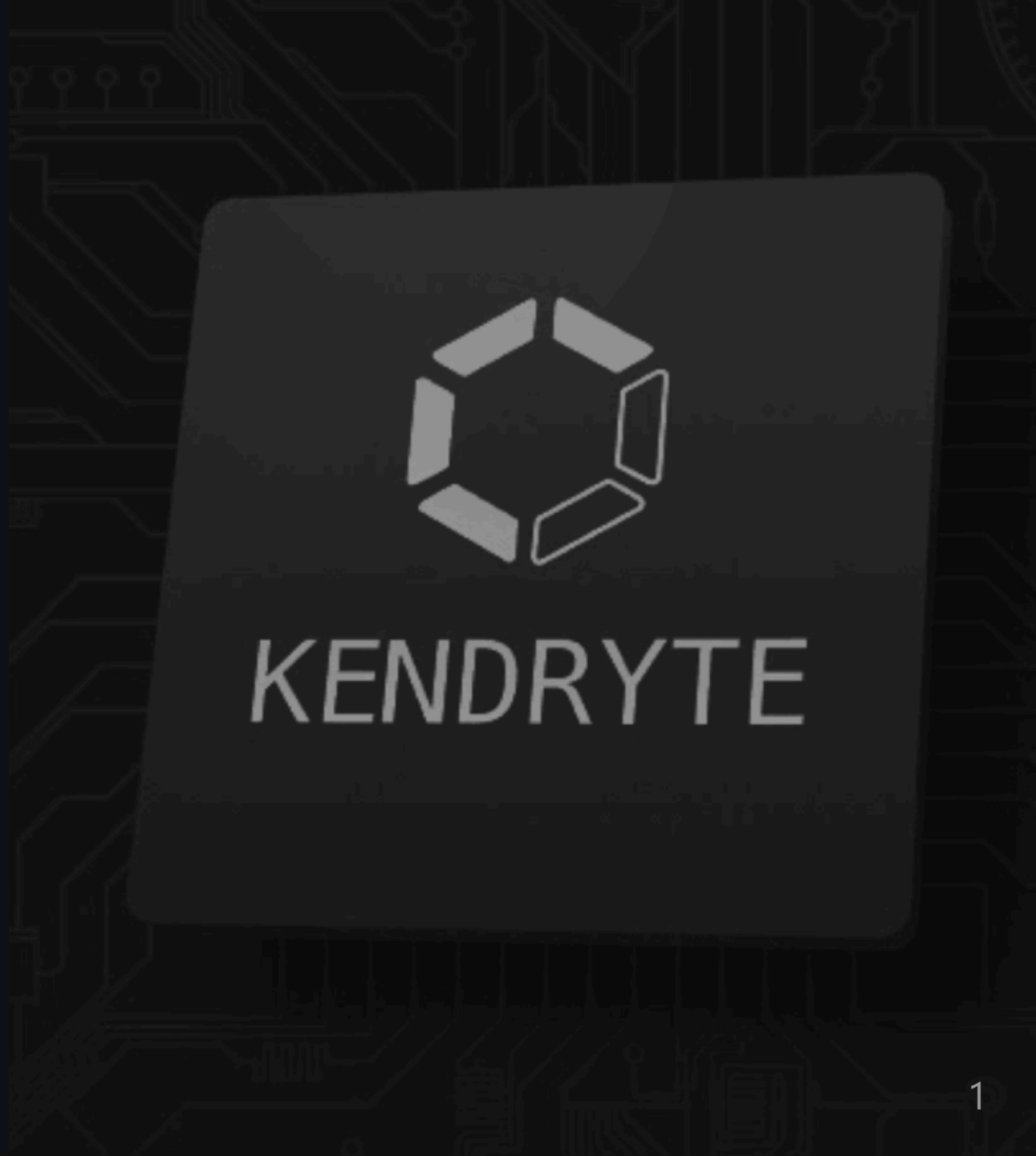
K210 AI Intro (MNIST)

Kexing Zhou



onnx / nncase / k210

onnx (Open Neural Network Exchange)

- 机器学习模型的 format 标准
- 库：`onnx`, `onnxsim`, `onnxruntime`
- 可视化工具：`netron`, `vscode-netron`

nncase 是一个为 AI 加速器设计的神经网络编译器。

- 1.0 版本提供了 K210 Runtime
- 最新版已不支持 K210

Example: MNIST inference

模型读入：

1. nncase 编译 `onnx` 到 `kmodel` 文件
2. 将 `kmodel` 文件链接
3. 用 `kpu_load_kmodel` 读入

```
INCBIN(model, "uint8_mnist.kmodel");
kpu_model_context_t mnist;
if(kpu_load_kmodel(&mnist, model_data)) {
    printf("\nmodel init error\n");
    while(1);
}
```

模型运行：

- 使用 `kpu_run_kmodel` 运行
- 用回调函数设置 `flag`
- 不需要设输入数据大小（模型里有）

```
flag = 0;
kpu_run_kmodel(
    &mnist,          // Context
    &input_data,     // 输入数据
    DMAC_CHANNEL5,   // 占用 DMA
    ai_done,         // 回调函数 (flag<=1)
    NULL             // 其它参数
);
while(!flag);
```

Example: MNIST inference

获取输出：

- 用 `kpu_get_output` 函数获取输出
- `output_ptr` 是指向 K210 AI 公共 IO 区域的指针

```
float * output_ptr; size_t output_size;
kpu_get_output(
    &model,
    0,
    &output_ptr,
    &output_size
);
```

关于 K210 AI RAM:

- 专用 RAM：2 MB，用于模型输入输出
- 通用 RAM：6 MB，用于储存模型权重
- 算子支持和加速能力参考 NNCASE 的老版本文档，并不是所有算子都能被加速

Example: MNIST inference

轮流显示 MNIST 的数字并输出推理结果

如何训练 + 部署自己的模型？

模型训练

使用 pytorch，构建一个简单的模型：

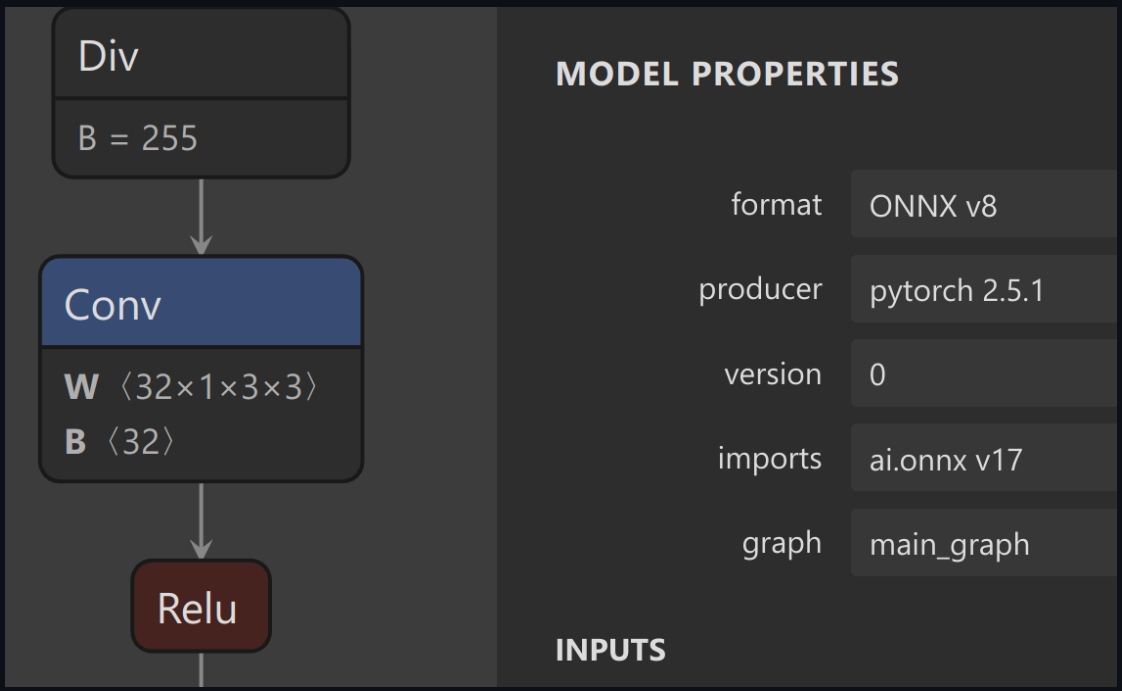
为了方便推理：

1. 输入 x 手动转换成 uint8
2. 在模型内将 x 转换回 float 再计算

```
class Net(nn.Module):
  def __init__(self):
    super().__init__()
    self.seq = nn.Sequential(
      nn.Conv2d(1,32,3), nn.BatchNorm2d(32),
      nn.ReLU(), nn.MaxPool2d(2),
      nn.Conv2d(32,64,3), nn.BatchNorm2d(64),
      nn.ReLU(), nn.MaxPool2d(2),
      nn.Conv2d(64,192,3), nn.BatchNorm2d(192),
      nn.ReLU(), nn.MaxPool2d(2),
      nn.Flatten(start_dim=1),
      nn.Linear(192,10)
    )
  def forward(self,x):
    # 为了方便推理，输入 x 是 uint
    # 先将其转换为 float，再归一化
    x = x.float() / 255
    x = self.seq(x)
    return x
```

模型导出

- 使用 pytorch 的 ONNX 功能导出模型
- 使用 `netron` 或 `vscode-netron` 检查




```python
model.eval()
input_tensor = (
    dataset[0][0].unsqueeze(0) * 255
).type(torch.uint8)

# 用一个张量导出计算图，生成 `onnx` 文件
torch.onnx.export(
    model,
    (input_tensor,),
    "my_model.onnx",
    input_names=["input"]
)

# 保存输入张量用于 nncase 量化
np.save('data.npy', np.array(input_tensor))
```

nncase 编译环境搭建

只有 nncase-v1 支持 kpu，需要安装 nncase 1.x 版本

- 只支持 3.8 版本的 Python

```
conda create -y -n nncase python==3.8
pip3 install nncase==1.9.0.20230322 onnx onnxsim scikit-learn
```

保证 k210 sdk 和编译用的 nncase 版本一致：

- `paddlepi` 默认用 `nncase-1.0.0-beta2`，这太老了
- 从 nncase 的 Release 里下载预编译的 runtime
  - `nncaseruntime-riscv64-none-k210.zip`
- 替换 sdk 里面 `nncase/v1` 为下载的 `1.9.0` 版本压缩包。

nncase 编译

编译的流程参考新版本 nncase 的 colab，但由于版本不一致，需要做一些修改：

```
# 没有 input file
# compile_options.input_file = ''

# 使用老版本的 量化选项
ptq_options = nncase.PTQTensorOptions()
ptq_options.calibrate_method = "no_clip" # "Kld"
ptq_options.samples_count = len(calib_data[0])
ptq_options.set_tensor_data(np.asarray(calib_data).tobytes())
```

完整版本

nncase 编译

如果不出意外，编译结果如右边所示

- 编译错误会 Runtime Error 然后闪退
- 可以用 nncase 自带仿真器验证

对比量化前后的输出差异

```
# 量化前
tensor([[ -3.5948,  -8.1992,  -6.5284,
           6.5337, -12.3460,   8.8779,
          -6.0821,  -7.9052,  -1.6330,
          -3.8634]])
# 量化后
array([[ -3.5790968,  -8.073776 ,  -6.5755496,
          6.4090815, -12.235517 ,   8.82289  ,
         -6.0761404,  -7.8240714,  -1.581461 ,
         -3.8288012]])
```

```
python compile.py my_model.onnx \
    -i data.npy -o out
```

```
SUMMARY
INPUTS
0        input    u8[1,1,28,28]
OUTPUTS
0        40       f32[1,10]

MEMORY USAGES
.input    784.00 B        (784 B)
.output    40.00 B        (40 B)
.data     123.27 KB       (126224 B)
MODEL     136.19 KB       (139456 B)
TOTAL     260.26 KB       (266504 B)
```

最简单工程

```c
#include "kpu.h"
#include "sleep.h"
#include "stdio.h"
#include "sysctl.h"
#define INCBIN_STYLE INCBIN_STYLE_SNAKE
#define INCBIN_PREFIX
INCBIN(model, "final.kmodel");
INCBIN(infer, "infer.bin");
kpu_model_context_t mnist;

volatile uint32_t g_ai_done_flag;
static void ai_done(void *ctx) { g_ai_done_flag = 1; }

int main(void) {
    sysctl_pll_set_freq(SYSCTL_PLL0, 800000000UL);
    sysctl_pll_set_freq(SYSCTL_PLL1, 400000000UL);
    sysctl_clock_enable(SYSCTL_CLOCK_AI);
    plic_init();
    printf("Loading model %s\n", model_data);
    if(kpu_load_kmodel(&mnist, model_data) != 0) {
        printf("Unable to load model\n");
        while(1);
    }
    sysctl_enable_irq();
    printf("Load model finish\n");
    for(int i = 0; i < 100; i++) {
        printf("Run example %d\n", i);
        printf("[01] Send data to model\n");
        g_ai_done_flag = 0;
        kpu_run_kmodel(
            &mnist, &infer_data[i * 28 * 28],
            DMAC_CHANNEL5, ai_done, NULL);
        printf("[02] Waiting finish\n");
        while(!g_ai_done_flag);
        float * output; size_t output_size;
        printf("[03] Get output\n");
        kpu_get_output(&mnist, 0, &output,
            &output_size);
        float max_value = output[0];
        int max_pos = 0;
        for(int j = 0; j < 10; j++) {
            if(output[j] > max_value) {
                max_value = output[j]; max_pos = j;
            }
        }
        printf("[04] Result: max_pos=%d\n", max_pos);
        msleep(1000);
    }
    return 0;
}
```

运行结果

```
Run example 0
[01] Send data to model
[02] Waiting finish
[03] Get output
[04] Result: max_pos=5
Run example 1
[01] Send data to model
[02] Waiting finish
[03] Get output
[04] Result: max_pos=0
Run example 2
[01] Send data to model
[02] Waiting finish
[03] Get output
[04] Result: max_pos=4
```

```
for i in range(5):
    print(dataset[i][1])

5
0
4
1
9
```