprocessor.hpp
- imagenet_cls_postprocessor.cpp

3.8 如何部署 YOLO11n

在本教程中，我们介绍如何使用 ESP-PPQ 对预训练的 YOLO11n 模型进行量化，并使用 ESP-DL 部署量化后的 YOLO11n 模型。

- 准备工作
- 模型量化
  - 预训练模型
  - 校准数据集
  - *8bit* 后量化
- 模型部署
  - 目标检测基类
  - 前处理
  - 后处理

3.8.1 准备工作

1. *安装 ESP_IDF*
2. *安装 ESP_PPQ*

3.8.2 模型量化

预训练模型

你可以从 Ultralytics release 下载预训练的 yolo11n 模型。

目前 ESP-PPQ 支持 ONNX、PyTorch、TensorFlow 模型。在量化过程中，PyTorch 和 TensorFlow 会先转化为 ONNX 模型，因此将与训练的 yolo11n 转化成 ONNX 模型。

具体来说，参考脚本：export_onnx.py 将预训练的 yolo11n 模型转换为 ONNX 模型。

在该脚本中，我们重载了 Detect 类的 forward 方法，具有以下优势：

- 更快的推理速度。与原始的 yolo11n 模型相比, 将推理过程中 Detect 里与解码边界框相关的操作移至后处理中完成, 从而显著减少了推理延迟。一方面，`Conv`，`Transpose`，`Slice`，`Split` 和 `Concat` 操作在推理过程中运行是非常耗时的。另一方面，在后处理阶段，模型推理的输出首先进行置信度筛选，然后再解码边界框，这大大减少了计算量，从而加快了整体推理速度。
- 更低的量化误差。ESP-PPQ 中的 `Concat` 和 `Add` 操作采用了联合量化。为了减少量化误差，由于 box 和 score 的范围差异较大，它们通过不同的分支输出，而不是拼接在一起。类似地，由于 `Add` 和 `Sub` 的输入的范围差异较大，相关计算被移到了后处理中进行，避免被量化。

校准数据集

校准数据集需要和模型输入格式一致，同时尽可能覆盖模型输入的所有可能情况，以便更好地量化模型。本示例中，我们使用的校准集为 calib_yolo11n 。

8bit 后量化

下面的量化设置通过 AutoQuant 搜索得到。要使用 AutoQuant，请更新 esp-ppq 为最新版本并参考 教程。

ESP32-P4 量化设置

```
quant_setting = QuantizationSettingFactory.espdl_setting()
quant_setting.quantize_activation_setting.calib_algorithm = 'percentile'

quant_setting.bias_correct = True
quant_setting.bias_correct_setting.interested_layers = []
quant_setting.bias_correct_setting.block_size = 4
quant_setting.bias_correct_setting.steps = 32

quant_setting.tqt_optimization = True
tqt_setting = quant_setting.tqt_optimization_setting
tqt_setting.lr = 1e-5
tqt_setting.steps = 500
tqt_setting.block_size = 4
tqt_setting.is_scale_trainable = True
tqt_setting.gamma = 0.0
tqt_setting.int_lambda = 0.0
tqt_setting.collecting_device = 'cuda'
```

ESP32-P4 量化误差

```
Analysing Graphwise Quantization Error(Phrase 2):: 100
↪%|
↪1/1 [00:00<00:00,  4.59it/s]
```

(续下页)

(接上页)

Layer	NOISE:SIGNAL POWER RATIO	
/model.3/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉	6.440%
/model.2/cv2/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉	6.035%
/model.2/m.0/cv2/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉	5.410%
/model.6/m.0/cv2/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉	5.329%
/model.10/m/m.0/attn/proj/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉▉▉▉▉	5.175%
/model.23/cv2.0/cv2.0.1/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉▉	4.425%
/model.4/m.0/cv1/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉▉	4.371%
/model.10/m/m.0/ffn/ffn.1/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉	4.270%
/model.4/cv2/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉	4.266%
/model.6/cv1/conv/Conv:	▉▉▉▉▉▉▉▉▉▉▉	4.050%
/model.8/m.0/cv2/conv/Conv:	▉▉▉▉▉▉▉▉▉▉	3.640%
/model.6/m.0/cv3/conv/Conv:	▉▉▉▉▉▉▉▉▉▉	3.530%
/model.5/conv/Conv:	▉▉▉▉▉▉▉▉▉▉	3.420%
/model.4/cv1/conv/Conv:	▉▉▉▉▉▉▉▉▉	3.341%
/model.6/cv2/conv/Conv:	▉▉▉▉▉▉▉▉▉	3.242%
/model.6/m.0/m/m.0/cv1/conv/Conv:	▉▉▉▉▉▉▉▉▉	3.111%
/model.23/cv2.1/cv2.1.1/conv/Conv:	▉▉▉▉▉▉▉▉	3.045%
/model.2/cv1/conv/Conv:	▉▉▉▉▉▉▉▉	2.928%
/model.23/cv3.0/cv3.0.1/cv3.0.1.1/conv/Conv:	▉▉▉▉▉▉▉	2.712%
/model.8/cv1/conv/Conv:	▉▉▉▉▉▉▉	2.700%
/model.2/m.0/cv1/conv/Conv:	▉▉▉▉▉▉▉	2.677%
/model.1/conv/Conv:	▉▉▉▉▉▉▉	2.523%
/model.23/cv2.1/cv2.1.0/conv/Conv:	▉▉▉▉▉▉▉	2.514%
/model.6/m.0/m/m.0/cv2/conv/Conv:	▉▉▉▉▉▉▉	2.474%
/model.23/cv2.0/cv2.0.0/conv/Conv:	▉▉▉▉▉▉	2.404%
/model.6/m.0/m/m.1/cv1/conv/Conv:	▉▉▉▉▉▉	2.325%
/model.10/m/m.0/attn/pe/conv/Conv:	▉▉▉▉▉▉	2.323%
/model.7/conv/Conv:	▉▉▉▉▉▉	2.322%
/model.23/cv3.1/cv3.1.1/cv3.1.1.1/conv/Conv:	▉▉▉▉▉▉	2.317%
/model.6/m.0/cv1/conv/Conv:	▉▉▉▉▉▉	2.279%
/model.4/m.0/cv2/conv/Conv:	▉▉▉▉▉▉	2.273%
/model.19/m.0/cv2/conv/Conv:	▉▉▉▉▉▉	2.273%
/model.16/m.0/cv2/conv/Conv:	▉▉▉▉▉▉	2.259%
/model.16/m.0/cv1/conv/Conv:	▉▉▉▉▉▉	2.122%
/model.8/m.0/cv1/conv/Conv:	▉▉▉▉▉	2.061%
/model.23/cv3.0/cv3.0.0/cv3.0.0.1/conv/Conv:	▉▉▉▉▉	2.031%
/model.13/m.0/cv2/conv/Conv:	▉▉▉▉▉	1.982%
/model.10/m/m.0/attn/qkv/conv/Conv:	▉▉▉▉▉	1.969%
/model.10/cv1/conv/Conv:	▉▉▉▉▉	1.939%
/model.19/cv2/conv/Conv:	▉▉▉▉▉	1.879%
/model.8/m.0/m/m.0/cv1/conv/Conv:	▉▉▉▉▉	1.868%
/model.23/cv3.2/cv3.2.1/cv3.2.1.1/conv/Conv:	▉▉▉▉▉	1.861%
/model.8/cv2/conv/Conv:	▉▉▉▉▉	1.835%
/model.17/conv/Conv:	▉▉▉▉▉	1.819%
/model.19/cv1/conv/Conv:	▉▉▉▉	1.773%
/model.16/cv2/conv/Conv:	▉▉▉▉	1.750%
/model.23/cv3.1/cv3.1.0/cv3.1.0.1/conv/Conv:	▉▉▉▉	1.740%
/model.22/m.0/cv2/conv/Conv:	▉▉▉▉	1.730%
/model.19/m.0/cv1/conv/Conv:	▉▉▉▉	1.706%
/model.23/cv2.0/cv2.0.2/Conv:	▉▉▉▉	1.661%
/model.13/cv1/conv/Conv:	▉▉▉▉	1.635%
/model.20/conv/Conv:	▉▉▉▉	1.635%
/model.13/m.0/cv1/conv/Conv:	▉▉▉▉	1.613%
/model.13/cv2/conv/Conv:	▉▉▉▉	1.594%
/model.23/cv2.2/cv2.2.0/conv/Conv:	▉▉▉▉	1.560%
/model.23/cv3.2/cv3.2.0/cv3.2.0.1/conv/Conv:	▉▉▉▉	1.538%

(续下页)

(接上页)

```
/model.6/m.0/m/m.1/cv2/conv/Conv:             | ████▌                | 1.513%
/model.23/cv3.2/cv3.2.0/cv3.2.0.0/conv/Conv:  | ████▌                | 1.511%
/model.22/cv1/conv/Conv:                      | ████                 | 1.456%
/model.8/m.0/m/m.0/cv2/conv/Conv:             | ████                 | 1.453%
/model.22/cv2/conv/Conv:                      | ████                 | 1.384%
/model.16/cv1/conv/Conv:                      | ████                 | 1.345%
/model.8/m.0/m/m.1/cv1/conv/Conv:             | ████                 | 1.336%
/model.8/m.0/cv3/conv/Conv:                   | ████                 | 1.329%
/model.23/cv2.2/cv2.2.1/conv/Conv:            | ████                 | 1.288%
/model.10/m/m.0/attn/MatMul:                  | ████                 | 1.260%
/model.22/m.0/cv3/conv/Conv:                  | ███                  | 1.117%
/model.10/cv2/conv/Conv:                      | ███                  | 1.094%
/model.23/cv2.1/cv2.1.2/Conv:                 | ███                  | 1.045%
/model.22/m.0/m/m.0/cv1/conv/Conv:            | ███                  | 1.025%
/model.22/m.0/m/m.0/cv2/conv/Conv:            | ███                  | 0.999%
/model.10/m/m.0/attn/MatMul_1:                | ███                  | 0.984%
/model.22/m.0/m/m.1/cv2/conv/Conv:            | ███                  | 0.889%
/model.23/cv3.1/cv3.1.0/cv3.1.0.0/conv/Conv:  | ███                  | 0.864%
/model.8/m.0/m/m.1/cv2/conv/Conv:             | ███                  | 0.839%
/model.9/cv2/conv/Conv:                       | ██                   | 0.800%
/model.23/cv3.2/cv3.2.1/cv3.2.1.0/conv/Conv:  | ██                   | 0.761%
/model.23/cv3.1/cv3.1.1/cv3.1.1.0/conv/Conv:  | ██                   | 0.759%
/model.22/m.0/cv1/conv/Conv:                  | ██                   | 0.750%
/model.22/m.0/m/m.1/cv1/conv/Conv:            | ██                   | 0.705%
/model.23/cv3.0/cv3.0.1/cv3.0.1.0/conv/Conv:  | ██                   | 0.677%
/model.10/m/m.0/ffn/ffn.0/conv/Conv:          | ██                   | 0.669%
/model.23/cv2.2/cv2.2.2/Conv:                 | ▌                    | 0.434%
/model.9/cv1/conv/Conv:                       | ▌                    | 0.390%
/model.23/cv3.0/cv3.0.0/cv3.0.0.0/conv/Conv:  | ▌                    | 0.341%
/model.0/conv/Conv:                           |                      | 0.157%
/model.23/cv3.2/cv3.2.2/Conv:                 |                      | 0.079%
/model.23/cv3.1/cv3.1.2/Conv:                 |                      | 0.068%
/model.23/cv3.0/cv3.0.2/Conv:                 |                      | 0.038%
Analysing Layerwise quantization error:: 100
↪%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████████
↪89/89 [00:12<00:00,  7.30it/s]
Layer                                         | NOISE:SIGNAL POWER RATIO
/model.1/conv/Conv:                           | ████████████████████ | 0.712%
/model.0/conv/Conv:                           | ██████████████       | 0.521%
/model.3/conv/Conv:                           | ██████████           | 0.369%
/model.2/cv2/conv/Conv:                       | ████▌                | 0.181%
/model.4/m.0/cv1/conv/Conv:                   | ███                  | 0.120%
/model.2/cv1/conv/Conv:                       | ███                  | 0.101%
/model.4/m.0/cv2/conv/Conv:                   | ███                  | 0.095%
/model.2/m.0/cv1/conv/Conv:                   | ██                   | 0.071%
/model.4/cv1/conv/Conv:                       | ██                   | 0.069%
/model.5/conv/Conv:                           | ▌                    | 0.052%
/model.4/cv2/conv/Conv:                       | ▌                    | 0.045%
/model.7/conv/Conv:                           | ▌                    | 0.036%
/model.10/cv2/conv/Conv:                      | ▌                    | 0.029%
/model.9/cv2/conv/Conv:                       | ▌                    | 0.029%
/model.6/cv1/conv/Conv:                       | ▌                    | 0.027%
/model.2/m.0/cv2/conv/Conv:                   | ▌                    | 0.026%
/model.13/cv1/conv/Conv:                      | ▌                    | 0.024%
/model.6/cv2/conv/Conv:                       | ▌                    | 0.022%
/model.10/cv1/conv/Conv:                      | ▌                    | 0.021%
/model.10/m/m.0/attn/pe/conv/Conv:            |                      | 0.017%
```

(续下页)

(接上页)

```
/model.8/cv2/conv/Conv:                          |                          | 0.017%
/model.16/cv1/conv/Conv:                         |                          | 0.016%
/model.13/cv2/conv/Conv:                         |                          | 0.015%
/model.23/cv2.2/cv2.2.0/conv/Conv:               |                          | 0.015%
/model.19/m.0/cv1/conv/Conv:                     |                          | 0.012%
/model.23/cv2.0/cv2.0.0/conv/Conv:               |                          | 0.012%
/model.9/cv1/conv/Conv:                          |                          | 0.011%
/model.16/cv2/conv/Conv:                         |                          | 0.011%
/model.13/m.0/cv1/conv/Conv:                     |                          | 0.010%
/model.19/cv2/conv/Conv:                         |                          | 0.010%
/model.23/cv2.1/cv2.1.0/conv/Conv:               |                          | 0.010%
/model.22/m.0/m/m.1/cv1/conv/Conv:               |                          | 0.009%
/model.10/m/m.0/attn/proj/conv/Conv:             |                          | 0.009%
/model.10/m/m.0/ffn/ffn.0/conv/Conv:             |                          | 0.008%
/model.23/cv2.2/cv2.2.1/conv/Conv:               |                          | 0.008%
/model.8/cv1/conv/Conv:                          |                          | 0.008%
/model.22/m.0/cv1/conv/Conv:                     |                          | 0.007%
/model.19/m.0/cv2/conv/Conv:                     |                          | 0.007%
/model.19/cv1/conv/Conv:                         |                          | 0.007%
/model.16/m.0/cv1/conv/Conv:                     |                          | 0.006%
/model.16/m.0/cv2/conv/Conv:                     |                          | 0.006%
/model.10/m/m.0/attn/qkv/conv/Conv:              |                          | 0.006%
/model.13/m.0/cv2/conv/Conv:                     |                          | 0.006%
/model.22/cv1/conv/Conv:                         |                          | 0.006%
/model.23/cv2.2/cv2.2.2/Conv:                    |                          | 0.006%
/model.23/cv2.0/cv2.0.2/Conv:                    |                          | 0.005%
/model.22/m.0/m/m.0/cv2/conv/Conv:               |                          | 0.005%
/model.8/m.0/cv1/conv/Conv:                      |                          | 0.005%
/model.23/cv2.1/cv2.1.2/Conv:                    |                          | 0.005%
/model.6/m.0/m/m.0/cv1/conv/Conv:                |                          | 0.005%
/model.6/m.0/m/m.1/cv1/conv/Conv:                |                          | 0.005%
/model.8/m.0/m/m.1/cv1/conv/Conv:                |                          | 0.004%
/model.8/m.0/m/m.0/cv1/conv/Conv:                |                          | 0.004%
/model.6/m.0/m/m.0/cv2/conv/Conv:                |                          | 0.004%
/model.20/conv/Conv:                             |                          | 0.004%
/model.10/m/m.0/ffn/ffn.1/conv/Conv:             |                          | 0.004%
/model.22/cv2/conv/Conv:                         |                          | 0.004%
/model.23/cv2.1/cv2.1.1/conv/Conv:               |                          | 0.004%
/model.23/cv2.0/cv2.0.1/conv/Conv:               |                          | 0.004%
/model.17/conv/Conv:                             |                          | 0.004%
/model.8/m.0/m/m.0/cv2/conv/Conv:                |                          | 0.004%
/model.6/m.0/cv3/conv/Conv:                      |                          | 0.003%
/model.6/m.0/cv1/conv/Conv:                      |                          | 0.003%
/model.23/cv3.1/cv3.1.1/cv3.1.1.1/conv/Conv:     |                          | 0.003%
/model.22/m.0/m/m.0/cv1/conv/Conv:               |                          | 0.003%
/model.6/m.0/m/m.1/cv2/conv/Conv:                |                          | 0.003%
/model.22/m.0/m/m.1/cv2/conv/Conv:               |                          | 0.002%
/model.22/m.0/cv3/conv/Conv:                     |                          | 0.002%
/model.23/cv3.2/cv3.2.0/cv3.2.0.1/conv/Conv:     |                          | 0.002%
/model.8/m.0/m/m.1/cv2/conv/Conv:                |                          | 0.002%
/model.8/m.0/cv3/conv/Conv:                      |                          | 0.002%
/model.10/m/m.0/attn/MatMul_1:                   |                          | 0.002%
/model.23/cv3.2/cv3.2.1/cv3.2.1.0/conv/Conv:     |                          | 0.001%
/model.23/cv3.1/cv3.1.1/cv3.1.1.0/conv/Conv:     |                          | 0.001%
/model.23/cv3.1/cv3.1.2/Conv:                    |                          | 0.001%
/model.23/cv3.0/cv3.0.1/cv3.0.1.0/conv/Conv:     |                          | 0.001%
/model.23/cv3.2/cv3.2.2/Conv:                    |                          | 0.001%
```

(续下页)

(接上页)

```
/model.23/cv3.0/cv3.0.0/cv3.0.0.1/conv/Conv: |                                   | 0.001%
/model.23/cv3.2/cv3.2.1/cv3.2.1.1/conv/Conv: |                                   | 0.001%
/model.23/cv3.2/cv3.2.0/cv3.2.0.0/conv/Conv: |                                   | 0.001%
/model.23/cv3.0/cv3.0.2/Conv:                |                                   | 0.001%
/model.23/cv3.1/cv3.1.0/cv3.1.0.1/conv/Conv: |                                   | 0.000%
/model.10/m/m.0/attn/MatMul:                 |                                   | 0.000%
/model.23/cv3.1/cv3.1.0/cv3.1.0.0/conv/Conv: |                                   | 0.000%
/model.8/m.0/cv2/conv/Conv:                  |                                   | 0.000%
/model.23/cv3.0/cv3.0.0/cv3.0.0.0/conv/Conv: |                                   | 0.000%
/model.6/m.0/cv2/conv/Conv:                  |                                   | 0.000%
/model.23/cv3.0/cv3.0.1/cv3.0.1.1/conv/Conv: |                                   | 0.000%
/model.22/m.0/cv2/conv/Conv:                 |                                   | 0.000%
```

ESP32-P4 量化结果

在相同输入下，量化后的模型在 COCO val2017 上的 mAP50-95 为 0.373，略低于浮点模型精度。

ESP32-S3 量化设置

```
quant_setting = QuantizationSettingFactory.espdl_setting()
quant_setting.quantize_activation_setting.calib_algorithm = 'percentile'

quant_setting.tqt_optimization = True
tqt_setting = quant_setting.tqt_optimization_setting
tqt_setting.lr = 1e-5
tqt_setting.steps = 800
tqt_setting.block_size = 4
tqt_setting.is_scale_trainable = True
tqt_setting.gamma = 0.0
tqt_setting.int_lambda = 0.0
tqt_setting.collecting_device = 'cuda'
```

ESP32-S3 量化误差

```
Analysing Graphwise Quantization Error(Phrase 2):: 100
↪%|████████████████████████████████████████████████████████████████████████████████████████████████████████████
↪1/1 [00:00<00:00,  4.25it/s]
Layer                                        | NOISE:SIGNAL POWER RATIO
/model.3/conv/Conv:                          | ████████████████████ | 7.039%
/model.2/cv2/conv/Conv:                      | ████████████████     | 5.965%
/model.10/m/m.0/attn/proj/conv/Conv:         | ████████████████     | 5.887%
/model.2/m.0/cv2/conv/Conv:                  | ███████████████      | 5.726%
/model.6/m.0/cv2/conv/Conv:                  | ███████████████      | 5.483%
/model.4/m.0/cv1/conv/Conv:                  | ████████████         | 4.815%
/model.23/cv2.0/cv2.0.1/conv/Conv:           | ████████████         | 4.815%
/model.10/m/m.0/ffn/ffn.1/conv/Conv:         | ███████████          | 4.569%
/model.4/cv2/conv/Conv:                      | ██████████           | 4.377%
/model.6/cv1/conv/Conv:                      | ██████████           | 4.048%
/model.6/m.0/cv3/conv/Conv:                  | ██████████           | 3.755%
/model.8/m.0/cv2/conv/Conv:                  | █████████            | 3.517%
/model.4/cv1/conv/Conv:                      | █████████            | 3.507%
/model.5/conv/Conv:                          | █████████            | 3.495%
/model.23/cv3.0/cv3.0.1/cv3.0.1.1/conv/Conv: | █████████            | 3.472%
/model.23/cv3.1/cv3.1.1/cv3.1.1.1/conv/Conv: | █████████            | 3.460%
/model.6/cv2/conv/Conv:                      | █████████            | 3.369%
/model.23/cv2.1/cv2.1.1/conv/Conv:           | ████████             | 3.262%
/model.6/m.0/m/m.0/cv1/conv/Conv:            | ████████             | 3.243%
```

(续下页)

(接上页)

```
/model.2/cv1/conv/Conv:                          | ███████ | 2.904%
/model.23/cv2.1/cv2.1.0/conv/Conv:               | ███████ | 2.770%
/model.8/cv1/conv/Conv:                          | ███████ | 2.730%
/model.23/cv3.0/cv3.0.0/cv3.0.0.1/conv/Conv:     | ███████ | 2.706%
/model.10/m/m.0/attn/pe/conv/Conv:               | ███████ | 2.671%
/model.23/cv2.0/cv2.0.0/conv/Conv:               | ██████  | 2.626%
/model.6/m.0/m/m.0/cv2/conv/Conv:                | ██████  | 2.595%
/model.7/conv/Conv:                              | ██████  | 2.547%
/model.1/conv/Conv:                              | ██████  | 2.519%
/model.19/m.0/cv2/conv/Conv:                     | ██████  | 2.476%
/model.6/m.0/m/m.1/cv1/conv/Conv:                | ██████  | 2.471%
/model.4/m.0/cv2/conv/Conv:                      | ██████  | 2.422%
/model.16/m.0/cv2/conv/Conv:                     | ██████  | 2.412%
/model.6/m.0/cv1/conv/Conv:                      | ██████  | 2.371%
/model.16/m.0/cv1/conv/Conv:                     | ██████  | 2.334%
/model.10/m/m.0/attn/qkv/conv/Conv:              | ██████  | 2.159%
/model.23/cv3.1/cv3.1.0/cv3.1.0.1/conv/Conv:     | ██████  | 2.150%
/model.10/cv1/conv/Conv:                         | ██████  | 2.135%
/model.23/cv2.2/cv2.2.0/conv/Conv:               | ██████  | 2.123%
/model.13/m.0/cv2/conv/Conv:                     | ██████  | 2.073%
/model.22/m.0/cv2/conv/Conv:                     | ██████  | 2.059%
/model.8/m.0/cv1/conv/Conv:                      | ██████  | 2.053%
/model.19/cv2/conv/Conv:                         | ██████  | 2.049%
/model.23/cv3.2/cv3.2.1/cv3.2.1.1/conv/Conv:     | ██████  | 2.041%
/model.13/cv2/conv/Conv:                         | ██████  | 2.027%
/model.17/conv/Conv:                             | ██████  | 2.001%
/model.16/cv2/conv/Conv:                         | █████   | 1.968%
/model.8/m.0/m/m.0/cv1/conv/Conv:                | █████   | 1.916%
/model.8/cv2/conv/Conv:                          | █████   | 1.888%
/model.23/cv2.0/cv2.0.2/Conv:                    | █████   | 1.870%
/model.19/m.0/cv1/conv/Conv:                     | █████   | 1.847%
/model.19/cv1/conv/Conv:                         | █████   | 1.846%
/model.22/cv1/conv/Conv:                         | █████   | 1.725%
/model.23/cv3.2/cv3.2.0/cv3.2.0.1/conv/Conv:     | █████   | 1.716%
/model.20/conv/Conv:                             | █████   | 1.710%
/model.23/cv3.2/cv3.2.0/cv3.2.0.0/conv/Conv:     | █████   | 1.681%
/model.13/cv1/conv/Conv:                         | █████   | 1.669%
/model.2/m.0/cv1/conv/Conv:                      | █████   | 1.646%
/model.13/m.0/cv1/conv/Conv:                     | █████   | 1.641%
/model.6/m.0/m/m.1/cv2/conv/Conv:                | ██████  | 1.639%
/model.22/cv2/conv/Conv:                         | ████    | 1.524%
/model.8/m.0/m/m.0/cv2/conv/Conv:                | ████    | 1.489%
/model.16/cv1/conv/Conv:                         | ████    | 1.457%
/model.8/m.0/cv3/conv/Conv:                      | ████    | 1.422%
/model.10/m/m.0/attn/MatMul:                     | ████    | 1.387%
/model.8/m.0/m/m.1/cv1/conv/Conv:                | ████    | 1.369%
/model.23/cv2.2/cv2.2.1/conv/Conv:               | ████    | 1.275%
/model.22/m.0/cv3/conv/Conv:                     | ███     | 1.259%
/model.10/m/m.0/attn/MatMul_1:                   | ███     | 1.195%
/model.10/cv2/conv/Conv:                         | ███     | 1.190%
/model.23/cv2.1/cv2.1.2/Conv:                    | ███     | 1.143%
/model.9/cv2/conv/Conv:                          | ███     | 1.026%
/model.22/m.0/m/m.1/cv2/conv/Conv:               | ███     | 0.995%
/model.22/m.0/m/m.0/cv2/conv/Conv:               | ███     | 0.994%
/model.22/m.0/m/m.0/cv1/conv/Conv:               | ███     | 0.990%
/model.23/cv3.1/cv3.1.0/cv3.1.0.0/conv/Conv:     | ███     | 0.977%
/model.8/m.0/m/m.1/cv2/conv/Conv:                | ███     | 0.966%
```

(续下页)

(接上页)

```
/model.23/cv3.1/cv3.1.1/cv3.1.1.0/conv/Conv: | ███                  | 0.964%
/model.23/cv3.0/cv3.0.1/cv3.0.1.0/conv/Conv: | ██                   | 0.910%
/model.22/m.0/m/m.1/cv1/conv/Conv:           | ██                   | 0.768%
/model.10/m/m.0/ffn/ffn.0/conv/Conv:         | ██                   | 0.763%
/model.23/cv3.2/cv3.2.1/cv3.2.1.0/conv/Conv: | ██                   | 0.752%
/model.22/m.0/cv1/conv/Conv:                 | ██                   | 0.725%
/model.23/cv3.0/cv3.0.0/cv3.0.0.0/conv/Conv: | ▉                    | 0.564%
/model.9/cv1/conv/Conv:                      | ▉                    | 0.381%
/model.23/cv2.2/cv2.2.2/Conv:                | ▉                    | 0.363%
/model.0/conv/Conv:                          |                      | 0.172%
/model.23/cv3.1/cv3.1.2/Conv:                |                      | 0.083%
/model.23/cv3.2/cv3.2.2/Conv:                |                      | 0.079%
/model.23/cv3.0/cv3.0.2/Conv:                |                      | 0.047%
Analysing Layerwise quantization error:: 100
↪%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████
↪89/89 [00:12<00:00,  7.40it/s]
Layer                                        | NOISE:SIGNAL POWER RATIO
/model.0/conv/Conv:                          | ████████████████████ | 0.451%
/model.1/conv/Conv:                          | ████████████████████ | 0.450%
/model.3/conv/Conv:                          | ██████████████████   | 0.402%
/model.2/cv2/conv/Conv:                      | ██████████████       | 0.312%
/model.9/cv2/conv/Conv:                      | ██████████           | 0.222%
/model.4/cv2/conv/Conv:                      | ████████             | 0.178%
/model.10/cv1/conv/Conv:                     | ███████              | 0.155%
/model.2/cv1/conv/Conv:                      | ██████               | 0.145%
/model.4/m.0/cv1/conv/Conv:                  | ██████               | 0.139%
/model.4/cv1/conv/Conv:                      | ██████               | 0.137%
/model.22/cv1/conv/Conv:                     | █████                | 0.119%
/model.5/conv/Conv:                          | █████                | 0.118%
/model.16/cv1/conv/Conv:                     | █████                | 0.118%
/model.4/m.0/cv2/conv/Conv:                  | █████                | 0.117%
/model.22/m.0/cv1/conv/Conv:                 | █████                | 0.104%
/model.2/m.0/cv1/conv/Conv:                  | ███                  | 0.068%
/model.7/conv/Conv:                          | ███                  | 0.057%
/model.13/cv2/conv/Conv:                     | ██                   | 0.053%
/model.13/cv1/conv/Conv:                     | ██                   | 0.047%
/model.16/cv2/conv/Conv:                     | ██                   | 0.044%
/model.10/cv2/conv/Conv:                     | ██                   | 0.043%
/model.6/cv2/conv/Conv:                      | ██                   | 0.042%
/model.19/cv2/conv/Conv:                     | ██                   | 0.041%
/model.6/cv1/conv/Conv:                      | ██                   | 0.039%
/model.6/m.0/m/m.1/cv1/conv/Conv:            | ▉                    | 0.030%
/model.2/m.0/cv2/conv/Conv:                  | ▉                    | 0.029%
/model.10/m/m.0/attn/qkv/conv/Conv:          | ▉                    | 0.029%
/model.10/m/m.0/attn/pe/conv/Conv:           | ▉                    | 0.027%
/model.8/cv2/conv/Conv:                      | ▉                    | 0.027%
/model.19/cv1/conv/Conv:                     | ▉                    | 0.026%
/model.19/m.0/cv1/conv/Conv:                 | ▉                    | 0.025%
/model.22/m.0/m/m.1/cv1/conv/Conv:           | ▉                    | 0.025%
/model.10/m/m.0/ffn/ffn.0/conv/Conv:         | ▉                    | 0.024%
/model.23/cv2.1/cv2.1.0/conv/Conv:           | ▉                    | 0.023%
/model.16/m.0/cv1/conv/Conv:                 | ▉                    | 0.021%
/model.23/cv2.0/cv2.0.0/conv/Conv:           | ▉                    | 0.021%
/model.13/m.0/cv2/conv/Conv:                 | ▉                    | 0.020%
/model.9/cv1/conv/Conv:                      | ▉                    | 0.020%
/model.23/cv2.2/cv2.2.0/conv/Conv:           | ▉                    | 0.019%
/model.10/m/m.0/attn/proj/conv/Conv:         | ▉                    | 0.018%
```

(续下页)

(接上页)

```
/model.8/cv1/conv/Conv:                          | ▉                    | 0.017%
/model.16/m.0/cv2/conv/Conv:                     | ▉                    | 0.017%
/model.22/cv2/conv/Conv:                         | ▉                    | 0.015%
/model.13/m.0/cv1/conv/Conv:                     | ▉                    | 0.014%
/model.6/m.0/cv3/conv/Conv:                      | ▉                    | 0.013%
/model.20/conv/Conv:                             | ▉                    | 0.012%
/model.22/m.0/cv3/conv/Conv:                     |                      | 0.011%
/model.23/cv2.2/cv2.2.1/conv/Conv:               |                      | 0.011%
/model.17/conv/Conv:                             |                      | 0.011%
/model.19/m.0/cv2/conv/Conv:                     |                      | 0.010%
/model.23/cv2.2/cv2.2.2/Conv:                    |                      | 0.010%
/model.23/cv2.1/cv2.1.2/Conv:                    |                      | 0.009%
/model.8/m.0/cv3/conv/Conv:                      |                      | 0.009%
/model.8/m.0/cv1/conv/Conv:                      |                      | 0.009%
/model.23/cv2.0/cv2.0.2/Conv:                    |                      | 0.009%
/model.22/m.0/m/m.0/cv2/conv/Conv:               |                      | 0.008%
/model.10/m/m.0/ffn/ffn.1/conv/Conv:             |                      | 0.008%
/model.6/m.0/m/m.0/cv2/conv/Conv:                |                      | 0.008%
/model.23/cv2.1/cv2.1.1/conv/Conv:               |                      | 0.007%
/model.8/m.0/m/m.0/cv1/conv/Conv:                |                      | 0.007%
/model.6/m.0/m/m.0/cv1/conv/Conv:                |                      | 0.006%
/model.8/m.0/m/m.1/cv1/conv/Conv:                |                      | 0.006%
/model.23/cv2.0/cv2.0.1/conv/Conv:               |                      | 0.006%
/model.8/m.0/m/m.0/cv2/conv/Conv:                |                      | 0.006%
/model.22/m.0/m/m.0/cv1/conv/Conv:               |                      | 0.006%
/model.23/cv3.1/cv3.1.1/cv3.1.1.0/conv/Conv:     |                      | 0.005%
/model.23/cv3.1/cv3.1.2/Conv:                    |                      | 0.004%
/model.6/m.0/m/m.1/cv2/conv/Conv:                |                      | 0.004%
/model.6/m.0/cv1/conv/Conv:                      |                      | 0.004%
/model.23/cv3.0/cv3.0.0/cv3.0.0.1/conv/Conv:     |                      | 0.004%
/model.23/cv3.2/cv3.2.0/cv3.2.0.1/conv/Conv:     |                      | 0.004%
/model.23/cv3.1/cv3.1.0/cv3.1.0.1/conv/Conv:     |                      | 0.004%
/model.8/m.0/m/m.1/cv2/conv/Conv:                |                      | 0.003%
/model.23/cv3.1/cv3.1.1/cv3.1.1.1/conv/Conv:     |                      | 0.003%
/model.22/m.0/m/m.1/cv2/conv/Conv:               |                      | 0.003%
/model.23/cv3.2/cv3.2.1/cv3.2.1.0/conv/Conv:     |                      | 0.003%
/model.23/cv3.0/cv3.0.0/cv3.0.0.0/conv/Conv:     |                      | 0.002%
/model.10/m/m.0/attn/MatMul_1:                   |                      | 0.002%
/model.23/cv3.2/cv3.2.1/cv3.2.1.1/conv/Conv:     |                      | 0.001%
/model.23/cv3.2/cv3.2.2/Conv:                    |                      | 0.001%
/model.23/cv3.0/cv3.0.1/cv3.0.1.0/conv/Conv:     |                      | 0.001%
/model.23/cv3.0/cv3.0.2/Conv:                    |                      | 0.001%
/model.23/cv3.0/cv3.0.1/cv3.0.1.1/conv/Conv:     |                      | 0.001%
/model.23/cv3.2/cv3.2.0/cv3.2.0.0/conv/Conv:     |                      | 0.001%
/model.6/m.0/cv2/conv/Conv:                      |                      | 0.001%
/model.22/m.0/cv2/conv/Conv:                     |                      | 0.000%
/model.8/m.0/cv2/conv/Conv:                      |                      | 0.000%
/model.23/cv3.1/cv3.1.0/cv3.1.0.0/conv/Conv:     |                      | 0.000%
/model.10/m/m.0/attn/MatMul:                     |                      | 0.000%
```

ESP32-S3 量化结果

在相同输入下, 量化后模型在 COCO val2017 上的 mAP50-95 为 0.37, 略低于 ESP32-P4 的量化结果。

备注: 如果想要更快的模型推理速度，并且可以接受一定程度的精度损失，可以考虑在量化 YOLO11N 的时候将输入大小设置为 320x320。不同分辨率下的模型推理速度可以在 README.md 中找到。

3.8.3 模型部署

参考示例

目标检测基类

- dl_detect_base.hpp
- dl_detect_base.cpp

前处理

`ImagePreprocessor` 类中封装了常用的图像前处理流程，包括 `color conversion`, `crop`, `resize`, `normalization`, `quantize`。

- dl_image_preprocessor.hpp
- dl_image_preprocessor.cpp

后处理

- dl_detect_postprocessor.hpp
- dl_detect_postprocessor.cpp
- dl_detect_yolo11_postprocessor.hpp
- dl_detect_yolo11_postprocessor.cpp

3.9 如何部署 YOLO11n-pose

在本教程中，我们介绍如何使用 ESP-PPQ 对预训练的 YOLO11n-pose 模型进行量化，并使用 ESP-DL 部署量化后的 YOLO11n-pose 模型。

- 准备工作
- 模型量化
  - 预训练模型
  - 校准数据集
  - *8bit* 后量化
  - 量化感知训练
- 模型部署
  - 目标检测基类
  - 前处理
  - 后处理

3.9.1 准备工作

1. *安装 ESP_IDF*
2. *安装 ESP_PPQ*

3.9.2 模型量化

预训练模型

你可以从 Ultralytics release 下载预训练的 yolo11n-pose 模型。

目前 ESP-PPQ 支持 ONNX、PyTorch、TensorFlow 模型。在量化过程中，PyTorch 和 TensorFlow 会先转化为 ONNX 模型，因此将与训练的 yolo11n-pose 转化成 ONNX 模型。

具体来说，参考脚本：export_onnx.py 将预训练的 yolo11n-pose 模型转换为 ONNX 模型。

在该脚本中，我们重载了 Pose 类的 forward 方法，具有以下优势：

- 更快的推理速度。与原始的 yolo11n-pose 模型相比，将推理过程中 Pose 里与解码边界框相关的操作移至后处理中完成，从而显著减少了推理延迟。一方面，`Conv`，`Transpose`，`Slice`，`Split` 和 `Concat` 操作在推理过程中运行是非常耗时的。另一方面，在后处理阶段，模型推理的输出首先进行置信度筛选，然后再解码边界框，这大大减少了计算量，从而加快了整体推理速度。
- 更低的量化误差。ESP-PPQ 中的 `Concat` 和 `Add` 操作采用了联合量化。为了减少量化误差，由于 box 和 score 的范围差异较大，它们通过不同的分支输出，而不是拼接在一起。类似地，由于 `Add` 和 `Sub` 的输入的范围差异较大，相关计算被移到了后处理中进行，避免被量化。

校准数据集

校准数据集需要和模型输入格式一致，同时尽可能覆盖模型输入的所有可能情况，以便更好地量化模型。本示例中，我们使用的校准集为 calib_yolo11n-pose 。

8bit 后量化

下面的量化设置通过 AutoQuant 搜索得到。要使用 AutoQuant，请更新 esp-ppq 为最新版本并参考 教程。

ESP32-P4 量化设置

```
quant_setting = QuantizationSettingFactory.espdl_setting()
quant_setting.quantize_activation_setting.calib_algorithm = 'percentile'

quant_setting.bias_correct = True
quant_setting.bias_correct_setting.interested_layers = []
quant_setting.bias_correct_setting.block_size = 2
quant_setting.bias_correct_setting.steps = 32
```

ESP32-P4 量化误差

```
Analysing Graphwise Quantization Error(Phrase 1):: 100
↪%|
↪2/2 [00:01<00:00,  1.03it/s]
Analysing Graphwise Quantization Error(Phrase 2):: 100
↪%|
↪2/2 [00:06<00:00,  3.00s/it]
```

(续下页)

(接上页)

```
Layer                                          | NOISE:SIGNAL POWER RATIO
/model.22/m.0/cv2/conv/Conv:                   | ████████████████████ | 9.927%
/model.23/cv3.2/cv3.2.0/cv3.2.0.0/conv/Conv:   | ███████████████████  | 9.484%
/model.23/cv4.1/cv4.1.0/conv/Conv:             | ██████████████████   | 9.145%
/model.23/cv3.2/cv3.2.0/cv3.2.0.1/conv/Conv:   | ████████████████     | 8.567%
/model.23/cv3.2/cv3.2.1/cv3.2.1.0/conv/Conv:   | ████████████████     | 8.372%
/model.20/conv/Conv:                           | ████████████████     | 8.366%
/model.23/cv2.0/cv2.0.1/conv/Conv:             | ███████████████      | 7.941%
/model.19/m.0/cv2/conv/Conv:                   | ███████████████      | 7.861%
/model.23/cv4.1/cv4.1.1/conv/Conv:             | ███████████████      | 7.778%
/model.23/cv3.1/cv3.1.1/cv3.1.1.1/conv/Conv:   | ███████████████      | 7.773%
/model.23/cv4.0/cv4.0.0/conv/Conv:             | ███████████████      | 7.769%
/model.23/cv4.0/cv4.0.1/conv/Conv:             | ███████████████      | 7.749%
/model.23/cv2.1/cv2.1.1/conv/Conv:             | ███████████████      | 7.704%
/model.10/m/m.0/ffn/ffn.1/conv/Conv:           | ██████████████       | 7.641%
/model.22/m.0/cv3/conv/Conv:                   | ██████████████       | 7.541%
/model.23/cv3.1/cv3.1.1/cv3.1.1.0/conv/Conv:   | ██████████████       | 7.432%
/model.23/cv3.1/cv3.1.0/cv3.1.0.1/conv/Conv:   | ██████████████       | 7.315%
/model.23/cv3.1/cv3.1.0/cv3.1.0.0/conv/Conv:   | ████████████         | 7.086%
/model.23/cv3.2/cv3.2.1/cv3.2.1.1/conv/Conv:   | ████████████         | 7.036%
/model.22/cv1/conv/Conv:                       | ████████████         | 6.485%
/model.19/cv2/conv/Conv:                       | ████████████         | 6.333%
/model.23/cv3.0/cv3.0.1/cv3.0.1.0/conv/Conv:   | ████████████         | 6.296%
/model.22/m.0/m/m.1/cv2/conv/Conv:             | ████████████         | 6.255%
/model.22/cv2/conv/Conv:                       | ███████████          | 6.196%
/model.17/conv/Conv:                           | ███████████          | 6.110%
/model.23/cv2.2/cv2.2.0/conv/Conv:             | ███████████          | 5.841%
/model.23/cv4.2/cv4.2.1/conv/Conv:             | ███████████          | 5.763%
/model.23/cv4.2/cv4.2.0/conv/Conv:             | ███████████          | 5.740%
/model.23/cv2.1/cv2.1.0/conv/Conv:             | ██████████           | 5.657%
/model.19/cv1/conv/Conv:                       | ██████████           | 5.583%
/model.23/cv2.2/cv2.2.1/conv/Conv:             | ██████████           | 5.552%
/model.23/cv3.0/cv3.0.1/cv3.0.1.1/conv/Conv:   | ██████████           | 5.254%
/model.6/m.0/cv2/conv/Conv:                    | ██████████           | 5.245%
/model.13/m.0/cv2/conv/Conv:                   | █████████            | 5.172%
/model.19/m.0/cv1/conv/Conv:                   | █████████            | 5.166%
/model.22/m.0/m/m.0/cv2/conv/Conv:             | █████████            | 5.136%
/model.23/cv3.0/cv3.0.0/cv3.0.0.1/conv/Conv:   | █████████            | 5.061%
/model.8/m.0/cv2/conv/Conv:                    | █████████            | 4.985%
/model.10/m/m.0/attn/proj/conv/Conv:           | █████████            | 4.962%
/model.22/m.0/m/m.0/cv1/conv/Conv:             | ████████             | 4.609%
/model.23/cv4.2/cv4.2.2/Conv:                  | ████████             | 4.572%
/model.22/m.0/m/m.1/cv1/conv/Conv:             | ████████             | 4.417%
/model.16/m.0/cv2/conv/Conv:                   | ████████             | 4.411%
/model.6/cv1/conv/Conv:                        | ████████             | 4.264%
/model.23/cv4.1/cv4.1.2/Conv:                  | ████████             | 4.264%
/model.10/m/m.0/attn/pe/conv/Conv:             | ███████              | 4.161%
/model.23/cv2.0/cv2.0.0/conv/Conv:             | ███████              | 4.063%
/model.3/conv/Conv:                            | ███████              | 3.764%
/model.16/cv2/conv/Conv:                       | ██████               | 3.694%
/model.8/cv1/conv/Conv:                        | ██████               | 3.689%
/model.13/cv2/conv/Conv:                       | ██████               | 3.543%
/model.23/cv4.0/cv4.0.2/Conv:                  | ██████               | 3.376%
/model.22/m.0/cv1/conv/Conv:                   | ██████               | 3.363%
/model.10/cv1/conv/Conv:                       | ██████               | 3.357%
/model.8/cv2/conv/Conv:                        | ██████               | 3.279%
/model.6/m.0/cv3/conv/Conv:                    | ██████               | 3.254%
```

(续下页)

(接上页)

```
/model.8/m.0/cv3/conv/Conv:                    | ██████                | 3.219%
/model.13/cv1/conv/Conv:                       | ██████                | 3.180%
/model.10/m/m.0/ffn/ffn.0/conv/Conv:           | ██████                | 3.142%
/model.13/m.0/cv1/conv/Conv:                   | ██████                | 3.129%
/model.10/m/m.0/attn/qkv/conv/Conv:            | ██████                | 3.074%
/model.16/m.0/cv1/conv/Conv:                   | ██████                | 3.061%
/model.2/m.0/cv2/conv/Conv:                    | ██████                | 3.024%
/model.4/cv1/conv/Conv:                        | ██████                | 2.990%
/model.6/m.0/m/m.0/cv2/conv/Conv:              | ██████                | 2.844%
/model.16/cv1/conv/Conv:                       | ██████                | 2.821%
/model.8/m.0/m/m.1/cv2/conv/Conv:              | ██████                | 2.807%
/model.4/cv2/conv/Conv:                        | ██████                | 2.781%
/model.4/m.0/cv1/conv/Conv:                    | ██████                | 2.742%
/model.10/cv2/conv/Conv:                       | ██████                | 2.627%
/model.23/cv3.0/cv3.0.0/cv3.0.0.0/conv/Conv:   | ██████                | 2.613%
/model.2/cv2/conv/Conv:                        | ██████                | 2.611%
/model.6/cv2/conv/Conv:                        | ██████                | 2.593%
/model.8/m.0/cv1/conv/Conv:                    | ██████                | 2.553%
/model.10/m/m.0/attn/MatMul_1:                 | ██████                | 2.547%
/model.7/conv/Conv:                            | ██████                | 2.447%
/model.5/conv/Conv:                            | ██████                | 2.433%
/model.10/m/m.0/attn/MatMul:                   | ██████                | 2.363%
/model.23/cv2.1/cv2.1.2/Conv:                  | ██████                | 2.344%
/model.6/m.0/m/m.0/cv1/conv/Conv:              | ██████                | 2.305%
/model.6/m.0/cv1/conv/Conv:                    | ████                  | 2.250%
/model.8/m.0/m/m.0/cv2/conv/Conv:              | ████                  | 2.247%
/model.2/cv1/conv/Conv:                        | ████                  | 2.080%
/model.8/m.0/m/m.1/cv1/conv/Conv:              | ████                  | 2.070%
/model.23/cv2.2/cv2.2.2/Conv:                  | ████                  | 1.977%
/model.6/m.0/m/m.1/cv1/conv/Conv:              | ████                  | 1.927%
/model.9/cv1/conv/Conv:                        | ████                  | 1.926%
/model.23/cv2.0/cv2.0.2/Conv:                  | ████                  | 1.859%
/model.8/m.0/m/m.0/cv1/conv/Conv:              | ███                   | 1.694%
/model.9/cv2/conv/Conv:                        | ███                   | 1.672%
/model.23/cv3.2/cv3.2.2/Conv:                  | ███                   | 1.499%
/model.4/m.0/cv2/conv/Conv:                    | ███                   | 1.491%
/model.6/m.0/m/m.1/cv2/conv/Conv:              | ███                   | 1.452%
/model.2/m.0/cv1/conv/Conv:                    | ██                    | 1.093%
/model.1/conv/Conv:                            | ██                    | 0.834%
/model.23/cv3.1/cv3.1.2/Conv:                  | ▌                     | 0.568%
/model.23/cv3.0/cv3.0.2/Conv:                  |                       | 0.128%
/model.0/conv/Conv:                            |                       | 0.046%
Analysing Layerwise quantization error:: 100
↪%|████████████████████████████████████████████████████████████████████████████████████████████████████████████
↪98/98 [04:07<00:00,  2.53s/it]
Layer                                          | NOISE:SIGNAL POWER RATIO
/model.0/conv/Conv:                            | ████████████████████  | 0.323%
/model.2/cv1/conv/Conv:                        | ██████████            | 0.155%
/model.1/conv/Conv:                            | ████████              | 0.128%
/model.2/cv2/conv/Conv:                        | ███████               | 0.104%
/model.8/cv1/conv/Conv:                        | ████                  | 0.070%
/model.9/cv2/conv/Conv:                        | ███                   | 0.049%
/model.6/m.0/m/m.0/cv2/conv/Conv:              | ▌                     | 0.016%
/model.3/conv/Conv:                            | ▌                     | 0.014%
/model.2/m.0/cv1/conv/Conv:                    | ▌                     | 0.014%
/model.2/m.0/cv2/conv/Conv:                    | ▌                     | 0.014%
/model.6/m.0/m/m.1/cv2/conv/Conv:              | ▌                     | 0.012%
```

(续下页)

(接上页)

```
/model.23/cv4.1/cv4.1.1/conv/Conv:            | ▉                    | 0.011%
/model.5/conv/Conv:                           | ▉                    | 0.010%
/model.9/cv1/conv/Conv:                       | ▉                    | 0.010%
/model.6/cv1/conv/Conv:                       | ▉                    | 0.010%
/model.4/cv2/conv/Conv:                       | ▉                    | 0.009%
/model.4/cv1/conv/Conv:                       | ▉                    | 0.009%
/model.16/m.0/cv2/conv/Conv:                  | ▉                    | 0.009%
/model.19/m.0/cv2/conv/Conv:                  | ▉                    | 0.009%
/model.10/m/m.0/attn/qkv/conv/Conv:           | ▉                    | 0.008%
/model.10/cv1/conv/Conv:                      |                      | 0.008%
/model.23/cv4.2/cv4.2.0/conv/Conv:            |                      | 0.008%
/model.13/m.0/cv1/conv/Conv:                  |                      | 0.008%
/model.13/cv1/conv/Conv:                      |                      | 0.007%
/model.8/m.0/cv3/conv/Conv:                   |                      | 0.007%
/model.23/cv3.2/cv3.2.0/cv3.2.0.1/conv/Conv:  |                      | 0.007%
/model.23/cv2.0/cv2.0.2/Conv:                 |                      | 0.007%
/model.8/m.0/m/m.1/cv1/conv/Conv:             |                      | 0.007%
/model.22/cv2/conv/Conv:                      |                      | 0.007%
/model.6/cv2/conv/Conv:                       |                      | 0.006%
/model.23/cv4.2/cv4.2.2/Conv:                 |                      | 0.006%
/model.16/cv2/conv/Conv:                      |                      | 0.006%
/model.23/cv4.1/cv4.1.0/conv/Conv:            |                      | 0.006%
/model.23/cv4.2/cv4.2.1/conv/Conv:            |                      | 0.006%
/model.8/cv2/conv/Conv:                       |                      | 0.006%
/model.16/cv1/conv/Conv:                      |                      | 0.006%
/model.13/cv2/conv/Conv:                      |                      | 0.006%
/model.23/cv3.2/cv3.2.1/cv3.2.1.1/conv/Conv:  |                      | 0.005%
/model.13/m.0/cv2/conv/Conv:                  |                      | 0.005%
/model.7/conv/Conv:                           |                      | 0.005%
/model.10/cv2/conv/Conv:                      |                      | 0.005%
/model.22/m.0/m/m.0/cv2/conv/Conv:            |                      | 0.005%
/model.23/cv3.2/cv3.2.1/cv3.2.1.0/conv/Conv:  |                      | 0.005%
/model.23/cv2.1/cv2.1.2/Conv:                 |                      | 0.005%
/model.8/m.0/m/m.1/cv2/conv/Conv:             |                      | 0.005%
/model.23/cv3.2/cv3.2.0/cv3.2.0.0/conv/Conv:  |                      | 0.005%
/model.19/cv2/conv/Conv:                      |                      | 0.004%
/model.4/m.0/cv2/conv/Conv:                   |                      | 0.004%
/model.8/m.0/cv1/conv/Conv:                   |                      | 0.004%
/model.23/cv2.2/cv2.2.1/conv/Conv:            |                      | 0.004%
/model.19/cv1/conv/Conv:                      |                      | 0.004%
/model.23/cv2.0/cv2.0.1/conv/Conv:            |                      | 0.004%
/model.10/m/m.0/attn/pe/conv/Conv:            |                      | 0.004%
/model.23/cv2.2/cv2.2.2/Conv:                 |                      | 0.004%
/model.22/m.0/m/m.1/cv2/conv/Conv:            |                      | 0.004%
/model.23/cv4.0/cv4.0.0/conv/Conv:            |                      | 0.004%
/model.19/m.0/cv1/conv/Conv:                  |                      | 0.003%
/model.10/m/m.0/attn/proj/conv/Conv:          |                      | 0.003%
/model.22/m.0/cv3/conv/Conv:                  |                      | 0.003%
/model.8/m.0/m/m.0/cv1/conv/Conv:             |                      | 0.003%
/model.23/cv2.1/cv2.1.0/conv/Conv:            |                      | 0.003%
/model.23/cv3.2/cv3.2.2/Conv:                 |                      | 0.002%
/model.10/m/m.0/attn/MatMul_1:                |                      | 0.002%
/model.4/m.0/cv1/conv/Conv:                   |                      | 0.002%
/model.23/cv4.1/cv4.1.2/Conv:                 |                      | 0.002%
/model.22/m.0/cv1/conv/Conv:                  |                      | 0.002%
/model.8/m.0/m/m.0/cv2/conv/Conv:             |                      | 0.002%
/model.22/cv1/conv/Conv:                      |                      | 0.002%
```

(续下页)

(接上页)

```
/model.23/cv4.0/cv4.0.2/Conv:                 |                        | 0.002%
/model.22/m.0/m/m.0/cv1/conv/Conv:            |                        | 0.002%
/model.22/m.0/m/m.1/cv1/conv/Conv:            |                        | 0.002%
/model.10/m/m.0/ffn/ffn.1/conv/Conv:          |                        | 0.002%
/model.23/cv4.0/cv4.0.1/conv/Conv:            |                        | 0.002%
/model.16/m.0/cv1/conv/Conv:                  |                        | 0.002%
/model.23/cv2.1/cv2.1.1/conv/Conv:            |                        | 0.002%
/model.6/m.0/cv1/conv/Conv:                   |                        | 0.002%
/model.6/m.0/cv3/conv/Conv:                   |                        | 0.002%
/model.23/cv2.0/cv2.0.0/conv/Conv:            |                        | 0.002%
/model.6/m.0/m/m.1/cv1/conv/Conv:             |                        | 0.002%
/model.6/m.0/m/m.0/cv1/conv/Conv:             |                        | 0.001%
/model.23/cv2.2/cv2.2.0/conv/Conv:            |                        | 0.001%
/model.10/m/m.0/ffn/ffn.0/conv/Conv:          |                        | 0.001%
/model.17/conv/Conv:                          |                        | 0.001%
/model.23/cv3.1/cv3.1.1/cv3.1.1.1/conv/Conv:  |                        | 0.001%
/model.23/cv3.1/cv3.1.1/cv3.1.1.0/conv/Conv:  |                        | 0.001%
/model.20/conv/Conv:                          |                        | 0.001%
/model.23/cv3.1/cv3.1.0/cv3.1.0.1/conv/Conv:  |                        | 0.001%
/model.23/cv3.0/cv3.0.1/cv3.0.1.1/conv/Conv:  |                        | 0.001%
/model.23/cv3.1/cv3.1.2/Conv:                 |                        | 0.000%
/model.23/cv3.1/cv3.1.0/cv3.1.0.0/conv/Conv:  |                        | 0.000%
/model.23/cv3.0/cv3.0.2/Conv:                 |                        | 0.000%
/model.23/cv3.0/cv3.0.1/cv3.0.1.0/conv/Conv:  |                        | 0.000%
/model.6/m.0/cv2/conv/Conv:                   |                        | 0.000%
/model.23/cv3.0/cv3.0.0/cv3.0.0.0/conv/Conv:  |                        | 0.000%
/model.8/m.0/cv2/conv/Conv:                   |                        | 0.000%
/model.23/cv3.0/cv3.0.0/cv3.0.0.1/conv/Conv:  |                        | 0.000%
/model.10/m/m.0/attn/MatMul:                  |                        | 0.000%
/model.22/m.0/cv2/conv/Conv:                  |                        | 0.000%
```

ESP32-P4 量化结果

在相同输入下，量化后的模型在 COCO 上的 Pose mAP50:95 为 43.9%，低于浮点模型（50.0%），存在一定的精度损失。

量化感知训练

为了进一步提高量化模型的精度，可以采用量化感知训练。本示例基于 8-bit 量化方式进行量化感知训练。

量化设置

- yolo11n-pose_qat.py
- trainer.py

ESP32-P4 量化误差

```
Layer                                         | NOISE:SIGNAL POWER RATIO
/model.22/m.0/cv2/conv/Conv:                  | ████████████████████   | 27.739%
/model.23/cv3.2/cv3.2.0/cv3.2.0.1/conv/Conv:  | ███████████████████    | 26.872%
/model.23/cv4.1/cv4.1.0/conv/Conv:            | ███████████████████    | 26.229%
/model.23/cv2.1/cv2.1.1/conv/Conv:            | ██████████████████     | 25.300%
/model.23/cv3.2/cv3.2.1/cv3.2.1.0/conv/Conv:  | ██████████████████     | 24.625%
/model.23/cv2.0/cv2.0.1/conv/Conv:            | ████████████████       | 23.751%
/model.20/conv/Conv:                          | ████████████████       | 23.320%
/model.23/cv3.2/cv3.2.0/cv3.2.0.0/conv/Conv:  | ████████████████       | 22.901%
```

(续下页)

(接上页)

```
/model.23/cv4.1/cv4.1.1/conv/Conv:            | ████████████████ | 22.516%
/model.10/m/m.0/ffn/ffn.1/conv/Conv:          | ████████████████ | 22.035%
/model.19/m.0/cv2/conv/Conv:                  | ████████████████ | 21.569%
/model.23/cv4.0/cv4.0.0/conv/Conv:            | ███████████████  | 21.199%
/model.23/cv3.1/cv3.1.0/cv3.1.0.1/conv/Conv:  | ███████████████  | 20.785%
/model.23/cv3.1/cv3.1.1/cv3.1.1.0/conv/Conv:  | ███████████████  | 20.597%
/model.23/cv3.1/cv3.1.1/cv3.1.1.1/conv/Conv:  | ███████████████  | 20.329%
/model.23/cv4.0/cv4.0.1/conv/Conv:            | ███████████████  | 20.179%
/model.23/cv3.1/cv3.1.0/cv3.1.0.0/conv/Conv:  | ██████████████   | 19.983%
/model.22/m.0/cv3/conv/Conv:                  | ██████████████   | 19.919%
/model.13/m.0/cv2/conv/Conv:                  | ██████████████   | 19.424%
/model.23/cv3.0/cv3.0.1/cv3.0.1.0/conv/Conv:  | ██████████████   | 18.893%
/model.19/cv2/conv/Conv:                      | ████████████     | 18.055%
/model.23/cv3.2/cv3.2.1/cv3.2.1.1/conv/Conv:  | ████████████     | 17.915%
/model.22/m.0/m/m.1/cv2/conv/Conv:            | ████████████     | 17.796%
/model.22/cv1/conv/Conv:                      | ████████████     | 17.777%
/model.23/cv4.2/cv4.2.1/conv/Conv:            | ████████████     | 17.573%
/model.19/cv1/conv/Conv:                      | ███████████      | 17.116%
/model.17/conv/Conv:                          | ███████████      | 16.869%
/model.22/cv2/conv/Conv:                      | ███████████      | 16.750%
/model.23/cv2.2/cv2.2.1/conv/Conv:            | ███████████      | 16.540%
/model.10/m/m.0/attn/proj/conv/Conv:          | ███████████      | 16.491%
/model.23/cv2.2/cv2.2.0/conv/Conv:            | ███████████      | 16.421%
/model.23/cv2.1/cv2.1.0/conv/Conv:            | ███████████      | 16.205%
/model.23/cv4.2/cv4.2.0/conv/Conv:            | ███████████      | 16.116%
/model.23/cv3.0/cv3.0.1/cv3.0.1.1/conv/Conv:  | ██████████       | 15.400%
/model.22/m.0/m/m.0/cv2/conv/Conv:            | ██████████       | 15.251%
/model.23/cv3.0/cv3.0.0/cv3.0.0.1/conv/Conv:  | ██████████       | 14.851%
/model.10/m/m.0/attn/pe/conv/Conv:            | ██████████       | 14.659%
/model.19/m.0/cv1/conv/Conv:                  | ██████████       | 14.289%
/model.22/m.0/m/m.1/cv1/conv/Conv:            | ████████         | 13.038%
/model.16/m.0/cv2/conv/Conv:                  | ████████         | 12.941%
/model.22/m.0/m/m.0/cv1/conv/Conv:            | ████████         | 12.791%
/model.23/cv4.2/cv4.2.2/Conv:                 | ████████         | 12.508%
/model.23/cv4.1/cv4.1.2/Conv:                 | ████████         | 12.226%
/model.13/cv1/conv/Conv:                      | ███████          | 11.821%
/model.13/cv2/conv/Conv:                      | ███████          | 11.612%
/model.13/m.0/cv1/conv/Conv:                  | ███████          | 11.515%
/model.10/m/m.0/attn/MatMul_1:                | ███████          | 11.303%
/model.16/cv2/conv/Conv:                      | ███████          | 11.028%
/model.10/m/m.0/attn/qkv/conv/Conv:           | ███████          | 10.951%
/model.10/cv1/conv/Conv:                      | ███████          | 10.755%
/model.23/cv2.0/cv2.0.0/conv/Conv:            | ███████          | 10.684%
/model.22/m.0/cv1/conv/Conv:                  | ██████           | 10.164%
/model.10/m/m.0/ffn/ffn.0/conv/Conv:          | ██████           | 9.968%
/model.16/m.0/cv1/conv/Conv:                  | ██████           | 9.656%
/model.23/cv4.0/cv4.0.2/Conv:                 | ██████           | 9.566%
/model.8/m.0/cv2/conv/Conv:                   | ██████           | 9.521%
/model.10/cv2/conv/Conv:                      | ██████           | 8.068%
/model.16/cv1/conv/Conv:                      | ██████           | 7.989%
/model.23/cv2.1/cv2.1.2/Conv:                 | ██████           | 7.969%
/model.8/m.0/cv3/conv/Conv:                   | ██████           | 7.725%
/model.23/cv3.0/cv3.0.0/cv3.0.0.0/conv/Conv:  | ████             | 7.570%
/model.8/m.0/m/m.0/cv2/conv/Conv:             | ████             | 7.339%
/model.8/m.0/m/m.1/cv2/conv/Conv:             | ████             | 7.283%
/model.8/cv2/conv/Conv:                       | ████             | 7.092%
/model.10/m/m.0/attn/MatMul:                  | ████             | 6.654%
```

(续下页)

(接上页)

```
/model.8/cv1/conv/Conv:                          | ██████                | 6.492%
/model.8/m.0/m/m.1/cv1/conv/Conv:                | ██████                | 6.451%
/model.23/cv2.0/cv2.0.2/Conv:                    | ████                  | 5.990%
/model.23/cv2.2/cv2.2.2/Conv:                    | ████                  | 5.902%
/model.6/m.0/m/m.0/cv2/conv/Conv:                | ████                  | 5.898%
/model.6/m.0/cv2/conv/Conv:                      | ████                  | 5.881%
/model.6/m.0/cv3/conv/Conv:                      | ████                  | 5.402%
/model.8/m.0/cv1/conv/Conv:                      | ████                  | 5.210%
/model.23/cv3.2/cv3.2.2/Conv:                    | ████                  | 5.126%
/model.6/cv1/conv/Conv:                          | ████                  | 4.983%
/model.9/cv2/conv/Conv:                          | ███                   | 4.616%
/model.9/cv1/conv/Conv:                          | ███                   | 3.934%
/model.7/conv/Conv:                              | ███                   | 3.906%
/model.3/conv/Conv:                              | ███                   | 3.654%
/model.6/cv2/conv/Conv:                          | ██                    | 3.429%
/model.8/m.0/m/m.0/cv1/conv/Conv:                | ██                    | 3.319%
/model.2/cv2/conv/Conv:                          | ██                    | 3.220%
/model.6/m.0/m/m.1/cv1/conv/Conv:                | ██                    | 3.191%
/model.6/m.0/m/m.0/cv1/conv/Conv:                | ██                    | 3.157%
/model.4/cv1/conv/Conv:                          | ██                    | 2.893%
/model.6/m.0/m/m.1/cv2/conv/Conv:                | ██                    | 2.792%
/model.6/m.0/cv1/conv/Conv:                      | ██                    | 2.761%
/model.5/conv/Conv:                              | ██                    | 2.629%
/model.4/cv2/conv/Conv:                          | ██                    | 2.298%
/model.2/cv1/conv/Conv:                          | ▌                     | 2.107%
/model.2/m.0/cv2/conv/Conv:                      | ▌                     | 2.095%
/model.4/m.0/cv1/conv/Conv:                      | ▌                     | 2.069%
/model.23/cv3.1/cv3.1.2/Conv:                    | ▌                     | 1.744%
/model.1/conv/Conv:                              | ▌                     | 1.631%
/model.2/m.0/cv1/conv/Conv:                      | ▌                     | 1.583%
/model.4/m.0/cv2/conv/Conv:                      | ▌                     | 1.126%
/model.23/cv3.0/cv3.0.2/Conv:                    |                       | 0.535%
/model.0/conv/Conv:                              |                       | 0.067%
Analysing Layerwise quantization error:: 100%|██████████| 98/98 [10:49<00:00,  6.63s/
↪it]
Layer                                            | NOISE:SIGNAL POWER RATIO
/model.9/cv2/conv/Conv:                          | ████████████████████  | 2.976%
/model.2/cv2/conv/Conv:                          | ██████████            | 1.610%
/model.3/conv/Conv:                              | ██████                | 0.854%
/model.2/cv1/conv/Conv:                          | ████                  | 0.543%
/model.1/conv/Conv:                              | ███                   | 0.487%
/model.8/cv1/conv/Conv:                          | ███                   | 0.414%
/model.4/cv2/conv/Conv:                          | ███                   | 0.397%
/model.0/conv/Conv:                              | ██                    | 0.364%
/model.6/m.0/cv3/conv/Conv:                      | ██                    | 0.230%
/model.5/conv/Conv:                              | ▌                     | 0.181%
/model.2/m.0/cv2/conv/Conv:                      | ▌                     | 0.144%
/model.13/cv2/conv/Conv:                         | ▌                     | 0.140%
/model.2/m.0/cv1/conv/Conv:                      | ▌                     | 0.138%
/model.4/cv1/conv/Conv:                          | ▌                     | 0.129%
/model.16/cv2/conv/Conv:                         | ▌                     | 0.122%
/model.23/cv4.2/cv4.2.0/conv/Conv:               | ▌                     | 0.120%
/model.4/m.0/cv1/conv/Conv:                      | ▌                     | 0.107%
/model.23/cv4.1/cv4.1.0/conv/Conv:               | ▌                     | 0.096%
/model.19/cv2/conv/Conv:                         | ▌                     | 0.078%
/model.23/cv2.2/cv2.2.2/Conv:                    | ▌                     | 0.076%
/model.4/m.0/cv2/conv/Conv:                      |                       | 0.071%
```

(续下页)

(接上页)

/model.8/m.0/m/m.1/cv1/conv/Conv:		0.071%
/model.6/cv2/conv/Conv:		0.067%
/model.6/cv1/conv/Conv:		0.066%
/model.17/conv/Conv:		0.060%
/model.23/cv4.2/cv4.2.1/conv/Conv:		0.057%
/model.22/m.0/m/m.1/cv1/conv/Conv:		0.056%
/model.16/cv1/conv/Conv:		0.051%
/model.10/cv1/conv/Conv:		0.050%
/model.23/cv4.2/cv4.2.2/Conv:		0.046%
/model.22/cv2/conv/Conv:		0.044%
/model.7/conv/Conv:		0.043%
/model.10/m/m.0/attn/pe/conv/Conv:		0.043%
/model.10/cv2/conv/Conv:		0.037%
/model.19/cv1/conv/Conv:		0.037%
/model.8/cv2/conv/Conv:		0.036%
/model.13/cv1/conv/Conv:		0.036%
/model.6/m.0/m/m.1/cv1/conv/Conv:		0.033%
/model.22/m.0/cv3/conv/Conv:		0.031%
/model.19/m.0/cv1/conv/Conv:		0.027%
/model.23/cv3.2/cv3.2.0/cv3.2.0.1/conv/Conv:		0.026%
/model.8/m.0/cv1/conv/Conv:		0.025%
/model.19/m.0/cv2/conv/Conv:		0.025%
/model.8/m.0/cv3/conv/Conv:		0.024%
/model.10/m/m.0/attn/qkv/conv/Conv:		0.023%
/model.8/m.0/m/m.0/cv1/conv/Conv:		0.023%
/model.22/m.0/cv1/conv/Conv:		0.021%
/model.6/m.0/m/m.0/cv1/conv/Conv:		0.021%
/model.23/cv2.0/cv2.0.0/conv/Conv:		0.020%
/model.6/m.0/cv1/conv/Conv:		0.020%
/model.23/cv4.0/cv4.0.0/conv/Conv:		0.019%
/model.9/cv1/conv/Conv:		0.018%
/model.23/cv4.1/cv4.1.2/Conv:		0.018%
/model.23/cv2.1/cv2.1.1/conv/Conv:		0.018%
/model.13/m.0/cv1/conv/Conv:		0.016%
/model.23/cv2.1/cv2.1.0/conv/Conv:		0.016%
/model.23/cv4.1/cv4.1.1/conv/Conv:		0.016%
/model.16/m.0/cv2/conv/Conv:		0.015%
/model.10/m/m.0/attn/proj/conv/Conv:		0.013%
/model.23/cv3.1/cv3.1.1/cv3.1.1.1/conv/Conv:		0.013%
/model.8/m.0/m/m.0/cv2/conv/Conv:		0.013%
/model.16/m.0/cv1/conv/Conv:		0.012%
/model.23/cv2.2/cv2.2.0/conv/Conv:		0.011%
/model.20/conv/Conv:		0.011%
/model.22/m.0/m/m.0/cv1/conv/Conv:		0.011%
/model.23/cv3.2/cv3.2.1/cv3.2.1.1/conv/Conv:		0.011%
/model.8/m.0/m/m.1/cv2/conv/Conv:		0.010%
/model.23/cv2.0/cv2.0.2/Conv:		0.009%
/model.10/m/m.0/attn/MatMul:		0.009%
/model.22/cv1/conv/Conv:		0.009%
/model.13/m.0/cv2/conv/Conv:		0.008%
/model.23/cv2.2/cv2.2.1/conv/Conv:		0.008%
/model.23/cv2.1/cv2.1.2/Conv:		0.007%
/model.23/cv3.2/cv3.2.1/cv3.2.1.0/conv/Conv:		0.007%
/model.22/m.0/m/m.1/cv2/conv/Conv:		0.007%
/model.6/m.0/m/m.0/cv2/conv/Conv:		0.006%
/model.22/m.0/m/m.0/cv2/conv/Conv:		0.006%
/model.23/cv4.0/cv4.0.1/conv/Conv:		0.005%

(续下页)

(接上页)

```
/model.23/cv3.2/cv3.2.0/cv3.2.0.0/conv/Conv: |                      | 0.005%
/model.23/cv4.0/cv4.0.2/Conv:                |                      | 0.004%
/model.6/m.0/m/m.1/cv2/conv/Conv:            |                      | 0.004%
/model.23/cv3.0/cv3.0.0/cv3.0.0.1/conv/Conv: |                      | 0.004%
/model.10/m/m.0/ffn/ffn.1/conv/Conv:         |                      | 0.003%
/model.23/cv3.2/cv3.2.2/Conv:                |                      | 0.003%
/model.10/m/m.0/attn/MatMul_1:               |                      | 0.002%
/model.10/m/m.0/ffn/ffn.0/conv/Conv:         |                      | 0.002%
/model.23/cv3.1/cv3.1.0/cv3.1.0.1/conv/Conv: |                      | 0.002%
/model.23/cv2.0/cv2.0.1/conv/Conv:           |                      | 0.002%
/model.23/cv3.1/cv3.1.1/cv3.1.1.0/conv/Conv: |                      | 0.001%
/model.23/cv3.0/cv3.0.2/Conv:                |                      | 0.001%
/model.23/cv3.1/cv3.1.2/Conv:                |                      | 0.001%
/model.23/cv3.0/cv3.0.1/cv3.0.1.0/conv/Conv: |                      | 0.001%
/model.23/cv3.1/cv3.1.0/cv3.1.0.0/conv/Conv: |                      | 0.001%
/model.23/cv3.0/cv3.0.0/cv3.0.0.0/conv/Conv: |                      | 0.000%
/model.6/m.0/cv2/conv/Conv:                  |                      | 0.000%
/model.23/cv3.0/cv3.0.1/cv3.0.1.1/conv/Conv: |                      | 0.000%
/model.8/m.0/cv2/conv/Conv:                  |                      | 0.000%
/model.22/m.0/cv2/conv/Conv:                 |                      | 0.000%
```

ESP32-P4 量化结果

在对 8-bit 量化应用量化感知训练后，在相同输入下，量化后的模型在 COCO 上的 Pose mAP50:95 提升至 44.2%；同时输出层的累计误差大幅减少。相比 8-bit 后量化方式，量化感知训练后的 8-bit 量化模型可以在相同的推理速度下达到最高的量化精度。

备注： 本文档提到的 mAP 计算结果均是基于 ultralytics version 8.4.50 得到的。

3.9.3 模型部署

example

目标检测基类

- dl_detect_base.hpp
- dl_detect_base.cpp

前处理

`ImagePreprocessor` 类中封装了常用的图像前处理流程，包括 `color conversion`, `crop`, `resize`, `normalization`, `quantize`。

- dl_image_preprocessor.hpp
- dl_image_preprocessor.cpp

后处理

- dl_detect_postprocessor.hpp
- dl_detect_postprocessor.cpp
- dl_pose_yolo11_postprocessor.hpp
- dl_pose_yolo11_postprocessor.cpp

3.10 如何部署流式模型

时间序列模型如今被应用在许多领域，例如，音频领域。而音频模型在部署时通常有两种模式：

- Offline 模式：模型需要一次性接收完整的音频数据（例如整个语音文件），然后进行整体处理。
- Streaming 模式：流式模式下，模型逐帧（逐块）接收音频数据，实时处理并输出中间结果。

在本教程中，我们来介绍如何使用 ESP-PPQ 量化流式模型，并使用 ESP-DL 部署量化后的流式模型。

- 准备工作
- 模型量化
  - 自动流式转换
  - 自动流式转换的工作原理
  - 手动流式缓存配置
- 模型部署

3.10.1 准备工作

1. 安装 *ESP_IDF*
2. 安装 *ESP_PPQ*

3.10.2 模型量化

参考示例

时间序列模型种类繁多，这里仅以 Temporal Convolutional Network(TCN) 为例，不熟悉的可自行查找资料了解，这里不过多介绍其细节。其它模型需根据自身情况，量体裁衣。

该示例代码中构建了一个 TCN 模型：models.py (模型非完整，仅用于演示)。

ESP-PPQ 提供了自动流式转换功能，可以简化创建流式模型的过程。通过 `auto_streaming=True` 参数，ESP-PPQ 自动处理流式推理所需的模型转换。

备注:

- Offline 模式，模型输入是一段完整数据，input shape 在时间维度上的 size 一般比较大（例如 `[1, 16, 15]`）。

- Streaming 模式，模型输入是连续数据，在时间维度上的 size 较小，匹配实时处理的块大小（例如 `[1, 16, 3]`）。

自动流式转换

ESP-PPQ 通过量化过程中的 `auto_streaming=True` 参数提供自动流式转换功能。启用此标志后，ESP-PPQ 会自动转换模型以支持流式推理：

1. 分析模型结构以识别适当的分块点
2. 创建内部状态管理以在块之间保持上下文
3. 生成适合流式场景的优化代码

自动流式转换的工作原理

ESP-PPQ 中的自动流式转换会分析模型图，并在关键位置插入 `StreamingCache` 节点以实现时间上下文保持。转换过程遵循以下原则：

1. 算子分类

- **支持流式的算子**：需要时间上下文的卷积、池化和转置卷积操作（例如 `Conv`、`AveragePool`、`MaxPool`、`ConvTranspose`）。
- **绕过算子**：不需要时间上下文的激活函数、数学运算、量化节点和其他操作（例如 `Relu`、`Add`、`MatMul`、`LayerNorm`）。

2. 窗口大小计算

对于支持流式的算子，ESP-PPQ 根据以下因素计算所需的缓存窗口大小：- Kernel size and dilation rates - Padding configuration - Stride values

窗口大小决定了需要缓存多少历史帧才能正确计算当前帧。

3. StreamingCache 节点插入

ESP-PPQ 在支持流式的算子之前插入 `StreamingCache` 节点。这些节点：- 维护历史帧的滑动窗口缓冲区 - 调整张量形状以容纳缓存窗口 - 保留原始操作的量化配置 - 管理帧轴对齐以进行正确的时间处理

4. 填充调整

对于流式操作，ESP-PPQ 调整填充配置：- 移除底部填充以防止前瞻到未来帧 - 保持对称或仅顶部填充以实现因果处理

限制和注意事项

- 自动转换开箱即用地支持基于卷积的时间操作
- 自定义操作或复杂的时间依赖关系可能需要手动配置流式表
- 转换假设时间维度沿轴 1（可通过 `streaming_table` 配置）

以下是如何使用自动流式功能的示例：

```
# 导出非流式模型
quant_ppq_graph = espdl_quantize_torch(
    model=model,
    espdl_export_file=ESPDL_MODEL_PATH,
    calib_dataloader=dataloader,
    calib_steps=32,  # 校准步数
    input_shape=INPUT_SHAPE,  # 离线模式的输入形状
    inputs=None,
```

(续下页)

(接上页)

```
    target=TARGET,  # 量化目标类型
    num_of_bits=NUM_OF_BITS,  # 量化位数
    dispatching_override=None,
    device=DEVICE,
    error_report=True,
    skip_export=False,
    export_test_values=True,
    verbose=1,  # 输出详细日志信息
)

# 使用自动转换导出流式模型
quant_ppq_graph = espdl_quantize_torch(
    model=model,
    espdl_export_file=ESPDL_STEAMING_MODEL_PATH,
    calib_dataloader=dataloader,
    calib_steps=32,
    input_shape=INPUT_SHAPE,
    inputs=None,
    target=TARGET,
    num_of_bits=NUM_OF_BITS,
    dispatching_override=None,
    device=DEVICE,
    error_report=True,
    skip_export=False,
    export_test_values=False,
    verbose=1,
    auto_streaming=True,  # 启用自动流式转换
    streaming_input_shape=[1, 16, 3],  # 流式模式的输入形状
    streaming_table=None,
)
```

手动流式缓存配置

对于 ESP-PPQ 流式转换功能不自动支持的算子（例如 Transpose、Reshape、Slice 等），您可以使用 `insert_streaming_cache_on_var` 函数手动插入 StreamingCache 节点。该函数允许您为无法自动插入 streamingCache 的变量指定缓存属性。

`insert_streaming_cache_on_var` 函数的签名如下：

```
def insert_streaming_cache_on_var(
    var_name: str,
    window_size: int,
    op_name: str = None,
    frame_axis: int = 1
) -> Dict[str, Any]
```

参数说明：- `var_name`：需要插入流式缓存的变量名称 - `window_size`：缓存窗口大小（需要缓存的帧数）- `op_name`：（可选）与变量关联的算子名称 - `frame_axis`：（可选）表示时间维度的轴，默认为 1

该函数返回一个包含流式缓存配置的字典，应将其添加到 `streaming_table` 列表中并传递给 `espdl_quantize_torch` 函数。

使用示例：

```
streaming_table = []
# 为无法自动插入 streamingCache 的变量手动指定缓存属性
```

(续下页)

(接上页)

```
streaming_table.append(
    insert_streaming_cache_on_var("/out_conv/Conv_output_0", output_frame_size - 1)
)
streaming_table.append(insert_streaming_cache_on_var("PPQ_Variable_0", 1, "/Slice"))

quant_ppq_graph = espdl_quantize_torch(
    model=model,
    espdl_export_file=ESPDL_STEAMING_MODEL_PATH,
    calib_dataloader=dataloader,
    calib_steps=32,
    input_shape=INPUT_SHAPE,
    inputs=None,
    target=TARGET,
    num_of_bits=NUM_OF_BITS,
    dispatching_override=None,
    device=DEVICE,
    error_report=True,
    skip_export=False,
    export_test_values=False,
    verbose=1,
    auto_streaming=True,
    streaming_input_shape=[1, 16, 3],
    streaming_table=streaming_table,  # 传递手动配置的流式表
)
```

3.10.3 模型部署

参考示例，该示例使用预生成的数据来模拟实时数据流。

备注： 基础的模型加载和推理方法，可参考其它文档，这里不再赘述：

- 如何加载和测试模型
- 如何进行模型推理

在流式模式下，模型按时间接收数据块，而不是要求一次性获得整个输入。流式模型依次处理这些块，同时在块之间保持内部状态。部署代码负责将输入分解为适当的块并将其馈送到模型。见 app_main.cpp 如下代码块：

```
dl::TensorBase *run_streaming_model(dl::Model *model, dl::TensorBase *test_input)
{
    std::map<std::string, dl::TensorBase *> model_inputs = model->get_inputs();
    dl::TensorBase *model_input = model_inputs.begin()->second;
    std::map<std::string, dl::TensorBase *> model_outputs = model->get_outputs();
    dl::TensorBase *model_output = model_outputs.begin()->second;

    if (!test_input) {
        ESP_LOGE(TAG,
                 "Model input doesn't have a corresponding test input. Please enable␣
↪export_test_values option "
                 "in esp-ppq when export espdl model.");
        return nullptr;
    }
```

(续下页)

(接上页)

```
    int test_input_size = test_input->get_bytes();
    uint8_t *test_input_ptr = (uint8_t *)test_input->data;
    int model_input_size = model_input->get_bytes();
    uint8_t *model_input_ptr = (uint8_t *)model_input->data;
    int chunks = test_input_size / model_input_size;
    for (int i = 0; i < chunks; i++) {
        // assign chunk data to model input
        memcpy(model_input_ptr, test_input_ptr + i * model_input_size, model_input_
↪size);
        model->run(model_input);
    }

    return model_output;
}
```

这种方法允许模型通过将长序列分解为更小、更易管理的块来高效处理。每个块依次馈送到模型中，内部状态自动维护以确保跨块的连续性。

备注:

- 块的数量是根据完整输入大小与流式模型输入大小的比率计算的。
- ESP-DL 流式模型自动处理内部状态管理，使部署变得简单。
- 流式模型的输出应与等效离线模型输出的最后部分匹配。

3.11 使用 TQT 量化模型

本文档说明为何以及如何在 ESP-PPQ 中使用 Trained Quantization Thresholds (TQT) 做量化，以获得更高的量化精度。注意 ESP-PPQ 必须是 1.2.7 版本或更新版本。

- *为何使用 TQT*
  - *后量化（PTQ）的局限性*
  - *量化感知训练（QAT）的复杂性*
  - *TQT 能带来什么*
- *如何使用 TQT*
  - *快速开始*
  - *TQTSetting 常用参数*
- *TQT 量化示例*
  - *YOLO26n 量化*
  - *MobileNetV2 量化*
- *常见问题*
  - *如何加速 TQT？*

- 可以不修改 *scale* 只训练模型权重吗？
- *TQT* 可以和 *Weight Equalization* 一起使用吗？

3.11.1 为何使用 TQT

TQT（Trained Quantization Thresholds）出自论文 Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks （Sambhav R. Jain, Albert Gural, Michael Wu, Chris H. Dick），发表于 **MLSys 2020**。由于 ESP32 系列芯片目前仅支持 **Per-Tensor/Per-Channel + Symmetric + Power-of-2** 量化策略生成的 `.espdl` 模型，我们在 ESP-PPQ 中引入了 TQT 的核心思想，通过 **标准反向传播与梯度下降**联合优化量化阈值（scale）和模型权重，以适配硬件约束。

在 ESP-PPQ 中，TQT 以 **TrainedQuantizationThresholdPass** 的形式实现，我们在 log 域对 scale 进行优化，并结合 ESP-DL 的 **Power-of-2** 约束做了适配（如 `int_lambda` ，STE 等）。

后量化（PTQ）的局限性

方式	优点	局限
PTQ	实现简单、无需训练	scale 由统计得到，未必最优；对敏感结构（如 YOLO26n head）误差较大
TQT	微调 scale/权重	需要一定校准数据与算力

量化感知训练（QAT）的复杂性

量化感知训练（QAT）通常在完整训练或微调阶段插入伪量化，能获得较好精度，但需要：

- **完整标注数据与较长训练、微调流程** (对于不使用标注数据进行训练的模型，QAT 也往往花费较长的训练、微调流程)；
- **较大算力与调参成本**；
- 与现有 PTQ/校准工具链的衔接往往需要额外工程。

TQT 则 **无需标签**：损失为浮点输出和量化输出的 MSE，只需校准集即可，操作简单，在保持 Power-of-2 约束下联合微调权重与 log2(scale)，通常用较少步数即可明显提升精度。

TQT 能带来什么

- **可学习的 scale (仍为 2 的 k 次方)**：在 log 域优化 `alpha = log2(scale)` ，数值稳定且自然满足 POWER_OF_2。但如果不满足条件或被禁用，则不会训练。
- **STE 与 range-precision 折中**：对阈值梯度使用 STE 并合理设计前向/反向行为，可在 **表示范围**与 **精度**之间取得更好折中；ESP-PPQ 中通过 alpha_ste 实现。
- **可学习权重**：权重可训练，可与 scale 一起做块级微调，进一步减小量化误差。
- **与 ESP-DL 对齐**：使用默认的 alpha_ste 前向时，前向用的就是整数 exponent；配合 `int_lambda` 可让学到的 alpha 更接近整数，便于导出与芯片端一致。
- **无需标签**：只需校准集即可做 scale/权重微调。

3.11.2 如何使用 TQT

快速开始

1. 使用 ESP-DL 默认 setting 并开启 TQT

```
from esp_ppq import QuantizationSettingFactory
from esp_ppq.api import espdl_quantize_onnx

quant_setting = QuantizationSettingFactory.espdl_setting()
quant_setting.tqt_optimization = True
quant_setting.tqt_optimization_setting.int_lambda = 0.25    # 可选：让 exponent␣
↪更接近整数
quant_setting.tqt_optimization_setting.steps = 500
quant_setting.tqt_optimization_setting.lr = 1e-5
quant_setting.tqt_optimization_setting.collecting_device = 'cuda'  # or 'cpu'
quant_setting.tqt_optimization_setting.block_size = 2
```

2. 传入 setting 做量化

```
from esp_ppq.api import ENABLE_CUDA_KERNEL
with ENABLE_CUDA_KERNEL():
    quant_ppq_graph = espdl_quantize_onnx(
        onnx_import_file=ONNX_PATH,
        espdl_export_file=ESPDL_MODLE_PATH,
        calib_dataloader=dataloader,
        calib_steps=32,
        input_shape=[1] + INPUT_SHAPE,
        target=TARGET,
        num_of_bits=NUM_OF_BITS,
        collate_fn=collate_fn,
        setting=quant_setting,
        device=DEVICE,
        error_report=False,
        skip_export=False,
        export_test_values=True,
        verbose=0,
        inputs=None,
    )
```

3. 导出与返回值

- **返回的 quant_ppq_graph** 里 scale 为 **TQT 微调后的 2 的整数次方**，可用于模型精度评估。
- **导出的.espdl 文件**：exponent 来自 int(log2(config.scale))，导出与芯片端推理会使用 TQT 优化后的 scale。

TQTSetting 常用参数

参数	类型	默认值	说明
interested_layers	List[str]	[]	指定要微调的 op 名称；空则对满足条件的 Conv/Gemm 等全部微调
steps	int	500	每个 block 的微调步数
lr	float	1e-5	学习率
block_size	int	4	图划分 block 的大小，影响稳定性与速度
is_scale_trainable	bool	True	是否微调 scale（需 POWER_OF_2 + LINEAR + SYMMETRICAL）。True：同时训练 scale 与 weights；False：仅训练 weights
gamma	float	0.0	正则：MSE(浮点输出, 量化输出)
int_lambda	float	0.0	正则：让 alpha 靠近 round(alpha)，取值范围为 [0.0, 1.0]
collecting_device	str	cpu	缓存校准数据的设备，有 GPU 可设为 cuda

3.11.3 TQT 量化示例

以下示例展示如何对 **YOLO26n** (检测) 和 **MobileNetV2** (分类) 模型启用 TQT 量化并导出 ESP-DL。校准数据与模型路径需按实际环境修改。

备注： 示例 YOLO26n 和 MobileNetV2 中的 P4 量化精度均是在 Per-Tensor 的量化粒度下测得的，现 P4 已支持对 Conv/Gemm 算子做 Per-Channel 量化。

YOLO26n 量化

准备

项目	说明
模型名称	YOLO26n (one2one 分支 end2end 推理)
任务	目标检测
输入形状	[1, 3, 640, 640] (NCHW)
ONNX	yolo26n_o2o.onnx
校准集	calib_yolo26n.zip
备注	也可以通过 one2many 分支做非 end2end 推理，ONNX 为 yolo26n_o2m.onnx ，量化方法与 end2end 模型一致

量化脚本

```
import os
from esp_ppq import QuantizationSettingFactory
from esp_ppq.api import espdl_quantize_onnx
from torch.utils.data import DataLoader
import torch
from torch.utils.data import Dataset
from torchvision import transforms
from PIL import Image
from onnxsim import simplify
import onnx
import zipfile
import urllib.request
from esp_ppq.api import ENABLE_CUDA_KERNEL

class CaliDataset(Dataset):
    def __init__(self, path, img_shape=640):
        super().__init__()
        self.transform = transforms.Compose(
            [
                transforms.Resize((img_shape, img_shape)),
                transforms.ToTensor(),
                transforms.Normalize(mean=[0, 0, 0], std=[1, 1, 1]),
            ]
        )

        self.imgs_path = []
        self.path = path
        for img_name in os.listdir(self.path):
            img_path = os.path.join(self.path, img_name)
            self.imgs_path.append(img_path)

    def __len__(self):
        return len(self.imgs_path)

    def __getitem__(self, idx):
        img = Image.open(self.imgs_path[idx])
        if img.mode == 'L':
            img = img.convert('RGB')
        img = self.transform(img)
        return img


def report_hook(blocknum, blocksize, total):
    downloaded = blocknum * blocksize
    percent = downloaded / total * 100
    print(f"\rDownloading calibration dataset: {percent:.2f}%", end="")


def quant_yolo26n(imgsz):
    BATCH_SIZE = 32
    INPUT_SHAPE = [3, imgsz, imgsz]
    DEVICE = "cpu"
    TARGET = "esp32p4" # or "esp32s3"
    NUM_OF_BITS = 8
```

(续下页)

(接上页)

```
    yolo26n_onnx_url = "https://dl.espressif.com/public/yolo26n_o2o.onnx"
    ONNX_PATH = "yolo26n_o2o.onnx"
    urllib.request.urlretrieve(
        yolo26n_onnx_url, "yolo26n_o2o.onnx", reporthook=report_hook
    )

    ESPDL_MODLE_PATH = "yolo26n_o2o_ptq_fq_tqt_p4_640.espdl"

    yolo26n_caib_url = "https://dl.espressif.com/public/calib_yolo26n.zip"
    CALIB_DIR = "calib_yolo26n"
    urllib.request.urlretrieve(
        yolo26n_caib_url, "calib_yolo26n.zip", reporthook=report_hook
    )
    with zipfile.ZipFile("calib_yolo26n.zip", "r") as zip_file:
        zip_file.extractall("./")

    model = onnx.load(ONNX_PATH)
    sim = True
    if sim:
        model, check = simplify(model)
        assert check, "Simplified ONNX model could not be validated"
    onnx.save(onnx.shape_inference.infer_shapes(model), ONNX_PATH)

    calibration_dataset = CaliDataset(CALIB_DIR, img_shape=imgsz)
    dataloader = DataLoader(
        dataset=calibration_dataset, batch_size=BATCH_SIZE, shuffle=False, num_
↪workers=8,
    )

    def collate_fn(batch: torch.Tensor) -> torch.Tensor:
        return batch.to(DEVICE)


    # default setting
    quant_setting = QuantizationSettingFactory.espdl_setting()
    # activation calibration algo
    quant_setting.quantize_activation_setting.calib_algorithm = "percentile"  # kl -->
↪ percentile to get better mAP
    # focused logits quantization
    quant_setting.quant_config_modify = True
    # o2o
    quant_setting.quant_config_modify_setting.custom_config = {
        "/model.23/one2one_cv3.0/one2one_cv3.0.2/Conv": 0.0625,
        "/model.23/one2one_cv3.1/one2one_cv3.1.2/Conv": 0.0625,
        "/model.23/one2one_cv3.2/one2one_cv3.2.2/Conv": 0.0625,
    }

    # o2m
    # quant_setting.quant_config_modify_setting.custom_config = {
    #     "/model.23/cv3.0/cv3.0.2/Conv": 0.0625,
    #     "/model.23/cv3.1/cv3.1.2/Conv": 0.0625,
    #     "/model.23/cv3.2/cv3.2.2/Conv": 0.0625,
    # }

    # TQT
    quant_setting.tqt_optimization = True
    quant_setting.tqt_optimization_setting.collecting_device = "cpu"
```

(续下页)

(接上页)

```
    quant_setting.tqt_optimization_setting.steps = 200  #300 for o2m
    quant_setting.tqt_optimization_setting.block_size = 2
    quant_setting.tqt_optimization_setting.lr = 1e-5

    quant_ppq_graph = espdl_quantize_onnx(
        onnx_import_file=ONNX_PATH,
        espdl_export_file=ESPDL_MODLE_PATH,
        calib_dataloader=dataloader,
        calib_steps=32,
        input_shape=[1] + INPUT_SHAPE,
        target=TARGET,
        num_of_bits=NUM_OF_BITS,
        collate_fn=collate_fn,
        setting=quant_setting,
        device=DEVICE,
        error_report=True,
        skip_export=False,
        export_test_values=False,
        verbose=0,
        inputs=None,
    )
    return quant_ppq_graph


if __name__ == "__main__":
    quant_yolo26n(imgsz=640)
```

Focused Logits Quantization

目标检测模型中，与 backbone/neck 不同，**检测头 (head) 输出**对量化误差非常敏感。同时，对于类似 YOLO26n 这样的检测模型，其 classification 分支的输出 logits 为 **Sigmoid 的输入**，经过 Sigmoid 可以得到类别置信度或分数。此时，若将 logits 与其它层使用同一套校准得到的 scale，容易得到 **过大的 scale (量化步长过粗)**，导致类别分数失真，mAP 下降。

- **Sigmoid 的有效输入范围有限**：σ(x) = 1/(1+exp(-x)) 在 x 的绝对值较大时很快饱和。x 很负时输出趋近 0，很正时趋近 1，只有中间一段（大致在 -4 到 4 内）变化明显。也就是说，对输出真正有区分度的 logits 只分布在这段 **有限范围**内，超出部分进入饱和区、几乎不再变化。
- **两边都有饱和区，检测中更关注正样本**：在目标检测里，我们更关心 **正样本** (有目标、高置信度)，对应 logits 偏正、Sigmoid 输出接近 1 的一侧。但无论正负两侧，只要 logits 被量化得过粗 (scale 过大、步长太大)，多个不同的 logits 会被舍入到同一量化档位：在饱和区附近会成片地被压成 0 或 1，在中间有效区则丢失细粒度区分，导致分数失真，mAP 掉点。

因此，对 **送入 Sigmoid 的 logits** (即示例中 one2one_cv3.0/1/2.2 这三个 head 分支的最后一层 Conv 输出) 做 **Focused logits quantization**：为这几层 **单独指定更细的 scale** (如 0.0625 = 2 的 -4 次方)，在保持 Power-of-2 的前提下减小量化步长，在 Sigmoid 的有效输入范围内保留更多档位，避免过早进入饱和、并保留正样本侧的区分度，从而稳定或提升 mAP。若使用其他检测模型或不同导出结构，可根据 Sigmoid 的输入来定位对应层并调整 custom_config。

ESP32-P4 上模型量化精度测试结果

在 COCO val2017 评估 yolo26n(size=640 pixels) 量化后的精度：

配置	mAP50-95(o2m)	mAP50-95(e2e)	备注
PTQ (无 TQT)	0.342	0.332	仅校准
PTQ + TQT	0.371	0.363	校准 + TQT 微调 scale/权重

测试结果表明，无论是对 end2end 推理模型，还是非 end2end 推理模型，TQT 均可以显著提升量化后模型的 mAP。

备注： 如果追求更快的模型推理速度，可以使用 size=512 pixels。实验结果表明，在 size=512 pixels 和 end2end 推理模式下，8-bit PTQ 量化模型的 mAP50-95 为 0.315，经过 TQT 后 mAP50-95 可提升至 0.341，精度提升了 2.6 个百分点。

MobileNetV2 量化

量化脚本

```
import os
import subprocess
from typing import Iterable, Tuple, List, Tuple

import torch
from datasets.imagenet_util import (
    evaluate_ppq_module_with_imagenet,
    load_imagenet_from_directory,
)
from esp_ppq import QuantizationSettingFactory, QuantizationSetting
from esp_ppq.api import espdl_quantize_onnx, get_target_platform
from torch.utils.data import DataLoader
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data.dataset import Subset
import urllib.request
import zipfile


def quant_setting_mobilenet_v2(
    onnx_path: str,
    optim_quant_method: List[str] = None,
) -> Tuple[QuantizationSetting, str]:
    '''
    Quantize onnx model with optim_quant_method.

    Args:
        optim_quant_method (List[str]): support 'MixedPrecision_quantization',
↪'LayerwiseEqualization_quantization'
        -'MixedPrecision_quantization': if some layers in model have larger errors in␣
↪8-bit quantization, dispathching
                                        the layers to 16-bit quantization. You can␣
```

(续下页)

(接上页)

```
↪remove or add layers according to your
                                        needs.
        -'LayerwiseEqualization_quantization':  using weight equalization strategy,␣
↪which is proposed by Markus Nagel.
                                                Refer to paper https://openaccess.
↪thecvf.com/content_ICCV_2019/papers/Nagel_Data-Free_Quantization_Through_Weight_
↪Equalization_and_Bias_Correction_ICCV_2019_paper.pdf for more information.
                                                Since ReLU6 exists in MobilenetV2,␣
↪convert ReLU6 to ReLU for better precision.

    Returns:
        [tuple]: [QuantizationSetting, str]
    '''
    quant_setting = QuantizationSettingFactory.espdl_setting()
    if optim_quant_method is not None:
        if "MixedPrecision_quantization" in optim_quant_method:
            # These layers have larger errors in 8-bit quantization, dispatching to␣
↪16-bit quantization.
            # You can remove or add layers according to your needs.
            quant_setting.dispatching_table.append(
                "/features/features.1/conv/conv.0/conv.0.0/Conv",
                get_target_platform(TARGET, 16),
            )
            quant_setting.dispatching_table.append(
                "/features/features.1/conv/conv.0/conv.0.2/Clip",
                get_target_platform(TARGET, 16),
            )
        elif "LayerwiseEqualization_quantization" in optim_quant_method:
            # layerwise equalization
            quant_setting.equalization = True
            quant_setting.equalization_setting.iterations = 4
            quant_setting.equalization_setting.value_threshold = 0.4
            quant_setting.equalization_setting.opt_level = 2
            quant_setting.equalization_setting.interested_layers = None
            # replace ReLU6 with ReLU
            onnx_path = onnx_path.replace("mobilenet_v2.onnx", "mobilenet_v2_relu.onnx
↪")
        else:
            raise ValueError(
                "Please set optim_quant_method correctly. Support 'MixedPrecision_
↪quantization', 'LayerwiseEqualization_quantization'"
            )

    return quant_setting, onnx_path


def collate_fn1(x: Tuple) -> torch.Tensor:
    return torch.cat([sample[0].unsqueeze(0) for sample in x], dim=0)


def collate_fn2(batch: torch.Tensor) -> torch.Tensor:
    return batch.to(DEVICE)


def report_hook(blocknum, blocksize, total):
    downloaded = blocknum * blocksize
    percent = downloaded / total * 100
```

(续下页)

(接上页)

```
    print(f"\rDownloading calibration dataset: {percent:.2f}%", end="")


if __name__ == "__main__":
    BATCH_SIZE = 32
    INPUT_SHAPE = [3, 224, 224]
    DEVICE = "cpu"  #  'cuda' or 'cpu', if you use cuda, please make sure that cuda␣
↪is available
    TARGET = "esp32p4"  #  'c', 'esp32s3' or 'esp32p4'
    NUM_OF_BITS = 8
    ONNX_PATH = "./models/torch/mobilenet_v2.onnx"  #'models/onnx/mobilenet_v2.onnx'
    ESPDL_MODEL_PATH = "models/onnx/mobilenet_v2.espdl"
    CALIB_DIR = "./imagenet"

    # Download mobilenet_v2 model from onnx models and dataset
    imagenet_url = "https://dl.espressif.com/public/imagenet_calib.zip"
    os.makedirs(CALIB_DIR, exist_ok=True)
    if not os.path.exists("imagenet_calib.zip"):
        urllib.request.urlretrieve(
            imagenet_url, "imagenet_calib.zip", reporthook=report_hook
        )
    if not os.path.exists(os.path.join(CALIB_DIR, "calib")):
        with zipfile.ZipFile("imagenet_calib.zip", "r") as zip_file:
            zip_file.extractall(CALIB_DIR)
    CALIB_DIR = os.path.join(CALIB_DIR, "calib")

    # ------------------------------------------
    # Prepare Calibration Dataset
    # ------------------------------------------
    if os.path.exists(CALIB_DIR):
        print(f"load imagenet calibration dataset from directory: {CALIB_DIR}")
        dataset = datasets.ImageFolder(
            CALIB_DIR,
            transforms.Compose(
                [
                    transforms.Resize(256),
                    transforms.CenterCrop(224),
                    transforms.ToTensor(),
                    transforms.Normalize(
                        mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]
                    ),
                ]
            ),
        )
        dataset = Subset(dataset, indices=[_ for _ in range(0, 1024)])
        dataloader = DataLoader(
            dataset=dataset,
            batch_size=BATCH_SIZE,
            shuffle=False,
            num_workers=4,
            pin_memory=False,
            collate_fn=collate_fn1,
        )
    else:
        # Random calibration dataset only for debug
        print("load random calibration dataset")
```

(续下页)

(接上页)

```
        def load_random_calibration_dataset() -> Iterable:
            return [torch.rand(size=INPUT_SHAPE) for _ in range(BATCH_SIZE)]

        # Load training data for creating a calibration dataloader.
        dataloader = DataLoader(
            dataset=load_random_calibration_dataset(),
            batch_size=BATCH_SIZE,
            shuffle=False,
        )

    # ------------------------------------------
    # Quantize ONNX Model.
    # ------------------------------------------

    quant_setting = QuantizationSettingFactory.espdl_setting()
    # TQT
    quant_setting.tqt_optimization = True
    quant_setting.tqt_optimization_setting.collecting_device = "cpu"
    quant_setting.tqt_optimization_setting.steps = 500
    quant_setting.tqt_optimization_setting.block_size = 4
    quant_setting.tqt_optimization_setting.lr = 1e-4

    quant_ppq_graph = espdl_quantize_onnx(
        onnx_import_file=ONNX_PATH,
        espdl_export_file=ESPDL_MODEL_PATH,
        calib_dataloader=dataloader,
        calib_steps=32,
        input_shape=[1] + INPUT_SHAPE,
        target=TARGET,
        num_of_bits=NUM_OF_BITS,
        collate_fn=collate_fn2,
        setting=quant_setting,
        device=DEVICE,
        error_report=True,
        skip_export=False,
        export_test_values=False,
        verbose=1,
    )

    # ------------------------------------------
    # Evaluate Quantized Model.
    # ------------------------------------------

    evaluate_ppq_module_with_imagenet(
        model=quant_ppq_graph,
        imagenet_validation_dir=CALIB_DIR,
        batchsize=BATCH_SIZE,
        device=DEVICE,
        verbose=1,
    )
```

量化结果及分析

在采用 TQT 策略对 8-bit 模型进行量化优化的情况下，模型的 Top-1 准确率为 71.775%，相比 ESP32P4 本身的 per-channel 的量化模型（Top-1 准确率为 70.850%），TQT 在 Top-1 精度上提升了 1.306 个百分点, 与 float32 模型的精度（71.878%）更加接近，表明即使在严格的 8-bit 量化约束下，仅通过对量化阈值进行可学习优化，TQT 也能够有效缓解量化误差带来的性能下降，使量化模型的精度接近浮点模型水平。

3.11.4 常见问题

如何加速 TQT？

有 GPU 时，把校准数据放到 GPU 上可加快速度。

先将 collecting_device 设为 cuda：

```
quant_setting.tqt_optimization_setting.collecting_device = "cuda"
```

再在 ENABLE_CUDA_KERNEL 中执行量化：

```
with ENABLE_CUDA_KERNEL():
        quant_ppq_graph = espdl_quantize_onnx(
            onnx_import_file=ONNX_PATH,
            espdl_export_file=ESPDL_MODLE_PATH,
            calib_dataloader=dataloader,
            calib_steps=32,
            input_shape=[1] + INPUT_SHAPE,
            target=TARGET,
            num_of_bits=NUM_OF_BITS,
            collate_fn=collate_fn,
            setting=quant_setting,
            device=DEVICE,
            error_report=False,
            skip_export=False,
            export_test_values=False,
            verbose=0,
            inputs=None,
        )
```

同时适当增大 block_size（如 2 ~ 4）可减少块数、缩短总时间，但过大可能不稳定，建议先试 4。

可以不修改 scale 只训练模型权重吗？

可以。将 is_scale_trainable 设置为 False 即可。

```
quant_setting.tqt_optimization_setting.is_scale_trainable = False
```

TQT 可以和 Weight Equalization 一起使用吗？

可以。特别是对于 espdet_pico 这样的模型，二者可以使用可以进一步提升量化效果。

```
quant_setting.tqt_optimization_setting.equalization = True
quant_setting.tqt_optimization_setting.tqt_optimization = True
```

CHAPTER 4

API Reference

4.1 Tensor API Reference

Tensor is the fundamental data type in esp-dl, used for storing multi-type data such as int8, int16, float, etc., similar to the tensor in PyTorch. We have implemented some common tensor operations. Please refer to the following APIs for details.

4.1.1 Header File

- esp-dl/dl/tensor/include/dl_tensor_base.hpp

4.1.2 Classes

class **`ExponentInfo`**

Exponent info for per-tensor / per-channel quantization.

Per-tensor: m_exponent only, zero heap allocation. Per-channel: heap-allocated m_exponents array. Provides operator int() so existing `DL_SCALE(tensor->exponent)` code works unchanged.

A failed per-channel allocation leaves the object in the invalid state reported by *is_valid()*; such an object describes no scale at all and its tensor must be rejected by the caller.

Public Functions

inline **ExponentInfo**(int exponent = 0)

Construct a per-tensor *ExponentInfo* with a single exponent value.

参数

exponent –The exponent value for per-tensor quantization. Default is 0.

inline **ExponentInfo**(const std::vector<int> &exponents)

Construct an *ExponentInfo* from a vector of exponents.

参数

exponents –Vector of exponent values. If size <= 1, uses per-tensor mode; otherwise uses per-channel mode with heap allocation.

inline **~ExponentInfo**()

Destroy the *ExponentInfo* object and free heap-allocated memory.

inline **ExponentInfo**(const *ExponentInfo* &other)

Copy constructor.

参数

other –The *ExponentInfo* object to copy from.

inline *ExponentInfo* &**operator=**(const *ExponentInfo* &other)

Copy assignment operator.

参数

other –The *ExponentInfo* object to assign from.

返回

Reference to this object.

inline **ExponentInfo**(*ExponentInfo* &&other) noexcept

Move constructor.

参数

other –The *ExponentInfo* object to move from.

inline *ExponentInfo* &**operator=**(*ExponentInfo* &&other) noexcept

Move assignment operator.

参数

other –The *ExponentInfo* object to move from.

返回

Reference to this object.

inline *ExponentInfo* &**operator=**(int value)

Assign a single integer value as per-tensor exponent.

参数

value –The exponent value to assign.

返回

Reference to this object.

inline int **get**(int ch = -1) const

Get exponent value.

参数

ch –Channel index. -1 (default) returns per-tensor value. ch >= 0 returns per-channel value if available, otherwise per-tensor.

返回

The exponent value for the specified channel or per-tensor exponent.

inline **operator int**() const

Implicit conversion to int, returns the per-tensor exponent value.

返回

The per-tensor exponent value.

inline bool **is_per_channel**() const

Check if using per-channel quantization.

返回

true if per-channel exponents are stored, false otherwise.

inline bool **is_valid**() const

Check whether this object describes a usable scale.

返回

false only when a per-channel exponent array could not be allocated. Quantized data governed by such an object cannot be interpreted, so its tensor must be rejected.

inline int **channel_size**() const

Get the number of channels.

返回

Number of channels for per-channel mode, 1 for per-tensor mode, or 0 when invalid.

inline const int ***data**() const

Get pointer to exponent data.

返回

Pointer to per-channel exponents array if available, otherwise pointer to per-tensor exponent.

inline bool **operator==**(const *ExponentInfo* &other) const

Compare two *ExponentInfo* objects for equality.

参数

other –The *ExponentInfo* object to compare with.

返回

true if both are valid and have the same exponent values, false otherwise.

inline bool **operator!=**(const *ExponentInfo* &other) const

Compare two *ExponentInfo* objects for inequality.

参数

other –The *ExponentInfo* object to compare with.

返回

true if exponent values differ, false if equal.

class **TensorBase**

This class is designed according to PyTorch Tensor. *TensorBase* is required to ensure that the first address are aligned to 16 bytes and the memory size should be a multiple of 16 bytes.

TODO:: Implement more functions

Public Functions

TensorBase(std::vector<int> shape, const void *element, int exponent = 0, dtype_t dtype = DATA_TYPE_FLOAT, bool deep = true, uint32_t caps = MALLOC_CAP_DEFAULT)

Construct a *TensorBase* object.

参数

- **shape** –Shape of tensor
- **element** –Pointer of data
- **exponent** –Exponent of tensor, default is 0
- **dtype** –Data type of element, default is float
- **deep** –True: malloc memory and copy data, false: use the pointer directly
- **caps** –Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

TensorBase(std::vector<int> shape, const void *element, const std::vector<int> &exponents, dtype_t dtype = DATA_TYPE_FLOAT, bool deep = true, uint32_t caps = MALLOC_CAP_DEFAULT)

Construct a *TensorBase* object with per-channel exponents.

> **警告:** If the per-channel exponents cannot be allocated, the tensor is left empty (data == nullptr, exponent.is_valid() == false) and must not be used.

参数

- **shape** –Shape of tensor
- **element** –Pointer of data
- **exponents** –Per-channel exponents
- **dtype** –Data type of element, default is float
- **deep** –True: malloc memory and copy data, false: use the pointer directly
- **caps** –Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

inline virtual **~TensorBase**()

Destroy the *TensorBase* object.

inline void ***operator new**(size_t size)

Allocate the tensor object itself.

MALLOC_CAP_DEFAULT prefers PSRAM when one is present and falls back to internal RAM otherwise, so a model holding many tensors does not spend internal RAM on the objects themselves.

参数

size –Size of the object in bytes

返回

Pointer to the allocated memory

inline void **operator delete**(void *ptr)

Deallocate the tensor object.

参数

ptr –Pointer previously returned by operator new

bool **assign**(*TensorBase* *tensor)

Assign tensor to this tensor.

参数

tensor –

返回

true if assign successfully, otherwise false.

bool **assign**(std::vector<int> shape, const void *element, int exponent, dtype_t dtype)

Assign data to this tensor.

参数

- **shape** –
- **element** –
- **exponent** –
- **dtype** –

返回

true if assign successfully, otherwise false.

inline int **get_size**()

Get the size of Tensor.

返回

the size of Tensor.

inline int **get_aligned_size**()

Get the aligned size of Tensor.

返回

the aligned size of Tensor.

inline size_t **get_dtype_bytes**()

Get the dtype size, in bytes.

返回

the size of dtype.

inline const char ***get_dtype_string**()

Get the dtype string of Tensor.

返回

the string of Tensor’ s dtype.

inline int **get_bytes**()

Get the bytes of Tensor.

返回

the bytes of Tensor.

inline int **get_aligned_bytes**()

Get the bytes of Tensor.

返回

the bytes of Tensor.

inline virtual void *get_element_ptr()

Get data pointer. If *cache(preload data pointer)* is not null, return cache pointer, otherwise return data pointer.

返回
the pointer of Tensor’s data

template<typename T>
inline *T* *get_element_ptr()

Get data pointer by the specified template. If *cache(preload data pointer)* is not null, return cache pointer, otherwise return data pointer.

返回
the pointer of Tensor’s data

TensorBase &set_element_ptr(void *data)

Set the data pointer of Tensor.

参数
data –point to data memory

返回
TensorBase& self

inline const std::vector<int> &get_shape() const

Get the shape of Tensor.

Returned by reference: operators query the shape several times per forward and the copies were costing more than the work they guarded.

返回
std::vector<int> the shape of Tensor

TensorBase &set_shape(const std::vector<int> shape)

Set the shape of Tensor.

参数
shape –the shape of Tensor.

返回
Tensor.

inline int get_exponent()

Get the exponent of Tensor.

返回
int the exponent of Tensor

inline dtype_t get_dtype()

Get the data type of Tensor.

返回
dtype_t the data type of Tensor

inline uint32_t get_caps()

Get the memory flags of Tensor.

返回
uint32_t the memory flags of Tensor

TensorBase ***reshape**(std::vector<int> shape)

Change a new shape to the Tensor without changing its data.

参数

shape –the target shape

返回

TensorBase *self

template<typename **T**>
TensorBase ***flip**(const std::vector<int> &axes)

Flip the input Tensor along the specified axes.

参数

axes –the specified axes

返回

TensorBase& self

TensorBase ***transpose**(*TensorBase* *input, std::vector<int> perm = {})

Reverse or permute the axes of the input Tensor.

参数

- **input** –the input Tensor
- **perm** –the new arrangement of the dims. if perm == {}, the dims arrangement will be reversed.

返回

TensorBase *self

template<typename **T**>
TensorBase ***transpose**(*T* *input_element, std::vector<int> &input_shape, std::vector<int> &input_axis_offset, std::vector<int> &perm)

Reverse or permute the axes of the input Tensor.

参数

- **input_element** –the input data pointer
- **input_shape** –the input data shape
- **input_axis_offset** –the input data axis offset
- **perm** –the new arrangement of the dims. if perm == {}, the dims arrangement will be reversed.

返回

TensorBase *self

bool **is_same_shape**(*TensorBase* *tensor)

Check the shape is the same as the shape of input.

参数

tensor –Input tensor pointer

返回

- true: same shape
- false: not

bool **equal**(*TensorBase* *tensor, float epsilon = 1e-6, bool verbose = false)

Compare the shape and data of two Tensor.

参数

- **tensor** –Input tensor
- **epsilon** –The max error of two element
- **verbose** –If true, print the detail of results

返回

true if two tensor is equal otherwise false

TensorBase ***slice**(const std::vector<int> &start, const std::vector<int> &end, const std::vector<int> &axes = {}, const std::vector<int> &step = {})

Produces a slice of the this tensor along multiple axes.

> **警告:** The length of start, end and step must be same as the shape of input tensor

参数

- **start** –Starting indicesd
- **end** –Ending indices
- **axes** –Axes that starts and ends apply to.
- **step** –Slice step, step = 1 if step is not specified

返回

TensorBase* Output tensor pointer, created by this slice function

template<typename **T**>
TensorBase ***pad**(*T* *input_element, const std::vector<int> &input_shape, const std::vector<int> &pads, const padding_mode_t mode, *TensorBase* *const_value = nullptr)

Pad input tensor.

参数

- **input_element** –Data pointer of input tensor
- **input_shape** –Shape of input tensor
- **pads** –The number of padding elements to add, pads format should be: [x1_begin, x2_begin, ⋯, x1_end, x2_end, ⋯]
- **mode** –Supported modes: constant(default), reflect, edge
- **const_value** –(Optional) A scalar value to be used if the mode chosen is constant

返回

Output tensor pointer

TensorBase ***pad**(*TensorBase* *input, const std::vector<int> &pads, const padding_mode_t mode, *TensorBase* *const_value = nullptr)

Pad input tensor.

参数

- **input** –Input tensor pointer

- **pads** –Padding elements to add, pads format should be: [x1_begin, x2_begin, ⋯, x1_end, x2_end,⋯]
- **mode** –Supported modes: constant(default), reflect, edge
- **const_value** –(Optional) A scalar value to be used if the mode chosen is constant

返回
Output tensor pointer

template<typename **T**>
bool **compare_elements**(const *T* *gt_elements, float epsilon = 1e-6, bool verbose = false)

Compare the elements of two Tensor.

参数

- **gt_elements** –The ground truth elements
- **epsilon** –The max error of two element
- **verbose** –If true, print the detail of results

返回
true if all elements are equal otherwise false

int **get_element_index**(const std::vector<int> &axis_index)

Get the index of element.

参数
axis_index –The coordinates of element

返回
int the index of element

std::vector<int> **get_element_coordinates**(int index)

Get the coordinates of element.

参数
index –The index of element

返回
The coordinates of element

template<typename **T**>
T **get_element**(int index)

Get a element of Tensor by index.

参数
index –The index of element

返回
The element of tensor

template<typename **T**>
T **get_element**(const std::vector<int> &axis_index)

Get a element of Tensor.

参数
axis_index –The index of element

返回
The element of tensor

void **memset**(int value)

Set a element of Tensor by index.

参数

value –The value of element

void **rand**()

Fill tensor data with random bytes from hardware RNG.

Uses esp_fill_random() to fill the tensor’s internal buffer with random bytes. Requires ESP-IDF (esp_random.h).

size_t **set_preload_addr**(void *addr, size_t size)

Set preload address of Tensor.

参数

- **addr** –The address of preload data, should be aligned to 16 bytes
- **size** –Size of the preload buffer, in bytes

返回

The number of bytes claimed at addr, or 0 if the buffer is too small

inline virtual void **preload**()

Copy the data of Tensor into its preload buffer.

Call after any in-place layout rewrite (reset_bias_layout, reset_param_layout) so the buffer holds the layout the kernels expect.

void **reset_bias_layout**(quant_type_t op_quant_type, bool is_depthwise)

Reset the layout of Tensor.

> **警告:** Only available for Convolution. Don’t use it unless you know exactly what it does.

参数

- **op_quant_type** –The quant type of operation
- **is_depthwise** –Whether is depthwise convolution

void **push**(*TensorBase* *new_tensor, int dim)

Push new_tensor to current tensor. The time series dimension size of new tensor must is lesser or equal than that of the current tensor.”.

参数

- **new_tensor** –The new tensor will be pushed
- **dim** –Specify the dimension on which to perform streaming stack pushes

virtual void **print**(bool print_data = false)

print the information of *TensorBase*

参数

print_data –Whether print the data

Public Members

int **size**
: size of element including padding

std::vector<int> **shape**
: shape of Tensor

dtype_t **dtype**
: data type of element

ExponentInfo **exponent**
: exponent of element (per-tensor or per-channel)

bool **auto_free**
: free element when object destroy

std::vector<int> **axis_offset**
: element offset of each axis

void ***data**
: data pointer

void ***cache**
: cache pointer，used for preload and do not need to free

uint32_t **caps**
: flags indicating the type of memory

Public Static Functions

static void **slice**(*TensorBase* *input, *TensorBase* *output, const std::vector<int> &start, const std::vector<int> &end, const std::vector<int> &axes = {}, const std::vector<int> &step = {})

Produces a slice along multiple axes.

> **警告:** The length of start, end and step must be same as the shape of input tensor

参数

- **input** –Input Tensor
- **output** –Output Tensor
- **start** –Starting indicesd
- **end** –Ending indices
- **axes** –Axes that starts and ends apply to.
- **step** –Slice step, step = 1 if step is not specified

4.2 Module API Reference

The Module is the base class for operators in esp-dl, and all operators inherit from this base class. This base class defines the basic interfaces for operators, enabling the model layer to automatically execute operators and manage memory planning.

4.2.1 Header File

- esp-dl/dl/module/include/dl_module_base.hpp

4.2.2 Classes

class **Module**

Base class for module.

Public Functions

Module(const char *name = NULL, module_inplace_t inplace = MODULE_NON_INPLACE, quant_type_t quant_type = QUANT_TYPE_NONE)

Construct a new *Module* object.

参数

- **name** –Name of module.
- **inplace** –Inplace operation mode
- **quant_type** –Quantization type

virtual **~Module**()

Destroy the *Module* object. Return resource.

inline void ***operator new**(size_t size)

Allocate the module object itself.

MALLOC_CAP_DEFAULT prefers PSRAM when one is present and falls back to internal RAM otherwise, so a model's modules do not spend internal RAM on the objects themselves.

参数

size –Size of the object in bytes

返回

Pointer to the allocated memory

inline void **operator delete**(void *ptr)

Deallocate the module object.

参数

ptr –Pointer previously returned by operator new

inline virtual std::vector<int> **get_outputs_index**()

Get the tensor index of this module's outputs.

返回

Tensor index of model's tensors

inline virtual bool **is_lut_module**() const

Whether this module uses a lookup table (LUT) as an extra input.

LUT modules load the table from the model initializer and append it to `m_inputs_index`.

返回

true if the module is a LUT module, false otherwise

virtual std::vector<std::vector<int» **get_output_shape**(std::vector<std::vector<int» &input_shapes) = 0

Calculate output shape by input shape.

参数

input_shapes –Input shapes

返回

outputs shapes

inline virtual std::vector<WorkspaceRequest> **get_workspace_requests**(std::vector<std::vector<int» &input_shapes)

Scratch memory needed while this module's forward runs.

The memory planner places each buffer with a lifetime of this node only and records its native workspace binding ID in `m_workspace_binding_ids`. Workspace bindings are separate from model tensors. The contents are undefined on entry and are not kept between calls. A module run without the planner (e.g. through *run()*) has no workspace bound in its temporary context and must provide its own memory.

When internal RAM is planned, a buffer competes for it like a tensor unless it is `psram_only`. A zero-size buffer gets a binding ID but no memory.

参数

input_shapes –Input shapes

返回

One request per workspace buffer; empty when none is needed

virtual void **forward**(*ModelContext* *context, runtime_mode_t mode = RUNTIME_MODE_AUTO) = 0

Build the module, high-level interface for *Module* layer.

参数

- **context** –*Model* context including all inputs and outputs and other runtime information
- **mode** –Runtime mode, default is RUNTIME_MODE_AUTO

inline virtual void **forward_args**(void *args)

Run the module, Low-level interface for base layer and multi-core processing.

参数

args –ArgsType, arithArgsType, resizeArgsType and so on

inline virtual void **print**()

print module information

inline virtual void **set_preload_addr**(void *addr, size_t size)

set preload RAM pointer

参数

- **addr** –Internal RAM address, should be aligned to 16 bytes
- **size** –The size of RAM address

inline virtual void **preload**()

Perform a preload operation.

> **警告:** Not implemented

inline virtual void **reset**()

reset all state of module, include inputs， outputs and preload cache setting

virtual void **run**(*TensorBase* *input, *TensorBase* *output, runtime_mode_t mode = RUNTIME_MODE_SINGLE_CORE)

Run the module with single input and single output.

参数

- **input** –Input tensor
- **output** –Output tensor
- **mode** –Runtime mode

virtual void **run**(std::vector<dl::*TensorBase**> inputs, std::vector<dl::*TensorBase**> outputs, runtime_mode_t mode = RUNTIME_MODE_SINGLE_CORE)

Run the module by inputs and outputs.

参数

- **inputs** –Input tensors
- **outputs** –Output tensors
- **mode** –Runtime mode

Public Members

char ***name**

Name of module.

module_inplace_t **inplace**

Inplace type.

quant_type_t **quant_type**

Quantization type.

std::vector<int> **m_inputs_index**

Tensor index of model’s tensors that used for inputs.

std::vector<int> **m_outputs_index**

Tensor index of model’s tensors that used for outputs.

std::vector<WorkspaceBindingId> **m_workspace_binding_ids**

Native workspace binding IDs assigned by the memory planner.

Public Static Functions

static inline *Module* *`deserialize`(fbs::*FbsModel* *fbs_model, std::string node_name)

create module instance by node serialization information

参数

- **`fbs_model`** –Flatbuffer’s model
- **`node_name`** –The node name in model’s graph

返回

The pointer of module instance

4.2.3 Header File

- esp-dl/dl/module/include/dl_module_creator.hpp

4.2.4 Classes

class **`ModuleCreator`**

Singleton class for registering modules.

Public Types

using **`Creator`** = std::function<*Module**(fbs::*FbsModel**, std::string)>

Module creator function type.

Public Functions

inline void **`register_module`**(const std::string &op_type, *Creator* creator)

Register a module creator to the module creator map This function allows for the dynamic registration of new module types and their corresponding creator functions at runtime. By associating the module type name with the creator function, the system can flexibly create instances of various modules.

参数

- **`op_type`** –The module type name, used as the key in the map
- **`creator`** –The module creator function, used to create modules of a specific type

inline *Module* *`create`(fbs::*FbsModel* *fbs_model, const std::string &op_type, const std::string name)

Create module instance pointer.

参数

- **`fbs_model`** –Flatbuffer model pointer
- **`op_type`** –Module/Operator type
- **`name`** –*Module* name

返回

Module instance pointer

void **register_dl_modules**()

Pre-register the already implemented modules Defined in dl_module_creator.cpp; the gated module includes live in dl_module_includes.inc and the registration rows in dl_module_register.inc, both generated from spec/ops.yml.

inline void **print**()

Print all modules has been registered.

inline void **clear**()

Clear all modules has been registered.

Public Static Functions

static inline *ModuleCreator* ***get_instance**()

Get instance of *ModuleCreator* by this function. It is only safe method to get instance of *ModuleCreator* because *ModuleCreator* is a singleton class.

返回

ModuleCreator instance pointer

4.3 Model API Reference

This section covers model loading and static memory planning, making it convenient for users to directly load and run ESPDL models.

4.3.1 Header File

- esp-dl/dl/model/include/dl_model_base.hpp

4.3.2 Macros

DL_LOG_INFER_LATENCY_INIT_WITH_SIZE(size)

DL_LOG_INFER_LATENCY_INIT()

DL_LOG_INFER_LATENCY_START()

DL_LOG_INFER_LATENCY_END()

DL_LOG_INFER_LATENCY_PRINT(prefix, key)

DL_LOG_INFER_LATENCY_END_PRINT(prefix, key)

DL_LOG_INFER_LATENCY_ARRAY_INIT_WITH_SIZE(n, size)

DL_LOG_INFER_LATENCY_ARRAY_INIT(n)

DL_LOG_INFER_LATENCY_ARRAY_START(i)

DL_LOG_INFER_LATENCY_ARRAY_END(i)

DL_LOG_INFER_LATENCY_ARRAY_PRINT(i, prefix, key)

DL_LOG_INFER_LATENCY_ARRAY_END_PRINT(i, prefix, key)

4.3.3 Classes

class **Model**

Neural Network *Model*.

Public Functions

inline bool **is_built**() const

Whether the current execution plan has successfully bound its variable arenas. A failed rebuild leaves the model safely non-runnable.

返回

true when the model is ready to run, otherwise false.

Model(const char *rodata_address_or_partition_label_or_path, fbs::model_location_type_t location = fbs::MODEL_LOCATION_IN_FLASH_RODATA, int max_internal_size = 0, memory_manager_t mm_type = MEMORY_MANAGER_GREEDY, const uint8_t *key = nullptr, bool param_copy = true, const std::map<std::string, std::vector<int» &input_shapes = {})

Create the *Model* object by rodata address or partition label.

参数

- **rodata_address_or_partition_label_or_path** –The address of model data while location is MODEL_LOCATION_IN_FLASH_RODATA. The label of partition while location is MODEL_LOCATION_IN_FLASH_PARTITION. The path of model while location is MODEL_LOCATION_IN_SDCARD.
- **location** –The model location.
- **max_internal_size** –In bytes. Limit the max internal size usage. Only take effect when there's a PSRAM, and you want to alloc memory on internal RAM first.
- **mm_type** –Type of memory manager
- **key** –The key of encrypted model.
- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.
- **input_shapes** –Optional map to override graph input shapes. The key is the graph input name and the value is the shape. If empty, the shapes stored in the model are used.

Model(const char *rodata_address_or_partition_label_or_path, int model_index, fbs::model_location_type_t location = fbs::MODEL_LOCATION_IN_FLASH_RODATA, int max_internal_size = 0, memory_manager_t mm_type = MEMORY_MANAGER_GREEDY, const uint8_t *key = nullptr, bool param_copy = true, const std::map<std::string, std::vector<int» &input_shapes = {})

Create the *Model* object by rodata address or partition label.

参数

- **rodata_address_or_partition_label_or_path** –The address of model data while location is MODEL_LOCATION_IN_FLASH_RODATA. The label of partition while location is MODEL_LOCATION_IN_FLASH_PARTITION. The path of model while location is MODEL_LOCATION_IN_SDCARD.

- **model_index** –The model index of packed models.
- **location** –The model location.
- **max_internal_size** –In bytes. Limit the max internal size usage. Only take effect when there's a PSRAM, and you want to alloc memory on internal RAM first.
- **mm_type** –Type of memory manager
- **key** –The key of encrypted model.
- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.
- **input_shapes** –Optional map to override graph input shapes. The key is the graph input name and the value is the shape. If empty, the shapes stored in the model are used.

Model(const char *rodata_address_or_partition_label_or_path, const char *model_name, fbs::model_location_type_t location = fbs::MODEL_LOCATION_IN_FLASH_RODATA, int max_internal_size = 0, memory_manager_t mm_type = MEMORY_MANAGER_GREEDY, const uint8_t *key = nullptr, bool param_copy = true, const std::map<std::string, std::vector<int» &input_shapes = {})

Create the *Model* object by rodata address or partition label.

参数

- **rodata_address_or_partition_label_or_path** –The address of model data while location is MODEL_LOCATION_IN_FLASH_RODATA. The label of partition while location is MODEL_LOCATION_IN_FLASH_PARTITION. The path of model while location is MODEL_LOCATION_IN_SDCARD.
- **model_name** –The model name of packed models.
- **location** –The model location.
- **max_internal_size** –In bytes. Limit the max internal size usage. Only take effect when there's a PSRAM, and you want to alloc memory on internal RAM first.
- **mm_type** –Type of memory manager
- **key** –The key of encrypted model.
- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.
- **input_shapes** –Optional map to override graph input shapes. The key is the graph input name and the value is the shape. If empty, the shapes stored in the model are used.

Model(fbs::*FbsModel* *fbs_model, int internal_size = 0, memory_manager_t mm_type = MEMORY_MANAGER_GREEDY, const std::map<std::string, std::vector<int» &input_shapes = {})

Create the *Model* object by fbs_model.

参数

- **fbs_model** –The fbs model.

- **internal_size** –Internal ram size, in bytes
- **mm_type** –Type of memory manager
- **input_shapes** –Optional map to override graph input shapes. The key is the graph input name and the value is the shape. If empty, the shapes stored in the model are used.

virtual **~Model**()

Destroy the *Model* object.

virtual esp_err_t **load**(const char *rodata_address_or_partition_label_or_path, fbs::model_location_type_t location = fbs::MODEL_LOCATION_IN_FLASH_RODATA, const uint8_t *key = nullptr, bool param_copy = true)

Load model graph and parameters from FLASH or sdcard.

参数

- **rodata_address_or_partition_label_or_path** –The address of model data while location is MODEL_LOCATION_IN_FLASH_RODATA. The label of partition while location is MODEL_LOCATION_IN_FLASH_PARTITION. The path of model while location is MODEL_LOCATION_IN_SDCARD.
- **location** –The model location.
- **key** –The key of encrypted model.
- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.

返回

- ESP_OK Success
- ESP_FAIL Failed

virtual esp_err_t **load**(const char *rodata_address_or_partition_label_or_path, fbs::model_location_type_t location = fbs::MODEL_LOCATION_IN_FLASH_RODATA, int model_index = 0, const uint8_t *key = nullptr, bool param_copy = true)

Load model graph and parameters from FLASH or sdcard.

参数

- **rodata_address_or_partition_label_or_path** –The address of model data while location is MODEL_LOCATION_IN_FLASH_RODATA. The label of partition while location is MODEL_LOCATION_IN_FLASH_PARTITION. The path of model while location is MODEL_LOCATION_IN_SDCARD.
- **location** –The model location.
- **model_index** –The model index of packed models.
- **key** –The key of encrypted model.
- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.

返回

- ESP_OK Success
- ESP_FAIL Failed

virtual esp_err_t **load**(const char *rodata_address_or_partition_label_or_path, fbs::model_location_type_t location = fbs::MODEL_LOCATION_IN_FLASH_RODATA, const char *model_name = nullptr, const uint8_t *key = nullptr, bool param_copy = true)

Load model graph and parameters from FLASH or sdcard.

参数

- **rodata_address_or_partition_label_or_path** –The address of model data while location is MODEL_LOCATION_IN_FLASH_RODATA. The label of partition while location is MODEL_LOCATION_IN_FLASH_PARTITION. The path of model while location is MODEL_LOCATION_IN_SDCARD.
- **location** –The model location.
- **model_name** –The model name of packed models.
- **key** –The key of encrypted model.
- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.

返回

- ESP_OK Success
- ESP_FAIL Failed

virtual esp_err_t **load**(fbs::*FbsModel* *fbs_model)

Load model graph and parameters from Flatbuffers model.

参数

fbs_model –The FlatBuffers model

返回

- ESP_OK Success
- ESP_FAIL Failed

virtual void **build**(size_t max_internal_size, memory_manager_t mm_type = MEMORY_MANAGER_GREEDY, bool preload = false, const std::map<std::string, std::vector<int» &input_shapes = {})

Allocate memory for the model.

参数

- **max_internal_size** –In bytes. Limit the max internal size usage. Only take effect when there’s a PSRAM, and you want to alloc memory on internal RAM first.
- **mm_type** –Type of memory manager
- **preload** –Whether to preload the model’s parameters to internal ram (not implemented yet)

- **input_shapes** –Optional map to override graph input shapes. The key is the graph input name and the value is the shape. If empty, the shapes stored in the model are used.

virtual void **run**(runtime_mode_t mode = RUNTIME_MODE_SINGLE_CORE)

Run the model module by module.

参数

mode –Runtime mode.

virtual void **run**(*TensorBase* *input, runtime_mode_t mode = RUNTIME_MODE_SINGLE_CORE)

Run the model module by module.

参数

- **input** –The model input.
- **mode** –Runtime mode.

virtual void **run**(std::map<std::string, *TensorBase**> &user_inputs, runtime_mode_t mode = RUNTIME_MODE_SINGLE_CORE, std::map<std::string, *TensorBase**> user_outputs = {})

Run the model module by module.

参数

- **user_inputs** –The model inputs.
- **mode** –Runtime mode.
- **user_outputs** –It's for debug to specify the output of the intermediate layer; Under normal use, there is no need to pass a value to this parameter. If no parameter is passed, the default is the graphical output, which can be obtained through *Model::get_outputs()*.

virtual void **run**(int stage_num, int stage_index, runtime_mode_t mode = RUNTIME_MODE_SINGLE_CORE)

Run a single stage of the model.

The execution plan is split into `stage_num` near-equal segments and only the segment indexed by `stage_index` is executed. The stage progress is managed by the caller (no state is kept inside the model), so a full inference is done by calling this for stage_index from 0 to stage_num - 1. Inputs/outputs are handled the same way as the other *run()* overloads (assign inputs before running stage 0, read outputs after running the last stage).

参数

- **stage_num** –Total number of stages to split the execution plan into (>= 1).
- **stage_index** –Index of the stage to execute, in range [0, stage_num).
- **mode** –Runtime mode.

void **minimize**()

Minimize the model.

esp_err_t **test**()

Test whether the model inference result is correct. The model should contain test_inputs and test_outputs. Enable export_test_values option in esp-ppq to use this api. The comparison epsilon is read from model metadata `test_max_error` (default 5e-4).

返回

esp_err_t

std::map<std::string, mem_info_t> **get_memory_info**()

Get memory info.

返回

Memory usage statistics on internal and PSRAM.

std::map<std::string, module_info> **get_module_info**()

Get module info.

返回

return Type and latency of each module.

void **print_module_info**(const std::map<std::string, module_info> &info, bool sort_module_by_latency = false)

Print the module info obtained by get_module_info function.

参数

- **info** –
- **sort_module_by_latency** –

void **profile_memory**()

Print model memory summary.

void **profile_module**(bool sort_module_by_latency = false)

Print module info summary. (Name, Type, Latency)

参数

sort_module_by_latency –True The module is printed in latency decreasing sort. False The module is printed in ONNX topological sort.

void **profile**(bool sort_module_by_latency = false)

Combination of profile_memory & profile_module.

参数

sort_module_by_latency –True The module is printed in latency decreasing sort. False The module is printed in ONNX topological sort.

virtual std::map<std::string, *TensorBase**> &**get_inputs**()

Get inputs of model.

返回

The map of model input's name and TensorBase*

virtual *TensorBase* ***get_input**()

Get the only input of model.

返回

TensorBase*

virtual *TensorBase* ***get_input**(const std::string &name)

Get input of model by name.

参数

name –input name

返回

TensorBase*

virtual *TensorBase* ***get_intermediate***(const std::string &name)

Get intermediate *TensorBase* of model.

备注： When using memory manager, the content of *TensorBase*' s data may be overwritten by the outputs of other

---

参数

name –The name of intermediate Tensor. operators.

返回

The intermediate TensorBase*.

virtual std::map<std::string, *TensorBase**> &***get_outputs***()

Get outputs of model.

返回

The map of model output' s name and TensorBase*

virtual *TensorBase* ***get_output***()

Get the only output of model.

返回

TensorBase*

virtual *TensorBase* ***get_output***(const std::string &name)

Get output of model by name.

参数

name –output name

返回

TensorBase*

std::string ***get_metadata_prop***(const std::string &key)

Get the model' s metadata prop.

参数

key –The key of metadata prop

返回

The value of metadata prop

virtual void ***print***()

Print the model.

virtual void ***reset***()

Reset the model. This function resets the model context and all modules, but does not reload the model or rebuild the execution plan.

inline virtual fbs::*FbsModel* ***get_fbs_model***()

Get the fbs model instance.

返回

fbs::FbsModel *

4.3.4 Header File

- esp-dl/dl/model/include/dl_model_context.hpp

4.3.5 Macros

CONTEXT_PARAMETER_OFFSET

Offset for parameter tensors

4.3.6 Classes

class **ModelContext**

Model Context class including variable tensors and parameters.

Public Functions

inline **ModelContext**()

Constructor for *ModelContext*. Initializes the PSRAM and internal root pointers to nullptr.

inline **~ModelContext**()

Destructor for *ModelContext*. Clears all resources and tensors.

int **add_tensor**(const std::string name, bool is_paramter = false, *TensorBase* *tensor = nullptr)

Adds a tensor to the parameter or variable list.

参数

- **name** –The name of the tensor.
- **is_paramter** –Whether the tensor is a parameter (default: false).
- **tensor** –Pointer to the *TensorBase* object (default: nullptr).

返回

int Returns the index of the added tensor.

int **push_back_tensor**(*TensorBase* *tensor, bool is_paramter = false)

Push back a tensor.

参数

- **tensor** –Pointer to the *TensorBase* object.
- **is_paramter** –Whether the tensor is a parameter (default: false).

返回

int Returns the index of the added tensor.

void **update_tensor**(int index, *TensorBase* *tensor)

Updates the tensor at the specified index.

参数

- **index** –The index of the tensor to update.
- **tensor** –Pointer to the new *TensorBase* object.

WorkspaceBindingId **create_workspace_binding**(const std::string &name)

Creates an empty native workspace binding.

Workspace bindings are not tensors and do not affect variable tensor indices or counts.

参数

name –Workspace name, used for memory reports.

返回

WorkspaceBindingId Stable ID for the binding.

void **bind_workspace**(WorkspaceBindingId binding_id, void *data, size_t size)

Binds a workspace ID to a region in a planner-owned root arena.

参数

- **binding_id** –Workspace binding ID.
- **data** –Workspace data pointer.
- **size** –Workspace capacity in bytes.

void ***get_bound_workspace**(WorkspaceBindingId binding_id, size_t &size)

Gets the arena region currently bound to a workspace ID.

参数

- **binding_id** –Workspace binding ID.
- **size** –Receives the workspace capacity, or zero for an invalid or unbound binding.

返回

void* Workspace data pointer, or nullptr for an invalid or unbound binding.

inline int **get_workspace_binding_count**() const

Gets the number of native workspace bindings.

返回

Number of workspace bindings.

inline const std::vector<*WorkspaceBinding*> &**get_workspace_bindings**() const

Gets all native workspace bindings. Bytes are already part of the variable arena.

返回

Read-only workspace binding list.

inline void **clear_workspace_bindings**()

Drops workspace binding metadata without freeing planner root arenas.

inline void **reset_variable_arena**()

Drops variable tensor views and their planner-owned arenas while preserving graph indices, parameter tensors, and name mappings.

Used before a model is rebuilt for a new set of input shapes. Each variable slot remains in place as nullptr until the memory manager binds its replacement view.

TensorBase ***get_tensor**(int index)

Gets the tensor by its index.

参数

index –The index of the tensor.

返回

TensorBase* Returns the pointer to the *TensorBase* object, or nullptr if the index is invalid.

TensorBase ***get_tensor**(const std::string &name)*

Gets the tensor by its name.

参数

name –The name of the tensor.

返回

TensorBase* Returns the pointer to the *TensorBase* object, or nullptr if the name is not found.

int **get_tensor_index**(const std::string &name)

Gets the tensor index by its name.

参数

name –The name of the tensor.

返回

int Returns index if the name is found, else -1

inline int **find_tensor_index**(const std::string &name) const

Silent variant of `get_tensor_index`: returns -1 without logging an error when the tensor name is not yet registered. Use this on hot paths (e.g. per-op input loop) where a miss is expected.

参数

name –The name of the tensor.

返回

int Returns index if the name is found, else -1

bool **has_tensor**(const std::string &name) const

Checks if a tensor exists by its name.

参数

name –The name of the tensor.

返回

bool Returns true if the tensor exists, false otherwise.

int **get_variable_index**(const std::string &name)

Gets the variable tensor index by its name.

参数

name –The name of the tensor.

返回

int Returns index if the name is found and is variable tensor, else -1

inline int **get_variable_count**()

Gets the count of variable tensors.

返回

int Returns the number of variable tensors.

inline int **get_parameter_count**()

Gets the count of parameter tensors.

返回

int Returns the number of parameter tensors.

bool **root_alloc**(size_t internal_size, size_t psram_size, int alignment = 16)

Allocates memory for PSRAM and internal roots.

参数

- **internal_size** –The size of the internal memory in bytes.
- **psram_size** –The size of the PSRAM memory in bytes.
- **alignment** –The alignment of the memory in bytes.

返回
Bool Return true if the allocation is successful, false otherwise.

inline void ***get_psram_root**()
Gets the pointer to the PSRAM root.

返回
Void* Returns the pointer to the PSRAM root.

inline void ***get_internal_root**()
Gets the pointer to the internal root.

返回
Void* Returns the pointer to the internal root.

size_t **get_parameter_memory_size**(mem_info_t &mem_info, bool copy)
Gets the size of the parameters in bytes.

参数

- **mem_info** –The size of the memory used by the parameters in bytes, filtered by copy option.
- **copy** –Filter the parameters by auto_free.

返回
size_t Returns the total size of the parameters memory in bytes.

size_t **get_variable_memory_size**(mem_info_t &mem_info)
Get the variable memory size object.

参数
mem_info –The size of the memory used by the variables in bytes.

返回
size_t Returns the total size of the variables memory in bytes.

inline void **root_free**()
Frees the memory allocated for PSRAM and internal roots. This function ensures proper cleanup of allocated memory.

inline void **root_reset**()
Resets the context by clearing variables, parameters, and name-to-index map. This is used to reset the state of the context for a new inference or after an inference.

inline void **minimize**()
Minimizes the context by clearing the name-to-index map. This is used to free unnecessary intermediate variables during the inference.

inline void **release_tensors**()
Drops all tensor references without deleting the tensors. Used by contexts that only borrow tensors owned by the caller.

inline void **clear**()
Clears all resources and tensors in the context. This includes clearing variables, parameters, name-to-index map, and freeing memory.

Public Members

std::vector<*TensorBase**> **m_variables**

Variable tensors of model, the first one is nullptr

std::vector<*TensorBase**> **m_parameters**

Parameters of model, the first one is nullptr

struct **WorkspaceBinding**

Non-owning view of a planner-managed native workspace.

Public Members

std::string **name**

"<node>:workspace<j>"

void ***data**

Pointer into a planner-owned root arena

size_t **size**

Workspace capacity in bytes

4.3.7 Header File

- esp-dl/dl/model/include/dl_memory_manager.hpp

4.3.8 Classes

class **MemoryManagerBase**

Memory manager base class, each model has its own memory manager TODO: share memory manager with different models.

Subclassed by *dl::MemoryManagerGreedy*

Public Functions

inline **MemoryManagerBase**(int alignment = 16)

Construct a new Memory Manager Base object.

参数

alignment –Memory address alignment

inline virtual **~MemoryManagerBase**()

Destroy the MemoryManager object. Return resource.

virtual bool **alloc**(fbs::*FbsModel* *fbs_model, std::vector<dl::module::*Module**> &execution_plan, *ModelContext* *context, const std::map<std::string, std::vector<int» &input_shapes = {}) = 0

Allocate memory for each tensor, include all input and output tensors.

参数

- **fbs_model** –FlatBuffer’ s *Model*
- **execution_plan** –Topological sorted module list
- **context** –*Model* context
- **input_shapes** –Optional map to override graph input shapes. The key is the graph input name and the value is the shape. If empty, the shapes stored in fbs_model are used.

返回

Bool Return true if the allocation is successful, false otherwise.

Public Members

int **alignment**

The root pointer needs to be aligned must be a power of two

class **MemoryInfo**

Common lifetime and placement metadata for planner-managed memory.

Both tensors and native workspaces use this representation so they can share the same greedy arena and lifetime-reuse algorithm.

Subclassed by *dl::TensorInfo*, *dl::WorkspaceInfo*

Public Functions

MemoryInfo(const std::string &name, int time_begin, int time_end, size_t size, bool is_internal = false)

Construct planner metadata for a tensor or native workspace.

参数

- **name** –Allocation name used in planner diagnostics.
- **time_begin** –First execution step that uses the allocation.
- **time_end** –Last execution step that uses the allocation.
- **size** –Allocation size in bytes.
- **is_internal** –Whether the allocation is initially assigned to internal RAM.

void **update_time**(int new_time)

Extend the allocation lifetime to include a new execution step.

参数

new_time –New lifetime end; -1 marks a persistent allocation.

inline bool **is_inplaced**()

Check whether this allocation shares storage with a leader.

返回

true when an in-place leader is assigned, otherwise false.

inline virtual bool **is_psram_only**() const

Check whether the allocation must stay in PSRAM.

返回

true when internal RAM placement is forbidden, otherwise false.

inline uint32_t **get_offset**()

Get the offset in the planner's PSRAM arena.

返回

PSRAM arena offset in bytes.

inline void **set_offset**(uint32_t offset)

Set the offset in the planner's PSRAM arena.

参数

offset –PSRAM arena offset in bytes.

inline uint32_t **get_internal_offset**()

Get the offset in the planner's internal-RAM arena.

返回

Internal-RAM arena offset in bytes.

inline bool **get_internal_state**()

Check whether the allocation is assigned to internal RAM.

返回

true for internal RAM, otherwise false.

inline void **set_internal_state**(bool is_internal)

Set whether the allocation is assigned to internal RAM.

参数

is_internal –true for internal RAM, false for PSRAM.

inline void **set_internal_offset**(uint32_t offset)

Assign an offset in the internal-RAM arena.

参数

offset –Internal-RAM arena offset in bytes.

inline int **get_time_end**()

Get the last execution step in the allocation lifetime.

返回

Lifetime end, or -1 for a persistent allocation.

inline int **get_time_begin**()

Get the first execution step in the allocation lifetime.

返回

Lifetime begin.

inline size_t **get_size**()

Get the allocation size.

返回

Allocation size in bytes.

inline std::string **get_name**()

Get the planner allocation name.

返回

Allocation name.

inline void **print**()

Print lifetime, size, and arena offsets for diagnostics.

class **TensorInfo** : public dl::*MemoryInfo*

Tensor-specific planner metadata.

Public Functions

TensorInfo(std::string &name, int time_begin, int time_end, std::vector<int> shape, dtype_t dtype, int exponent, bool is_internal = false)

Construct a new Tensor Info object.

参数

- **name** –Tensor name
- **time_begin** –Tensor lifetime begin
- **time_end** –Tensor lifetime end
- **shape** –Tensor shape
- **dtype** –Tensor dtype
- **exponent** –Tensor exponent
- **is_internal** –Is tensor in internal RAM or not

inline **~TensorInfo**()

Destroy the Tensor Info object.

void **set_inplace_leader_tensor**(*TensorInfo* *tensor)

Set the inplace leader tensor object.

参数

tensor –Inplace leader tensor

inline void **set_inplace_follower_dirty_tensor**(*TensorInfo* *tensor)

Set the inplace follower dirty tensor object.

参数

tensor –Inplace follower dirty tensor

inline void **set_inplace_follower_clean_tensor**(*TensorInfo* *tensor)

Set the inplace follower clean tensor object.

参数

tensor –Inplace follower clean tensor

inline std::pair<*TensorInfo**, *TensorInfo**> **get_inplace_follower_tensor**()

Get the inplace follower tensor object.

返回

std::pair<TensorInfo *, TensorInfo *>

TensorBase *`create_tensor`(void *internal_root, void *psram_root)

Create a *TensorBase* view over the planner-owned tensor storage.

参数

- **`internal_root`** –Base address of the internal-RAM arena.
- **`psram_root`** –Base address of the PSRAM arena.

返回

TensorBase view bound to the selected planner arena.

inline std::vector<int> **`get_shape`**()

Get the tensor shape.

返回

std::vector<int>

class **`WorkspaceInfo`** : public dl::*MemoryInfo*

Native workspace planner metadata.

Public Functions

inline **`WorkspaceInfo`**(const std::string &name, int time_begin, int time_end, size_t size, WorkspaceBindingId workspace_binding_id, bool psram_only)

Construct metadata for a module-native workspace.

参数

- **`name`** –Workspace name used in planner diagnostics.
- **`time_begin`** –First execution step that uses the workspace.
- **`time_end`** –Last execution step that uses the workspace.
- **`size`** –Workspace size in bytes.
- **`workspace_binding_id`** –Binding used to publish the planned address.
- **`psram_only`** –Whether internal RAM placement is forbidden.

inline WorkspaceBindingId **`get_workspace_binding_id`**() const

Get the model-context binding for this workspace.

返回

Workspace binding identifier.

inline virtual bool **`is_psram_only`**() const override

Check whether the workspace must stay in PSRAM.

返回

true when internal RAM placement is forbidden, otherwise false.

void *`get_data`(void *internal_root, void *psram_root)

Resolve the workspace address from the planner root arenas.

参数

- **`internal_root`** –Base address of the internal-RAM arena.
- **`psram_root`** –Base address of the PSRAM arena.

返回
Pointer to the planned workspace storage.

class **MemoryChunk**

Memory chunk used to simulate allocation of tensors and workspaces.

Public Functions

MemoryChunk(size_t size, int is_free, int alignment = 16)

Construct a new Memory Chunk object.

参数

- **size** –Memory chunk size
- **is_free** –Whether free or not
- **alignment** –Memory chunk alignment

MemoryChunk(*MemoryInfo* *memory, int alignment = 16)

Construct a new Memory Chunk object.

参数

- **memory** –Planner allocation metadata
- **alignment** –Memory chunk alignment

inline **~MemoryChunk**()

Destroy the Memory Chunk object.

MemoryChunk ***merge_free_chunk**(*MemoryChunk* *chunk)

Merge continuous free chunk.

参数
chunk –

返回
MemoryChunk*

MemoryChunk ***insert**(*MemoryInfo* *memory)

Insert a planner allocation into a free chunk.

参数
memory –

返回
MemoryChunk*

MemoryChunk ***extend**(*MemoryInfo* *memory)

Extend free chunk and insert a planner allocation.

参数
memory –

返回
MemoryChunk*

inline void **free**()

Free memory chunk.

Public Members

size_t **size**

Memory chunk size

bool **is_free**

Whether memory chunk is free or not

int **offset**

Offset relative to root pointer

int **alignment**

Memory address alignment

MemoryInfo ***memory**

Info of the planner allocation occupying this chunk

4.3.9 Header File

- esp-dl/dl/model/include/dl_memory_manager_greedy.hpp

4.3.10 Classes

class **MemoryManagerGreedy** : public dl::*MemoryManagerBase*

Greedy memory manager that allocates memory for tensors in execution order, prioritizing internal RAM allocation first.

Public Functions

inline **MemoryManagerGreedy**(int max_internal_size, int alignment = 16)

Constructs a greedy memory manager with specified constraints.

参数

- **max_internal_size** –Maximum allowed internal RAM usage in bytes
- **alignment** –Memory address alignment requirement (default: 16 bytes)

inline **~MemoryManagerGreedy**()

Destructor that releases all managed memory resources.

virtual bool **alloc**(fbs::*FbsModel* *fbs_model, std::vector<dl::module::*Module**> &execution_plan, *ModelContext* *context, const std::map<std::string, std::vector<int» &input_shapes = {}) override

Allocates memory for all network tensors following greedy strategy.

参数

- **fbs_model** –FlatBuffer model containing network architecture

- **execution_plan** –Execution graph ordered by computation dependencies
- **context** –Device-specific runtime configuration
- **input_shapes** –Optional map to override shapes of graph inputs for allocation

返回

bool True if successful allocation, false if memory insufficient

void **free**()

Releases all allocated memory including tensor buffers and memory pools.

4.4 Fbs API Reference

The esp-dl model utilizes FlatBuffers to store information about parameters and the computation graph. Taking into account the encryption requirements of some models, this part has not been open-sourced. However, we provide a set of APIs to facilitate users in loading and parsing esp-dl models.

4.4.1 Header File

- esp-dl/fbs_loader/include/fbs_loader.hpp

4.4.2 Classes

class **FbsLoader**

Class for parser the flatbuffers.

Public Functions

FbsLoader(const char *rodata_address_or_partition_label_or_path = nullptr, model_location_type_t location = MODEL_LOCATION_IN_FLASH_RODATA)

Construct a new *FbsLoader* object.

参数

- **rodata_address_or_partition_label_or_path** –The address of model data while location is MODEL_LOCATION_IN_FLASH_RODATA. The label of partition while location is MODEL_LOCATION_IN_FLASH_PARTITION. The path of model while location is MODEL_LOCATION_IN_SDCARD.
- **location** –The model location.

~FbsLoader()

Destroy the *FbsLoader* object.

FbsModel ***load**(const uint8_t *key = nullptr, bool param_copy = true)

Load the model. If there are multiple sub-models, the first sub-model will be loaded.

参数

- **key** –NULL or a 128-bit AES key, like {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f}

- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.

返回

Return nullptr if loading fails. Otherwise return the pointer of *FbsModel*.

FbsModel ***load**(const int model_index, const uint8_t *key = nullptr, bool param_copy = true)

Load the model by model index.

参数

- **model_index** –The index of model.
- **key** –NULL or a 128-bit AES key, like {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f}.
- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.

返回

Return nullptr if loading fails. Otherwise return the pointer of *FbsModel*.

FbsModel ***load**(const char *model_name, const uint8_t *key = nullptr, bool param_copy = true)

Load the model by model name.

参数

- **model_name** –The name of model.
- **key** –NULL or a 128-bit AES key, like {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f}
- **param_copy** –Set to false to avoid copy model parameters from FLASH to PSRAM. Only set this param to false when your PSRAM resource is very tight. This saves PSRAM and sacrifices the performance of model inference because the frequency of PSRAM is higher than FLASH. Only takes effect when MODEL_LOCATION_IN_FLASH_RODATA(CONFIG_SPIRAM_RODATA not set) or MODEL_LOCATION_IN_FLASH_PARTITION.

返回

Return nullptr if loading fails. Otherwise return the pointer of *FbsModel*.

int **get_model_num**()

Get the number of models.

返回

The number of models

void **list_models**()

List all model’s name.

std::string **get_model_name**(int model_index)

Get the name of a packed model by its index.

备注： Only valid for packed multi-model formats (PDL1/PDL2/PDL3).

参数

model_index –The index of the model in the packed package.

返回

The model entry name, or an empty string when the flatbuffers is empty, the index is out of range, or the loaded file is not a packed multi-model format.

const char ***get_model_location_string**()

Get the model location string.

返回

The model location string.

fbs_file_format_t **get_model_format**()

Get the format of the loaded model/package.

返回

The format of the model, or FBS_FILE_FORMAT_UNK if the flatbuffers is empty or the format is unrecognized.

esp_err_t **get_package_version**(char *out_version, size_t out_size)

Get the package version string of a PDL3 package.

备注： Only valid for the PDL3 format.

参数

- **out_version** –Output buffer that receives the '\0' terminated version string.
- **out_size** –Size of the output buffer in bytes. Must be at least 16 bytes to hold the full field.

返回

ESP_OK on success. ESP_ERR_NOT_SUPPORTED if the package is not PDL3. ESP_ERR_INVALID_ARG / ESP_ERR_INVALID_SIZE on bad arguments.

uint32_t **get_package_size**()

Get the package_size field of a PDL3 package.

备注： Only valid for the PDL3 format.

返回

The valid byte count of the PDL3 package, or 0 if the package is not PDL3.

esp_err_t **get_package_sha256**(uint8_t out_sha256[32])

Get the package_sha256 field stored in a PDL3 package header.

备注： Only valid for the PDL3 format. This returns the digest stored in the package, it does not recompute it.

参数

out_sha256 –Output buffer that receives the 32-byte digest.

返回

ESP_OK on success. ESP_ERR_NOT_SUPPORTED if the package is not PDL3. ESP_ERR_INVALID_ARG on bad arguments.

esp_err_t **calc_package_sha256**(uint8_t out_sha256[32])

Recompute the SHA256 digest of a PDL3 package.

The digest is computed over the package bytes [0, package_size), where the 32 bytes of the package_sha256 field are treated as all zeros. Only the PDL3 header and the data within package_size are read.

参数

out_sha256 –Output buffer that receives the recomputed 32-byte digest.

返回

ESP_OK on success. ESP_ERR_NOT_SUPPORTED if the package is not PDL3. ESP_ERR_INVALID_ARG on bad arguments.

bool **verify_package_sha256**()

Verify the integrity of a PDL3 package.

Recomputes the SHA256 digest (see calc_package_sha256) and compares it with the package_sha256 field stored in the header.

返回

true if the package is PDL3 and the recomputed digest matches the stored digest, false otherwise.

4.4.3 Header File

- esp-dl/fbs_loader/include/fbs_model.hpp

4.4.4 Classes

class **FbsModel**

Flatbuffer model object.

Public Functions

FbsModel(const void *data, size_t size, model_location_type_t location, bool encrypt, bool rodata_move, bool auto_free, bool param_copy)

Construct a new *FbsModel* object.

参数

- **data** –The data of model flatbuffers.
- **size** –The size of model flatbuffers in bytes.
- **location** –The location of model flatbuffers.
- **encrypt** –Whether the model flatbuffers is encrypted or not.
- **rodata_move** –Whether the model flatbuffers is moved from FLASH rodata to PSRAM.
- **auto_free** –Whether to free the model flatbuffers data when destroy this class instance.
- **param_copy** –Whether to copy the parameter in flatbuffers.

~FbsModel()

Destroy the *FbsModel* object.

void **print**()

Print the model information.

std::vector<std::string> **topological_sort**()

Return vector of node name in the order of execution.

返回

topological sort of node name.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, int &ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回

esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, float &ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回

esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, std::string &ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回

esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, std::vector<int> &ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回

esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, std::vector<float> &ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回

esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, dl::quant_type_t &ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回

esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, dl::activation_type_t &ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回
esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, dl::resize_mode_t &ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回
esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_attribute**(std::string node_name, std::string attribute_name, dl::*TensorBase* *&ret_value)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **attribute_name** –The name of attribute.
- **ret_value** –The attribute value.

返回
esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_input_shape**(std::string node_name, int index, std::vector<int> &ret_value)

Get operation input shape.

参数

- **node_name** –The name of operation.
- **index** –The index of inputs
- **ret_value** –Return shape value.

返回
esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_output_shape**(std::string node_name, int index, std::vector<int> &ret_value)

Get operation output shape.

参数

- **node_name** –The name of operation.
- **index** –The index of outputs
- **ret_value** –Return shape value.

返回
esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

esp_err_t **get_operation_inputs_and_outputs**(std::string node_name, std::vector<std::string> &inputs, std::vector<std::string> &outputs)

Get the attribute of node.

参数

- **node_name** –The name of operation.
- **inputs** –The vector of operation inputs.
- **outputs** –The vector of operation outputs.

返回
esp_err_t Return ESP_OK if get successfully. Otherwise return ESP_FAIL.

std::string **get_operation_type**(std::string node_name)

Get operation type, "Conv", "Linear" etc.

参数
node_name –The name of operation

返回
The type of operation.

dl::*TensorBase* ***get_operation_parameter**(std::string node_name, int index = 1, uint32_t caps = MALLOC_CAP_DEFAULT)

Return if the variable is a parameter.

参数

- **node_name** –The name of operation
- **index** –The index of the variable
- **caps** –Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned

返回
*dl::TensorBase**

dl::*TensorBase* ***get_parameter**(std::string tensor_name, uint32_t caps = MALLOC_CAP_DEFAULT)

Get a parameter tensor by initializer name.

参数

- **tensor_name** –The initializer name.
- **caps** –Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned.

返回
*dl::TensorBase**

esp_err_t **get_operation_lut_name**(std::string node_name, std::string &lut_name, std::string attribute_name = "lut")

Resolve and validate the initializer name referenced by a LUT attribute.

参数

- **node_name** –The name of operation.
- **lut_name** –The resolved LUT initializer name.

- **attribute_name** –The name of LUT attribute.

返回
esp_err_t ESP_OK when the attribute references an existing initializer.

dl::*TensorBase* ***get_operation_lut**(std::string node_name, uint32_t caps = MALLOC_CAP_DEFAULT, std::string attribute_name = "lut")

Get LUT(Look Up Table) if the operation has LUT.

参数

- **node_name** –The name of operation
- **caps** –Bitwise OR of MALLOC_CAP_* flags indicating the type of memory to be returned
- **attribute_name** –The name of LUT attribute

返回
*dl::TensorBase**

bool **is_parameter**(std::string name)

return true if the variable is a parameter

参数
name –Variable name

返回
true if the variable is a parameter else false

const void ***get_tensor_raw_data**(std::string tensor_name)

Get the raw data of FlatBuffers::Dl::Tensor.

参数
tensor_name –The name of Tensor.

返回
uint8_t * The pointer of raw data.

dl::dtype_t **get_tensor_dtype**(std::string tensor_name)

Get the element type of tensor tensor.

参数
tensor_name –The tensor name.

返回
FlatBuffers::Dl::TensorDataType

std::vector<int> **get_tensor_shape**(std::string tensor_name)

Get the shape of tensor.

参数
tensor_name –The name of tensor.

返回
std::vector<int> The shape of tensor.

std::vector<int> **get_tensor_exponents**(std::string tensor_name)

Get the exponents of tensor.

警告: When quantization is PER_CHANNEL, the size of exponents is same as out_channels. When quantization is PER_TENSOR, the size of exponents is 1.

参数
tensor_name –The name of tensor.

返回
The exponents of tensor.

dl::dtype_t **get_value_info_dtype**(std::string var_name)

Get the element type of value_info.

参数
var_name –The value_info name.

返回
dl::dtype_t

std::vector<int> **get_value_info_shape**(std::string var_name)

Get the shape of value_info.

参数
var_name –The value_info name.

返回
the shape of value_info.

int **get_value_info_exponent**(std::string var_name)

Get the exponent of value_info. Only support PER_TENSOR quantization.

参数
var_name –The value_info name.

返回
the exponent of value_info

const void ***get_test_input_tensor_raw_data**(std::string tensor_name)

Get the raw data of test input tensor.

参数
tensor_name –The name of test input tensor.

返回
uint8_t * The pointer of raw data.

const void ***get_test_output_tensor_raw_data**(std::string tensor_name)

Get the raw data of test output tensor.

参数
tensor_name –The name of test output tensor.

返回
uint8_t * The pointer of raw data.

dl::*TensorBase* ***get_test_input_tensor**(std::string tensor_name)

Get the test input tensor.

参数
tensor_name –The name of test input tensor.

返回
The pointer of tensor.

dl::*TensorBase* ***get_test_output_tensor**(std::string tensor_name)

Get the test output tensor.

参数

tensor_name –The name of test output tensor.

返回

The pointer of tensor.

std::vector<std::string> **get_test_outputs_name**()

Get the name of test outputs.

返回

the name of test outputs

std::vector<std::string> **get_graph_inputs**()

Get the graph inputs.

返回

the name of inputs

std::vector<std::string> **get_graph_outputs**()

Get the graph outputs.

返回

the name of ounputs

void **clear_map**()

Clear all map.

void **load_map**()

Load all map.

std::string **get_model_name**()

Get the model name.

返回

the name of model

int64_t **get_model_version**()

Get the model version.

返回

The version of model

std::string **get_model_doc_string**()

Get the model doc string.

返回

The doc string of model

std::string **get_model_metadata_prop**(const std::string &key)

Get the model’s metadata prop.

参数

key –The key of metadata prop

返回

The value of metadata prop

void **get_model_size**(size_t *internal_size, size_t *psram_size, size_t *psram_rodata_size, size_t *flash_size)

Get the model size.

参数

- **internal_size** –Flatbuffers model internal RAM usage
- **psram_size** –Flatbuffers model PSRAM usage
- **psram_rodata_size** –Flatbuffers model PSRAM rodate usage. If CONFIG_SPIRAM_RODATA option is on, \ Flatbuffers model in FLASH rodata will be copied to PSRAM
- **flash_size** –Flatbuffers model FLASH usage

Public Members

bool **m_param_copy**

copy flatbuffers param or not.